ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Μελέτη και Σύγκριση Επιδόσεων Κατανεμημένων Βάσεων Δεδομένων Σε Υπολογιστικές Υπηρεσίες Νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Π. Σεριάτος

Επιβλέπων :Θεοδώρα Α. Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π

Αθήνα, Οκτώβριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Μελέτη και Σύγκριση Επιδόσεων Κατανεμημένων Βάσεων Δεδομένων Σε Υπολογιστικές Υπηρεσίες Νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Π. Σεριάτος

Επιβλέπων :Θεοδώρα Α. Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2 Οκτωβρίου 2014

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Δημήτριος Ασκούνης	Συμεών Παπαβασιλείου
Καθηγήτρια Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2014

.....

Γεώργιος Π. Σεριάτος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η ραγδαία αύξηση του όγκου των δεδομένων τα τελευταία χρόνια έχει οδηγήσει στην ανάπτυξη νέων συστημάτων βάσεων δεδομένων. Οι παραδοσιακές σχεσιακές βάσεις παρά την ευρεία αποδοχή τους και την πληθώρα χαρακτηριστικών που προσφέρουν, παρουσιάζουν αδυναμίες όταν καλούνται να διαχειριστούν ιδιαίτερα μεγάλο όγκο δεδομένων. Προκειμένου να γίνει εφικτή η διαχείρισή και η εξαγωγή χρήσιμων συμπερασμάτων από τα δεδομένα αυτά, τα νέα συστήματα βάσεων δεδομένων διαφοροποιούνται από τα παραδοσιακά, προσαρμόζοντας τη δομή των δεδομένων και εγκαταλείποντας κάποια λειτουργικότητα προκειμένου να εξασφαλίσουν τη δυνατότητα οριζόντιας κλιμάκωσης και κατανομής των δεδομένων σε πολλαπλούς κόμβους για παράλληλη αποθήκευση και επεξεργασία.

Σκοπός αυτής της διπλωματικής εργασίας είναι η μελέτη και η ανάλυση της αρχιτεκτονικής τριών τέτοιων βάσεων, της HBase της Cassandra και της MongoDB, καθώς και η σύγκριση των επιδόσεων τους σε περιβάλλον υπολογιστικής υπηρεσίας νέφους με χρήση του μετροπρογράμματος YCSB για διάφορα σενάρια χρήσης.

Λέξεις Κλειδιά

Κατανεμημένες Βάσεις Δεδομένων , NoSQL, HBase, Cassandra, MongoDB, YCSB, θεώρημα CAP, ACID εγγυήσεις.

Abstract

The rapid growth of unstructured data over the last few years, has led to the emergence of new database management systems. Traditional relational databases, despite their wide adoption and plethora of features, begin to show weaknesses when having to deal with very large amounts of data. In order to manage those data and extract useful information out of them, the new database management systems are differentiated from the traditional ones, by relaxing the data schema constrains and the transaction guarantees in order to store and manage the data in parallel and to achieve linear scalability.

The goal of this diploma thesis is to study and analyze the architecture of three NoSQL systems: Hbase, Cassandra and MongoDB and compare their performance under a cloud service by using the YCSB benchmark.

Key Words

Distributed Database Management Systems, NoSQL, HBase, Cassandra, MongoDB, YCSB, CAP theorem, Cloud Service, ACID properties.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την καθηγήτρια μου κα. Θεοδώρα Βαρβαρίγου για την εμπιστοσύνη που μου έδειξε, την ενθάρρυνση και φυσικά τη δυνατότητα που μου παρείχε να ασχοληθώ με τεχνολογίες αιχμής μέσω του εργαστηρίου της.

Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα τον μεταδιδακτορικό κ. Γιώργο Κουσιουρή για την συνεχή καθοδήγηση, την υπομονή, τις πολύτιμες συμβουλές και τη συνολική συμβολή του στην ολοκλήρωση της εργασίας μου.

Τέλος θα ήθελα να ευχαριστήσω πολύ την οικογένειά μου για την συνεχή υποστήριξη σε όλα τα βήματα μου καθώς και τους φίλους μου που ήταν κοντά μου σε όλη αυτή την προσπάθεια και μου έδιναν κουράγιο.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

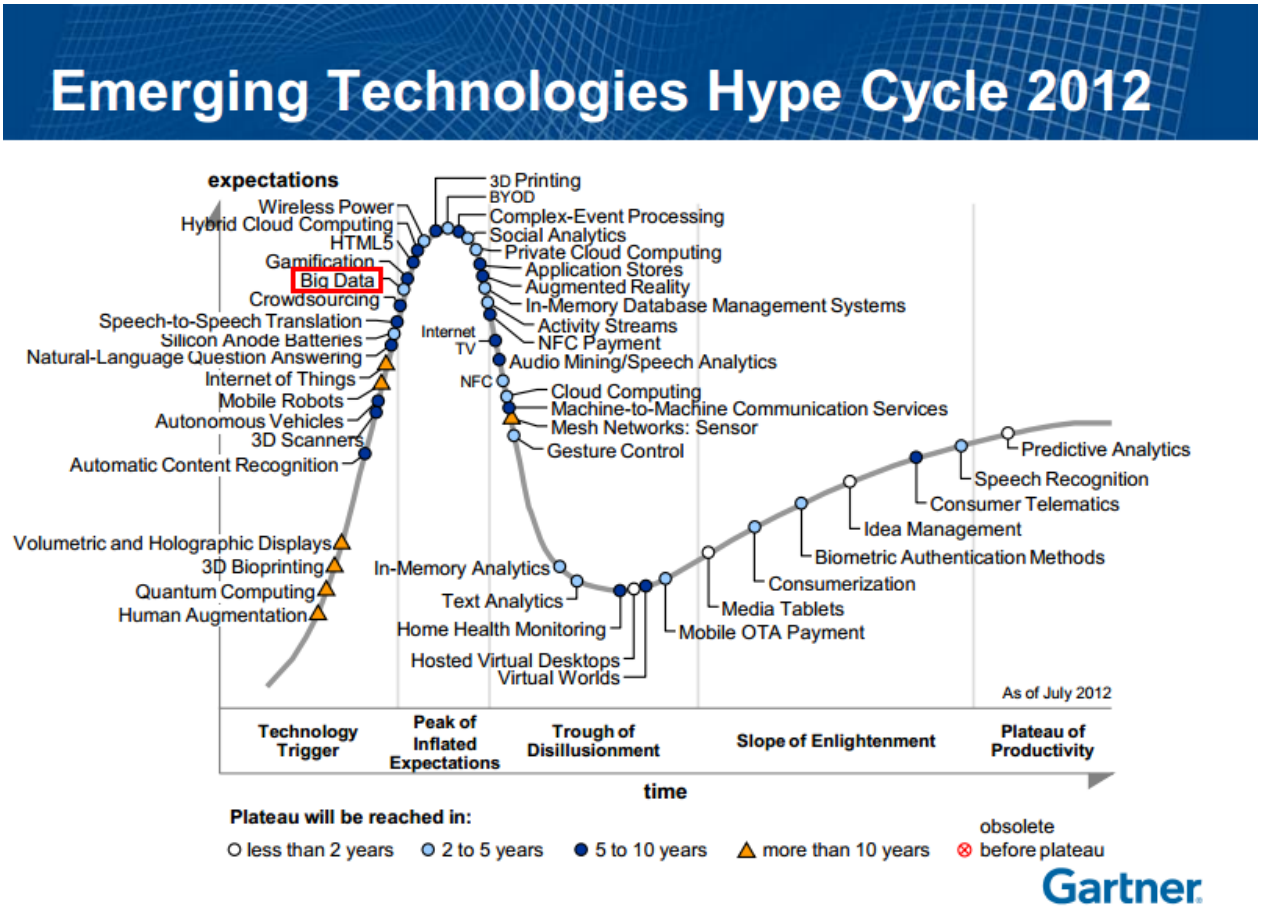
Κεφάλαιο 1
Εισαγωγή

1.1 Εποχή των Δεδομένων

Τα τελευταία χρόνια μπορούμε να πούμε ότι ζούμε στην εποχή των δεδομένων. Υπολογίζεται ότι το συνολικό μέγεθος των δεδομένων στο "ψηφιακό σύμπαν" το 2012 ανήλθε στα 2.84 zettabytes (δισεκατομμύρια terabytes) με τις προβλέψεις για το 2020 να φτάνουν τα 40 ZB [1]. Το facebook συγκεντρώνει 300 petabytes (χιλιάδες terabytes) δεδομένων από τα οποία επεξεργάζεται τουλάχιστον 1 PB κάθε μήνα και αριθμεί πάνω από ένα δισεκατομμύριο ενεργούς χρήστες[2]. Το κέντρο δεδομένων στον Μεγάλο Επιταχυντή Αδρονίων (LHC) στο CERN συγκεντρώνει κάθε χρόνο 15 petabytes δεδομένων για επεξεργασία [3]. Ο όρος “Big Data” χρησιμοποιείται για μία από τις πιο αναδυόμενες τεχνολογίες στον χώρο της πληροφορικής (Σχήμα 1.1) προκειμένου να περιγράψει τη συγκέντρωση και την ανάλυση ιδιαίτερα μεγάλων όγκων δεδομένων και την εξαγωγή συμπερασμάτων, συσχετίσεων και τάσεων που αυτά παρουσιάζουν.

Πολλοί τομείς του επιστημονικού χώρου όπως η μετεωρολογία, η γονιδιωματική, η εξομοίωση φυσικών πειραμάτων κ.α. περιλαμβάνουν τέτοιους πολύ μεγάλους όγκους δεδομένων στα προβλήματα που αντιμετωπίζουν. Μεγάλες εταιρίες του διαδικτύου όπως η Google, η Amazon και το Facebook και πολλές ακόμη συγκεντρώνουν δεδομένα από εκατομμύρια χρήστες και η ανάγκη διαχείρισής τους τις οδήγησε στη δημιουργία και την ανάπτυξη νέων συστημάτων γι αυτόν τον σκοπό αφού αυτά δεν μπορούσαν να αξιοποιηθούν αποτελεσματικά από τα παραδοσιακά συστήματα βάσεων δεδομένων . Επίσης αναγκαιότητα για επεξεργασία “Big Data” προκύπτει και σε τομείς της οικονομίας αλλά και σε επιχειρήσεις που αναγνωρίζουν την αξία της νέας αυτής τεχνολογίας.

Η αποθήκευση των δεδομένων έγινε σταδιακά αρκετά φτηνότερη και παράλληλα πλέον υπάρχουν τα κατάλληλα εργαλεία για την επεξεργασία τους και την εξαγωγή χρήσιμων συμπερασμάτων από αυτά. Έτσι, όλο και περισσότερες επιχειρήσεις χρησιμοποιούν λύσεις επιχειρησιακής νοημοσύνης (Business intelligence) αναγνωρίζοντας ότι η συγκέντρωση και η ανάλυση δεδομένων μπορεί να τις βοηθήσει να κινηθούν προς τις σωστές κατευθύνσεις και να πάρουν καλύτερες αποφάσεις, τόσο σε ότι αφορά τη λειτουργία τους, όσο και απέναντι στον ανταγωνισμό. Η επιχειρησιακή νοημοσύνη περιλαμβάνει λειτουργίες όπως εξόρυξη δεδομένων (data mining) καθώς και εξαγωγή συσχετίσεων των δεδομένων και τάσεων που αυτά παρουσιάζουν (analytics).



Σχήμα 1.1: Big Data Hype Cycle

Πηγή: http://www.forbes.com/sites/danmunro/2013/04/28/big-problem-with-little-data/

Οι παραδοσιακές σχεσιακές βάσεις παρουσιάζουν αδυναμίες στις παραπάνω περιπτώσεις, αφενός λόγω του όγκου των δεδομένων, αφετέρου λόγω της δομής τους που συχνά δεν είναι προκαθορισμένη. Το κενό αυτό έρχονται να καλύψουν τα τελευταία χρόνια συστήματα που μπαίνουν στην γενική κατηγορία NoSQL τα οποία εγκαταλείπουν κάποια από τα γνώριμα χαρακτηρίστηκα των σχεσιακών βάσεων προκειμένου να εξασφαλίζουν τη δυνατότητα παράλληλης-κατανεμημένης αποθήκευσης και επεξεργασίας δεδομένων χωρίς πολλούς περιορισμούς ως προς την δομή που αυτά πρέπει να έχουν.

Κεφάλαιο 2
Χαρακτηριστικά συστημάτων RDBMS και NoSQL

2.1 RDBMS

Οι σχεσιακές βάσεις δεδομένων κυριαρχούν στον χώρο των βάσεων δεδομένων τα τελευταία 30 χρόνια γεγονός που δεν είναι τυχαίο, αφού παρέχουν μία πληθώρα χαρακτηριστικών που τις καθιστούν κατάλληλες και αποδοτικές για πολλές εφαρμογές. Το 1970 ο Edgar F. Codd πρότεινε και θεμελίωσε το σχεσιακό μοντέλο όσο ακόμα δούλευε στην IBM[4]. Όμως η εταιρία του άργησε να υλοποιήσει τις ιδέες του με το SystemR, με αποτέλεσμα, η πρώτη εμπορικά διαθέσιμη σχεσιακή βάση δεδομένων, βασισμένη πάνω στο μοντέλο του, να είναι η Oracle το 1979. Η εξέλιξή τους από τότε ήταν ραγδαία και ενσωματώθηκαν ευρέως σε κάθε είδους εφαρμογή.

Τα βασικά τους χαρακτηριστικά που οδήγησαν στην καθιέρωσή τους είναι τα εξής:

- Απλότητα σε σύγκριση με τα παλαιότερα μοντέλα δεδομένων, όπως το δικτυακό και το ιεραρχικό,η οποία διευκόλυνε τη δουλειά των προγραμματιστών.
- Εκτέλεση δηλωτικών επερωτήσεων μέσω γλώσσας ανώτερου επιπέδου (SQL) για το τι ενέργεια θέλει ο χρήστης να κάνει, χωρίς να ορίζει το μηχανισμό με τον οποίο αυτή θα πραγματοποιηθεί.
- Αποφεύγεται η επανάληψη δεδομένων με κανονικοποιημένες βάσεις και σωστό σχεδιασμό.
- Δυνατότητα επιβολής κανόνων ακεραιότητας στα δεδομένα.
- Δυνατότητα επιβολής κανόνων ασφάλειας.
- Αξιοπιστία στις συναλλαγές που πραγματοποιούνται στη βάση αφού εξασφαλίζονται ιδιότητες όπως ατομικότητα ,συνέπεια, απομόνωση, μονιμότητα (ACID).
- Δυνατότητα ταυτόχρονης πρόσβασης από διαφορετικά σημεία στα δεδομένα, ενώ παράλληλα διασφαλίζεται η συνέπεια (consistency).

Οι σχεσιακές βάσεις παρ' όλα αυτά αρχίζουν να παρουσιάζουν αδυναμίες στις επιδόσεις όταν ο όγκος των δεδομένων, των χρηστών και των ερωτημάτων που πραγματοποιούνται γίνεται μεγάλος. Για να αντιμετωπιστούν οι αυξημένες απαιτήσεις υιοθετούνται πρακτικές όπως η προσθήκη εξυπηρετητή σκλάβου (slave server), η προσθήκη κρυφής μνήμης (cache), η

αντικατάσταση των εξυπηρετητών με άλλους υψηλότερων τεχνικών προδιαγραφών καθώς και ο κατακερματισμός (sharding) της βάσης. Αυτά όμως έχουν σαν αποτέλεσμα να αυξάνονται δυσανάλογα η πολυπλοκότητα και το κόστος και πιθανώς δεν δίνουν πάντα και το επιθυμητό αποτέλεσμα. Επίσης, αδυναμία παρουσιάζουν σε περιπτώσεις στις οποίες η δομή των δεδομένων, είτε είναι τέτοια που από τη φύση της δεν ταιριάζει στην δομή με τους πίνακες των σχεσιακών βάσεων, είτε δεν μπορεί να καθοριστεί επαρκώς από την αρχή λειτουργίας ενός συστήματος και επομένως χρειάζεται να υπάρχει αυξημένη ευελιξία. Η αλλαγή της δομής των δεδομένων σε ένα σχεσιακό σύστημα είναι μία διαδικασία που χρειάζεται κάποιο χρόνο, κατά τον οποίο η βάση είτε θα είναι εκτός λειτουργίας (downtime) είτε θα παρέχει πολύ περιορισμένες λειτουργίες. Στα NoSQL συστήματα αυτοί οι περιορισμοί είναι συνήθως μηδαμινοί.

2.2 NoSQL

Για να αντιμετωπιστούν οι δυσκολίες που αναφέρθηκαν, αναπτύχθηκαν συστήματα τα οποία μπήκαν στη γενική κατηγορία που ονομάζεται NoSQL. Παρ' ότι ο όρος δεν περιγράφει τις ιδιότητες αυτών των συστημάτων, χρησιμοποιείται από τη στιγμή που κάτι διαφέρει από τις παραδοσιακές σχεσιακές βάσεις. Μάλιστα επειδή σε αρκετές περιπτώσεις αναπτύσσονται διεπαφές οι οποίες δίνουν τη δυνατότητα στον χρήστη να εκτελεί γνώριμα ερωτήματα τύπου SQL στα δεδομένα, ο όρος NoSQL σταδιακά έχει αρχίζει να διαβάζεται ως Not-only-SQL, δηλαδή όχι-μόνο-SQL. Οι διαφορές από σύστημα σε σύστημα μπορεί να είναι αρκετές, όμως σε αυτή την κατηγορία βρίσκει κανείς χαρακτηριστικά όπως:

- Ικανότητα διαχείρισης μεγάλου όγκου δεδομένων (big-data, web-scale data).
- Ελεύθερη δομή του συστήματος δεδομένων.
- Τα συστήματα είναι κατανεμημένα (distributed).
- Υποστήριξη εύκολης επανάληψης-αντιγραφής (replication) των δεδομένων
- Παροχή μηχανισμών που επιτρέπουν την αυτόματη επαναφορά του συστήματος σε περιπτώσεις που κάποιος κόμβος σταματάει να είναι διαθέσιμος (πχ. κατάτμηση δικτύου ή αποτυχία υλικού) .
- Οριζόντια αναβάθμιση των συστημάτων, δηλαδή για να ανταποκριθεί το σύστημα σε επιπλέον ανάγκες αρκεί η προσθήκη επιπλέον μηχανημάτων στα ήδη υπάρχοντα. Αυτή αποτελεί διαφορετική προσέγγιση σε σχέση με τη συνήθη κάθετη αναβάθμιση των συστημάτων RDBMS η οποία γίνεται με αναβάθμιση επεξεργαστών, προσθήκη RAM

κ.α. και η οποία έχει σαν αποτέλεσμα το κόστος να αυξάνεται δυσανάλογα σε σχέση με τις δυνατότητες που προστίθενται. Επιπλέον, η οριζόντια δυνατότητα αναβάθμισης δεν βάζει περιορισμό στο όριο που μπορεί να αναπτυχθεί ένα τέτοιο σύστημα και θεωρητικά μπορούν να προστεθούν όσοι κόμβοι και αν χρειάζονται. Επίσης δίνει και δυνατότητα εύκολης υποβάθμισης του σε περίπτωση που οι ανάγκες μειωθούν.

- Υποστηρίζονται οι λεγόμενες BASE (σε αντιδιαστολή με τις ACID) ιδιότητες στις συναλλαγές που πραγματοποιούνται στη βάση. Το BASE αποτελεί ακρώνυμο των όρων **B**asic **A**vailability, **S**oft-state, **E**ventual-consistency. Basic availability σημαίνει ότι ανά πάσα στιγμή το σύστημα είναι διαθέσιμο αλλά όχι κατ' ανάγκη όλα τα στοιχεία του. Soft-state σημαίνει ότι τα δεδομένα δεν είναι πάντα αποθηκευμένα σε δίσκο, αλλά αν χρειαστεί μπορούν να ξαναδημιουργηθούν. Αυτή η τακτική ακολουθείται και από τα 3 συστήματα που εξετάζουμε σε αυτή την εργασία προκειμένου αυτά να παρέχουν εγγυήσεις μονιμότητας στα δεδομένα χωρίς να επηρεάζουν σημαντικά την απόδοση του συστήματος. Μια διαφορετική ερμηνεία είναι ότι το σύστημα μπορεί να αλλάξει χωρίς κάποια παρεμβολή από τον χρήστη. Σε κάποιες περιπτώσεις ανάλογα τις επιλογές που κάνει ο προγραμματιστής μπορεί να σημαίνει ότι και κάποια δεδομένα μπορεί να διαγράφονται εφόσον δεν ανανεώνονται. Τέλος, ο όρος eventual consistensy σημαίνει ότι κατά βάση δεν υπάρχει ανά πάσα στιγμή συνέπεια στα δεδομένα αλλά εν τέλει. Ο όρος BASE είναι επινοημένος από τον Eric Brewer[5] και επειδή είναι γενικός, κάθε σύστημα θα πρέπει να εξετάζεται ξεχωριστά για τις ιδιότητες που έχει και τις δυνατότητες που δίνει.

Τα NoSQL συστήματα δεν εμφανίστηκαν για να αντικαταστήσουν τα σχεσιακά, αλλά για να τα συμπληρώσουν. Όπως και σε πολλές άλλες περιπτώσεις έτσι και εδώ, το ζητούμενο είναι να γίνει η επιλογή του κατάλληλου εργαλείου για την λύση του κάθε προβλήματος. Έτσι στις περισσότερες περιπτώσεις είναι απαραίτητο να γίνει μια εκτίμηση των δυνατοτήτων των διαθέσιμων συστημάτων και στη συνέχεια να υλοποιείται ένας συνδυασμός τους ώστε κάθε επιμέρους ανάγκη να καλύπτεται από το σύστημα που της ταιριάζει καλύτερα. Έτσι, όσο τα δεδομένα του διαδικτύου αυξάνονται με εκθετικούς ρυθμούς, τόσο πιο πολλές θα είναι και οι περιπτώσεις όπου θα καθίσταται καλή επιλογή η χρήση τους. Η ανάπτυξη των NoSQL συστημάτων είναι πολύ γρήγορη τα τελευταία χρόνια. Τη στιγμή της συγγραφής, η λίστα με τις NoSQL βάσεις δεδομένων περιλαμβάνει πάνω από 150 καταχωρήσεις[6] όταν πριν από τρία χρόνια οι καταχωρήσεις έφταναν τις 35. Ο διαχωρισμός τους σε κατηγορίες γίνεται συνήθως με βάση, είτε τις δυνατότητες τους, είτε με τον τρόπο με τον οποίο αποθηκεύονται τα δεδομένα.

Κεφάλαιο 3
Είδη NoSQL

3.1 Διαχωρισμός με βάση σχεδιαστικές επιλογές

Από τη στιγμή που κάποιος αποφασίσει πως οι παραδοσιακές σχεσιακές βάσεις δεν ταιριάζουν σε κάποια από τις ανάγκες του και στραφεί στις NoSQL, θα δει πως η ποικιλία απ' όπου καλείται να επιλέξει είναι πολύ μεγάλη.

3.1.1 Θεώρημα CAP

Ένας συνήθης διαχωρισμός που γίνεται βασίζεται στο πολυσυζητημένο θεώρημα CAP γνωστό και ως θεώρημα Brewer[5]. Το 2000 στο συμπόσιο με θέμα τις αρχές των κατανεμημένων συστημάτων (PODS), ο Eric A. Brewer εξέφρασε την άποψη ότι κάθε διαμοιρασμένο σύστημα μπορεί να έχει δύο από τις τρεις εξής βασικές ιδιότητες:

- **C**onsistency - Συνέπεια.
- **A**vailability – Διαθεσιμότητα.
- Tolerance to network **P**artitions - Ανοχή στις κατατμήσεις που οφείλονται στο δίκτυο.

Στη συνέχεια το 2002 οι N. Lynch και S. Gilbert έδωσαν μία απόδειξη για την εκτίμηση του Brewer οπότε και αυτή καθιερώθηκε σαν θεώρημα CAP [7].Αναλυτικότερα, η ιδιότητα της συνέπειας έχει την έννοια, ότι κάθε ερώτημα που γίνεται στη βάση, θα διαβάζει τα τελευταία δεδομένα που καταχωρήθηκαν σε αυτήν και όχι παλαιότερα. Για την απόδειξη του CAP θεωρήματος χρησιμοποιήθηκε η έννοια της ατομικότητας(Α από τις ACID ιδιότητες). Δηλαδή, το σύστημα θα πρέπει να συμπεριφέρεται σαν να αποτελείτο από έναν κόμβο στον οποίο εγγραφές και αναγνώσεις εξυπηρετούνται με τη σειρά που πραγματοποιήθηκαν. Η ιδιότητα της διαθεσιμότητας έχει την έννοια ότι κάθε αίτημα που γίνεται στη βάση θα πρέπει να λαμβάνει απάντηση, είτε αυτό έχει επιτύχει, είτε έχει αποτύχει. Τέλος η ανοχή στις κατατμήσεις σημαίνει ότι σε περίπτωση που κάποιο μέρος του συστήματος αποκοπεί από το υπόλοιπο λόγω βλάβης στο δίκτυο (ή και βλάβης σε κάποια μηχανήματα), τότε το σύστημα θα πρέπει να είναι σε θέση να συνεχίσει να εξυπηρετεί αιτήματα και να λειτουργεί. Τα επόμενα χρόνια, που αναπτύχθηκαν σε πολύ μεγαλύτερο βαθμό τέτοιου είδους συστήματα, έγινε κοινώς αποδεκτό, ότι οι δημιουργοί των συστημάτων θα έπρεπε όντως να επιλέξουν ποιες δύο από τις τρεις ιδιότητες θα υποστήριζε αποτελεσματικά το σύστημά τους σε περιπτώσεις που κάτι δεν πήγαινε καλά. Η ανοχή στις

κατατμήσεις ήταν κάτι που από τη φύση τους έχουν ανάγκη τα κατανεμημένα συστήματα, επομένως έπρεπε να επιλέξουν σύμφωνα με τις ανάγκες για τις οποίες προοριζόταν το σύστημα, για το ποια από τις δύο ιδιότητες Availability - Consistency θα υποστήριζαν κατά προτεραιότητα (και φυσικά όχι κατ' αποκλειστικότητα).

Παρά το γεγονός ότι το θεώρημα CAP ορίζει ότι πρέπει να διαλέξει κανείς ανάμεσα σε διαθεσιμότητα (Availability) και σε συνέπεια (Consistency) αν θέλει να έχει ανοχή στις κατατμήσεις, αυτό δεν σημαίνει ότι η επιλογή της διαθεσιμότητας θα έχει σαν αποτέλεσμα να μην εξασφαλίζεται καμία συνοχή. Αυτό που συμβαίνει είναι ότι χρησιμοποιούνται μηχανισμοί οι οποίοι παρέχουν συνέπεια τελικώς (eventual consistency), δηλαδή τα δεδομένα που ενημερώνονται σε κάποιο κόμβο θα ενημερωθούν σταδιακά και στους υπόλοιπους κόμβους μέσω των μηχανισμών αυτών . Αυτή η προσέγγιση μπορεί να μην παρέχει εγγυήσεις ότι τα δεδομένα που διαβάζονται σε κάποια χρονική στιγμή είναι τα πιο ενημερωμένα, αλλά τουλάχιστον κάνει το μεγαλύτερο μέρος της βάσης να είναι σε συνεπή κατάσταση.

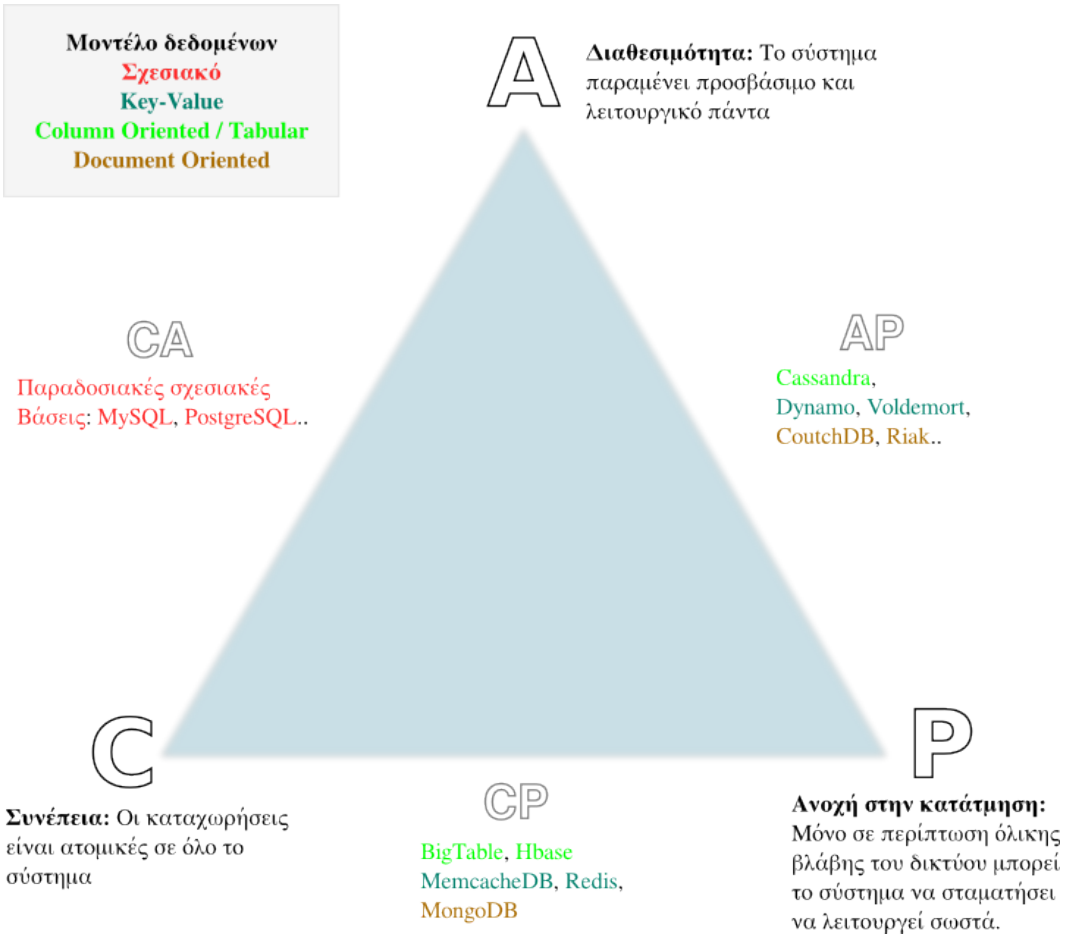
Στην πράξη αρκετά από αυτά τα συστήματα δίνουν τη δυνατότητα στο χρήστη της εφαρμογής (client) να αλλάξει την προκαθορισμένη συμπεριφορά του συστήματος, ανάλογα με τις ανάγκες του. Έτσι ένα σύστημα όπως το Cassandra που ανήκει ξεκάθαρα στην κατηγορία AP (διαθέσιμο και με ανοχή στις κατατμήσεις) μπορεί να ρυθμιστεί για συγκεκριμένες ενέργειες να δίνει περισσότερες εγγυήσεις συνέπειας εγκαταλείποντας την διαθεσιμότητα σε περίπτωση κατάτμησης. Αντίστοιχα συστήματα όπως τα HBase και MongoDB εντάσσονται στην κατηγορία CP, όμως σε περίπτωση κατάτμησης ή βλάβης υλικού οι υπόλοιποι κόμβοι μπορούν να συνεχίσουν να ανταποκρίνονται στα αιτήματα που γίνονται στη βάση. Με την αυστηρή όμως έννοια του CAP θεωρούνται μη διαθέσιμα σε μία τέτοια περίπτωση. Σε γενικές γραμμές τα περισσότερα συστήματα παρέχουν επιλογές και μπορούν να ρυθμιστούν κατάλληλα και να αλλάξουν την κατηγορία CAP στην οποία ανήκουν ανάλογα με τις ιδιαίτερες ανάγκες που προκύπτουν.

Στην παρακάτω λίστα παρατίθενται μερικά από τα πιο γνωστά συστήματα βάσεων δεδομένων, και πώς αυτά, έχοντας την προκαθορισμένη τους συμπεριφορά, εντάσσονται στις τρεις κατηγορίες CA, CP, AP, που προκύπτουν από το θεώρημα:

1. **CA**: Παραδοσιακά RDBMS (MySQL, Postgres κλπ), Aster Data, Greenplum, Vertica, Oracle Coherence κ.α.
2. **CP**: BigTable, Hbase, Hypertable, MongoDB, Terrastore, MemcacheDB, Redis, Scalaris, BerkeleyDB κ.α.

3. **AP**: Cassandra, CouchDB, Riak, SimpleDB, Dynamo, Voldemort, Tokyo Cabinet, KAI κ.α.

Τα παραπάνω αποτυπώνονται παραστατικά στο σχήμα 3.1.



Σχήμα 3.1: Θεώρημα CAP

(Πηγή:http://blog.nahurst.com/visual-guide-to-nosql-systems)

3.1.2 PACELC: Συμπλήρωση του CAP

Το θεώρημα CAP παρ' ότι είναι γενικώς αποδεκτό, δεν εξετάζει όλες τις δυνατές επιλογές δυνατοτήτων που κάνουν οι σχεδιαστές αυτών τον συστημάτων. Έτσι υπάρχουν κάποια τα οποία δεν μπορούν να ενταχθούν ικανοποιητικά σε κάποια από τις τρεις κατηγορίες που δημιουργεί το θεώρημα. Ο D.Abadi σε μία ανάρτηση του[8] αναλύει κάποιες ασάφειες που δημιουργεί το απλοποιημένο μοντέλο του CAP θεωρήματος και προτείνει έναν πιο γενικό τρόπο περιγραφής των NoSQL συστημάτων. Αρχικά, κάνει την παρατήρηση ότι τα συστήματα που μπαίνουν στην κατηγορία AP έχουν την τάση να έχουν χαλαρωμένη συνέπεια C σε όλες τις περιπτώσεις, ακόμα και όταν δεν υπάρχει κατάτμηση λόγω δικτύου(P). Αντίθετα τα συστήματα που μπαίνουν στην

κατηγορία CP, σταματούν να είναι διαθέσιμα μόνο σε περιπτώσεις που υπάρχει κατάτμηση, οπότε και προσπαθούν να διατηρήσουν τη συνέπεια στα δεδομένα. Αυτή η ασυμμετρία, κατά την εκτίμηση του, κάνει τα συστήματα CA και CP να είναι αντίστοιχα σε ότι αφορά την περιγραφή τους από το CAP θεώρημα. Η περιγραφή που προτείνει προσθέτει ένα βασικό χαρακτηριστικό που σε όλες τις περιπτώσεις είναι επιθυμητό και σε κάποιες περιπτώσεις απαραίτητο: ο χαμηλός χρόνος αντίδρασης (latency). Σαν παράδειγμα φέρνει το σύστημα PNUTS (εμπορικά Yahoo Sherpa), το οποίο φαίνεται να μην μπορεί να ενταχθεί στην κατηγοριοποίηση του CAP, αφού στις περιπτώσεις που δεν υπάρχει κατάτμηση λόγω δικτύου, το σύστημα χαλαρώνει την συνέπεια προς χάριν των χαμηλών χρόνων αντίδρασης (latency), ενώ σε περιπτώσεις που υπάρχει κατάτμηση εγκαταλείπει την διαθεσιμότητα έτσι ώστε να μην υπάρχει περαιτέρω χαλάρωση της συνέπειας. Συμπερασματικά, ο όρος που προτείνει είναι ο PACELC και η περιγραφή είναι η εξής:

Εάν υπάρχει κατάτμηση **P**, το σύστημα επιλέγει να είναι διαθέσιμο **A** ή να έχει συνέπεια **C** στα δεδομένα του; Αλλιως (**E**lse) το σύστημα διαλέγει να έχει χαμηλούς χρόνους αντίδρασης **L** ή διαλέγει να έχει συνέπεια **C** στα δεδομένα του; Πιο περιγραφικά θα μπορούσαμε να γράψουμε:

P?(A|C):(L|C)

Παρ' ότι συνήθως τα συστήματα είναι PA/EL και PC/EC και υπάρχει συσχέτιση στις επιλογές που αφορούν τη συνέπεια, υπάρχουν και συστήματα όπως το PNUTS το οποίο αλλάζει τις προτεραιότητες του σε περίπτωση που υπάρχει κατάτμηση και μπαίνει στην κατηγορία PC/EL.

3.2 Διαχωρισμός με βάση τη δομή των δεδομένων

Για να μπορέσουν τα κατανεμημένα συστήματα να έχουν αποδοτική οριζόντια κλιμάκωση, έπρεπε να απομακρυνθούν από τους παραδοσιακούς πίνακες των σχεσιακών συστημάτων οι οποίοι μεγάλωναν προσθέτοντας γραμμές. Έτσι υιοθετήθηκαν μοντέλα τα οποία έκαναν την κατανομή τον δεδομένων σε πολλά μηχανήματα απλούστερη αλλά και αποδοτικότερη. Οι βασικότερες κατηγορίες είναι οι εξής:

- Ζεύγη Κλειδού – Τιμής (Key – Value)
- Πίνακες οργανωμένοι κατά στήλες (Column - Oriented / Tabular)
- Οργάνωση κατά αρχεία (Document - Oriented)
- Γράφου (Graph)
- Αντικειμενοστρεφείς (Object Oriented)

3.2.1 Κλειδί-Τιμή (Key-Value)

Το μοντέλο κλειδιού - τιμής είναι το πιο απλό από όλες τις κατηγορίες που εξετάζουμε. Τα δεδομένα αποθηκεύονται σε ζεύγη κλειδιού - τιμής, με τρόπο παρόμοιο με μία απεικόνιση (map) ή με έναν πίνακα κατακερματισμού (hash table). Κάποια συστήματα μπορεί να επιτρέπουν η τιμή να είναι κάτι πιο πολύπλοκο, όπως για παράδειγμα μία λίστα. Επίσης μπορεί να ενσωματώνουν κάποιες δυνατότητες για τον χειρισμό των κλειδιών και των τιμών. Το γεγονός ότι το μοντέλο είναι τόσο απλό, κάνει τις βάσεις αυτού του είδους να πετυχαίνουν πολύ καλές επιδόσεις όταν η συγκεκριμένη δομή ταιριάζει στο πρόβλημα. Τα δεδομένα για να είναι κατάλληλα για μια τέτοια βάση, δεν θα πρέπει να έχουν υψηλά επίπεδα συσχετίσεων μεταξύ τους. Αν το ζητούμενο είναι να υπάρχει δυνατότητα για πολύπλοκα ερωτήματα στη βάση, τότε το μοντέλο αυτό δεν διευκολύνει.

Από τα πιο γνωστά συστήματα που ανήκουν σε αυτή την κατηγορία είναι τα memcached, membase, cachedb, Voldemort, Redis και Riak.

3.2.2 Αποθήκευση σε πίνακες κατά στήλη (Column-Oriented/Tabular)

Οι column-oriented βάσεις δεδομένων από άποψη δομής βρίσκονται κάπου ανάμεσα στους σχεσιακούς πίνακες και στο μοντέλο κλειδιού - τιμής. Θα μπορούσαμε να πούμε ότι αποτελούνται από πίνακες οι οποίοι όμως διαφέρουν από τους παραδοσιακούς των σχεσιακών βάσεων ως προς τον τρόπο με τον οποίο αποθηκεύουν τα δεδομένα. Η διαφορά βρίσκεται στο ότι στους πίνακες κατά στήλες, τα δεδομένα από μία στήλη αποθηκεύονται μαζί στο δίσκο (σε αντίθεση με τους πίνακες των σχεσιακών βάσεων που αποθηκεύονταν κατά γραμμές). Αυτή η αλλαγή, παρ' ότι φαίνεται μικρή δημιουργεί πολύ διαφορετικές συνθήκες και ιδιότητες στο σύστημα. Η προσθήκη επιπλέον στηλών είναι μια διαδικασία που δεν κοστίζει, κάθε γραμμή μπορεί να έχει διαφορετικές στήλες ή και καθόλου, και επίσης οι πίνακες μπορούν να παραμένουν αραιοί χωρίς αυτό όμως να έχει επίπτωση στον αποθηκευτικό χώρο που απαιτούν.

Αποθήκευση κατά **σειρές**:

row1	1	http://foobar.com	S2asd	FooBar	<html><head><title>FooBar...
row2	2	http://foobaz.org	asd21	FooBaz	<html><head><title>FooBaz...
row3	3	http://barbaz.com	HJuiR	Barbaz	<html><head><title>Barbaz...

...

SQL πίνακας:

url_id INTEGER PK	url VARCHAR(4096)	ref_short_id CHAR(8)	title VARCHAR(200)	content TEXT
1	http://foobar.com	S2asd	FooBar	<html><head><title>FooBar...
2	http://foobaz.org	asd21	FooBaz	<html><head><title>FooBaz...
3	http://barbaz.com	HJuiR	Barbaz	<html><head><title>Barbaz...

...

Αποθήκευση κατά **στήλες**:

Col1:url	http://foobar.com	http://foobaz.org	http://barbaz.com	...
Col2: url_short_id	S2asd	asd21	HJuiR	...
Col3:title	FooBar	FooBaz	Barbaz	...
Col4:content	<html><head><title>FooBar...	<html><head><title>FooBaz...	<html><head><title>Barbaz...	...

Σχήμα 3.2: Αποθήκευση κατά Σειρές και κατά Στήλες

(Πηγή:http://www.narendranaidu.com/2014/02/ruminating-on-column-oriented-data.html)

Ένα σημαντικό χαρακτηριστικό που έχουν τέτοια συστήματα είναι οι ενσωματωμένες δυνατότητες για συμπίεση των δεδομένων, αφού η οργάνωση κατά στήλες κάνει τα δεδομένα που βρίσκονται σε μία στήλη να είναι όμοια και έτσι οι αλγόριθμοι συμπίεσης πετυχαίνουν καλύτερα αποτελέσματα. Το χαρακτηριστικό αυτό γίνεται πολύ σημαντικό όταν ο όγκος των δεδομένων είναι μεγάλος. Είναι γενικά καλό με τις συγκεκριμένες βάσεις να υπάρχει ένα πλάνο για το πώς θα χρησιμοποιηθούν τα δεδομένα και τι ερωτήματα θα γίνονται σε αυτά, έτσι ώστε να γίνει κατάλληλη επιλογή της δομής των δεδομένων και να είναι αποδοτική η βάση στη συνέχεια. Αν δεν είναι δυνατόν να γίνει μία τέτοια εκτίμηση εκ των προτέρων, τότε τα συγκεκριμένα συστήματα μπορεί να μην αναδεικνύουν τα δυνατά τους χαρακτηριστικά.

Σε αυτή την κατηγορία ανήκουν συστήματα όπως: BigTable, Hbase, Hypertable, Cassandra κ.α.

3.2.3 Βάση Αρχείων (Document-Oriented)

Οι βάσεις οργανωμένες κατά αρχεία, όπως λέει και το όνομά τους, χρησιμοποιούν αρχεία για να αποθηκεύσουν τα δεδομένα. Τα αρχεία αυτά έχουν ένα μοναδικό αναγνωριστικό ID το καθένα και συνήθως η βάση διατηρεί ευρετήριο (index) ώστε να γίνεται γρήγορα η αναζήτηση και η ανάκτηση τους. Σαν τιμή μπορεί να περιέχουν κάθε είδους δεδομένα με μοναδικό περιορισμό να μπορούν να εκφραστούν σαν αρχείο, γεγονός που παρέχει μεγάλη ευελιξία. Τα αρχεία είναι συνήθως τύπου JSON (JavaScript Object Notation) είτε παραλλαγές όπως BSON (Binary JSON) τα οποία είναι κατάλληλα τόσο για την μεταφορά και την αποθήκευση, όσο και για την ανάγνωση και τον χειρισμό των δεδομένων, αφού είναι ένα πρότυπο με το οποίο τα δεδομένα περιγράφουν από μόνα τους τη δομή τους, χωρίς αυτό να προσθέτει κάποιο περιορισμό στη βάση. Λόγω της φύσης των αρχείων, συνήθως τέτοιου είδους συστήματα ταιριάζουν με αντικειμενοστραφείς γλώσσες προγραμματισμού. Μία ιδιαιτερότητα των βάσεων που οργανώνονται κατά αρχεία είναι ότι τα δεδομένα κατά βάση δεν είναι κανονικοποιημένα όπως στις σχεσιακές βάσεις, αλλά κάθε αρχείο προβλέπεται να διατηρεί τις περισσότερες αν όχι όλες τις πληροφορίες που μπορεί να χρειαστούν όταν πραγματοποιείται η ενέργεια που το αφορά.

Σε αυτή την κατηγορία ανήκουν συστήματα όπως : MongoDB, CouchDB, Couchbase, RethinkDB, Terrastore, Elasticsearch κ.α.

3.2.4 Γράφου (Graph)

Οι βάσεις δεδομένων αυτής της κατηγορίας δεν χρησιμοποιούνται συχνά, αφού είναι κατάλληλη επιλογή μόνο σε περιπτώσεις οπού τα δεδομένα παρουσιάζουν μεγάλη συσχέτιση μεταξύ τους και η συσχέτιση αυτή πρέπει να αποτυπωθεί. Μοντελοποιούνται με κόμβους και με σχέσεις που τους συνδέουν. Χαρακτηριστική λειτουργία σε τέτοιου είδους βάσεις είναι να διασχίζονται οι κόμβοι χρησιμοποιώντας τις σχέσεις που τους συνδέουν. Τέλος, οι κόμβοι και οι σχέσεις μπορούν να έχουν ιδιότητες στις οποίες αποθηκεύονται δεδομένα. Η κλασσική χρήση τέτοιων συστημάτων είναι σε κοινωνικά δίκτυα, όπου οι σχέσεις μεταξύ των χρηστών είναι πολλές και πολύπλοκες. Σε τέτοιες περιπτώσεις τα υπόλοιπα συστήματα βάσεων δεδομένων ταιριάζουν αρκετά λιγότερο και προσθέτουν δυσκολίες. Το γεγονός όμως ότι οι εξαρτήσεις μεταξύ των δεδομένων είναι τόσο πολλές, καθιστά δύσκολο το να γίνουν αυτά τα συστήματα κατανεμημένα, αφού σε περίπτωση κατάτμησης λόγω δικτύου οι σχέσεις που δεν θα είναι πλέον προσβάσιμες μπορεί να είναι πάρα πολλές.

Σε αυτή την κατηγορία ανήκουν συστήματα όπως : Neo4j, FlockDB, OrientDB, AllegroGraph, GraphDB...

3.2.5 Αντικειμενοστρεφείς βάσεις (object-oriented databases)

Οι αντικειμενοστρεφείς βάσεις δεδομένων αποθηκεύουν τα δεδομένα τους, όπως υποδεικνύει το όνομά τους με τη μορφή αντικειμένων, όπως αυτά χρησιμοποιούνται στις αντικειμενοστρεφείς γλώσσες προγραμματισμού. Αυτό κάνει πιο άμεση και απλή τη χρησιμοποίησή τους από εφαρμογές γραμμένες σε τέτοιου είδους γλώσσες, αφού δε χρειάζεται η χρήση επιπλέον κώδικα για την αντιστοίχιση των δεδομένων με την εφαρμογή. Το γεγονός ότι δε χρειάζεται κάποια μετάφραση των αντικειμένων που χρησιμοποιούνται, σε κάποια άλλη μορφή (πχ κατάλληλη για σχεσιακές βάσεις), κάνει τις αντικειμενοστρεφείς βάσεις να παρουσιάζουν συχνά αυξημένη επίδοση σε σχέση με άλλα συστήματα. Επίσης δεν είναι απαραίτητη η δημιουργία κλειδιών για να επιτευχθεί η συσχέτιση και η αντιστοίχιση των δεδομένων, τα οποία μπορούν να αναζητηθούν με τη χρήση δεικτών (pointers).

Οι αντικειμενοστρεφείς βάσεις δεδομένων έχουν εμφανιστεί από τα μέσα της δεκαετίας του 70 και βρήκαν ανταπόκριση σε κάποιες εφαρμογές όπως CAD (Computer Aided Design), CAM (Computer Aided Manufacturing) και σε ενσωματωμένα συστήματα, όμως ποτέ δεν έγινε εκτεταμένη χρήση τους. Αυτό οφείλεται σε κάποια μειονεκτήματα που παρουσιάζουν σε σχέση με άλλα συστήματα. Το κυριότερο από αυτά είναι ότι η βάση που δημιουργείται συσχετίζεται στενά με την εφαρμογή που τη χρησιμοποιεί και πιθανώς και με τη γλώσσα στην οποία είναι γραμμένη η εφαρμογή. Αυτό κάνει τη βάση λιγότερο ευέλικτη σε σχέση με άλλες εναλλακτικές. Επίσης για απλές περιπτώσεις βάσεων που περιέχουν απλές σχέσεις, οι αντικειμενοστρεφείς βάσεις μπορεί να είναι λιγότερο αποδοτικές και πιο πολύπλοκες σε σχέση με ένα σύστημα RDBMS.

Σε αυτή την κατηγορία ανήκουν συστήματα όπως : VelocityDB, Versand, Objectivity, Starcounter, Perst, HSS Database, Sterling κ.α.

3.3 Διαχωρισμός σε master-slave και peer-to-peer αρχιτεκτονικές

Ο διαχωρισμός αυτός αναφέρεται στον τρόπο με τον οποίο συνεργάζονται τα στοιχεία από τα οποία αποτελούνται τα συστήματα. Στις master-slave αρχιτεκτονικές υπάρχει κάποιος κόμβος ο οποίος αναλαμβάνει το συντονισμό και τη διαχείριση των υπολοίπων slave κόμβων. Αυτή η τακτική προσφέρει περισσότερες δυνατότητες ελέγχου των λειτουργιών που πραγματοποιούνται στη βάση και δίνει δυνατότητες για περισσότερες εγγυήσεις συνέπειας στα δεδομένα αφού

υπάρχει κάποιος κεντρικός έλεγχος σε αυτά. Ο κόμβος master όμως αποτελεί αδύναμο σημείο του κατανεμημένου αυτού συστήματος αφού στην περίπτωση που αυτός σταματήσει να είναι διαθέσιμος λόγω κάποιας αποτυχίας του υλικού ή οποιουδήποτε άλλου λόγου, τότε επηρεάζεται σημαντικά η λειτουργία όλων των slave κόμβων που εξαρτώνται από αυτόν.

Αντίθετα σε συστήματα peer-to-peer, όλοι οι κόμβοι είναι ισότιμοι και επομένως η απώλεια ενός οποιουδήποτε κόμβου μπορεί να αναπληρωθεί από αυτούς που απομένουν χωρίς να επηρεαστεί η λειτουργία του υπόλοιπου συστήματος σημαντικά. Αυτό προσφέρει αυξημένες εγγυήσεις διαθεσιμότητας του συστήματος αλλά και περισσότερη απλότητα στον τρόπο με τον οποίο ένα κατανεμημένο σύστημα πραγματοποιεί οριζόντια αναβάθμιση. Όμως έχει το μειονέκτημα ότι για να εξασφαλιστεί η συνέπεια στα δεδομένα απαιτούνται πιο πολύπλοκοι μηχανισμοί, περισσότεροι έλεγχοι των δεδομένων που περιέχονται σε διαφορετικούς κόμβους προκειμένου να επιστραφεί το πιο πρόσφατο αποτέλεσμα και τελικά σημαντική αύξηση στον χρόνο ανταπόκρισης του συστήματος.

Από τα συστήματα της επιλογής η HBase κάνει χρήση της master-slave αρχιτεκτονικής ενώ η Cassandra της peer-to-peer και στο επόμενο κεφάλαιο παρουσιάζονται αναλυτικότερα οι διαφορές που εισάγει αυτή η επιλογή τους.

Κεφάλαιο 4
Περιγραφή συστημάτων

Η επιλογή των συστημάτων έγινε με κριτήριο να υπάρχει ποικιλία στα χαρακτηριστικά τους και στις σχεδιαστικές επιλογές τους, έτσι ώστε να εντοπίσουμε ποιες από αυτές είναι αυτές που προκαλούν διακύμανση στην απόδοση τους. Στο παρόν κεφάλαιο αναλύονται τα χαρακτηριστικά των συστημάτων που επιλέχθηκαν.

4.1 HBase

4.1.1 Γενικά

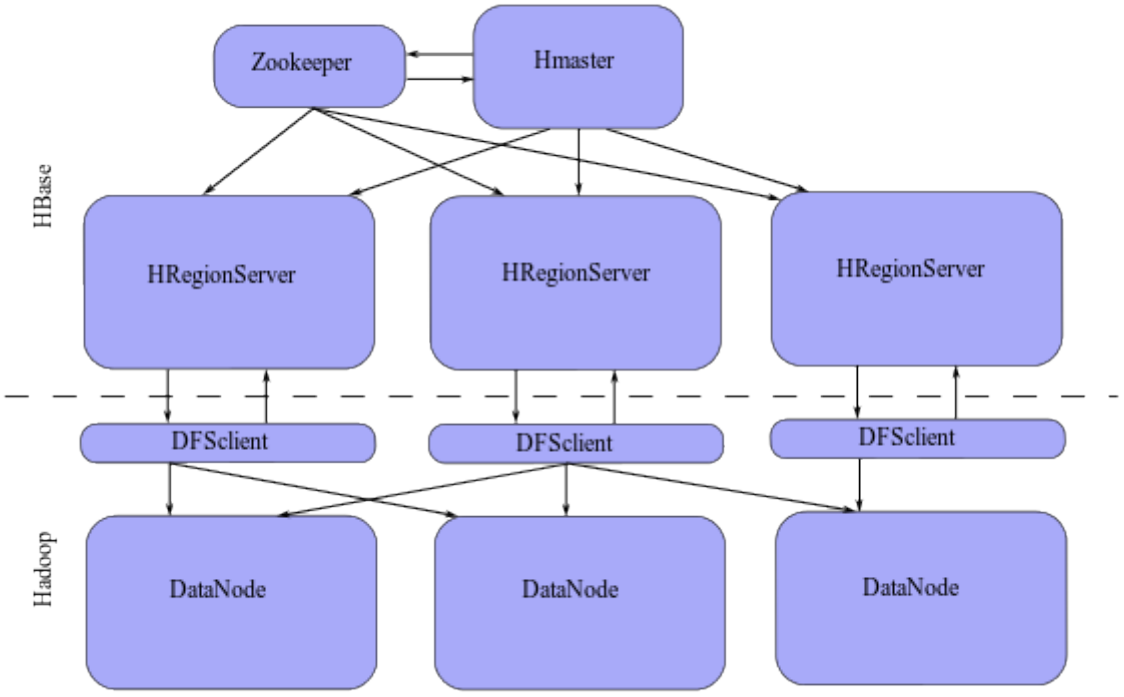
Η HBase είναι μία βάση ανοιχτού λογισμικού γραμμένη σε Java η οποία αναπτύσσεται από τον οργανισμό Apache. Έχει δημιουργηθεί με πρότυπο το BigTable της Google όπως αυτό δημοσιεύτηκε το 2006[9]. Χρησιμοποιεί για την αποθήκευση των δεδομένων της το Hadoop Distributed File System (HDFS) το οποίο με τη σειρά του έχει δημιουργηθεί με πρότυπο τη δομή του Google File System (GFS) όπως αυτό δημοσιεύτηκε το 2003[10]. Τα ειδικά χαρακτηριστικά που προσφέρει περιλαμβάνουν:

- Δυνατότητες οριζόντιας αναβάθμισης του συστήματος
- Συνέπεια σε εγγραφές και αναγνώσεις
- Κατάτμηση των δεδομένων αυτόματα στους κόμβους με δυνατότητες παραμετροποίησης
- Αυτόματη επαναφορά του συστήματος σε κανονική λειτουργία σε περιπτώσεις που κάποιος κόμβος σταματάει να είναι διαθέσιμος
- Ενσωματωμένες δυνατότητες επεξεργασίας των αποθηκευμένων δεδομένων με χρήση του Hadoop MapReduce.
- Java API για πρόσβαση από τους χρήστες
- Block cache και Bloom φίλτρα για αποδοτικά ερωτήματα στη βάση σε πραγματικό χρόνο

Χρησιμοποιείται από μεγάλες εταιρίες όπως, EBay, Yahoo!, Facebook, Trend Micro, RocketFuel Flurry, Ancestry.com, OCLC και άλλες.

4.1.2 Αρχιτεκτονική

Ένα σύστημα Hbase αποτελείται από έναν κεντρικό κόμβο master που ονομάζεται HMaster, ο οποίος αναλαμβάνει την διαχείριση των υπολοίπων slave κόμβων που ονομάζονται Regionservers. Επίσης χρησιμοποιεί ένα σύστημα συντονισμού που ονομάζεται Zookeeper και μπορεί να αναλαμβάνει τη λειτουργία του η ίδια η Hbase ή να είναι στημένο από τον διαχειριστή ανεξάρτητα από αυτή. Ο HMaster είναι υπεύθυνος για το ξεκίνημα του συστήματος, αναθέτει στους διαθέσιμους Regionservers τις περιοχές (regions) των δεδομένων πάνω στις οποίες αυτοί λειτουργούν, φροντίζοντας παράλληλα για την ισοκατανομή των δεδομένων. Επίσης, αναλαμβάνει τη διαχείριση των περιπτώσεων όπου κάποιοι Regionservers σταματούν να είναι διαθέσιμοι και πρέπει το σύστημα να επανέλθει σε φυσιολογική λειτουργία. Με τη σειρά τους οι Regionservers αναλαμβάνουν όλες τις λειτουργίες που ζητάει ο χρήστης και αφορούν τα δεδομένα. Είναι υπεύθυνοι για τον διαχωρισμό (split) των περιοχών που διαχειρίζονται εφόσον τα δεδομένα ξεπεράσουν κάποιο προκαθορισμένο όριο και την ενημέρωση του Hmaster ώστε αυτός να πραγματοποιήσει ανακατανομή των περιοχών αν χρειάζεται. Τέλος το σύστημα Zookeeper παρέχει πληροφορίες στους χρήστες προκειμένου να συνδεθούν με τους κόμβους όπου θέλουν να πραγματοποιήσουν λειτουργίες στη βάση. Επίσης παρέχει πληροφορίες στον master για τα δεδομένα και τις λειτουργίες των regionservers. Οι πληροφορίες αυτές αποθηκεύονται με μορφή που θυμίζει ιεραρχικό σύστημα δεδομένων και χρησιμοποιούνται για να εντοπίζονται μη διαθέσιμοι κόμβοι, αποτυχίες και μερικές αποτυχίες (partial failures). Μία γενική εικόνα των μερών από τα οποία αποτελείται ένα σύστημα HBase φαίνονται στο σχήμα 4.1.



Σχήμα 4.1: HBase: Γενική εικόνα συστήματος

Πηγή:http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

Η master-slave αρχιτεκτονική που χρησιμοποιείται από την HBase καθώς και από το HDFS, εισάγουν στο σύστημα έναν κόμβο ο οποίος αποτελεί Single Point Of Failure, δηλαδή στην περίπτωση που σταματήσει να είναι διαθέσιμος ο κεντρικός κόμβος HMaster της HBase ή ο Namenode του HDFS, τότε επηρεάζεται σημαντικά όλη η λειτουργία του συστήματος. Στην περίπτωση του HDFS, για να περιοριστούν οι συνέπειες από ένα τέτοιο ενδεχόμενο, παρέχεται η δυνατότητα να λειτουργούν παράλληλα με τον κύριο NameNode, κάποιος δευτερεύων κεντρικος κόμβος (SecondaryNameNode), ο οποίος εκτελεί κάποιες ενέργειες ώστε η επαναφορά του συστήματος να γίνει όσο το δυνατόν πιο άμεσα και να περιοριστούν οι ενέργειες που απαιτούνται, όταν ο NameNode επανέλθει σε λειτουργία. Επιπλέον σε εκδόσεις μεγαλύτερες της 2.0.0 υπάρχει η δυνατότητα ύπαρξης επιπλέον κεντρικών κόμβων οι οποίοι θα ακολουθούν τη λειτουργία και είναι έτοιμοι να αναλάβουν τη συνέχιση της λειτουργίας του συστήματος ακόμη και αυτόματα, αν αυτός σταματήσει απροσδόκητα να λειτουργεί ή αν απαιτούνται εργασίες συντήρησης. Αντίστοιχα στην HBase υπάρχει δυνατότητα να λειτουργούν παραπάνω από ένας HMasters οι οποίοι είναι έτοιμοι να συνεχίσουν να εξυπηρετούν τις ανάγκες αν ο κύριος σταματήσει να είναι διαθέσιμος. Επειδή η επικοινωνία των χρηστών με τα δεδομένα περνά μέσω

του συστήματος Zookeeper και στη συνέχεια γίνεται απευθείας με τους Regionservers, αν ο Hmaster σταματήσει να λειτουργεί απροσδόκητα πιθανώς το σύστημα να είναι σε θέση να λειτουργεί για κάποιο χρονικό διάστημα χωρίς το γεγονός να γίνει αντιληπτό από τους χρήστες. Ο Hmaster όμως θα πρέπει να επαναφέρεται σε λειτουργία άμεσα καθώς οι ενέργειες που πραγματοποιεί είναι απαραίτητες για τη συνέχιση της λειτουργίας του συστήματος.

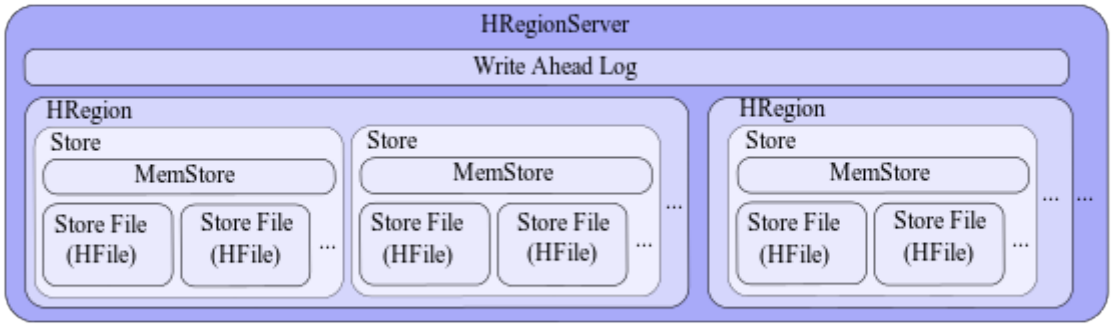
4.1.3 Εσωτερικοί μηχανισμοί

Η HBase κάνει χρήση διαφόρων μηχανισμών ώστε να εξασφαλίζει την αποδοτική χρήση των αποθηκευτικών μέσων που έχει στη διάθεσή της. Κάθε κόμβος HRegionServer διατηρεί ένα αρχείο καταγραφής WAL (Write Ahead Log) τύπου HLog και αναλαμβάνει την εξυπηρέτηση κάποιων περιοχών δεδομένων που ονομάζονται Regions. Για κάθε τέτοια περιοχή διατηρεί στη μνήμη μία δομή MemStore που λειτουργεί σαν write-back cache και ένα ή περισσότερα Store Files τα οποία αποθηκεύονται στη συνέχεια στον δίσκο με τη μορφή Hfiles. Αρχικά οι ενέργειες που προσθέτουν, τροποποιούν ή διαγράφουν δεδομένα στον κόμβο καταγράφονται σειριακά μία προς μία στο WAL και στη συνέχεια εκτελούνται στη μνήμη MemStore. Η καταγραφή των ενεργειών στο WAL γίνεται προκειμένου να ικανοποιούνται οι εγγυήσεις μονιμότητας που παρέχει το σύστημα και να διατηρείται η δυνατότητα επανάληψης τους στους σε περίπτωση που κάποιος RegionServer σταματήσει να λειτουργεί και δεν έχουν προλάβει όλα τα δεδομένα που βρίσκονται στη μνήμη να μονιμοποιηθούν στο δίσκο. Η εκτέλεση στη συνέχεια στο MemStore προσφέρει τα εξής πλεονεκτήματα:

- Η μνήμη ενημερώνεται γρήγορα και η προσθήκη επιπλέον γραμμών στον πίνακα ή η ενημέρωση καινούριων που υπάρχουν σε αυτή δεν προσθέτει σημαντικές καθυστερήσεις στο χρόνο απόκρισης. Διατηρούνται σε αυτή ταξινομημένες μόνο οι πιο πρόσφατες τιμές και πιθανές αναγνώσεις σε δεδομένα που υπάρχουν σε αυτή εκτελούνται άμεσα χωρίς αναζήτηση στο δίσκο. Επίσης η ταξινόμηση που πραγματοποιείται στη μνήμη έχει μικρό κόστος.
- Όταν το μέγεθος των δεδομένων που διατηρούνται σε αυτή τη μνήμη ξεπεράσει κάποιο ορισμένο όριο, τότε αυτά καταγράφονται σε ένα HFile το οποίο στη συνέχεια αποθηκεύεται στο HDFS, και το HDFS με τη σειρά του αναλαμβάνει την δημιουργία αντιγράφων και την κατανομή τους στους διάφορους κόμβους. Η εγγραφή γίνεται αξιοποιώντας τις ταχύτητες σειριακής εγγραφής στο δίσκο.

Από τη στιγμή που οι εγγραφές που υπήρχαν στο MemStore καταγραφούν και μονιμοποιηθούν στον δίσκο, ενημερώνεται το WAL ώστε να μπορούν να διαγραφούν από αυτό οι ενέργειες που

αφορούσαν το συγκεκριμένο τμήμα δεδομένων και να διατηρείται μικρό το μέγεθός του. Επίσης δημιουργείται μία καινούρια δομή MemStore για να αναλάβει την εξυπηρέτηση νέων ενεργειών. Αυτή η διαδικασία επαναλαμβάνεται όσο αυξάνονται τα δεδομένα που εισάγονται στο σύστημα οπότε και δημιουργούνται μια σειρά από HFiles που αποθηκεύονται στο δίσκο. Τα παραπάνω στοιχεία που αποτελούν τις βασικές δομές ενός RegionServer φαίνονται στο σχήμα 4.2.



Σχήμα 4.2: HBase: Εσωτερικοί μηχανισμοί HregionServer

Πηγή: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

Τα HFiles όπως αναφέρθηκε αποτελούνται από ταξινομημένα δεδομένα. Στην πραγματικότητα αποτελούν LSM-Trees (Log-Structured Merge-Trees) [11], τα οποία θυμίζουν στη δομή τα κλασσικά B-trees [12] που χρησιμοποιούνται από αρκετές άλλες βάσεις δεδομένων, όμως είναι προσαρμοσμένα ώστε να κάνουν αποδοτική την χρήση των αποθηκευτικών μέσων και να εκμεταλλεύονται οι σειριακές δυνατότητες ανάγνωσης δεδομένων. Από τη στιγμή που θα καταγραφούν στον δίσκο δεν αλλάζουν παρά μόνο κατά τη διαδικασία όπου συγχωνεύονται με τα υπόλοιπα HFiles. Η διαδικασία της συγχώνευσης έχει ως στόχο να διατηρηθεί μικρός ο αριθμός των HFiles και να διατηρείται μικρός ο αριθμός αναζητήσεων στο δίσκο σε περιπτώσεις ανάγνωσης. Για το σκοπό αυτό τρέχει στο παρασκήνιο μία διεργασία που πραγματοποιεί "μερική" συμπύκνωση (minor compaction) και ενώνει τα μικρότερα HFiles σε μεγαλύτερα. Το γεγονός ότι τα δεδομένα είναι ήδη ταξινομημένα, βοηθά στο να γίνεται η συμπύκνωση αποδοτικά και η συμπύκνωση να μην προσθέτει ιδιαίτερες καθυστερήσεις στην απόκριση του συστήματος. Ακόμη και αν εκτελούνται λειτουργίες διαγραφής δεδομένων τότε αυτές δεν πραγματοποιούνται απευθείας στα δεδομένα που έχουν αποθηκευτεί στο δίσκο προκειμένου να ελαχιστοποιούνται οι αναζητήσεις στον δίσκο της θέσης των δεδομένων. Αντί αυτού

δημιουργούνται νέες καταχωρήσεις κλειδιού-τιμής για τα συγκεκριμένα δεδομένα που περιέχουν μία σήμανση (tombstone) και η οποία υποδεικνύει ότι το συγκεκριμένο δεδομένο έχει διαγραφεί. Τα δεδομένα διαγράφονται οριστικά από τον δίσκο κατά την διαδικασία της καθολικής συμπύκνωσης (major compaction) η οποία πραγματοποιεί ανά τακτά χρονικά διαστήματα σε όλα τα δεδομένα, και είναι απαραίτητη για τη διατήρηση της καλής επίδοσης.

4.1.4 Δομή Δεδομένων

Αξιοποιεί την αποθήκευση στο δίσκο κατά στήλες (column-oriented) η οποία της παρέχει τα χαρακτηριστικά και τα πλεονεκτήματα που αναφέρθηκαν σε προηγούμενο κεφάλαιο. Συγκεκριμένα, αποθηκεύει τα δεδομένα χρησιμοποιώντας πίνακες οι οποίοι αποτελούνται από γραμμές και στήλες. Κάθε γραμμή συσχετίζεται με ένα μοναδικό κλειδί για ταξινόμηση και μπορεί να έχει οποιοδήποτε αριθμό στηλών αλλά και διαφορετικές εκδοχές (versions) της ίδιας στήλης με βάση κάποια χρονοσήμανση (time stamp). Οι στήλες με τη σειρά τους κατηγοριοποιούνται σε οικογένειες στηλών (column families), πληροφορία που προσφέρει καλύτερη σημασιολογική ερμηνεία των δεδομένων και χρησιμοποιείται ώστε να βρίσκονται παρόμοια δεδομένα μαζί στον δίσκο παρέχοντας πλεονεκτήματα όπως καλύτερη συμπίεση ή αποδοτικότερο caching στη μνήμη. Οι πίνακες είναι στην πραγματικότητα μία δομή από ταξινομημένες λίστες, στις οποίες οι ετικέτες της γραμμής, της οικογένειας στηλών, της στήλης και της χρονοσήμανσης, χρησιμοποιούνται για την πρόσβαση στην τιμές που περιέχονται στον πίνακα, κάνοντας έτσι τη δομή των δεδομένων να θυμίζει δομές Κλειδιού-Τιμής. Έτσι οι πίνακες μπορούν να είναι αραιοί (sparse) και κάθε γραμμή περιέχει μόνο τις στήλες οι οποίες έχουν δεδομένα. Στους πίνακες που ακολουθούν (Πηγές: [13],[14]) φαίνεται ένα τέτοιο παράδειγμα και πώς οι διαφορετικές στήλες αποθηκεύονται από την HBase. Ο πίνακας webtable (4.1) αποτελείται από δύο οικογένειες στηλών που ονομάζονται contents και anchor. Η οικογένεια anchor στο παράδειγμα αυτό περιέχει δύο στήλες (anchor:cssnsi.com, anchor:my.look.ca) και η οικογένεια contents περιέχει μία στήλη (contents:html).

Row Key	**Time Stamp**	**ColumnFamily “contents”**	**ColumnFamily “anchor”**
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

Πίνακας 4.1: Παράδειγμα χρήσης δεδομένων από την HBase: Πίνακας Webtable

Στην πραγματικότητα ο παραπάνω αραιός πίνακας αποθηκεύεται από την HBase όπως στους παρακάτω πίνακες (4.2,4.3) όπου φαίνεται η κατηγοριοποίηση ανά οικογένεια στηλών. Επιπλέον μπορούν να προστεθούν καινούριες στήλες σε κάθε οικογένεια στηλών χωρίς αυτές να έχουν οριστεί νωρίτερα. Η ταξινόμηση στις καταχωρήσεις γίνεται με την εξής σειρά: ανά σειρά, ανά οικογένεια στηλών, ανά χρονοσήμανση. Αν κατά την αναζήτηση μίας τιμής δεν οριστεί η εκδοχή της (version) με βάση τη χρονοσήμανση, τότε θα επιστραφεί η πιο πρόσφατη τιμή.

Row Key	**Time Stamp**	**Column Family anchor**
"com.cnn.www"	t9	anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

Πίνακας 4.2: Παράδειγμα χρήσης δεδομένων από την HBase: Οικογένεια Στηλών anchor

Row Key	**Time Stamp**	**ColumnFamily "contents:"**
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Πίνακας 4.3: Παράδειγμα χρήσης δεδομένων από την HBase: Οικογένεια Στηλών contents

4.1.5 ACID Εγγυήσεις

Η HBase προσφέρει ACID εγγυήσεις κυρίως για τις λειτουργίες που πραγματοποιούνται ανά γραμμή [15]. Η μέθοδος με την οποία πραγματοποιεί έλεγχο ταυτοχρονισμού (concurrency control) βασίζεται στην χρονοσήμανση των λειτουργιών αυτών (MVCC) [16].

Παρέχει ατομικότητα ανά γραμμή δεδομένων, δηλαδή όλες οι αλλαγές που πραγματοποιούνται σε μία γραμμή είναι ατομικές ακόμη και αν αυτές οι αλλαγές αφορούν παραπάνω από μία στήλες. Δηλαδή το αποτέλεσμα μιας αλλαγής θα είναι είτε επιτυχία, είτε αποτυχία ή ότι τελείωσε ο χρόνος που της είχε δοθεί για να επιστρέψει αποτέλεσμα. Στην περίπτωση που το αποτέλεσμα που επιστρέφεται είναι ότι τελείωσε ο χρόνος, τότε αυτή η αλλαγή είτε θα έχει επιτύχει, είτε θα έχει αποτύχει, δηλαδή δεν επιτρέπεται να επιτύχει-αποτύχει μερικώς. Αν ένας χρήστης πραγματοποιεί αλλαγές σε παραπάνω από μία γραμμές παράλληλα, τότε το αποτέλεσμα δεν εξασφαλίζεται ότι ικανοποιεί την ατομικότητα, αφού οι επιμέρους λειτουργίες σε κάθε γραμμή

μπορεί να επιστρέψουν οποιοδήποτε από τα αποτελέσματα που αναφέρθηκαν. Αντίστοιχα, αν δύο χρήστες πραγματοποιήσουν αλλαγές σε μία γραμμή σχεδόν ταυτόχρονα, πχ ο χρήστης Α πραγματοποιήσει μία ανάθεση τιμών "a=1,b=1,c=1" για μία γραμμή, και ο χρήστης Β "a=2,b=2,c=2", τότε το αποτέλεσμα θα είναι είτε αυτό που καταχώρησε ο χρήστης Α, είτε αυτό που καταχώρησε ο χρήστης Β και δεν μπορεί να είναι κάτι του τύπου "a=1,b=2,c=1".

Ως προς την εγγύηση της συνέπειας (consistency) και της απομόνωσης (isolation) η HBase εξασφαλίζει ότι το αποτέλεσμα από κάποια ανάγνωση σε κάποια γραμμή θα επιστρέψει μία ολοκληρωμένη κατάσταση της γραμμής αυτής όπως υπήρχε κάποια στιγμή στη βάση, ακόμη και αν πραγματοποιούνται αλλαγές στη γραμμή αυτή ταυτόχρονα με την ανάγνωση. Οι αλλαγές στην κατάσταση μίας γραμμής πραγματοποιούνται με τη χρονολογική σειρά που καταχωρούνται. Αντίστοιχα με πριν για λειτουργίες scan στη βάση οι οποίες αφορούν παραπάνω από μία γραμμές δεν μπορεί να εξασφαλιστεί ότι το συνολικό αποτέλεσμα θα είναι μία συνεπής εικόνα των δεδομένων της βάσης.

Ως προς την εγγύηση της μονιμότητας (durability) εξασφαλίζει ότι όλες οι αναγνώσεις επιστρέφουν δεδομένα τα οποία έχουν μονιμοποιηθεί. Όλες οι ενέργειες που πραγματοποιούνται στη βάση και επιστρέφουν κωδικό επιτυχίας θα μονιμοποιηθούν και αντίστοιχα καμία ενέργεια που επιστρέφει κωδικό αποτυχίας δε θα μονιμοποιηθεί.

4.2 Cassandra

4.2.1 Γενικά

Η Cassandra είναι μία βάση η οποία συνδυάζει την κατανεμημένη αρχιτεκτονική του Dynamo της Amazon [17] χρησιμοποιώντας μία δομή δεδομένων αντίστοιχη με αυτή του BigTable της Google [9]. Είναι ανοιχτού λογισμικού γραμμένη σε γλώσσα Java και αναπτύχθηκε αρχικά από το Facebook για να υλοποιήσει το χαρακτηριστικό της αναζήτησης στο inbox των χρηστών. Έχει σχεδιαστεί και αυτό όπως και τα περισσότερα NoSQL συστήματα για να αντιμετωπίζει την πραγματικότητα: ότι σε ένα κατανεμημένο σύστημα είναι δεδομένο ότι θα υπάρχουν αποτυχίες υλικού και κατατμήσεις. Τα βασικά χαρακτηριστικά που προσφέρει είναι τα εξής:

- Ομοιογενώς κατανεμημένο σύστημα, που αποτελείται από ισότιμους κόμβους, χωρίς Single Point of Failure
- Ελαστική δυνατότητα οριζόντιας αναβάθμισης- υποβάθμισης του συστήματος
- Υψηλή διαθεσιμότητα του συστήματος με ανοχή στις κατατμήσεις και τις αποτυχίες, με

αυτόματη διαχείριση τέτοιων περιπτώσεων.

- Δυνατότητες προσαρμόσιμης συνέπειας στα δεδομένα
- Αλληλεπίδραση με τη βάση μέσω μίας γλώσσας που ονομάζεται CQL και έχει παρόμοια σύνταξη με την SQL για ευκολία στην εκμάθηση και την χρήση.

Χρησιμοποιείται σε πάνω από 1500 εταιρίες[18] ανάμεσα στις οποίες βρίσκονται και πολύ γνωστά ονόματα όπως Facebook, Twitter, Cisco, Hulu, Rackspace, Digg, Cloudkick, Reddit, CERN, eBay και Instagram.

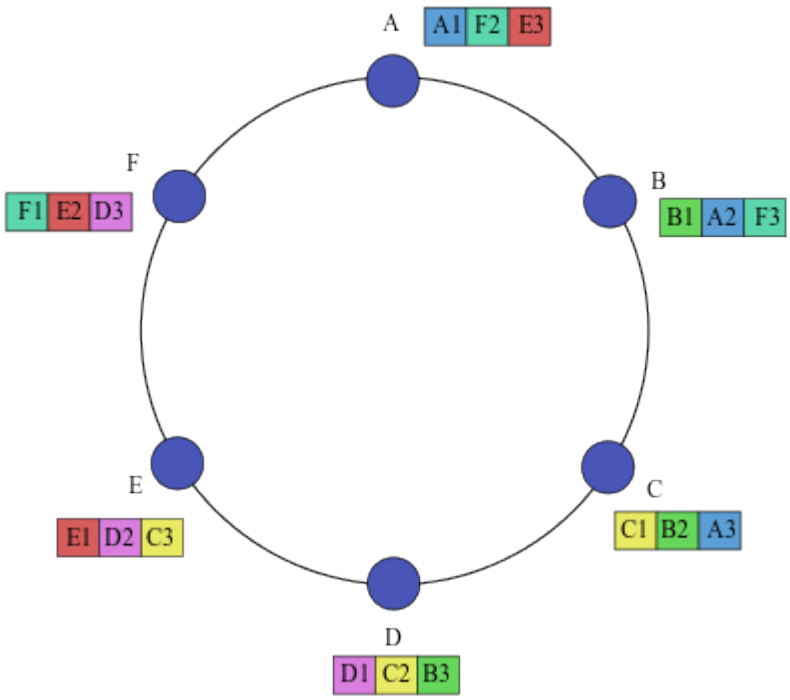
4.2.2 Αρχιτεκτονική

Ένα από τα βασικά χαρακτηριστικά ενός συστήματος Cassandra είναι ότι όλοι οι κόμβοι που το αποτελούν είναι ισότιμοι (peer to peer). Σε κάθε κάθε έναν κόμβο ξεκινάει μία διεργασία CassandraDeamon η οποία αρκεί για να προσθέσει τον κόμβο στο cluster να έχει κάποια πληροφορία για το πού βρίσκεται ένας άλλος κόμβος του συστήματος. Στην πράξη ορίζονται κάποιοι κόμβοι που ονομάζονται seeds και με τους οποίους επικοινωνεί ένας καινούριος κόμβος για να λάβει πληροφορίες για την κατάσταση του συστήματος και να συνδεθεί σε αυτό. Έτσι η προσθήκη ενός καινούριου κόμβου στον δακτύλιο αποτελεί μία απλή διαδικασία και το σύστημα Cassandra ήταν με διαφορά το πιο απλό από τα τρία στο στήσιμό του.

Κανένας κόμβος δεν επιτελεί κάποια ιδιαίτερη λειτουργία σε σχέση με τους υπόλοιπους γεγονός που του δίνει κάποια ιδιαίτερα χαρακτηριστικά. Το κυριότερο από αυτά είναι ότι δεν υπάρχει στο σύστημα κάποιος κόμβος ο οποίος αν σταματήσει να λειτουργεί θα επηρεάσει σημαντικά τη λειτουργία του(Single Point Of Failure) όπως συμβαίνει σε συστήματα που ακολουθούν master-slave αρχιτεκτονικές και την οποία χρησιμοποιούν τα άλλα δύο συστήματα της επιλογής. Στα δύο άλλα συστήματα της επιλογής, σε περίπτωση που ένας master κόμβος σταματήσει να είναι διαθέσιμος τότε, εφόσον έχει υπάρξει ο κατάλληλος σχεδιασμός, υπάρχουν δυνατότητες να ανακτηθεί η πλειοψηφία των δεδομένων και το σύστημα να επανέλθει σε φυσιολογική λειτουργία. Όμως αυτό συνήθως απαιτεί παρέμβαση από τον διαχειριστή του συστήματος και έχει και σαν αποτέλεσμα να μεσολαβεί κάποιος χρόνος κατά τον οποίο το σύστημα(ή το μέρος του συστήματος για την MongoDB) δεν είναι διαθέσιμο. Αντίθετα στην Cassandra όπου όλοι οι κόμβοι είναι ισότιμοι, η απώλεια ενός οποιουδήποτε κόμβου δεν επηρεάζει τη διαθεσιμότητα του συστήματος. Το τίμημα αυτής της αυξημένης διαθεσιμότητας είναι ότι, στη γενική περίπτωση, δεν παρέχονται εγγυήσεις για τη συνέπεια των δεδομένων ανά πάσα στιγμή. Το σύστημα απλώς εγγυάται ότι τα δεδομένα θα είναι συνεπή κάποια στιγμή και αυτό μπορεί να συμβαίνει ήδη τη

στιγμή που πραγματοποιείται μία ανάγνωση ή να συμβεί στο μέλλον(eventual consistency).

Σε κάθε κόμβο ανατίθεται ένα μοναδικό αναγνωριστικό (token) το οποίο τον εντάσσει σε έναν δακτύλιο που δημιουργείται και χρησιμοποιείται για να προσδιοριστεί από το σύστημα η κατανομή των δεδομένων σε αυτό. Στην απλή περίπτωση το σύστημα αναθέτει το αυτό αναγνωριστικό κατά την εκκίνηση του κόμβου. Σε πιο πολύπλοκες περιπτώσεις, όπως παραδείγματος χάριν αν το cluster κατανέμεται σε παραπάνω από μία τοποθεσίες , αυτό το αναγνωριστικό μπορεί να οριστεί από τον διαχειριστή ώστε ο δακτύλιος που θα δημιουργηθεί να εξασφαλίζει καλύτερη απόδοση του συστήματος με αξιοποίηση της τοπικότητας των δεδομένων, καθώς τη δυνατότητα συνέχισης της ομαλής λειτουργίας του σε περίπτωση που μία ολόκληρη τοποθεσία στην οποία υπάρχουν δεδομένα γίνει μη διαθέσιμη. Στη συνέχεια κατανέμει τα δεδομένα κυκλικά λαμβάνοντας υπόψιν τον αριθμό των επαναλήψεων των δεδομένων (replication factor) που έχει οριστεί. Το σύστημα που προκύπτει για ένα απλό παράδειγμα 6 κόμβων A, B, C, D, E, F, με παράγοντα επανάληψης των δεδομένων 3, έχει τη μορφή που φαίνεται στο σχήμα 4.3.



Σχήμα 4.3: Cassandra: Δακτύλιος κόμβων και κατανομή δεδομένων

Πηγή: http://www.datastax.com/docs/1.1/cluster_architecture/partitioning

Επίσης παρέχεται η δυνατότητα να χρησιμοποιηθεί η πληροφορία που αφορά σε ποια τοποθεσία βρίσκονται τα δεδομένα, αλλά και πώς είναι οργανωμένα τα racks μέσα στην ίδια τοποθεσία ώστε να γίνεται όσο το δυνατόν αποδοτικότερη η κατανομή των δεδομένων. Οι πληροφορία αυτή ονομάζεται snitch και χρησιμοποιείται αντιστοιχίζοντας κατάλληλες IP στους αντίστοιχους κόμβους.

Η Cassandra χρησιμοποιεί ένα πρωτόκολλο gossip (κουτσομπολιού) ώστε κάθε κόμβος να γνωρίζει την κατάσταση του υπόλοιπου δικτύου και να εντοπίζονται μη διαθέσιμοι κόμβοι. Το πρωτόκολλο αυτό έχει πάρει την ονομασία του από το χαρακτηριστικό ότι κάθε άτομο επιλέγει με ποιο άλλο άτομο θα ανταλλάξει πληροφορίες. Αυτό προσφέρει μία πιο ευέλικτη μέθοδο εντοπισμού νεκρών κόμβων σε σχέση με τα απλά κλασσικά συστήματα. Τα πιο απλά συστήματα χρησιμοποιούν heartbeats, δηλαδή στέλνουν ανά τακτά χρονικά διαστήματα κάποια σήματα στους κόμβους και εφόσον οι κόμβοι δεν επιστρέφουν απάντηση ότι λειτουργούν κανονικά τότε θεωρούνται νεκροί από το υπόλοιπο σύστημα. Η διαφορά του πρωτοκόλλου που χρησιμοποιείται από την Cassandra είναι ότι το σύστημα εντοπισμού αποτυχιών διατηρεί για κάθε κόμβο μια τιμή που δείχνει την πιθανότητα(την υποψία) αυτός ο κόμβος είναι μη διαθέσιμος. Αν η τιμή αυτή ξεπεράσει κάποιο όριο τότε μόνο ο κόμβος θεωρείται μη διαθέσιμος από το υπόλοιπο σύστημα. Αν ένας κόμβος Α δεν μπορεί να επικοινωνήσει απευθείας με κάποιον κόμβο Β τότε για να υπολογίσει την πιθανότητα να μην είναι διαθέσιμος λαμβάνει υπόψιν του και πληροφορίες από άλλους κόμβους για την δυνατότητά τους να συνδεθούν με τον κόμβο Β. Με αυτό τον τρόπο συνυπολογίζονται πιθανές διακυμάνσεις του δικτύου που συνδέει τους κόμβους, το φορτίο που εξυπηρετεί κάποιος κόμβος, και ιστορικές συνθήκες, μειώνοντας έτσι την πιθανότητα για λανθασμένα συμπεράσματα.

Ο χρήστης μπορεί να συνδεθεί με οποιονδήποτε κόμβο του συστήματος προκειμένου να ζητήσει δεδομένα. Αν συνδεθεί με κάποιον κόμβο ο οποίος δεν περιέχει τα δεδομένα που ζητάει, τότε αυτός ο κόμβος λειτουργεί σαν συντονιστής, ώστε να φέρει τα δεδομένα στον χρήστη από τον κόμβο που τα περιέχει αλλά και να μεταφέρει πιθανές εγγραφές στον κόμβο που προορίζονται.

Σε περίπτωση που κάποιος κόμβος σταματήσει να είναι διαθέσιμος τότε τίθεται κάποιο χρονικό όριο κατά το οποίο το υπόλοιπο σύστημα αναλαμβάνει να διατηρεί δεδομένα για τις λειτουργίες που θα επιτελούνταν σε αυτόν. Συγκεκριμένα αν ένας χρήστης συνδεθεί με έναν κόμβο Α και ζητήσει να γράψει δεδομένα που προορίζονται για έναν κόμβο Β ο οποίος δεν είναι διαθέσιμος, τότε ο κόμβος Α θα αποθηκεύσει αυτές τις πληροφορίες με τη μορφή hints και θα ενημερώσει τον κόμβο Β για τις λειτουργίες που δεν πραγματοποίησε μόλις αυτός ξαναγίνει διαθέσιμος στέλνοντάς του μηνύματα που ονομάζονται hinted hadoffs. Σε περίπτωση που εξαντληθεί το

χρονικό όριο που δίνεται στον κόμβο Β να επανέλθει σε φυσιολογική λειτουργία τότε, οι υπόλοιπο κόμβοι σταματούν να αποθηκεύουν hints που τον αφορούν και πρέπει είτε να διαγραφεί ο κόμβος είτε αν επανέλθει σε λειτουργία αργότερα να εκτελεστούν κάποιες λειτουργίες για την επαναφορά των δεδομένων του και την εκτέλεση των λειτουργιών που δεν εκτέλεσε κατά το διάστημα αυτό. Στην περίπτωση αυτή, όπου κάποιος κόμβος δεν είναι διαθέσιμος, αν ο παράγοντας αντιγραφής των δεδομένων είναι μεγαλύτερος από 1 τότε οι λειτουργίες που εκτελεί ο χρήστης πραγματοποιούνται κανονικά στους υπόλοιπους κόμβους που διατηρούν αντίγραφα και είναι διαθέσιμοι.

4.2.3 Εσωτερικοί μηχανισμοί

Η Cassandra, έχοντας και αυτή σχεδιαστεί με βάση το BigTable της Google, παρουσιάζει πολλές ομοιότητες με την HBase ως προς τον τρόπο με τον οποίο διαχειρίζεται τα δεδομένα.. Οι διάφοροι μηχανισμοί που χρησιμοποιεί βασικό κριτήριο έχουν την καλύτερη δυνατή επίδοση του συστήματος και την αποδοτικότερη χρησιμοποίηση των διαθέσιμων αποθηκευτικών μέσων. Κάθε κόμβος χρησιμοποιεί ένα commit log το οποίο αποθηκεύει απευθείας στο δίσκο τις ενέργειες που πραγματοποιούν οι χρήστες της βάσης στα δεδομένα. Αυτό γίνεται για να ικανοποιούνται οι εγγυήσεις της μονιμότητας στα δεδομένα της βάσης και οι συγκεκριμένες καταχωρήσεις μπορούν να χρησιμοποιηθούν για επαναφορά των δεδομένων ακόμη και σε περίπτωση απώλειας ρεύματος σε κάποιο κόμβο. Επίσης διατηρείται μία δομή memtable η οποία λειτουργεί σαν write-back cache και η οποία δίνει επίδοση στο σύστημα τόσο σε περιπτώσεις εγγραφών όσο και σε περιπτώσεις ανάγνωσης. Το memtable αποτελείται από γραμμές δεδομένων στις οποίες μπορεί να γίνει αναζήτηση βάση κλειδιού. Η διαδικασία που ακολουθείται για να γίνουν εγγραφές στον δίσκο είναι η εξής:

- Αρχικά η εγγραφή καταγράφεται στο commit log στο δίσκο ώστε το σύστημα να ικανοποιεί τις εγγυήσεις μονιμότητας προτού θεωρηθεί επιτυχής.
- Επίσης η εγγραφή εκτελείται στο memtable το οποίο διατηρεί μόνο τις πιο πρόσφατες τιμές των δεδομένων.
- Αν οι εγγραφές ξεπεράσουν ένα όριο που ορίζεται από τον διαχειριστή τότε το memtable αποθηκεύεται στον δίσκο με τη μορφή SSTables. Τα SSTables είναι ταξινομημένοι πίνακες οι οποίοι διατηρούν τις τελευταίες ενημερώσεις στα δεδομένα. Για κάθε SSTable διατηρούνται bloom filters στη μνήμη για επιπλέον επίδοσή και ελαχιστοποίηση των αναζητήσεων στον δίσκο. Όλες οι εγγραφές στον δίσκο γίνονται αξιοποιώντας την ταχύτητα σειριακών εγγραφών του δίσκου(sequential) και τα δεδομένα που περιέχουν τα

SSTables δεν καταλήγουν απευθείας στην θέση που τους ανήκει στα υπόλοιπα δεδομένα της βάσης αφού κάτι τέτοιο θα εισήγαγε καθυστερήσεις σε κάθε εγγραφή από τον χρόνο αναζήτησης της θέσης κάθε δεδομένου στο δίσκο.

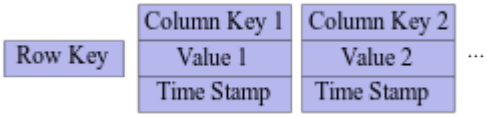
- Στη συνέχεια χρησιμοποιείται μια διαδικασία που λέγεται compaction (συμπύκνωση) κατά την οποία συγχωνεύονται τα διάφορα SSTables που υπάρχουν στον δίσκο. Αυτή η διαδικασία είναι αποδοτική αφού τα SSTables περιέχουν ήδη ταξινομημένα δεδομένα και επίσης περιορίζονται οι πόροι που δεσμεύει από το σύστημα ώστε να γίνεται με τρόπο που να μην επιβαρύνει σημαντικά ανά πάσα στιγμή τον κόμβο και να μην είναι ορατή στον χρήστη του συστήματος. Κατά τη διαδικασία αυτή δημιουργούνται επίσης νέοι δείκτες, και πραγματοποιείται και η πραγματική διαγραφή δεδομένων που μέχρι το προηγούμενο στάδιο με τα SSTables είχαν απλώς σημαδευτεί από το σύστημα ότι προορίζονται για διαγραφή με μία σήμανση που ονομάζεται tombstone.

Αντίστοιχα και με τις αναγνώσεις στόχος των μηχανισμών που χρησιμοποιούνται είναι να γίνονται όσο το δυνατόν λιγότερες αλληλεπιδράσεις με τον δίσκο. Κάθε ανάγνωση πιθανώς να χρειάζεται να πραγματοποιήσει αναζήτηση σε πάνω από ένα SSTables προκειμένου να επιστρέψει αποτέλεσμα. Αρχικά χρησιμοποιούνται τα Bloom Filters τα οποία είναι ένας γρήγορος τρόπος για να απορριφθούν τα SSTables που σίγουρα δεν περιέχουν τα δεδομένα. Στη συνέχεια πραγματοποιείται αναζήτηση σε μία κρυφή μνήμη που περιέχει κλειδιά από τιμές που είναι αποθηκευμένες στο τμήμα των δεδομένων που έχει ο κόμβος (partition key cache). Αν το δεδομένο που ζητάει ο χρήστης είναι σε αυτή τη μνήμη τότε στη συνέχεια το σύστημα αναζητά την τοποθεσία του τμήματος των συμπιεσμένων δεδομένων που περιέχει ζητούμενο δεδομένο από ένα άλλο ευρετήριο που βρίσκεται στη μνήμη(compression offsets). Στη συνέχεια με αυτή την πληροφορία επικοινωνεί για πρώτη φορά με το δίσκο, διαβάζει το τμήμα που περιέχει το ζητούμενο αποτέλεσμα και στη συνέχεια επιστρέφει το αποτέλεσμα στον χρήστη. Αν το δεδομένο δεν περιέχεται στην partition key cache τότε μεσολαβεί και ένα στάδιο όπου το σύστημα αναζητά το δεδομένο στο ευρετήριο των δεδομένων του δίσκου.

4.2.4 Δομή Δεδομένων

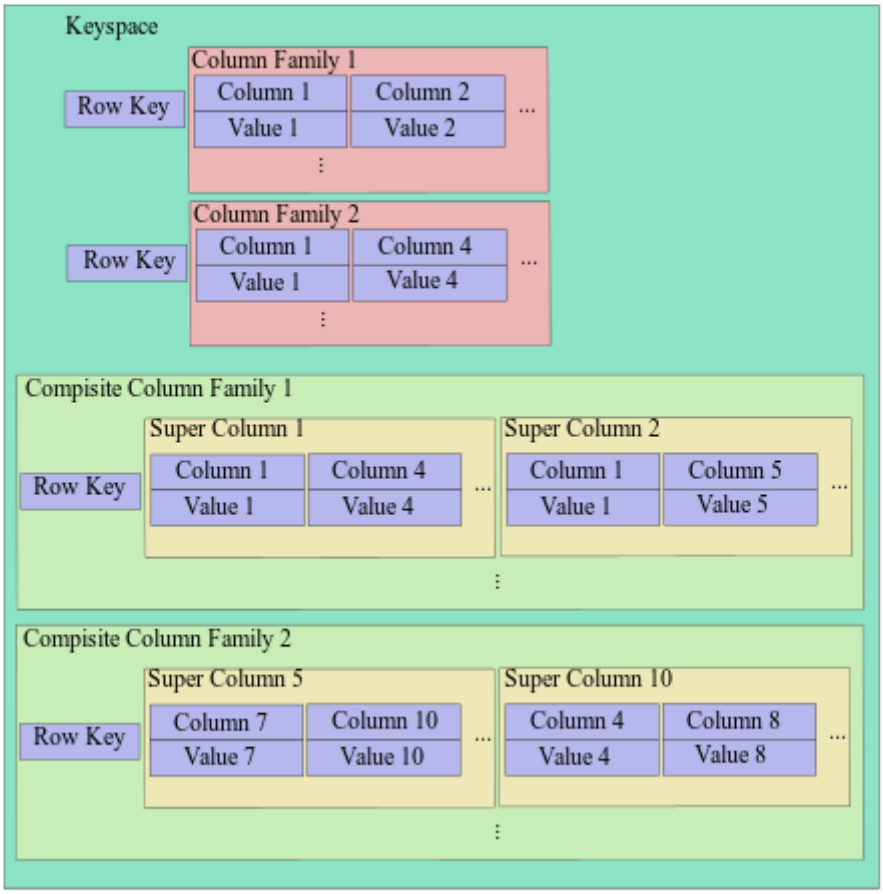
Η δομή των δεδομένων είναι εμπνευσμένη από τη δομή του BigTable της Google και επομένως παρουσιάζει αρκετές ομοιότητες με αυτή της HBase. Αντίστοιχα και η βάση Cassandra οργανώνει τα δεδομένα της ανά στήλες και έχει δυνατότητα να διατηρεί αραιούς πίνακες καταλαμβάνοντας χώρο μόνο για τις στήλες εκείνες που έχουν δεδομένα. Αρχικά ορίζεται ένα keyspace το οποίο είναι αντίστοιχο με το όνομα της βάσης, και συνήθως ορίζεται ένα ανά

εφαρμογή. Οι γραμμές προσδιορίζονται από ένα κλειδί και αποτελούν λίστες οι οποίες περιέχουν κάποιο αριθμό στηλών. Κάθε στήλη προσδιορίζεται με τη σειρά της από ένα μοναδικό κλειδί και περιέχει κάποια τιμή και κάποια χρονοσήμανση.



Σχήμα 4.4: Cassandra: Γραμμή Δεδομένων

Τα κλειδιά αυτά των γραμμών και των στηλών, εν αντιθέσει με άλλα συστήματα, μπορούν εκτός από κάποιο αλφαριθμητικό να είναι και ακέραιοι, UUIDs ή και πίνακες από bytes. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα να αποθηκεύονται χρήσιμες πληροφορίες και στα κλειδιά της βάσης και όχι μόνο στις τιμές τους. Επίσης ορίζονται οικογένειες στηλών οι οποίες δείχνουν ποια δεδομένα σχετίζονται μεταξύ τους και προσδιορίζουν έτσι κάτι ανάλογο με τους πίνακες των σχεσιακών βάσεων δεδομένων. Αυτές οι οικογένειες στηλών μπορούν να είναι είτε στατικές είτε δυναμικές. Οι στατικές θυμίζουν περισσότερο τη συμπεριφορά των πινάκων των σχεσιακών βάσεων δεδομένων, αφού έχουν εκ των προτέρων προσδιορισμένες τις στήλες τους και έτσι κάθε γραμμή γεμίζει με συγκεκριμένα δεδομένα. Η διαφορά από τις σχεσιακές βάσεις και σε αυτή την περίπτωση παραμένει ότι αποθηκεύονται στο δίσκο μόνο οι στήλες που περιέχουν τιμές. Οι δυναμικές αντίθετα δεν έχουν περιορισμούς και ο χρήστης μπορεί να προσθέτει καινούριες στήλες σε κάθε γραμμή ανάλογα με τις ανάγκες, ορίζοντας αυτός το όνομα της στήλης. Ένα επιπλέον χαρακτηριστικό σε σχέση με το μοντέλο δεδομένων της HBase είναι ότι μπορούν να προσδιοριστούν σύνθετες οικογένειες στηλών (composite column families) οι οποίες χρησιμοποιούνται σαν ένα επιπλέον επίπεδο απεικόνισης, και μπορούν να περιέχουν είτε κανονικές στήλες είτε υπέρ-στήλες (super-columns). Με τη σειρά τους οι υπέρ-στήλες μπορούν να περιέχουν κάποιον αριθμό από στήλες που μπορούν να επιλέγονται από τις διάφορες γραμμές με βάση κάποια συνθήκη. Έτσι οι σύνθετες οικογένειες στηλών χρησιμοποιούνται με σκοπό να αντιστοιχίζονται και να παρουσιάζονται δεδομένα από πολλαπλές οικογένειες στηλών συγκεντρωμένα. Τα παραπάνω φαίνονται πιο παραστατικά στο σχήμα 4.5 όπου έχουν δοθεί τυχαία ονόματα στις διάφορες δομές και έχουν παραλειφθεί τα κελιά της χρονοσήμανσης χάριν απλότητας:



Σχήμα 4.5: Cassandra: Οικογένειες και Σύνθετες Οικογένειες Στηλών

Πηγές: http://www.datastax.com/docs/1.1/ddl/column_family

http://en.wikipedia.org/wiki/Super_column_family

4.2.5 ACID εγγυήσεις

Η Cassandra παρέχει ατομικότητα σε επίπεδο γραμμής, που σημαίνει ότι η προσθήκη και η ενημέρωση στηλών σε κάποια γραμμή αντιμετωπίζεται σαν μία εγγραφή[19]. Αντίστοιχα με την HBase όμως δεν εξασφαλίζει την ατομικότητα σε περιπτώσεις που ο χρήστης θέλει να ενημερώσει παραπάνω από μία γραμμές με τη μορφή μίας ενέργειας. Επίσης δεν επιστρέφει την βάση στην προηγούμενη κατάστασή της αν μια εγγραφή επιτύχει σε έναν κόμβο αλλά αποτύχει σε κάποιον άλλον κόμβο που διατηρεί αντίγραφο των δεδομένων. Υπάρχει δυνατότητα ελέγχου της επιτυχίας-αποτυχίας της εγγραφής στους διάφορους κόμβους που αποτελούν το σύνολο με τα αντίγραφα (replication set), όμως αυτό δεν αποτρέπει τα δεδομένα από το να εγγραφούν στους

κόμβους όπου η εγγραφή έγινε με επιτυχία.

Ο χρήστης μπορεί να προσαρμόσει τη συνέπεια που ζητάει από τα δεδομένα και να διαλέξει ανάμεσα σε διάφορα επίπεδα συνέπειας ανάλογα με τις ανάγκες. Η ερμηνεία αυτών των διαφορετικών επιπέδων προκύπτει σε συνδυασμό με τον παράγοντα αντιγραφής των δεδομένων (replication factor) και η τιμή που δίνεται καθορίζει πόσοι κόμβους από αυτούς που τελικά θα λάβουν τα δεδομένα πρέπει να επιβεβαιώσουν την εγγραφή ώστε αυτή να θεωρηθεί επιτυχής. Αντίστοιχα για τις αναγνώσεις δείχνει από πόσους κόμβους πρέπει να ληφθούν τιμές, ώστε στη συνέχεια να γίνει επιλογή της πιο πρόσφατης τιμής και να επιστραφεί σαν αποτέλεσμα της ανάγνωσης. Σαν quorum ορίζεται η τιμή: {(παράγοντας αντιγραφής / 2) + 1}. Η διαίρεση είναι ακέραια και το αποτέλεσμα δίνει πόσοι πρέπει να είναι οι κόμβοι, ώστε να προκύπτει πλειοψηφία. Παραδείγματος χάριν για replication factor = 3 η τιμή quorum θα ισούται με (3/2)+1=1+1=2. Για περιπτώσεις εγγραφής τα πιθανά επίπεδα που μπορεί να ορίσει ο χρήστης φαίνονται στον πίνακα 4.4:

Επίπεδο	**Περιγραφή**
ANY	Αυτή η επιλογή προσδίδει στο σύστημα τη μεγαλύτερη δυνατή διαθεσιμότητα. Για να γίνει μία εγγραφή δεκτή από το σύστημα αρκεί ο κόμβος-συντονιστής με τον οποίο συνδέεται ο χρήστης να κάνει μια καταγραφή της εγγραφής με τη μορφή hinted-handoff, ακόμα και αν όλοι οι κόμβοι στους οποίους απευθύνεται η εγγραφή είναι μη διαθέσιμοι τη στιγμή της εγγραφής.
ONE	Η εγγραφή πρέπει να καταγραφεί στο commit log και το memory table τουλάχιστον ενός κόμβου
TWO	Η εγγραφή πρέπει να καταγραφεί στο commit log και το memory table τουλάχιστον δύο κόμβων
THREE	Η εγγραφή πρέπει να καταγραφεί στο commit log και το memory table τουλάχιστον τριών κόμβων
QUORUM	Η εγγραφή πρέπει να καταγραφεί στο commit log και το memory table σε τουλάχιστον τόσους κόμβους όσο είναι η τιμή του quorum.
LOCAL_QUORUM	Η τιμή του quorum σε αυτή την περίπτωση υπολογίζεται από τους κόμβους που βρίσκονται στην ίδια τοποθεσία με τον κόμβο που επικοινωνεί ο χρήστης. Χρησιμοποιείται ώστε να αποφεύγεται η καθυστέρηση που θα προέκυπτε, αν έπρεπε κόμβοι του συστήματος που βρίσκονται σε κάποια άλλη τοποθεσία(data center) να επιβεβαιώσουν την εγγραφή.
EACH_QUORUM	Η εγγραφή πρέπει να καταγραφεί σε quorum κόμβους σε κάθε μία από τις τοποθεσίες που βρίσκεται το σύστημα(local_quorum για κάθε τοποθεσία).
ALL	Η εγγραφή πρέπει να καταγραφεί στο commit log και το memory table όλων των κόμβων.

Πίνακας 4.4: Cassandra: Επίπεδα Συνέπειας Εγγραφών

Σε περίπτωση αναγνώσεων οι επιλογές είναι αντίστοιχες με τη διαφορά ότι δεν υπάρχει δυνατότητα να γίνει ανάγνωση από δεδομένα που βρίσκονται στο σύστημα σε μορφή hinted handoffs και δεν έχουν μονιμοποιηθεί ακόμα σε κάποιον κόμβο. Η τιμή της συνέπειας δείχνει από πόσους κόμβους, που περιέχουν επαναλήψεις των δεδομένων, ζητάει ο κόμβος συντονιστής τα δεδομένα, ώστε βάση χρονοσήμανσης να επιστρέψει στον χρήστη τα πιο πρόσφατα. Οι πιθανές τιμές φαίνονται στον πίνακα 4.5:

Επίπεδο	**Περιγραφή**
ONE	Επιστρέφει το αποτέλεσμα από τον πιο κοντινό κόμβο όπως αυτός επιλέγεται με βάση την πληροφορία snitch. Παράλληλα στέλνεται στο παρασκήνιο μία εντολή η οποία φροντίζει να κάνει τους υπόλοιπους κόμβους συνεπείς ως προς τα δεδομένα που αφορούσε η ανάγνωση
TWO	Επιστρέφεται το πιο πρόσφατο αποτέλεσμα από δύο από τους κοντινότερους κόμβους.
THREE	Επιστρέφεται το πιο πρόσφατο αποτέλεσμα από τρεις από τους κοντινότερους κόμβους.
QUORUM	Επιστρέφεται το πιο πρόσφατο αποτέλεσμα από αυτά που επιστρέφουν μία πλειοψηφία (quorum) κόμβων.
LOCAL_QUORUM	Επιστρέφεται το πιο πρόσφατο αποτέλεσμα από αυτά που επιστρέφουν μία πλειοψηφία (quorum) κόμβων από την ίδια τοποθεσία με τον κόμβο-συντονιστή.
EACH_QUORUM	Επιστρέφεται το πιο πρόσφατο αποτέλεσμα από αυτά που επιστρέφουν οι ανά τοποθεσία πλειοψηφίες (quorum) κόμβων(local_quorum για κάθε τοποθεσία).
ALL	Επιστρέφεται το πιο πρόσφατο αποτέλεσμα από όλους τους κόμβους.

Πίνακας 4.5: Cassandra: Επίπεδα Συνέπειας Αναγνώσεων

Με βάση τα παραπάνω, αν ένας χρήστης που πραγματοποιεί εγγραφές και αναγνώσεις στα δεδομένα, επιθυμεί να λαμβάνει κάθε φορά τα πιο πρόσφατα-συνεπή αποτελέσματα τότε θα πρέπει να φροντίσει να ισχύει η ακόλουθη σχέση:

```
(κόμβοι_εγγραφής + κόμβοι_ανάγνωσης) > παράγοντας_αντιγραφής
```

Παραδείγματος χάριν αν σε ένα σύστημα έχει οριστεί ο παράγοντας αντιγραφής να ισούται με 3 τότε για να έχει συνέπεια στα δεδομένα που διαβάζει μπορεί να ορίσει:

- Να μονιμοποιούνται οι εγγραφές σε 1 κόμβο(συνέπεια εγγραφής ONE) αλλά να διαβάζονται και από τους 3(συνέπεια ανάγνωσης ALL/THREE) για να επιλέγεται το πιο πρόσφατο.
- Να μονιμοποιούνται οι εγγραφές σε 2 κόμβους (συνέπεια εγγραφής TWO/QUORUM) και να διαβάζονται από 2 κόμβους (συνέπεια ανάγνωσης TWO/QUORUM).
- Να μονιμοποιούνται και στους 3 κόμβους οι εγγραφές (συνέπεια εγγραφής ALL/THREE) και να διαβάζονται από οποιονδήποτε (συνέπεια ανάγνωσης ONE).

Αν υπάρχει ανάγκη για αυξημένη συνέπεια, αυτό εισάγει κόστος στον χρόνο απόκρισης (latency) του συστήματος επομένως η παραπάνω σχέση θα πρέπει να λαμβάνεται υπόψιν ώστε να μην πραγματοποιούνται παραπάνω έλεγχοι από το σύστημα από όσους πραγματικά χρειάζονται για να εξασφαλιστεί η συνέπεια. Επίσης θα πρέπει να λαμβάνονται υπόψιν τα περιθώρια που δίνονται στο σύστημα να υπάρχει κάποιος κόμβος που δεν είναι διαθέσιμος πράγμα που εξασφαλίζεται με επιλογές που δεν απαιτούν επιβεβαίωση κάποιας ενέργειας από όλους τους κόμβους που συμμετέχουν σε κάποιο replica set.

Η εγγύηση της απομόνωσης (isolation) παρέχεται και αυτή με τη σειρά της ανά γραμμή. Δηλαδή αν ένας χρήστης πραγματοποιεί μία αλλαγή σε κάποιες στήλες μίας γραμμής τότε το αποτέλεσμα της αλλαγής θα είναι ορατό στους υπόλοιπους χρήστες μόνο αφού ολοκληρωθεί.

Οι εγγραφές στο σύστημα Cassandra προσφέρουν την εγγύηση της μονιμότητας αφού κάθε εγγραφή σε κάποιον κόμβο καταγράφεται και στη μνήμη αλλά και στο commit log στο δίσκο προτού η εγγραφή θεωρηθεί επιτυχής. Αν προκύψει αποτυχία στον κόμβο, πριν προλάβουν τα περιεχόμενα των πινάκων που διατηρούνται στη μνήμη να μονιμοποιηθούν στο δίσκο, τότε όταν ο κόμβος επανέλθει σε λειτουργία, χρησιμοποιούνται τα δεδομένα που έχουν αποθηκευτεί στο commit log για να επανεκτελεστούν οι λειτουργίες που δεν είχαν ακόμη μονιμοποιηθεί στο δίσκο Επίσης χρησιμοποιούνται τα hinted-handoffs που αποθήκευαν οι υπόλοιποι κόμβοι στο διάστημα που αυτός δεν ήταν διαθέσιμος, ώστε να φτάσουν τα δεδομένα στην πιο πρόσφατη εκδοχή τους και να είναι σύμφωνα με τα δεδομένα του υπόλοιπου συστήματος. Επιπλέον η διατήρηση αντιγράφων σε άλλους κόμβους κάνει την εγγύηση της μονιμότητας πιο ισχυρή.

4.3 MongoDB

4.3.1 Γενικά

Η MongoDB είναι μία βάση αρχείων η οποία ξεκίνησε να δημιουργείται το 2007 από την εταιρία 10gen (πλέον MongoDB) σαν μέρος ενός προϊόντος PaaS (Platform as a Service). Αργότερα η εταιρία επικέντρωσε τις προσπάθειές της μόνο στην βάση, βλέποντας τις προοπτικές που αυτή είχε. Είναι ανοιχτού λογισμικού γραμμένη σε γλώσσα C++ και σχεδιάστηκε για να καλύπτει αποτελεσματικά τις ανάγκες web εφαρμογών και με την προοπτική να μπορεί να κλιμακωθεί οριζόντια ανάλογα με τις ανάγκες, συνδυάζοντας δυνατά χαρακτηριστικά σχεσιακών βάσεων και NoSQL συστημάτων. Τα δεδομένα της αποθηκεύονται με τη μορφή αρχείων BSON (Binary

JSON). Τα κυριότερα χαρακτηριστικά της MongoDB είναι τα εξής:

- Δυνατότητες οριζόντιας αναβάθμισης με αυτόματη ισοκατανομή των δεδομένων στους κόμβους.
- Δυνατότητες αυτόματης επανάληψης των δεδομένων, με αυτόματη διαχείριση περιπτώσεων που κάποιος κόμβος σταματά να είναι διαθέσιμος.
- Η δομή των δεδομένων η οποία χρησιμοποιεί δίνει τη δυνατότητα να αποθηκεύονται τα δεδομένα που αφορούν κάποια οντότητα συγκεντρωμένα σε ένα αρχείο χωρίς να χρειάζονται πολύπλοκα JOIN queries προκειμένου να ανακτηθούν από τη βάση.
- Μεγάλη ευελιξία στο είδος των δεδομένων που μπορούν να αποθηκευτούν σε ένα αρχείο χωρίς να χρειάζεται να έχουν οριστεί εκ των προτέρων το είδος των τιμών που θα αποθηκευτούν.
- Υποστηρίζει secondary indexes για γρήγορες αναζητήσεις στη βάση.
- JavaScript Shell για αλληλεπίδραση και διαχείριση της βάσης με κάποιες επιπλέον προσθήκες για περισσότερη συμβατότητα με γνώριμες SQL εντολές και χρηστικότητα.

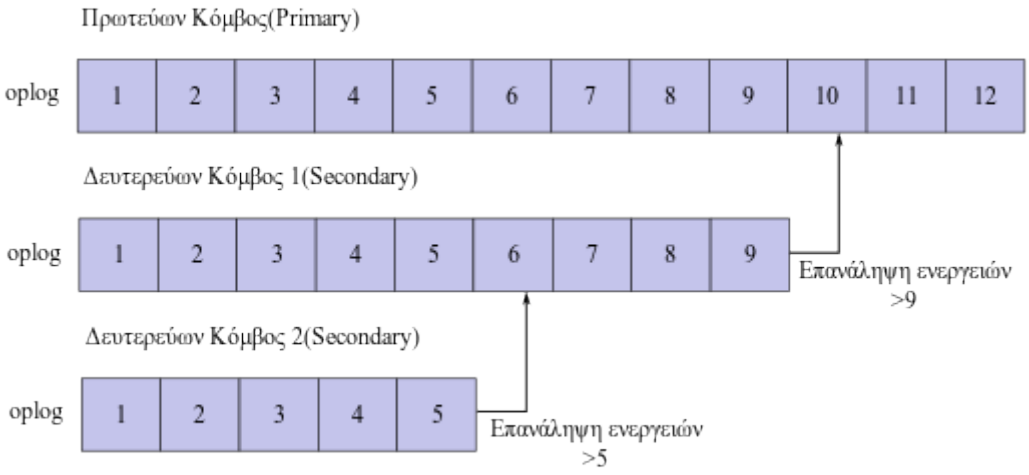
Αποτελεί μία από τις πιο διαδεδομένες λύσεις NoSQL με τον δικτυακό τόπο http://db-engines.com να την κατατάσσει στην πρώτη θέση των NoSQL λύσεων τη στιγμή της συγγραφής βασιζόμενο σε στατιστικά από αναζητήσεις στο διαδίκτυο, τεχνολογικές συζητήσεις, στατιστικά προσφοράς και ζήτησης θέσεων εργασίας και άλλα. Χρησιμοποιείται από εταιρίες όπως LinkedIn, McAfee, SAP, Sourceforge, Guardian.co.uk, eBay, Cisco, CERN και πολλές ακόμα [20].

4.3.2 Αρχιτεκτονική

Η αρχιτεκτονική ενός συστήματος MongoDB χωρίζεται σε δύο βασικά μέρη και προσαρμόζεται ανάλογα με τις ανάγκες. Το ένα μέρος αφορά την επανάληψη των δεδομένων (Replica Sets) και τον μηχανισμό με τον οποίο αυτά αντιγράφονται σε διαφορετικούς κόμβους, αλλά και με τον οποίο διαχειρίζονται καταστάσεις όπου κάποιος κόμβος σταματά να είναι διαθέσιμος. Το δεύτερο μέρος αφορά τον διαχωρισμό και την κατανομή των δεδομένων σε παραπάνω από έναν κόμβους (Sharding) ώστε να παρέχονται οι δυνατότητες αποδοτικής κλιμάκωσης του συστήματος.

Τα Replica Sets αποτελούνται από έναν πρωτεύοντα κόμβο (primary node) ο οποίος αναλαμβάνει την εξυπηρέτηση των εγγραφών και των αναγνώσεων για τα δεδομένα που αποθηκεύει, και έναν ή περισσότερους δευτερεύοντες κόμβους (secondary nodes) οι οποίοι φροντίζουν να διατηρούν

τα ίδια δεδομένα με αυτόν. Η αντιγραφή των δεδομένων στους δευτερεύοντες κόμβους γίνεται εκτελώντας μία προς μία τις εντολές που εκτελούνται στον πρωτεύοντα, και είναι αποθηκευμένες στο αρχείο καταγραφής των ενεργειών του (oplog). Στη γενική περίπτωση, η επανάληψη των ενεργειών στους δευτερεύοντες κόμβους γίνεται ασύγχρονα, δηλαδή ακολουθούν τις ενέργειες του με χρονολογική σειρά, όμως αυτό πραγματοποιείται αναγκαστικά με κάποια καθυστέρηση χωρίς όμως να επηρεάζουν τη λειτουργία του κυρίαρχου κόμβου μέχρι να δημιουργηθούν όλα τα αντίγραφα στους υπόλοιπους κόμβους. Έτσι, ενώ υπάρχει δυνατότητα οι δευτερεύοντες κόμβοι να εξυπηρετούν αναγνώσεις στα δεδομένα (για καλύτερες επιδόσεις ανάγνωσης από το σύστημα), αυτές οι αναγνώσεις δεν παρέχουν εγγυήσεις ότι τα δεδομένα θα είναι τα πιο πρόσφατα-συνεπή. Η διαδικασία της ασύγχρονης επανάληψης των ενεργειών φαίνεται πιο παραστατικά στο σχήμα 4.6 όπου κάθε κόμβος συγχρονίζει τα δεδομένα του από κάποιον άλλο ο οποίος έχει πιο πολλές εντολές αποθηκευμένες στο αρχείο καταγραφής ενεργειών. Επίσης φαίνεται ότι κάποιος δευτερεύων κόμβος μπορεί να συγχρονίζει τα δεδομένα του από κάποιον άλλο δευτερεύοντα κατανέμοντας το φορτίο αυτής της διαδικασίας στους επιμέρους κόμβους και απαλλάσσοντας τον πρωτεύοντα από αυτό.



Σχήμα 4.6: MongoDB:Διαδικασία αντιγραφής ενεργειών

Όλοι οι κόμβοι που περιλαμβάνονται σε ένα τέτοιο σύνολο, ανταγωνίζονται προκειμένου να γίνουν οι κυρίαρχοι κόμβοι του συνόλου. Σε περίπτωση που ο πρωτεύων κόμβος σταματήσει να είναι διαθέσιμος, τότε ξεκινά αυτόματα μία διαδικασία ψηφοφορίας ανάμεσα στους υπόλοιπους και εκλέγεται καινούριος κυρίαρχος κόμβος ο οποίος αναλαμβάνει στη συνέχεια την εξυπηρέτηση των εντολών στα δεδομένα που έχει το Replica Set. Στην ψηφοφορία αυτή προκειμένου κάποιος κόμβος να γίνει πρωτεύων θα πρέπει να εξασφαλίσει ψήφους από

τουλάχιστον τους μισούς κόμβους του συνόλου. Αυτό γίνεται για να αποκλείεται η πιθανότητα τα μέλη του συνόλου να χωριστούν σε δύο τμήματα λόγω κάποιας βλάβης δικτύου που ενώνει δύο τοποθεσίες παραδείγματος χάριν, εκλέγοντας στη συνέχεια το κάθε τμήμα τον δικό του πρωτεύοντα κόμβο. Στην ψηφοφορία αυτή μπορεί να συμμετέχει και ένας ειδικός κόμβος που ονομάζεται Arbiter (ρυθμιστής) ο οποίος χρησιμοποιείται μόνο για να επιλύει ισοψηφίες χωρίς να αποθηκεύει δεδομένα.

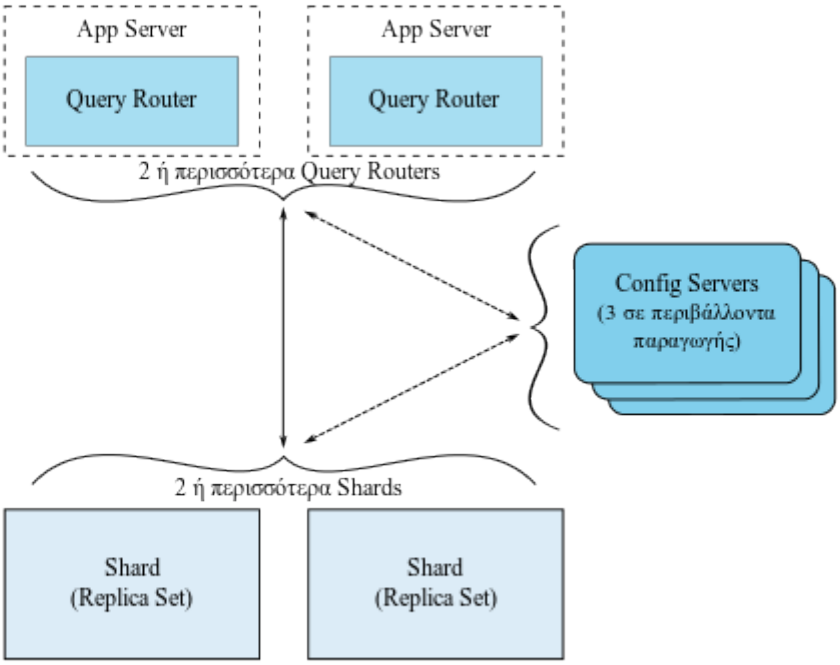
Το μέρος το οποίο φροντίζει για την ισοκατανομή των δεδομένων και την οριζόντια κλιμάκωση αποτελείται από τρία μέρη. Τα Shards, τους Config Servers και τα Query Routers.

Η MongoDB χωρίζει τα δεδομένα που αποθηκεύει η βάση σε τμήματα που ονομάζονται chunks και στη συνέχεια φροντίζει για την ισοκατανομή τους στους διαθέσιμους κόμβους. Shards ονομάζονται τα μηχανήματα που αναλαμβάνουν να εξυπηρετήσουν τις λειτουργίες στα τμήματα δεδομένων που τους ανατίθενται. Ένα Shard μπορεί να αποτελείται από ένα μοναδικό μηχάνημα, ή από κάποιο σύνολο μηχανημάτων που διαμορφώνουν ένα Replica Set προκειμένου να εξασφαλίζεται η σταθερή λειτουργία με ανοχή σε αποτυχίες υλικού, κατατμήσεις δικτύου και σε ό,τι άλλες περιπτώσεις κάποιος κόμβος σταματά να εξυπηρετεί τις ανάγκες του συστήματος.

Τα Query Routers είναι υπεύθυνα για την αλληλεπίδραση των χρηστών με τη βάση κατευθύνοντας τις ενέργειες που εκτελούν αυτοί στα αντίστοιχα μηχανήματα που περιέχουν τα ζητούμενα δεδομένα. Επίσης εκτελούν τις απαραίτητες λειτουργίες για να διατηρηθεί η ισοκατανομή των δεδομένων στους επιμέρους κόμβους του συστήματος. Αν σε κάποιο Shard υπάρχουν περισσότερα δεδομένα από ότι σε κάποιο άλλο τότε ξεκινάει μία διεργασία που ονομάζεται balancer και αναλαμβάνει την μεταφορά τμημάτων δεδομένων ώστε να αποκατασταθεί η ισοκατανομή. Αρχικά αντιγράφονται τα αποθηκευμένα τμήματα (chunks) στο καινούριο Shard. Έπειτα ενημερώνονται τα δεδομένα αυτών των τμημάτων με ότι ενέργειες εκτελέστηκαν πάνω σε αυτά κατά το διάστημα που γινόταν η μεταφορά τους, στη συνέχεια ενημερώνονται οι Config Servers για τη νέα θέση των δεδομένων και εφόσον όλα αυτά εκτελεστούν με επιτυχία, διαγράφονται τα δεδομένα από το αρχικό Shard. Ο διαχωρισμός των δεδομένων σε κομμάτια (chunks) γίνεται χρησιμοποιώντας κάποιο κλειδί που ορίζεται στα δεδομένα και μπορεί είτε να προσδιοριστεί από τον διαχειριστή, ο οποίος γνωρίζοντας την μορφή του κλειδιού εξασφαλίζει την σωστή τοποθέτηση και κατανομή των δεδομένων στα διάφορα Shards, είτε να γίνει διαχωρισμός με βάση κάποιο hash ενός κλειδιού το οποίο σε διαφορετική περίπτωση δε θα εξασφάλιζε καλή κατανομή των δεδομένων. Με τη χρήση όμως της συνάρτησης κατακερματισμού πιθανώς να χάνονται κάποια πλεονεκτήματα τοπικότητας των δεδομένων.

Οι Config Servers διατηρούν τα μεταδεδομένα του συστήματος, παρέχοντας πληροφορίες στα Query Routers για το πού βρίσκονται τα δεδομένα. Έτσι όταν κάποιος χρήστης θέλει να εκτελέσει κάποιες ενέργειες σε κάποια δεδομένα, επικοινωνεί με κάποιο Query Router, αυτό στη συνέχεια λαμβάνει πληροφορίες από τους Config Servers για το ποιο/ποιά Shard τα περιέχουν και στη συνέχεια φέρνει σε επαφή το χρήστη με τα αντίστοιχα Shards. Οι Config Servers πρέπει να είναι τρεις σε περιβάλλοντα παραγωγής προκειμένου να εξασφαλίζεται η διαθεσιμότητά τους.

Η γενική εικόνα ενός συστήματος MongoDB φαίνεται στο σχήμα 4.7.



Σχήμα 4.7: MongoDB: Γενική Εικόνα Συστήματος

(Πηγή: http://docs.mongodb.org/v2.4/)

4.3.3 Εσωτερικοί Μηχανισμοί

Η MongoDB προκειμένου να αυξήσει την επίδοσή της, δεν αποθηκεύει απ' ευθείας τα αρχεία που δημιουργούνται ή τροποποιούνται στον δίσκο, αλλά χρησιμοποιεί την προσωρινή μνήμη RAM. Σε αυτή δημιουργεί μία απεικόνιση των αρχείων (memory-mapped files) που περιέχει η βάση η οποία ονομάζεται shared view, την οποία εν γένει διαχειρίζεται το λειτουργικό σύστημα,

φορτώνοντας από τον δίσκο και γράφοντας σε αυτόν αρχεία ανάλογα με τις ανάγκες. Η MongoDB, με βάση τις προεπιλεγμένες ρυθμίσεις, προκαλεί μονιμοποίηση των δεδομένων αυτών στον δίσκο ανά 60 δευτερόλεπτα. Το χρονικό διάστημα αυτό μπορεί να είναι μικρότερο, αν το ορίσει ο διαχειριστής ή αν το λειτουργικό σύστημα το κρίνει αναγκαίο(κυρίως σε περιπτώσεις που η διαθέσιμη μνήμη είναι περιορισμένη)

Προκειμένου να βελτιωθούν οι εγγυήσεις μονιμότητας των δεδομένων σε διάφορα σενάρια αποτυχίας, χρησιμοποιείται ένας μηχανισμός journaling[21]. Κατά τη λειτουργία του journaling χρησιμοποιείται μία ακόμα απεικόνιση στη μνήμη που ονομάζεται private view. Κάθε καινούρια καταγραφή που πραγματοποιείται στη βάση αρχικά καταγράφεται στην private view η οποία εξυπηρετεί και λειτουργίες ανάγνωσης στα δεδομένα αφού περιέχει τα πιο πρόσφατα. Στη συνέχεια ανά διαστήματα 100ms, όποιες εγγραφές έχουν πραγματοποιηθεί, καταγράφονται σειριακά σε αρχεία στον δίσκο που ονομάζονται journals. Αφού οι ενέργειες καταγραφούν στο journal στη συνέχεια εκτελούνται στη shared view οπότε και τα αρχεία που βρίσκονται σε αυτή τη μνήμη είναι τροποποιημένα σε σχέση με τα αρχεία που βρίσκονται στο δίσκο μέχρι να πραγματοποιηθεί ο επόμενος συγχρονισμός με αυτά. Χρησιμοποιώντας τα αρχεία journal, η MongoDB είναι σε θέση να επαναλάβει στη shared view ότι ενέργειες είχαν πραγματοποιηθεί σε περίπτωση που για οποιοδήποτε λόγο διακοπεί η λειτουργία της εφαρμογής και να ανακτήσει έτσι ότι δεδομένα δεν είχαν προλάβει να μονιμοποιηθούν στα αρχεία του δίσκου. Έτσι εξασφαλίζεται ότι σε καμία περίπτωση δε θα χαθούν δεδομένα που αφορούν διάστημα παραπάνω από 100ms. Το χρονικό αυτό διάστημα μπορεί να μειωθεί από τον διαχειριστή της βάσης αν τα δεδομένα είναι ιδιαίτερα σημαντικά ή να αυξηθεί σε περίπτωση που είναι επιθυμητή η μείωση του αντίκτυπου που έχει στην επίδοση του συστήματος η χρησιμοποίηση του journal. Επίσης υπάρχει δυνατότητα από την εφαρμογή που χρησιμοποιεί τη βάση να ζητήσει επιβεβαίωση ότι κάποια πολύ σημαντικά δεδομένα μονιμοποιήθηκαν στο δίσκο, κάνοντας το σύστημα να μειώνει το χρονικό διάστημα καταγραφής στα 30ms. Αυτή η τακτική όμως αυξάνει σημαντικά τους χρόνους απόκρισης του συστήματος (latency). Από τη στιγμή που τα δεδομένα μονιμοποιηθούν στον δίσκο μέσω της shared view, ενημερώνονται τα αρχεία journal που περιέχουν τα αντίστοιχα δεδομένα ώστε να διαγραφούν ή να ανακυκλωθούν για περαιτέρω χρήση από το σύστημα.

Επίσης η MongoDB φροντίζει να δεσμεύει εκ των προτέρων χώρο στον δίσκο (preallocation) για τα αρχεία που θα αποθηκευτούν σε κάποια βάση (αλλά και για τα journal αρχεία) έτσι ώστε να αποφεύγονται οι σημαντικές καθυστερήσεις που προκύπτουν όταν πρέπει να δεσμευτεί χώρος στον δίσκο τη στιγμή που αποθηκεύεται ένα αρχείο, αλλά και για να περιορίζεται ο κατακερματισμός των δεδομένων στον δίσκο. Για τον ίδιο σκοπό, φροντίζει επίσης να αφήνει

κάποιον διαθέσιμο χώρο ελεύθερο μετά από κάποιο αρχείο (padding), ώστε πιθανές ενημερώσεις και προσθήκες σε αυτό να μην έχουν σαν αποτέλεσμα την αναζήτηση νέας θέσης στον δίσκο και τον κατακερματισμό της βάσης.

Όλες οι παραπάνω τεχνικές (Journaling, Preallocation, Padding) έχουν σαν αποτέλεσμα την αυξημένη χρήση του διαθέσιμου αποθηκευτικού χώρου. Αυτό γίνεται όμως με σκοπό την αυξημένη απόδοση του συστήματος και σε βάσεις με μεγάλο όγκο δεδομένων, η αναντιστοιχία στον πραγματικό όγκο των δεδομένων που αποθηκεύονται και στον όγκο που καταλαμβάνει η βάση δεν γίνεται τόσο έντονα αισθητή.

4.3.4 Δομή Δεδομένων

Στη MongoDB όλα τα δεδομένα αποθηκεύονται με τη μορφή αρχείων BSON. Ένα αρχείο θα μπορούσε να χαρακτηριστεί το αντίστοιχο μίας γραμμής των σχεσιακών βάσεων δεδομένων, όμως στην πράξη είναι πολύ πιο περιγραφικός ο τρόπος που αποθηκεύει δεδομένα και εκτός από ζεύγη κλειδιού-τιμής μπορεί να περιέχει λίστες, αναφορές σε άλλα αρχεία της βάσης ή ακόμα και ενσωματωμένα αρχεία. Τα αρχεία οργανώνονται σε συλλογές αρχείων (collections) οι οποίες θα μπορούσαν να χαρακτηριστούν το αντίστοιχο των σχεσιακών πινάκων. Τέλος κάθε σύστημα MongoDB μπορεί να φιλοξενεί αρκετές βάσεις οι οποίες με τη σειρά τους αποτελούνται από συλλογές.

Τα αρχεία BSON αποτελούν μία παραλλαγή των αρχείων JSON (Binary JSON). Παρακάτω φαίνεται ένα παράδειγμα δύο τέτοιων αρχείων:

```
{
  _id : 111111111,
  name: "Mathima1",
  kathightes: [ "Kathigitis1" , "Kathigitis2" ],
  wres: 6
  programma:{
             deytera:"15:00-18:00",
             pempth:"16:00-19:00"
            }
}


{
  _id: 222222222,
  name: "Mathitis1",
  am: 03100000
  mathima_id: "111111111"

}
```

Πίνακας 4.6: MongoDB: Παράδειγμα αρχείων - Δομή Δεδομένων

Στο παραπάνω παράδειγμα φαίνονται διάφορα χαρακτηριστικά των αρχείων της MongoDB. Το πρώτο αρχείο που θα μπορούσε να αποτελεί μέρος μιας συλλογής μαθημάτων. Φαίνεται ότι κάθε αρχείο περιέχει μια ένα μοναδικό αναγνωριστικό _id. Επίσης φαίνεται πως ορίζονται τα ζεύγη κλειδιού-τιμής, με τις τιμές να μπορούν να είναι αλφαριθμητικά("Mathima1"), αριθμοί (wres: 6), πίνακες τιμών (kathightes: ["Kathigitis1" , "Kathigitis2"]), ή ακόμα και ενσωματωμένα αρχεία μέσα στο ίδιο αρχείο(programma). Επίσης από το δεύτερο αρχείο, που θα μπορούσε να αποτελεί μέρος μιας συλλογής μαθητών, φαίνεται πώς η τιμή μπορεί να είναι αναφορά σε κάποιο άλλο αρχείο επιτρέποντας έτσι τη δημιουργία συσχετίσεων μεταξύ των διαφορετικών αρχείων. Η δομή αυτή των αρχείων επιτρέπει να αποτυπώνονται μέσα στο ίδιο αρχείο μίας οντότητας μια σειρά από δεδομένα και σχέσεις που την αφορούν. Με αυτό τον τρόπο η ανάκτηση των δεδομένων που αφορούν την οντότητα αυτή γίνεται εύκολα και αποφεύγονται πολλαπλά JOINS μεταξύ πινάκων, και πολλαπλές αναγνώσεις από τον δίσκο προκειμένου να συγκεντρωθούν τα αποτελέσματα. Επίσης όταν τα δεδομένα βρίσκονται σε ένα αρχείο παρέχονται εγγυήσεις ατομικότητας για τις ενέργειες που πραγματοποιούνται σε αυτό που θα σχολιαστούν στην επόμενη ενότητα. Εναλλακτικά μπορούν να ορίζονται αναφορές σε άλλα αντικείμενα όπως στο παραπάνω παράδειγμα για να αποφεύγεται η επανάληψη των δεδομένων με το κόστος όμως ότι θα χρειάζονται παραπάνω από μία αναγνώσεις για να ανακτηθούν. Γενικά οι βάσεις στη MongoDB δεν είναι κανονικοποιημένες. Το πόσο συχνή θα είναι η επανάληψη κάποιων δεδομένων στη βάση αποτελεί και αντικείμενο σχεδιασμού και πρέπει να αντισταθμίζονται τα πλεονεκτήματα με τα μειονεκτήματα από τον τρόπο με τον οποίο αποθηκεύονται τα δεδομένα.

Σε ότι αφορά τη σύγκριση των BSON αρχείων με τα τυπικά JSON αρχεία, τα πρώτα είναι σχεδιασμένα έτσι ώστε στις περισσότερες περιπτώσεις μπορούν να αναπαριστούν τα δεδομένα που περιέχουν αρκετά πιο αποδοτικά σε σχέση με τα JSON, ενώ στη χειρότερη περίπτωση είναι λίγο λιγότερο αποδοτικά από αυτά. Επίσης έχουν κάποια επιπλέον χαρακτηριστικά ώστε να γίνεται πιο γρήγορα η προσπέλαση των δεδομένων τους και είναι σχεδιασμένα με τέτοιο τρόπο ώστε να είναι γρήγορη κωδικοποίηση και η αποκωδικοποίησή τους.

4.3.5 ACID εγγυήσεις

Όλες οι εγγραφές είναι ατομικές σε επίπεδο αρχείου. Αυτό περιλαμβάνει όλες τις μετατροπές που αφορούν ένα αρχείο, ακόμη και αν αυτό περιέχει ενσωματωμένα αρχεία. Για ενέργειες που αφορούν τροποποιήσεις σε παραπάνω από ένα αρχεία η εγγύηση της ατομικότητας δε μπορεί να

είναι δεδομένη αφού μπορεί να υπάρξει παρεμβολή από ενέργειες άλλων χρηστών της βάσης πάνω σε αυτά τα αρχεία. Αρκετά από τα προβλήματα που απαιτούν εγγυήσεις ατομικότητας μπορούν να λυθούν οργανώνοντας τα αρχεία με τέτοιο τρόπο ώστε οι ενέργειες να πραγματοποιούνται εντός του ίδιου αρχείου και η δομή των αρχείων της MongoDB με τις δυνατότητες για ενσωματωμένα αρχεία επιτρέπει μία τέτοια προσέγγιση. Εναλλακτικά υπάρχει δυνατότητα να πραγματοποιηθούν κάποιες αλλαγές που αφορούν παραπάνω από ένα αρχεία χρησιμοποιώντας μεθόδους που δίνουν κάποια από τα χαρακτηριστικά των two-phase-commits των σχεσιακών βάσεων. Ορίζονται κάποιες νέες τιμές ώστε να αποτρέπονται άλλες εφαρμογές από το να πραγματοποιούν αλλαγές όσο διαρκούν οι αλλαγές στα αρχεία και κάποιες άλλες για να καταγράφεται η εξέλιξη των αλλαγών και να υπάρχει δυνατότητα επιστροφής στην αρχική κατάσταση της βάσης αν κάτι δεν πάει καλά.

Σε ότι αφορά τη συνέπεια των δεδομένων, η προεπιλεγμένη λειτουργία της MongoDB παρέχει εγγυήσεις συνέπειας στα δεδομένα αφού οι αναγνώσεις πραγματοποιούνται μόνο από τον πρωτεύοντα κόμβο του κάθε Replication Set, και ο κόμβος αυτός περιέχει πάντα τα πιο πρόσφατα δεδομένα. Υπάρχει δυνατότητα όμως οι αναγνώσεις να εξυπηρετούνται και από τους δευτερεύοντες κόμβους αυξάνοντας έτσι τις επιδόσεις ανάγνωσης αλλά παρέχοντας συνέπεια εν τέλει (eventual consistency).

Ως προς την εγγύηση της απομόνωσης οι ενέργειες εγγραφής και ενημέρωσης σε επίπεδο ενός αρχείου γίνονται απομονωμένες, δηλαδή κάποια ανάγνωση δεν μπορεί να διαβάσει ενδιάμεσες καταστάσεις ενός αρχείου αν πραγματοποιείται ταυτόχρονα με κάποια ενημέρωση. Παρ' όλα αυτά μία ανάγνωση μπορεί να διαβάσει ένα αρχείο από τη μνήμη (private view) προτού αυτό καταγραφεί στο journal και μονιμοποιηθεί στον δίσκο. Έτσι δεν αποκλείεται η πιθανότητα μία ανάγνωση να διαβάσει κάποια δεδομένα αμέσως πριν σταματήσει απροσδόκητα η λειτουργία της εφαρμογής, και όταν επανέλθει σε λειτουργία αυτά τα δεδομένα να μην υπάρχουν στη βάση.

Τέλος, ως προς την εγγύηση της μονιμότητας των δεδομένων[22], οι διαδικασίες που αφορούν το journal και εξασφαλίζουν την καταγραφή των δεδομένων στον δίσκο, περιγράφηκαν σε προηγούμενη ενότητα. Από τη στιγμή που κάποιο δεδομένο έχει καταγραφεί στο journal εξασφαλίζεται ότι έχει μονιμοποιηθεί. Στις περιπτώσεις που χρησιμοποιούνται replica sets για να θεωρείται κάποιο δεδομένο ότι έχει πραγματικά μονιμοποιηθεί, θα πρέπει να έχει περαστεί σε μία πλειοψηφία των κόμβων που το αποτελούν. Αυτό συμβαίνει γιατί σε περίπτωση αποτυχίας παραδείγματος χάριν του πρωτεύοντα κόμβου, οι δευτερεύοντες μπορεί να συνεχίσουν τη λειτουργία τους χωρίς το δεδομένο αυτό, και όταν ο κόμβος επανέλθει να διαγράψει κάποια από τα δεδομένα του προκειμένου να είναι σε συνεπή κατάσταση με το υπόλοιπο replica set.

Κεφάλαιο 5
Στόχος και Περιγραφή του Πειράματος

5.1 Yahoo! Cloud Serving Benchmark (YCSB)

YCSB version: 0.1.4

Το Yahoo! Cloud Serving Benchmark (YCSB) [23] είναι ένα μετροπρόγραμμα το οποίο όπως δηλώνει και το όνομά του συγκρίνει τις επιδόσεις συστημάτων τα οποία εξυπηρετούν τις ανάγκες υπηρεσιών cloud. Το πιο συνηθισμένο σενάριο χρήσης είναι η ανάγνωση και η εγγραφή δεδομένων online, με απλό παράδειγμα να είναι η χρήση μίας ιστοσελίδας, όπου πραγματοποιούνται εγγραφές στη βάση δεδομένων για τη δημιουργία της και αναγνώσεις για την εμφάνισή της στους χρήστες. Παρέχει ένα σύνολο από προκαθορισμένα φορτία εργασίας, τα οποία αποτελούνται από ένα μείγμα από εγγραφές, αναγνώσεις και αναζητήσεις δεδομένων, και στη συνέχεια εφαρμόζονται πάνω στα διάφορα εγκατεστημένα συστήματα και επιστρέφει στατιστικά για το πόσο γρήγορα εκτελέστηκαν οι συγκεκριμένες εργασίες. Επίσης δίνει τη δυνατότητα να καθοριστούν από τον χρήστη διαφορετικά από τα προκαθορισμένα σενάρια χρήσης, αλλάζοντας τις αναλογίες των εγγραφών , των αναγνώσεων και των αναζητήσεων, ώστε τα αποτελέσματα να αντικατοπτρίζουν με μεγαλύτερη ακρίβεια την απόδοση των συστημάτων όταν αυτά λειτουργούν με βάση τις ξεχωριστές ανάγκες.

Το YCSB είναι ουσιαστικά ένας client ο οποίος ανάλογα με τις παραμέτρους που του δίνονται παράγει μια σειρά από γενικές εντολές ανάγνωσης, εγγραφής, ενημέρωσης και αναζήτησης δεδομένων στη βάση. Επίσης αποτελείται από διάφορες βιβλιοθήκες οι οποίες χρησιμοποιούνται για να μετατρέπονται αυτές οι εντολές σε μορφή που να επιτρέπει την αλληλεπίδραση με τις διάφορες βάσεις που εξετάζονται. Στον client ορίζεται εκτός των άλλων, μία παράμετρος που καθορίζει πόσες ενέργειες εκτελεί στη βάση ανά δευτερόλεπτο(Throughput: Ops/sec) οπότε και αυτός είναι ο επιθυμητός ρυθμός απόδοσης της βάσης. Στη συνέχεια καταγράφει πόσες από αυτές τις ενέργειες εξυπηρετεί η βάση ανά δευτερόλεπτο καθώς και τον χρόνο απόκρισης (latency) για κάθε μία από αυτές. Τα αποτελέσματα από κάθε τέτοιο διαδικασία μέτρησης είναι ένας μέσος όρος για το throuput που πέτυχε η βάση, ένας μέσος όρος για τον χρόνο απόκρισης και μία κατανομή των ενεργειών με βάση τους χρόνους απόκρισης. Έτσι εκτελώντας κάθε φορτίο για μια σειρά από επιθυμητούς ρυθμούς απόδοσης προκύπτουν τα διαγράμματα που δείχνουν τη σχέση Throughput(Ops/s) / Latency(ms) όσο αυξάνονται οι απαιτήσεις των χρηστών

σε ενέργειες στη βάση.

Τα φορτία που χρησιμοποιήθηκαν ήταν τα προκαθορισμένα και φαίνονται στον πίνακα 5.1.

Φορτίο (Workload)	**Ενέργειες**	**Κατανομή επιλογής καταχωρήσεων**	**Παράδειγμα εφαρμογής**
A - Update heavy	Αναγνώσεις: 50% Ενημερώσεις: 50%	Zipfian	Αποθήκευσή της Session ώστε να καταγράφονται οι πρόσφατες ενέργειες των χρηστών.
B - Read heavy	Αναγνώσεις: 95% Ενημερώσεις: 5%	Zipfian	Επισήμανση φωτογραφιών (Photo tagging). Οι περισσότερες λειτουργίες αποτελούν αναγνώσεις των επισημάνσεων και λιγότερες οι ενημερώσεις των δεδομένων με νέες επισημάνσεις.
C - Read only	Αναγνώσεις: 100%	Zipfian	Χρησιμοποίηση ως cache με τα δεδομένα να φτιάχνονται από κάποιο άλλο σύστημα.
D - Read latest	Αναγνώσεις: 95% Εισαγωγές: 5%,	Latest	Καταστάσεις χρηστών σε κοινωνικά δίκτυα, όπου υπάρχουν λιγότερες καταχωρήσεις συγκριτικά με τις αναγνώσεις και οι αναγνώσεις ζητούν τα πιο πρόσφατα αποτελέσματα
E - Short ranges	Αναζητήσεις: 95% Εισαγωγές: 5%	Zipfian/Uniform	Συζητήσεις σε φόρουμ που οργανώνονται σε θέματα, και κάθε αναζήτηση γίνεται με σκοπό να ανακτηθούν οι καταχωρήσεις που αφορούν κάθε θέμα.
F – Read – modify - write	Αναγνώσεις 50% Ανάγνωση - Μετατροπή - Εγγραφή 50%	Zipfian	Μία βάση χρηστών όπου οι καταγραφές διαβάζονται από τους χρήστες, τροποποιούνται και αποθηκεύονται πίσω στη βάση.

Πίνακας 5.1: Φορτία YCSB

Η διακριτή κατανομή Zipfian [24] που χρησιμοποιείται από τα περισσότερα φορτία περιγράφεται από τον τύπο:

$$f(k;s,N)=\frac{1/k^s}{\sum_{n=1}^{N}(1/n^s)}$$

όπου N ο αριθμός των στοιχείων σε έναν πληθυσμό, k ο βαθμός κάθε στοιχείου και s ο εκθέτης που χαρακτηρίζει την κατανομή. Ο παραπάνω τύπος δείχνει τη συχνότητα με την οποία εμφανίζονται τα διάφορα στοιχεία του πληθυσμού ανάλογα με τον βαθμό τους και τις υπόλοιπες παραμέτρους της κατανομής. Η κατανομή αυτή έχει διαπιστωθεί ότι περιγράφει αρκετά καθημερινά φαινόμενα. Στην απλή περίπτωση όπου s = 1 δείχνει ότι το στοιχείο που εμφανίζεται τις περισσότερες φορές σε έναν πληθυσμό (βαθμός=1) έχει συχνότητα εμφάνισης διπλάσια από το στοιχείο που είναι δεύτερο σε συχνότητα (βαθμός=2), τριπλάσια από το στοιχείο που είναι τρίτο σε συχνότητα(βαθμός=3) κ.ο.κ. Με αυτόν τον τρόπο επιλέγονται και τα δεδομένα πάνω στα οποία εκτελούνται οι ενέργειες, αντικατοπτρίζοντας το πραγματικό σενάριο χρήσης, οπού κάποια δεδομένα χρησιμοποιούνται αρκετά περισσότερο από κάποια άλλα.

Η κατανομή Uniform όπως δηλώνει και το όνομά της είναι η ομοιογενής ενώ η Latest χρησιμοποιείται ώστε να επιλέγονται τα πιο πρόσφατα δεδομένα της βάσης σε συνδυασμό με καταχωρήσεις που γίνονται από τα φορτία που τη χρησιμοποιούν.

Αρχικά πραγματοποιήθηκε εισαγωγή των δεδομένων στη βάση πάνω στα οποία εκτελούνταν στη συνέχεια οι ενέργειες που καθορίζονταν στα υπόλοιπα φορτία. Επειδή τα φορτία D και E εισάγουν καινούρια δεδομένα στη βάση και αλλάζουν τα δεδομένα που περιέχει αυτή, ο τρόπος που χρησιμοποιήθηκε ο client ήταν ο εξής:

- Αρχικά εισήχθησαν όσα δεδομένα χρησιμοποιήθηκαν στη συνέχεια των μετρήσεων δίνοντας στον YCSB-client την παράμετρο load.
- Στη συνέχεια εκτελέστηκαν τα φορτία A, B, C, F και τέλος το φορτίο D
- Διαγράφηκαν ότι δεδομένα είχε η βάση και πραγματοποιήθηκε νέα εισαγωγή των αρχικών δεδομένων.
- Τέλος εκτελέστηκε το φορτίο E.

Οι μετρήσεις εκτελέστηκαν αυξάνοντας τον επιθυμητό ρυθμό απόδοσης με βήματα χιλίων ενεργειών ανά δευτερόλεπτο. Κάθε μέτρηση επαναλήφθηκε τέσσερις φορές προκειμένου να υπάρχει δυνατότητα να αποκλειστούν μετρήσεις που παρουσίαζαν μεγάλες αποκλίσεις από την μέση συμπεριφορά του συστήματος και μπορεί να οφείλονταν σε αστάθμητους παράγοντες. Το γεγονός ότι το σύστημα που χρησιμοποιήθηκε βρισκόταν σε υπηρεσία cloud εισήγαγε τέτοιου είδους παράγοντες αφού η απόδοση τόσο του επεξεργαστή όσο και των αποθηκευτικών μέσων θα μπορούσε να επηρεαστεί από την ταυτόχρονη χρήση τους από άλλους χρήστες της υπηρεσίας.

5.2 BonFIRE Project

Οι δοκιμές πραγματοποιήθηκαν στο σύστημα BonFIRE [25]. Ο Οργανισμός Bonfire δημιουργήθηκε με τη συμμετοχή μεγάλων επιχειρήσεων του κλάδου(ATOS, HP, SAP), ερευνητικών(IT Innovation, FhG Fokus, Inria, i2CAT) και ακαδημαϊκών (UCM, EPCC, HLRS Stuttgart, iMinds, TUB) φορέων με σκοπό να δώσει τη δυνατότητα για πειράματα και έρευνα σχετικά με Cloud υπηρεσίες που κατανέμονται σε περισσότερες από μία τοποθεσίες οι οποίες περιέχουν ετερογενείς διαθέσιμους πόρους.

Η διαχείριση των διαθέσιμων πόρων γίνεται μέσω μίας διεπαφής που βασίζεται στην Open Cloud Computing Interface(OCCI) και προσφέρει έναν ομογενή τρόπο αλληλεπίδρασης με ετερογενείς υποδομές που περιλαμβάνει το σύστημα BonFIRE (OpenNebula, Virtual Wall, Cells).

Για τους σκοπούς της εργασίας χρησιμοποιήθηκαν οι εξής πόροι:

Ένας κόμβος (1) με συγκριτικά αυξημένες δυνατότητες που ανέλαβε την εξυπηρέτηση των επιπλέον αναγκών που απαιτούσαν οι λειτουργίες των master κόμβων σε HBase και HDFS και έγινε προσπάθεια να επιφορτιστεί με επιπλέον λειτουργίες στα υπόλοιπα δύο συστήματα ώστε να αξιοποιείται κατά το δυνατόν περισσότερο και από αυτά. Τα χαρακτηριστικά του ήταν τα εξής:

- 4 Physical Cores
- 10 GB RAM

Πέντε κόμβοι (2-6) για την δημιουργία του συνολικού συστήματος αποτελούμενοι ο καθένας από:

- 2 Physical Cores
- 4 GB RAM

Ένας κόμβος (7) που ανέλαβε τον ρόλο του YCSB-Client. Δημιουργήθηκε σε ξεχωριστό κόμβο προκειμένου να μην επιβαρύνει με την λειτουργία του τα υπό εξέταση συστήματα. Τα χαρακτηριστικά του ήταν τα εξής:

- 4 Physical Cores
- 1 GB RAM.

Σε κάθε κόμβο (πλην του Client) προστέθηκαν 10GB αποθηκευτικού χώρου με σύστημα αρχείων ext4 πάνω στον οποίο ορίστηκαν να αποθηκεύονται όλα τα δεδομένα από τη λειτουργία των συστημάτων.

Όλοι οι κόμβοι του συστήματος χρησιμοποιούσαν λειτουργικό σύστημα Debian Squeeze v6 (linux kernel: 2.6.32-5-amd64).

Η διαθέσιμη μνήμη που υπήρχε δυνατότητα να δεσμευτεί ήταν δυστυχώς περιορισμένη και όλα τα συστήματα που εξετάζονται είχαν αυξημένες προτεινόμενες απαιτήσεις σε αυτόν τον τομέα προκειμένου να παρουσιάζουν τις μέγιστες δυνατότητές τους. Συνοπτικά τα τεχνικά χαρακτηριστικά των κόμβων φαίνονται στον πίνακα 5.2:

	Κόμβος 1	Κόμβοι 2,3,4,5,6	Κόμβος 7
CPU	4 Physical Cores (AMD Opteron 6176)	2 Physical Cores ο καθένας (2:AMD Opteron 6176, 3-6: Intel Xeon E5620)	4 Physical Cores (AMD Opteron 6176)
RAM	10 GB	4 GB	1 GB
Disk	10GB ext3 : Λειτουργικό + Λογισμικό 10GB ext4 : Δεδομένα	10GB ext3 : Λειτουργικό + Λογισμικό 10GB ext4 : Δεδομένα	10GB ext3 : Λειτουργικό + Λογισμικό
OS	Debian Squeeze v6 (kernel:2.6.32-5-amd64)	Debian Squeeze v6 (kernel:2.6.32-5-amd64)	Debian Squeeze v6 (kernel:2.6.32-5-amd64)

Πίνακας 5.2: Τεχνικά χαρακτηριστικά κόμβων

5.3 Συστήματα

Η κατανομή των διεργασιών στους κόμβους των διαφόρων συστημάτων, καθώς και οι παράμετροι που δόθηκαν σε αυτά και καθόριζαν τη λειτουργία τους έγινε με κριτήριο, αφενός τα διαθέσιμα μηχανήματα να αξιοποιούνται όσο το δυνατόν αντίστοιχα, αφετέρου τα συστήματα να εκτελούν αντίστοιχες λειτουργίες κατά την εκτέλεση του YCSB, ώστε τα αποτελέσματα των μετρήσεων να αντικατοπτρίζουν την απόδοση των συστημάτων χρησιμοποιώντας τους ίδιους διαθέσιμους πόρους και παράγοντας το ίδιο τελικό αποτέλεσμα. Ο παράγοντας επανάληψης των δεδομένων ορίστηκε ίσος με τρία (3). Παρότι κατά την μελέτη των συστημάτων και τον προγραμματισμό του πειράματος, βρέθηκαν αρκετοί τρόποι παραμετροποίησης και βελτίωσης της απόδοσης τους, έγινε προσπάθεια να χρησιμοποιηθούν μόνο οι απαραίτητες για τη σωστή λειτουργία ρυθμίσεις, ώστε τα αποτελέσματα να αντικατοπτρίζουν την προκαθορισμένη συμπεριφορά του συστήματος, να υπάρχει δυνατότητα αναπαραγωγής τους και να μην εξαρτώνται από ικανότητες παραμετροποίησης που είχαμε.

5.3.1 HBase

HBase version: 0.94.17

Hadoop version: 1.2.1

Zookeeper version: 3.4.5

Java version: 1.6.0.26

Η κατανομή των απαραίτητων διεργασιών για τη λειτουργία του συστήματος έγινε ως εξής:

Όλοι οι κόμβοι (1-6) συμμετείχαν στην αποθήκευση δεδομένων και στην εξυπηρέτηση αιτημάτων από τον client οπότε και εκτελούσαν τις λειτουργίες DataNone και RegionServer. Στον κόμβο (1) ανατέθηκε επιπρόσθετα η λειτουργία του NameNode και του SecondaryNameNode του Hadoop καθώς και η λειτουργία του Hmaster και του Zookeeper της HBase.

Το στήσιμο της HBase ήταν σχετικά ξεκάθαρο αν και περιελάμβανε και το στήσιμο του HDFS καθώς και του Zookeeper. Χρειάστηκε να οριστούν μια σειρά από παράμετροι στα αρχεία ρυθμίσεων των κεντρικών κόμβων οι οποίοι στη συνέχεια αναλάμβαναν με βάση αυτές τις ρυθμίσεις να ξεκινήσουν τις απαραίτητες διεργασίες σε όλους τους κόμβους του συστήματος. Για να το επιτύχουν αυτό ήταν απαραίτητο να επιτρέπεται η επικοινωνία των κεντρικών κόμβων της HBase και του HDFS με τους όλους τους υπόλοιπους κόμβους, με χρήση ssh χωρίς εισαγωγή κωδικού.

Η HBase ήταν το σύστημα που παρουσίασε τα περισσότερα προβλήματα κατά την προσπάθεια να στηθεί και να τρέξει σωστά και τα διάφορα σφάλματα που προέκυπταν δεν ήταν πάντοτε περιγραφικά ως προς το τι δεν πήγαινε καλά. Ιδιαίτερη μέριμνα χρειάστηκε ώστε να μην επιτρέπεται η χρησιμοποίηση του πρωτοκόλλου IPv6 καθώς και να αφαιρούνται κατά τη χρήση της οι καταχωρήσεις στο hosts αρχείο που περιείχαν τη διεύθυνση loopback(127.0.0.1) καθώς η έκδοση της HBase δεν μπορούσε διαχειριστεί σωστά την πληροφορία και γινόταν διακοπή της επικοινωνίας με τους δευτερεύοντες κόμβους [26]. Επίσης ήταν το σύστημα που έκανε πιο αισθητή την ανάγκη του για περισσότερη μνήμη RAM.

5.3.2 Cassandra

Cassandra version: 2.05

Java version: 1.7.0

Στο σύστημα Cassandra κάθε κόμβος (1-6) εκτελούσε μία διεργασία CassandraDeamon.

Η κατανομή των διεργασιών και το στήσιμο της βάσης Cassandra ήταν με διαφορά η πιο απλή διαδικασία από τις υπόλοιπες. Το μόνο που χρειάστηκε προκειμένου να τρέξει σωστά ήταν η ρύθμιση κάποιων παραμέτρων (seed nodes) που καθόριζαν την επικοινωνία του κάθε κόμβου με το υπόλοιπο σύστημα αλλά και με τους χρήστες του συστήματος και τέλος το ξεκίνημα της διεργασίας σε κάθε κόμβο ο οποίος στη συνέχεια φρόντιζε αυτόματα να προστεθεί στο υπόλοιπο σύστημα.

5.3.3 MongoDB

MongoDB version: 2.4.10

Στην περίπτωση της MongoDB το στήσιμο του συστήματος απαιτούσε μια σειρά από χειροκίνητες ενέργειες.

Προκειμένου να εξασφαλιστεί ο παράγοντας επανάληψης των δεδομένων να είναι ίσος με 3 χρειάστηκε να ξεκινήσουν σε κάθε κόμβο 3 διεργασίες mongod με τις κατάλληλες παραμέτρους ώστε η κάθε μία από αυτές να αποτελεί μέρος από κάποιο διαφορετικό replica set, όμως μόνο μία από αυτές να είναι πρωτεύων κόμβος και οι άλλες δύο να είναι δευτερεύοντες κόμβοι άλλων replica set του συστήματος. Στη συνέχεια αφού είχαν ξεκινήσει οι απαραίτητες διεργασίες έγινε εκκίνηση των replica sets ώστε να μπορούν να χρησιμοποιηθούν το καθένα από αυτά σαν ένα shard. Η λειτουργία του Config Server ανατέθηκε στον κόμβο (1) και αποτελούσε μία διεργασία mongod με τις κατάλληλες παραμέτρους. Τέλος έγινε η προσθήκη των shards καθως και εκκίνηση μίας διεργασίας Mongo Router (mongos) στα VMs 1 έως 4.

Η κατανομή των μελών των 6 replica set (18 διεργασίες) που δημιουργήθηκαν συνολικά, έγινε με τέτοιο τρόπο ώστε το σύστημα να κατανέμει τα δεδομένα αντίστοιχα με έναν δακτύλιο της Cassandra (Σχήμα 4.3).Αναλυτικά οι διεργασίες φαίνονται στον πίνακα 5.3 όπου με RS συμβολίζονται οι διεργασίες mongod που αποτελούσαν τα μέρη των Replica Sets.

Κόμβος (1)	**Κόμβος (2)**	**Κόμβος (3)**	**Κόμβος (4)**	**Κόμβος (5)**	**Κόμβος (6)**
RS1-Primary	RS2-Primary	RS3-Primary	RS4-Primary	RS5-Primary	RS6-Primary
RS6-Sec.-1	RS1-Sec.-1	RS2-Sec.-1	RS3-Sec.-1	RS4-Sec.-1	RS5-Sec.-1
RS5-Sec.-2	RS6-Sec.-2	RS1-Sec.-2	RS2-Sec.-2	RS3-Sec.-2	RS4-Sec.-2
Config Server					
Mongo Router	Mongo Router	Mongo Router	Mongo Router		

Πίνακας 5.3: Κατανομή Διεργασιών Συστήματος MongoDB

Τέλος η επιλογή του συστήματος ext4 για τον χώρο αποθήκευσης των δεδομένων έγινε με κριτήριο τις ανάγκες της MongoDB για την οποία το παλιότερο ext3 σύστημα αρχείων θεωρείται ακατάλληλο από άποψη επίδοσης. Ο τρόπος που η βάση αυτή επιλέγει να αποθηκεύει τα δεδομένα της με τις τεχνικές preallocation και padding των αρχείων καθώς και η χρήση του Journal δεσμεύουν αρκετά παραπάνω διαθέσιμο χώρο στον δίσκο συγκριτικά με τα δεδομένα που αποθηκεύονται σε αυτή. Αυτή η διαφορά γίνεται αισθητή κυρίως κατά το ξεκίνημα της βάσης και και σε περιπτώσεις όπου η χρησιμοποιούμενη βάση είναι μικρή, όπως στη δικιά μας περίπτωση όπου ο χώρος που είχαμε διαθέσιμος σε κάθε κόμβο ήταν μόλις 10GB. Το ποσοστό του δεσμευμένου χώρου που δεν περιέχει δεδομένα γίνεται όλο και μικρότερο όσο η βάση μεγαλώνει.

Τα προκαθορισμένα μεγέθη αρχείων που χρησιμοποιούνται είναι τα εξής:

- 16MB για το αρχείο .ns που διατηρεί το namespace
- 64MB preallocated χώρου για το πρώτο αρχείο για την αποθήκευση δεδομένων
- 128MB για το δεύτερο αρχείο
- για κάθε επόμενο αρχείο το μέγεθος του χώρου που δεσμεύεται είναι το διπλάσιο του προηγούμενου, δηλαδή 256MB για το τρίτο αρχείο, 512MB για το τέταρτο κ.ο.κ. με το μέγεθος αυτό να φτάνει μέχρι τα 2GB.
- 1 GB για κάθε Journal αρχείο (δημιουργούνται 3 με το ξεκίνημα του συστήματος, ένα για χρήση και 2 γίνονται preallocated)
- 5% του διαθέσιμου αποθηκευτικού χώρου για χρήση από το oplog του συστήματος

Τα παραπάνω έκαναν αδύνατη τη χρησιμοποίηση της MongoDB με το σύστημα που είχαμε στη διάθεσή μας οπότε κάποιες επιλογές ώστε να περιοριστεί το αρχικό μέγεθος της βάσης. Αυτές ήταν οι “--smallfiles” και “--oplogSize 128” που ορίστηκαν στο ξεκίνημα των διεργασιών mongod. Με την επιλογή “--smallfiles” τα παραπάνω μεγέθη διαμορφώθηκαν ως εξής:

- 16MB για το αρχείο .ns που διατηρεί το namespace
- 16MB για το πρώτο αρχείο που γίνεται preallocated για να αποθηκεύσει δεδομένα
- 128MB για το δεύτερο και 128MB για το τρίτο. 256MB για το τέταρτο και 256MB για το πέμπτο.

- 512MB για όλα τα επόμενα αρχεία δεδομένων
- 128MB για κάθε ένα από τα 3 Journal αρχεία
- Τέλος η επιλογή "--oplogSize 128" έκανε το μέγεθος κάθε oplog να είναι 128 MB.

Επισημαίνεται ότι αυτά τα μεγέθη δεσμεύονταν για κάθε διεργασία mongod επομένως δεσμεύονταν 3 φορές σε κάθε κόμβο για να επιτύχουμε τον παράγοντα επανάληψης 3 όπως φαίνεται στον πίνακα 5.3. Το αποτέλεσμα όλων των παραπάνω ήταν με το ξεκίνημα του συστήματός μας και προτού ακόμη καταχωρηθούν δεδομένα να δεσμεύονται 673MB χώρου για κάθε διεργασία δηλαδή κάτι παραπάνω από 2GB κάθε κόμβο. Μετά την εισαγωγή των δεδομένων και λόγω της ύπαρξης των preallocated αρχείων δεσμευμένος χώρος άγγιζε τα 7GB σε κάθε κόμβο και ήταν ακόμη περισσότερα στον κόμβο (1) όπου λειτουργούσε και ο config server.

Για τον λόγο αυτό τα δεδομένα που εισήχθησαν στα συστήματα έφτασαν μέχρι τις 1.800.000 καταγραφές. Κάθε καταγραφή αποτελείτο από 10 πεδία και κάθε πεδίο από 100 bytes. Έτσι κάθε καταγραφή περιείχε 1KB δεδομένων και επιπροσθέτως κάποιο κλειδί του οποίου η μορφή ήταν συνδεδεμένη με το κάθε σύστημα. Επομένως τα πραγματικά δεδομένα στα συστήματα της επιλογής μας έφτασαν αναγκαστικά μέχρι μόλις 5.4GB λαμβάνοντας υπόψιν και τον παράγοντα επανάληψης των δεδομένων.

5.4 Αυτοματοποίηση της διαδικασίας και Scripts

Για την δημιουργία του συνόλου των πόρων στο σύστημα BonFIRE παρέχονταν διάφορες επιλογές για αυτοματοποίηση της διαδικασίας. Αυτή που χρησιμοποιήθηκε ήταν η περιγραφή των επιθυμητών πόρων μέσω ενός JSON αρχείου που ονομάζεται experiment descriptor, το οποίο στη συνέχεια μεταφορτώνεται μέσω της δικτυακής διεπαφής στο σύστημα και αυτό με τη σειρά του δημιουργεί όλους τους πόρους που έχουν οριστεί στο αρχείο. Στη συνέχεια η διαχείριση των κόμβων γινόταν μέσω ssh.

Η διαδικασία της εκτέλεσης του πειράματος αυτοματοποιήθηκε με τη χρήση διαφόρων Bash scripts τα οποία συνδυάζονταν κατάλληλα ώστε το σύνολο των μετρήσεων πραγματοποιείται τελικά με την εκτέλεση μιας εντολής(./bench-all-N.sh). Το κεντρικό αυτό σενάριο εντολών καθώς και τα περισσότερα που χρησιμοποιήθηκαν βρίσκονταν στον κόμβο (1) master. Για τη διαχείριση των υπολοίπων κόμβων και την εκκίνηση των απαραίτητων διεργασιών και σεναρίων εντολών σε αυτούς, χρησιμοποιήθηκαν πολλαπλές εντολές μέσω ssh. Το σενάριο εντολών "bench-all-N.sh" παίρνει σαν όρισμα τον αριθμό των κόμβων που θέλουμε να χρησιμοποιήσουμε

και υλοποιήθηκε με αυτόν τον τρόπο ώστε να χρησιμοποιείται και για δοκιμαστικούς σκοπούς με χρήση λιγότερων από 6 κόμβους. Στο script αυτό εκτελούνται οι παρακάτω ενέργειες:

1. Ορίζονται αρχικά οι παράμετροι εκτέλεσης του πειράματος που ήταν κοινοί για τα συστήματα, όπως ο αριθμός των εγγραφών στη βάση, το εύρος των επιθυμητών throughput που παράγει ο YCSB Client, ο αριθμός των επαναλήψεων των μετρήσεων για κάθε επιθυμητό troughput, ο αριθμός των threads που θα εκτελεί ο Client κ.α.
2. Καλείται ένα script "fixhosts.sh" το οποίο αναλάμβανε την ενημέρωση των hosts αρχείων όλων των κόμβων με τα κατάλληλα ονόματα και τις χρησιμοποιούμενες διευθύνσεις IP ώστε να γίνεται απροβλημάτιστα η επικοινωνία μεταξύ των κόμβων των συστημάτων.
3. Καλείται ένα script "starthbase-N.sh" με παράμετρο τον αριθμό των κόμβων ο οποίος δίνεται στο αρχικό script. Αυτό με τη σειρά του πραγματοποιεί μια σειρά από ενέργειες προκειμένου να γίνει εκκίνηση του συστήματος HBase.
    - Αρχικά τροποποιεί τα αρχεία ρυθμίσεων της HBase και του HDFS ορίζοντας τους κόμβους που θα χρησιμοποιηθούν από το σύστημα (RegionServers και DataNodes).
    - Στη συνέχεια κάνει comment out την καταχώρηση loopback στα αρχεία hosts όλων των κόμβων καλώντας scripts που βρίσκονται σε κάθε κόμβο (1)-(6) και έχουν τα απαραίτητα δικαιώματα για την ολοκλήρωση αυτής της ενέργειας
    - Πραγματοποιεί format στον NameNode του συστήματος HDFS προετοιμάζοντας το κατανεμημένο σύστημα αρχείων
    - Εκκινεί τη λειτουργία του HDFS
    - Εκκινεί τη λειτουργία του Zookeeper
    - Εκκινεί τη λειτουργία της HBase
    - Προσθέτει στην βάση HBase έναν πίνακα "usertable" καθώς και μια οικογένεια στηλών "family" που ήταν απαραίτητα για τον YCSB Client.
4. Γίνεται μέσω ssh η εκκίνηση ενός script "bench-hbase-N.sh" το οποίο εκτελείται στον κόμβο Client δίνοντας μια σειρά από παραμέτρους που ορίστηκαν στο στάδιο 1. Αυτό το σενάριο εντολών πραγματοποιεί τις παρακάτω ενέργειες:
    - Γεμίζει τη βάση με δεδομένα

- Πραγματοποιεί μετρήσεις για τα wokloads A, B, C, F, D σύμφωνα με τις παραμέτρους που του έχουν δοθεί και για τον κατάλληλο αριθμό επαναλήψεων και μαζεύει τα αποτελέσματα ανά κατηγορίες δίνοντας κατάλληλα ονόματα στα παραγόμενα αρχεία.

5. Καλείται ένα script “stophbase-N.sh” το οποίο πραγματοποιεί τις εξής λειτουργίες:
   - Διακόπτει τη λειτουργία της HBase
   - Διακόπτει τη λειτουργία του Zookeeper
   - Διακόπτει τη λειτουργία του HDFS
   - Διαγράφει ότι δεδομένα είχαν δημιουργηθεί από τη λειτουργία του συστήματος
   - Επαναφέρει τα hosts αρχεία των κόμβων στην αρχική τους κατάσταση
6. Τα βήματα 3 και 5 χρησιμοποιήθηκαν ξανά για την εκτέλεση των μετρήσεων για το workload E για το οποίο έπρεπε εισαχθούν εκ νέου τα δεδομένα στη βάση και να εκτελεστεί με κάποιες ιδιαίτερες παραμέτρους (αρκετά χαμηλότερο throughput).
7. Στη συνέχεια εκτελούνται οι αντίστοιχες ενέργειες για την βάση Cassandra. Το script “startcassandra-N.sh” εκκινούσε απλώς μέσω ssh μια διεργασία cassandra σε κάθε διαθέσιμο κόμβο και τέλος δημιουργούσε ένα Keyspace “ycsb” με τον κατάλληλο replication factor (3) και μέσα σε αυτό έναν πίνακα “usertable” για χρήση από τον YCSB Client.
8. Εκτελούνται τα workloads A, B, C, F, D με χρήση του αντίστοιχου script “benchcassandra-N.sh”.
9. Διακόπτεται η λειτουργία της Cassandra και καθαρίζονται όλοι οι κόμβοι από τα δεδομένα που παρήχθησαν
10. Επαναλαμβάνονται τα βήματα 7 και 9 με την εκτέλεση στο ενδιάμεσο μετρήσεων για το workload E (benchcassandraE.sh)
11. Στη συνέχεια γίνεται εκκίνηση του συστήματος MongoDB μέσω του script “startmongo-N.sh”. Το σενάριο αυτό καλεί μια σειρά από άλλα scripts αφού υπό κανονικές συνθήκες το σύστημα MongoDB χρειαζόταν πολλές χειροκίνητες ενέργειες προκειμένου να στηθεί και δεν παρείχε κάποιον συνολικό, αυτοματοποιημένο τρόπο ορισμού των κόμβων και των λειτουργιών καθενός από αυτού. Παρείχε τη δυνατότητα να ξεκινάει αυτόματα μια διεργασία mongod με τις κατάλληλες παραμέτρους κατά την

εκκίνηση του λειτουργικού κάθε κόμβου, όμως αυτό δεν ήταν αρκετό για τις ανάγκες μας αφού κάθε σε κάθε κόμβο ανατέθηκαν παραπάνω από μία λειτουργίες και επίσης ήταν απαραίτητο το σύστημα να ξεκινά και να σταματά ανάλογα με τις ανάγκες χωρίς να παρεμβάλλεται στη λειτουργία των άλλων βάσεων. Έτσι το σενάριο εντολών "startmongo-N.sh" πραγματοποιεί τις παρακάτω ενέργειες:

- Καλεί το σενάριο εντολών "mstartprocesses-N.sh" το οποίο εκκινεί όλες τις απαραίτητες διεργασίες που αργότερα προστίθενται στα replica sets. Χρησιμοποιεί για αυτό τον σκοπό την πληροφορία για το πλήθος των διαθέσιμων κόμβων και με κατάλληλη αριθμητική ορίζει τα ports που χρησιμοποιεί κάθε μία διεργασία ανάλογα με τη λειτουργία της. Οι διεργασίες που εκκινεί φαίνονται στον πίνακα 5.3 με ονόματα "RS-*"..
- Καλεί το σενάριο εντολών "minitiatemongors-N.sh" το οποίο χρησιμοποιώντας την αντίστοιχη αριθμητική εκκινεί την λειτουργία των Replica Sets μέσω του πρωτεύοντος μέλους σε κάθε κόμβο και στη συνέχεια προσθέτει τα δευτερεύοντα μέλη που βρίσκονται στους άλλους κόμβους.
- Αφού δημιουργούνται τα Replica Sets στη συνέχεια καλεί το σενάριο εντολών "maddshards-N.sh" το οποίο αρχικά εκκινεί τον Config Server στον κόμβο (1) καθώς και μια σειρά από Mongo Routers στους κόμβους (1) εως (4). Στη συνέχεια συνδέεται μέσω ενός Mongo Router στον Config Server και προσθέτει τα Replica Sets σαν Shards στη βάση.
- Τέλος καλεί το σενάριο εντολών "menablesharding.sh" το οποίο δημιουργεί για τις ανάγκες του YCSB Client μία βάση "ycsb", μία συλλογή δεδομένων "usertable" και ενεργοποιεί την επιλογή "shardcollection" ώστε η βάση αυτή να κατανέμει τα δεδομένα της ισομερώς σε όλα τα διαθέσιμα shards.

12. Στη συνέχεια, όπως και στα παραπάνω συστήματα, εκκινεί στον client το σενάριο εντολών "bench-mongo-N.sh" που πραγματοποιεί μετρήσεις για τα workloads A, B, C, F, D και συλλέγει τα αποτελέσματα.
13. Η βάση καθαρίζεται από ότι δεδομένα έχει δημιουργήσει και στη συνέχεια εκτελούνται οι μετρήσεις για το workload E.
14. Τέλος τα αποτελέσματα συλλέγονται, συμπιέζονται και αποστέλλονται στο e-mail μας για επεξεργασία.

5.5 Διαδικασία Μετρήσεων

Ο YCSB Client, όπως αναφέραμε, παράγει ερωτήματα στη βάση με ρυθμό που καθορίζεται από τις παραμέτρους της κάθε εκτέλεσης. Στην πραγματικότητα οι ρυθμοί απόδοσης των συστημάτων έφταναν μέχρι κάποιο σημείο, το οποίο καθοριζόταν από τις αντοχές του συστήματος. Έτσι, παρότι οι επιθυμητοί ρυθμοί πού δίνονταν στις παραμέτρους ξεκινούσαν από τα 1000 Ops/sec και αυξάνονταν κάθε φορά κατά 1000 μονάδες, τα αποτελέσματα στα διαγράμματα του κεφαλαίου 6 δεν ευθυγραμμίζονται με αυτές τις τιμές απόλυτα. Κάθε μέτρηση (για ένα συγκεκριμένο σύστημα, ένα συγκεκριμένο φορτίο και έναν συγκεκριμένο επιθυμητό ρυθμό απόδοσης) πραγματοποιείτο τέσσερις φορές προκειμένου να αποκλειστούν κατά την επεξεργασία των αποτελεσμάτων οι μετρήσεις που παρουσίαζαν μεγάλη απόκλιση συγκριτικά με τις άλλες και το αποτέλεσμα να υπολογίζεται σαν μια μέση τιμή των υπολοίπων μετρήσεων. Οι αποκλίσεις αυτές οφείλονταν σε αστάθμητους παράγοντες που εισήγαγε η υπολογιστική υπηρεσία νέφους και με αυτόν τον τρόπο επιχειρήθηκε η μικρότερη δυνατή επιρροή τους στα αποτελέσματά μας.

Η κάθε μέτρηση περιελάμβανε και έλεγχο του χρόνου που απαιτούσε προκειμένου να ολοκληρωθεί. Ο διαθέσιμος χρόνος που αφιερώναμε σε κάθε μέτρηση υπολογιζόταν σαν συνάρτηση των ενεργειών που πραγματοποιούνταν στην βάση και του επιθυμητού ρυθμού απόδοσης. Αν ο ρυθμός απόδοσης που πετύχαινε η βάση ήταν μικρότερος από τα 2/3 του επιθυμητού ρυθμού απόδοσης τότε διακόπταμε το συγκεκριμένο πείραμα αφού δεν θα παρείχε παραπάνω πληροφορία στα διαγράμματά μας και παραπάνω μετρήσεις απλώς θα καθυστερούσαν το συνολικό πείραμα.

Παρακάτω παραθέτουμε ένα απόσπασμα κώδικα BASH που εκτελείτο στον κόμβο client στο script “benchcassandra-N.sh” που αναφέρθηκε στο υποκεφάλαιο 5.4 και ήταν υπεύθυνο για τη σειρά μετρήσεων του φορτίου A στη βάση Cassandra.

```
let "opcount= 50 * $tstart"
let "giventime = 3 * $opcount / $tstart"
for i in `seq  $tstart $tstep $tend`;
do
  printf -v k "%05d" $i
  let "opcount= 50 * $i"
  for j in `seq 1 $times`;
  do
  sleep $coolingtime
  thetime=`date +%d-%m::%H:%M:%S`
  timeout $giventime ./CASSANDRA-YCSB/bin/ycsb run cassandra-10 -P
./CASSANDRA-YCSB/workloads/workloada \
  -p recordcount=$recordcount -p operationcount=$opcount -p
```

```
hosts=$chosts -p target=$i -p threadcount=$threads -s >
./RESULTS/a/CRun-t:$k-run:$j-$thetime.txt
  if [ $(wc -l < ./RESULTS/a/CRun-t:$k-run:$j-$thetime.txt) -le 10 ]
  then
  echo "timed out after $giventime" >> ./RESULTS/a/CRun-t:$k-run:
$j-$thetime.txt
  break 2
  fi
  let "giventime = 3 * $opcount  / 2 / $i"
  done
done
```

$opcount : Ο αριθμός των ενεργειών που θα πραγματοποιήσει η επόμενη εκτέλεση

$tstart : Ο αρχικός επιθυμητός ρυθμός απόδοσης (1000 Ops/sec)

$giventime: Ο διαθέσιμος χρόνος που θα δοθεί στην επόμενη μέτρηση προκειμένου να ολοκληρωθεί. Κατά την πρώτη εκτέλεση ο διαθέσιμος χρόνος ήταν τέτοιος ώστε ο πραγματικός ρυθμός απόδοσης της βάσης να μπορεί να είναι μέχρι και 1/3 του επιθυμητού ρυθμού απόδοσης. Στη συνέχεια ο ρυθμός απόδοσης της βάσης μπορούσε να είναι μέχρι και 2/3 του επιθυμητού ρυθμού απόδοσης. Αν ήταν μικρότερος διακοπτόταν η συγκεκριμένη σειρά μετρήσεων για να μην εισάγει καθυστερήσεις στο συνολικό πείραμα χωρίς να προσφέρει παραπάνω δεδομένα στα αποτελέσματά μας.

Στη συνέχεια υπάρχουν δύο βασικά for loops. Το εξωτερικό είναι υπεύθυνο ώστε να αυξάνει τον επιθυμητό ρυθμό απόδοσης από την τιμή $tstart (1000) μέχρι την τιμή $tend(14000, τιμή αρκετά μεγάλη, ώστε το script να διακόπτεται όταν το σύστημα δεν πιάνει παραπάνω ρυθμό απόδοσης) με βήματα $tstep (1000). Η τιμή $i δίνεται στη συνέχεια με την παράμετρο target=$i στην εντολή που εκτελεί την μέτρηση παράγοντας ενέργειες με αυτόν τον ρυθμό. Το εσωτερικό loop αναλάμβανε να πραγματοποιήσει επανάληψη της μέτρησης $times φορές και χρησιμοποιήσαμε την τιμή 4. Η διάρκεια κάθε μεμονωμένης εκτέλεσης ήταν ιδανικά 50 δευτερόλεπτα, γεγονός που οδηγούσε σε 200.000 μεμονωμένες λειτουργίες στην περίπτωση του μικρότερου ρυθμού απόδοσης.

$coolingtime : Χρόνος που δινόταν ανάμεσα στις επαναλήψεις των μετρήσεων

$thetime : Η ώρα που ξεκίνησε η εκτέλεση της μέτρησης για να φαίνεται στη συνέχεια στο όνομά του αρχείου που παράγει ο YCSB Client με τα αποτελέσματα.

$recordcount : Ο αριθμός των καταχωρήσεων στη βάση (1.000.000 και 1.800.000)

$chosts : Οι κόμβοι οι οποίοι συμμετείχαν στον δακτύλιο της Cassandra

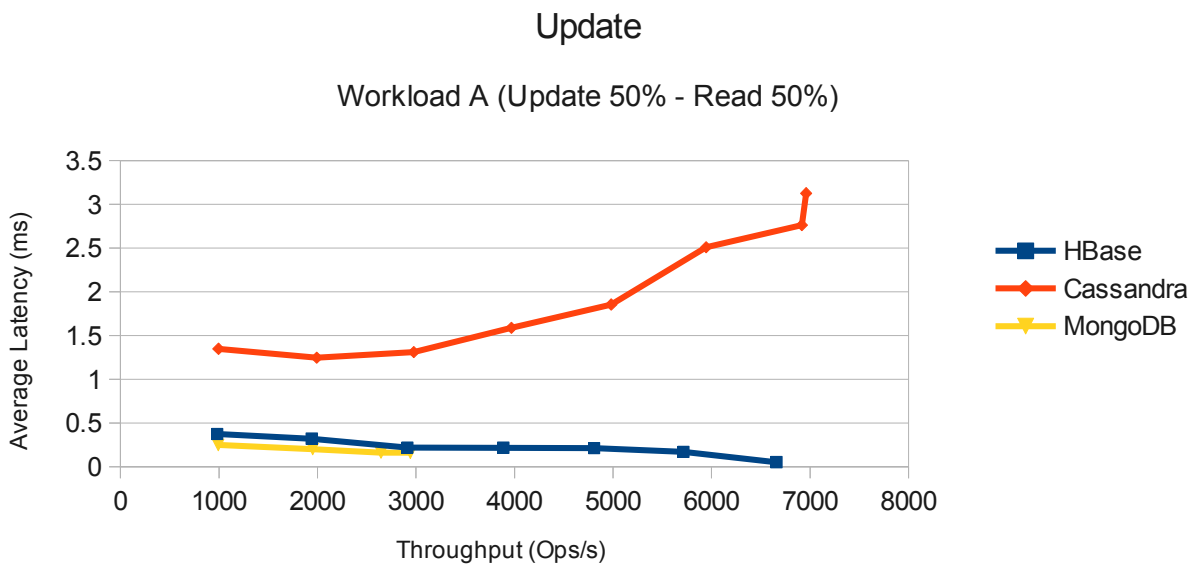
$threadcount : Ο αριθμός των threads που χρησιμοποιούσε ο Client.

Κεφάλαιο 6
Πειραματικές Μετρήσεις

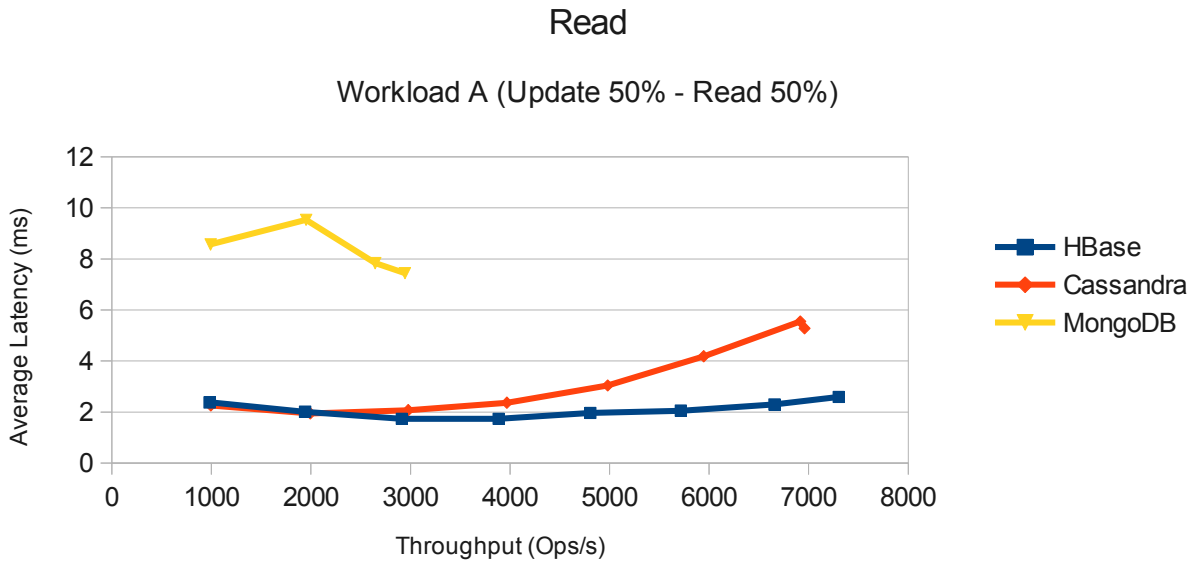
6.1 Φορτίο Α

Στα σχήματα 6.1 και 6.2 φαίνονται τα αποτελέσματα των μετρήσεων για το φορτίο Α το οποίο αποτελείται από ένα μείγμα 50% ενημερώσεων και 50% αναγνώσεων. Τα συστήματα HBase και Cassandra πέτυχαν αρκετά μεγαλύτερο ρυθμό απόδοσης συγκριτικά με το σύστημα MongoDB. Το στοιχείο του φορτίου που προκαλεί καθυστερήσεις στο σύστημα MongoDB και είναι ο λόγος της χαμηλής απόδοσής του είναι ο αυξημένος αριθμός από ενημερώσεις. Αυτό το στοιχείο δεν αποτυπώνεται στον χρόνο απόκρισης των ενημερώσεων αφού το σύστημα επιστρέφει επιτυχία της ενημέρωσης, όπως περιγράφηκε στο κεφάλαιο 4.3.3, όταν αυτή καταγραφεί στη μνήμη RAM παρέχοντας σχετικά χαλαρές εγγυήσεις μονιμότητας. Η μονιμοποίηση των δεδομένων πραγματοποιείται ανά 100ms με τη χρήση του μηχανισμού Journal και ο συγχρονισμός των αρχείων που βρίσκονται στη RAM με τα αρχεία του δίσκου πραγματοποιείται ανά 60 δευτερόλεπτα. Σε συνδυασμό όμως με την περιορισμένη μνήμη των διαθέσιμων συστημάτων φαίνεται ότι το λειτουργικό πραγματοποιούσε συγχρονισμό των αρχείων σε διαστήματα μικρότερα των 60 δευτερολέπτων προκειμένου να εξασφαλίσει χώρο στη μνήμη και να φέρει τα αρχεία που ήταν απαραίτητα για τις αναγνώσεις. Αυτό αποτυπώνεται στους σχετικά μεγάλους χρόνους απόκρισης των αναγνώσεων συγκριτικά με άλλα φορτία που εκτελέστηκαν στην MongoDB και οι αναγνώσεις παρουσίαζαν αρκετά καλύτερη συμπεριφορά (φορτία B, C, D) με αρκετά χαμηλότερους χρόνους απόκρισης.

Αντίθετα τα συστήματα HBase και Cassandra δεν επηρεάζονται από τις ενημερώσεις στα δεδομένα αφού όπως περιγράφηκε στα κεφάλαια 4.1.3 και 4.2.3, τα συστήματα αυτά δεν ανακαλούν τα δεδομένα που επιθυμούν να ενημερώσουν από τον δίσκο αλλά αποθηκεύουν τις αλλαγές σε καινούρια αρχεία (HFiles και SSTables αντίστοιχα) χρησιμοποιώντας τις δυνατότητες σειριακής εγγραφής των δίσκων. Η ενημέρωση των δεδομένων γίνεται στο παρασκήνιο συγχωνεύοντας τα αρχεία αυτά, χωρίς να επηρεάζεται σημαντικά η απόδοση των συστημάτων από τους χρόνους αναζήτησης της θέσης των αρχείων στον δίσκο. Έτσι πετυχαίνουν αντίστοιχα καλές επιδόσεις, με τη βάση Cassandra να έχει ελαφρώς μεγαλύτερους χρόνους απόκρισης πιθανώς λόγω του κόμβου-συντονιστή που μεσολαβεί κατά την αλληλεπίδραση ενός χρήστη με τον κόμβο που περιέχει τα δεδομένα.



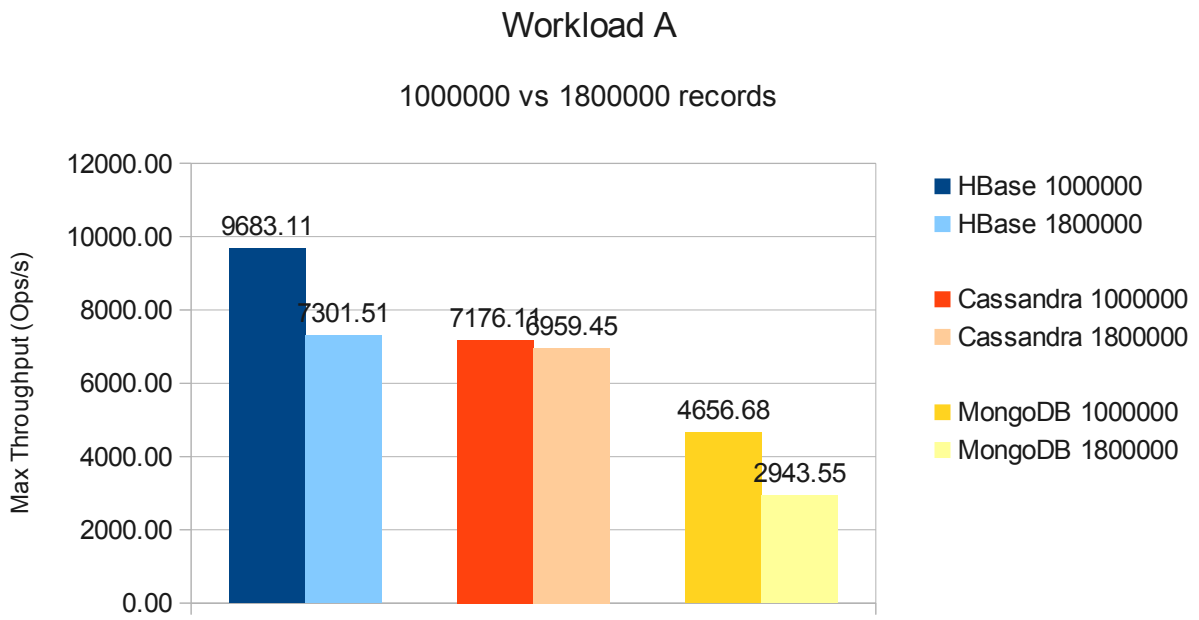
Σχήμα 6.1: Φορτίο Α: Ενημερώσεις 50%



Σχήμα 6.2: Φορτίο Α: Αναγνώσεις 50%

Σε ότι αφορά την επίδραση του όγκου των δεδομένων στη βάση (σχήμα 6.3) η MongoDB φαίνεται να επηρεάζεται σημαντικά για τους λόγους που αναφέρθηκαν παραπάνω και η απόδοση της μειώνεται κατά ~37% όταν οι εγγραφές στη βάση έγιναν 1.800.000. Έτσι όταν τα δεδομένα είναι λιγότερα οι μηχανισμοί caching έχουν μεγαλύτερο hit ratio και οι βάσεις αποδίδουν

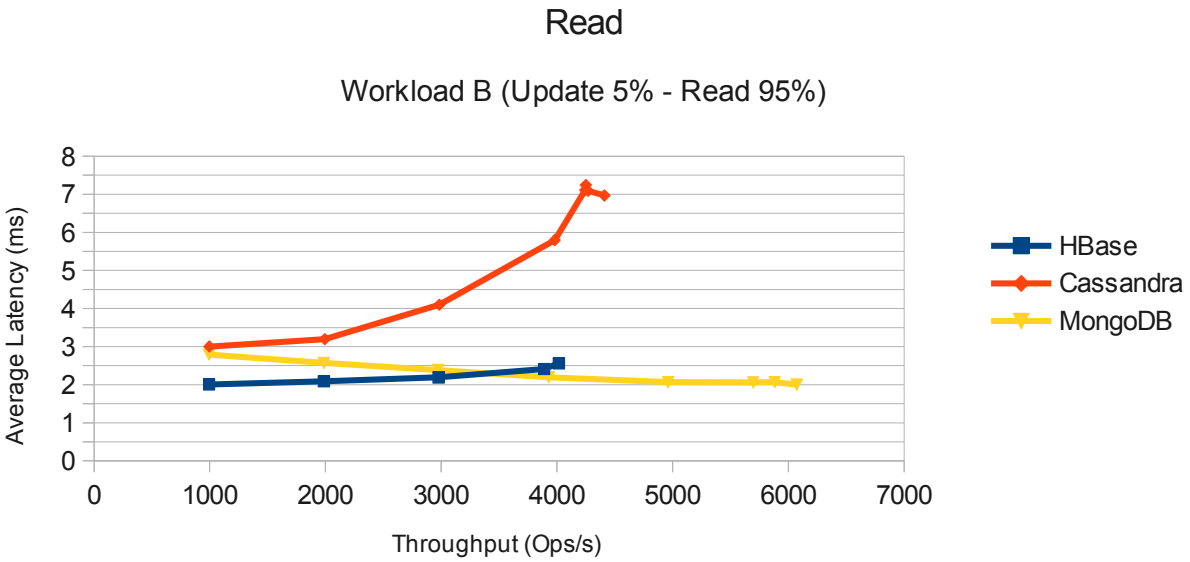
σημαντικά καλύτερα. Σημαντική ήταν και η πτώση κατά 25% της απόδοσης για την βάση HBase η οποία έδειχνε σε πολλές περιπτώσεις την ανάγκη της για περισσότερη RAM. Η Cassandra δεν επηρεάστηκε σημαντικά αφού η πτώση της επίδοσης ήταν της τάξης του 3%.



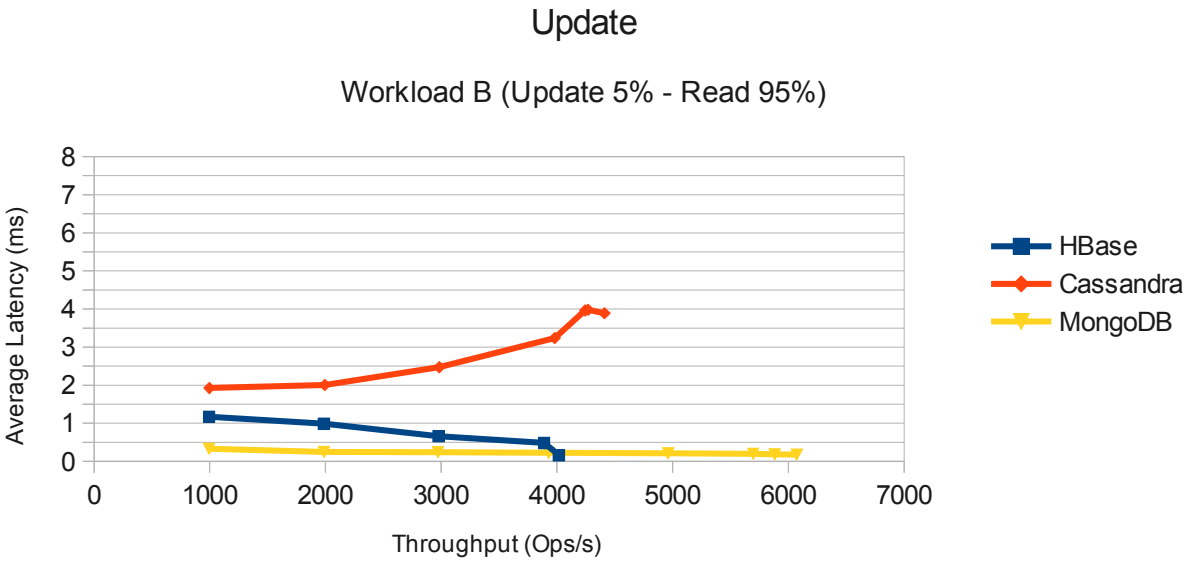
Σχήμα 6.3: Φορτίο Α: Σύγκριση μέγιστων επιδόσεων με 1.000.000 και 1.800.000 καταχωρήσεις

6.2 Φορτίο Β

Στο φορτίο Β όπου οι ενημερώσεις αποτελούσαν το 5% και οι αναγνώσεις το 95% των ενεργειών η MongoDB πήγε αρκετά καλύτερα συγκριτικά με τα άλλα δύο συστήματα που είχαν παρόμοια συμπεριφορά. Τα memory-mapped αρχεία σε συνδυασμό με τη Zipfian κατανομή που επιλέγει κάποια δεδομένα πάνω στα οποία εκτελούνται οι ενέργειες αρκετά περισσότερες φορές από κάποια άλλα, έκαναν τους μηχανισμούς caching του λειτουργικού να αποδίδουν ιδιαίτερα καλά. Η απουσία ενημερώσεων συγκριτικά με το φορτίο Α φαίνεται ότι απάλλαξε σε μεγάλο βαθμό τη βάση από καθυστερήσεις. Η HBase και η Cassandra είχαν μειωμένη απόδοση συγκριτικά με το φορτίο Α. Αυτό οφείλεται αφενός στην περιορισμένη μνήμη RAM που αφιέρωναν για τους μηχανισμούς caching, αφού η ποσότητα αυτή προκύπτει σαν ποσοστό του μεγέθους της Heap (Προκαθορισμένη τιμή 1GB) που χρησιμοποιούν τα συστήματα σε κάθε κόμβο. Αντίθετα η MongoDB, αφήνοντας το λειτουργικό να διαχειριστεί την προσωρινή μνήμη δέσμευε δυναμικά όλη τη διαθέσιμη μνήμη που δε χρησιμοποιούσαν οι υπόλοιπες λειτουργίες του κόμβου. Η διαφορά που οφείλεται στην περιορισμένη μνήμη RAM αποτυπώνεται και στο σχήμα 6.6 από τη σύγκριση των μέγιστων επιδόσεων των συστημάτων όταν οι εγγραφές στη βάση ήταν 1.000.000 και 1.800.000 αντίστοιχα. Ένας δεύτερος λόγος που παρατηρείται μειωμένη απόδοση για το σύστημα HBase είναι πως η ανάκληση των δεδομένων από τον δίσκο στην προσωρινή μνήμη γίνεται εντός του Heap της διεργασίας Java. Έτσι φορτία που αποτελούνται από πολλές αναγνώσεις προκαλούν συχνά κατακερματισμό της Heap και διαχειρίζονται από τον Garbage Collector (συλλέκτη απορριμάτων) τον οποίο επιβαρύνουν σημαντικά. Η HBase σε μεταγενέστερες εκδόσεις της 0.96.3 δίνει τη δυνατότητα η προσωρινή μνήμη “blockcache”, η οποία είναι το μέρος της μνήμης που αναλαμβάνει την διαδικασία caching δεδομένων, να λειτουργεί ανεξάρτητα από το Heap της διεργασίας του RegionServer εκμεταλλευόμενη έτσι περισσότερη διαθέσιμη μνήμη χωρίς αυτό να επιβαρύνει την λειτουργία του GC [27]. Η Cassandra στην προκαθορισμένη λειτουργία της δεν αποθηκεύει δεδομένα στην cache, αλλά κλειδιά δεδομένων κάνοντας έτσι πιο γρήγορη την αναζήτηση τους όταν αυτά ζητηθούν όπως περιγράφηκε στην ενότητα 4.2.3.

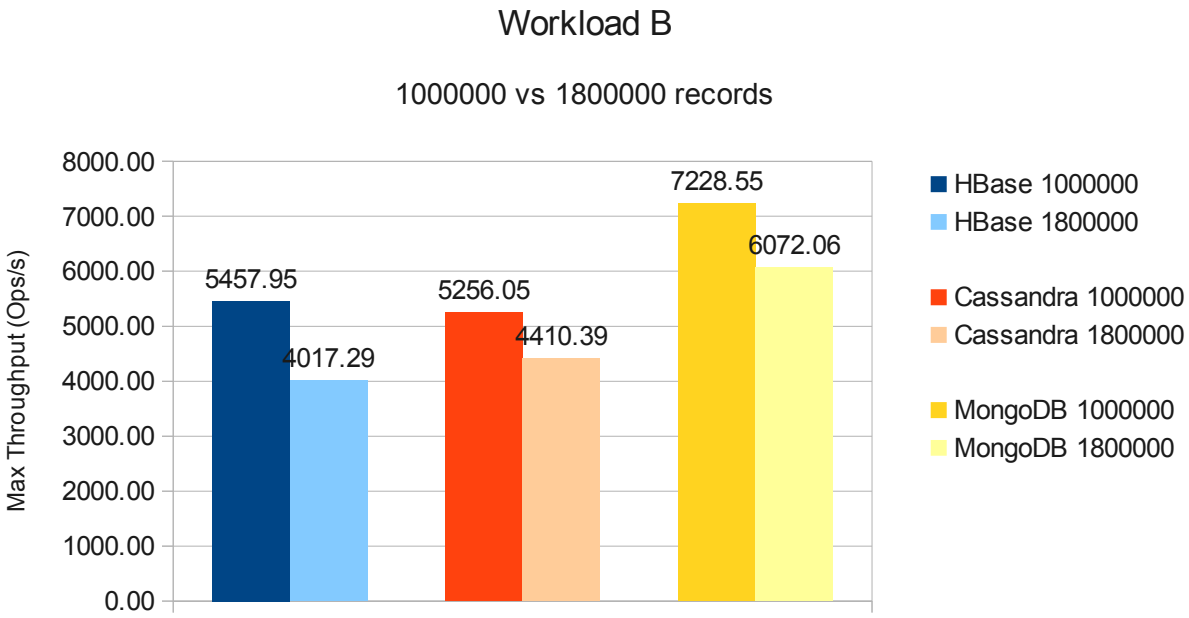


Σχήμα 6.4: Φορτίο Β: Αναγνώσεις 95%



Σχήμα 6.5: Φορτίο Β: Ενημερώσεις 5%

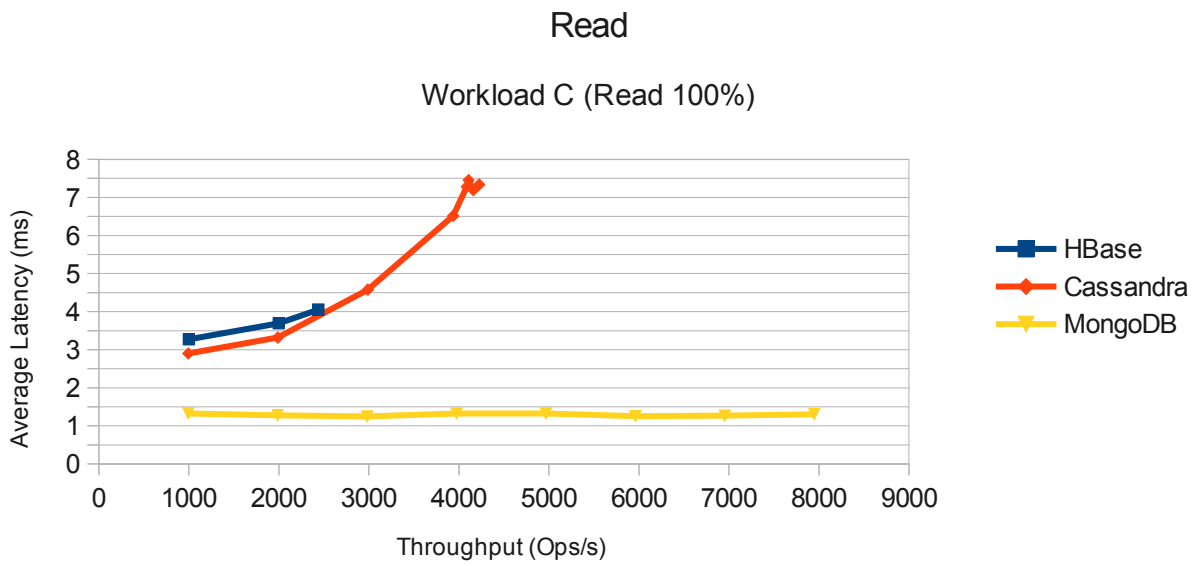
Η απόδοση της HBase μειώθηκε κατά ~26% της Cassandra κατά 16% και της MongoDB κατά 19% όταν αυξήθηκαν τα δεδομένα όπως φαίνεται στο σχήμα 6.6.



Σχήμα 6.6: Φορτίο Β: Σύγκριση μέγιστων επιδόσεων με 1.000.000 και 1.800.000 καταχωρήσεις

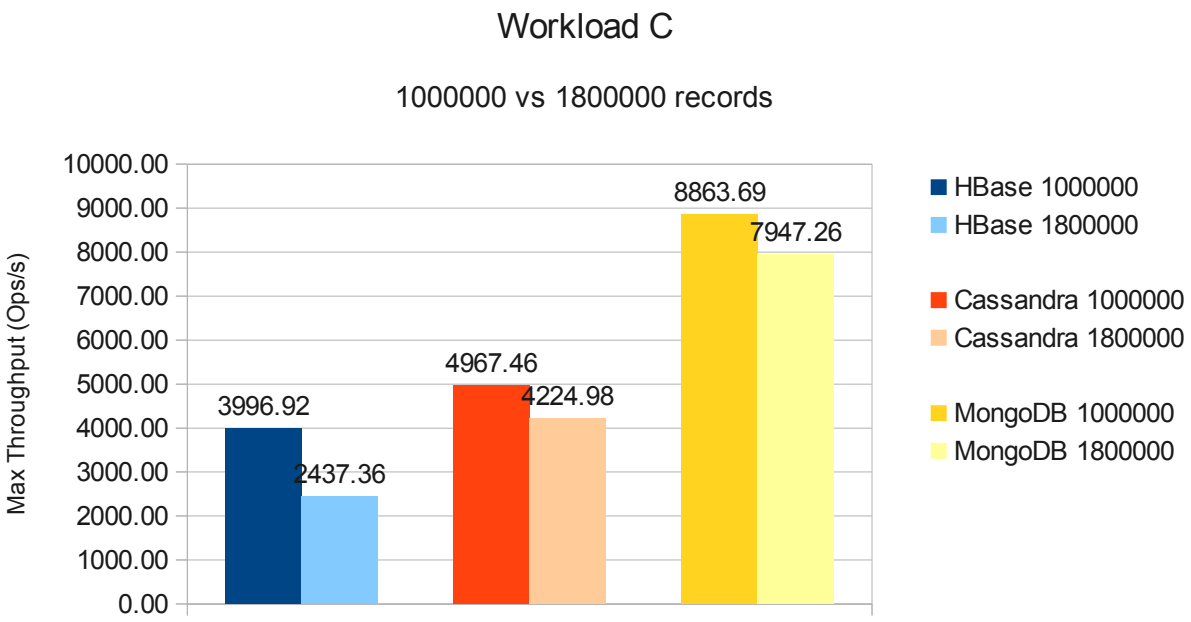
6.3 Φορτίο C

Το φορτίο C αποτελείται αποκλειστικά από αναγνώσεις. Στο φορτίο αυτό φαίνονται ξεκάθαρα διάφορα από τα θέματα που αναφέρθηκαν στα δύο προηγούμενα φορτία (Σχήμα 6.7). Η MongoDB πετυχαίνει πολύ καλή απόδοση με ιδιαίτερα χαμηλό χρόνο απόκρισης μιας και οι μηχανισμοί caching λειτουργούν απροβλημάτιστα απουσία ενημερώσεων και ανάγκης συγχρονισμού των αρχείων που βρίσκονται στη μνήμη με τον δίσκο. Αντίθετα η HBase έδειξε την αδυναμία της να πραγματοποιήσει αποτελεσματικό caching των δεδομένων κάτω από τις συνθήκες της περιορισμένης μνήμης RAM που είχαμε στη διάθεσή μας για τους λόγους που αναφέρθηκαν στο προηγούμενο φορτίο. Φυσικά σε πραγματικές συνθήκες όπου θα είχαμε στη διάθεση μας αρκετά περισσότερα GB μνήμης και θα προσαρμόζαμε τις παραμέτρους του συστήματος στις ανάγκες του φορτίου που μας ενδιέφερε τα αποτελέσματα θα ήταν αρκετά διαφορετικά. Η βάση Cassandra κρατώντας στην προσωρινή μνήμη μόνο τα κλειδιά των τιμών που χρησιμοποιούνται περισσότερο δεν επηρεάζεται τόσο έντονα από τον μεγάλο όγκο των αναγνώσεων. Η αναζήτηση όμως τελικά των δεδομένων στον δίσκο την κάνει να αποδίδει αρκετά λιγότερο συγκριτικά με την MongoDB. Για την Cassandra παρέχεται η δυνατότητα να αποθηκεύονται στην προσωρινή μνήμη γραμμές δεδομένων που ζητούνται προς ανάγνωση, δυνατότητα που θα βοηθούσε σημαντικά στο συγκεκριμένο φορτίο λόγω της κατανομής Zipfian των ενεργειών, όμως κάτι τέτοιο δε χρησιμοποιήθηκε αφού δεν ήταν στις προκαθορισμένες ρυθμίσεις.



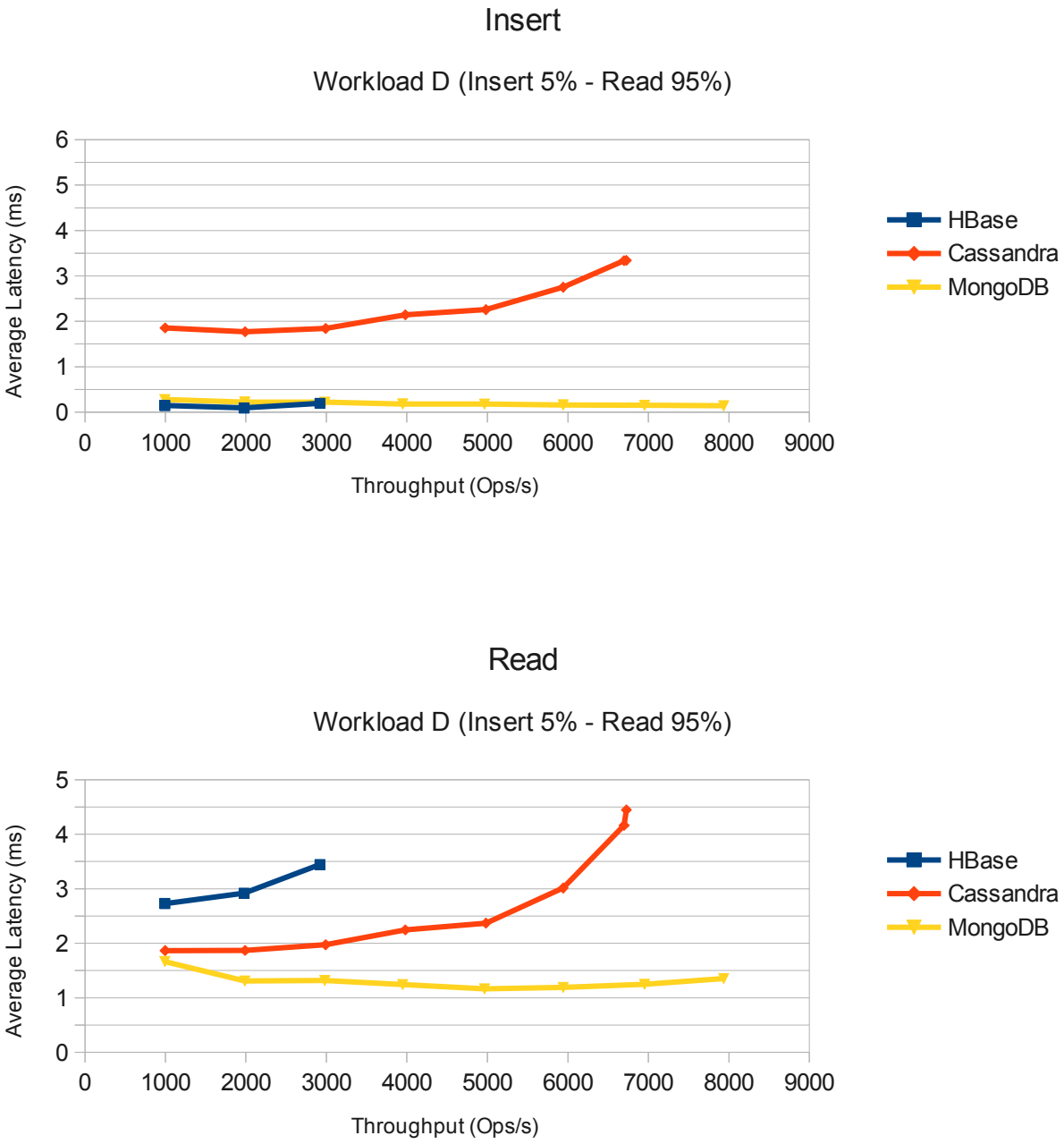
Σχήμα 6.7: Φορτίο C: Αναγνώσεις 100%

Όπως ήταν αναμενόμενο λόγω των παραπάνω, ο αυξημένος όγκος δεδομένων για το φορτίο C, είχε μεγαλύτερη επίπτωση στο σύστημα HBase που παρουσίασε μείωση της απόδοσής του κατά 39%. Οι βάσεις Cassandra και MongoDB επηρεάστηκαν αρκετά λιγότερο με πτώση της απόδοσης κατά 17.5% και 11.5% αντίστοιχα.



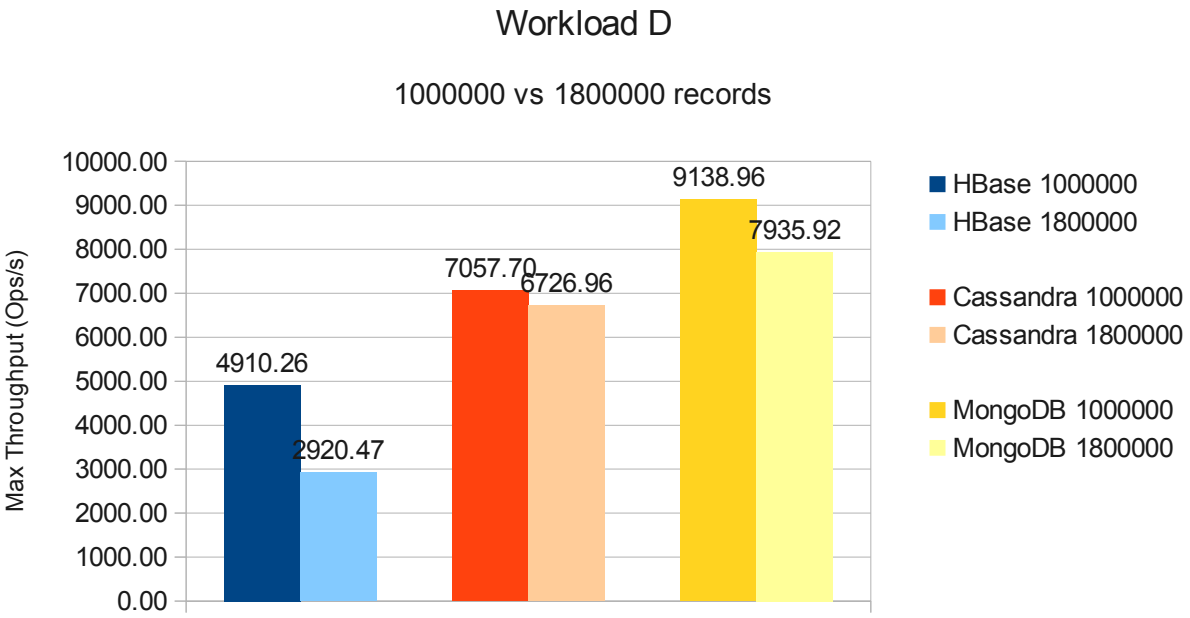
6.4 Φορτίο D

Το φορτίο D αποτελείται από 95% αναγνώσεις και 5% εισαγωγές δεδομένων στη βάση. Μια σημαντική διαφορά συγκριτικά με το φορτίο B είναι ότι οι αναγνώσεις γίνονται πάνω στα πιό πρόσφατα δεδομένα ακολουθώντας κατανομή Latest και όχι Zipfian. Η MongoDB επωφελούμενη από την χρήση memory-mapped αρχείων παρουσίασε και σε αυτό το φορτίο πολύ καλή επίδοση. Η εισαγωγή δεδομένων δε φαίνεται να επηρέασε την απόδοση του συστήματος αφού όπως αναφέρθηκε, η MongoDB χρησιμοποιεί μηχανισμούς ώστε να δεσμεύει εκ των προτέρων τον χώρο που πιθανώς να χρειαστεί για καινούριες εισαγωγές δεδομένων. Οι εισαγωγές στα συστήματα HBase και Cassandra πραγματοποιούνται όπως περιγράφηκαν στα κεφάλαια 4.1.3 και 4.2.3 δημιουργώντας καινούρια αρχεία (HFiles και SSTables αντίστοιχα) χωρίς να αναζητούν κατά την στιγμή της εισαγωγής την ακριβή θέση του δεδομένου στον δίσκο. Η χρήση της κατανομής Latest έκανε την Cassandra να αποδίδει σημαντικά καλύτερα συγκριτικά με το φορτίο B και να πλησιάζει σε απόδοση την MongoDB. Αυτό πιθανώς να οφείλεται στο γεγονός ότι κάποιος σημαντικός αριθμός αναγνώσεων εξυπηρετούνταν από τα δεδομένα που βρίσκονταν λόγω της κατανομής των αναγνώσεων ακόμη στο memtable του συστήματος, δηλαδή στην προσωρινή μνήμη. Θα περιμέναμε αντίστοιχη συμπεριφορά και από το σύστημα HBase, όμως φαίνεται ότι ο μεγάλος αριθμός από αναγνώσεις επιβάρυνε και επηρέασε όπως στα προηγούμενα φορτία τη λειτουργία του συστήματος, προσθέτοντας σημαντικό φορτίο στον Garbage Collector του JVM. Αντίθετα στην Cassandra το μέρος της μνήμης που χρησιμοποιείται για προσωρινή αποθήκευση των κλειδιών βρίσκεται εκτός σωρού (off-Heap).



Σχήμα 6.10: Φορτίο D: Αναγνώσεις 95%

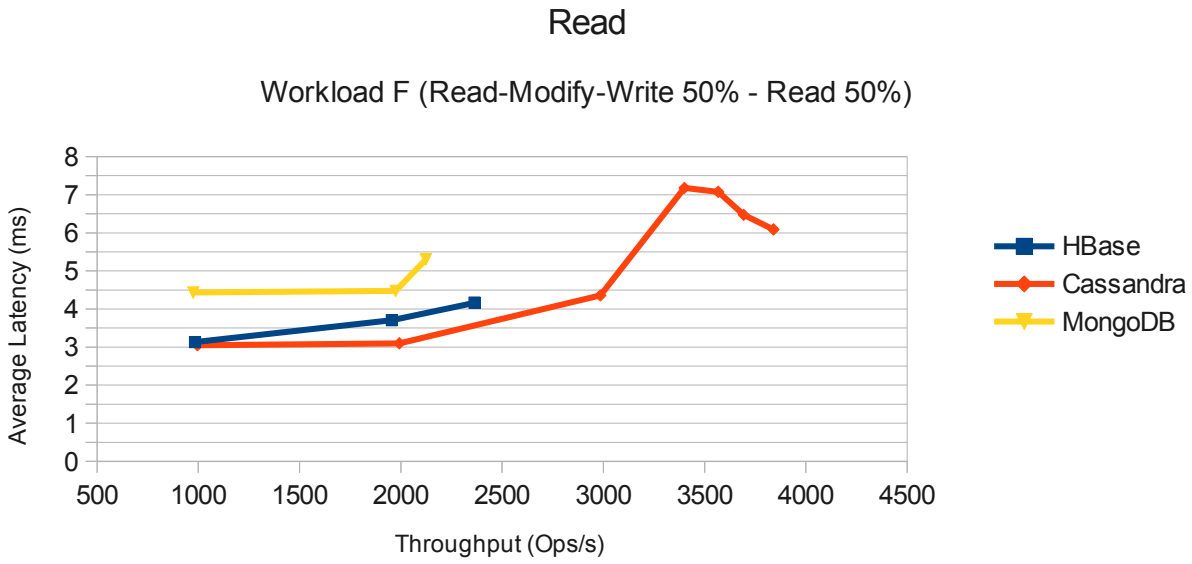
Η HBase παρουσίασε πτώση της απόδοσης κατά 40% όταν αυξήθηκε ο όγκος των δεδομένων στη βάση παρουσιάζοντας προβληματική συμπεριφορά λόγω της περιορισμένης μνήμης. Στις βάσεις Cassandra και MongoDB η πτώση της απόδοσης ήταν 5% και 15.1% αντίστοιχα.

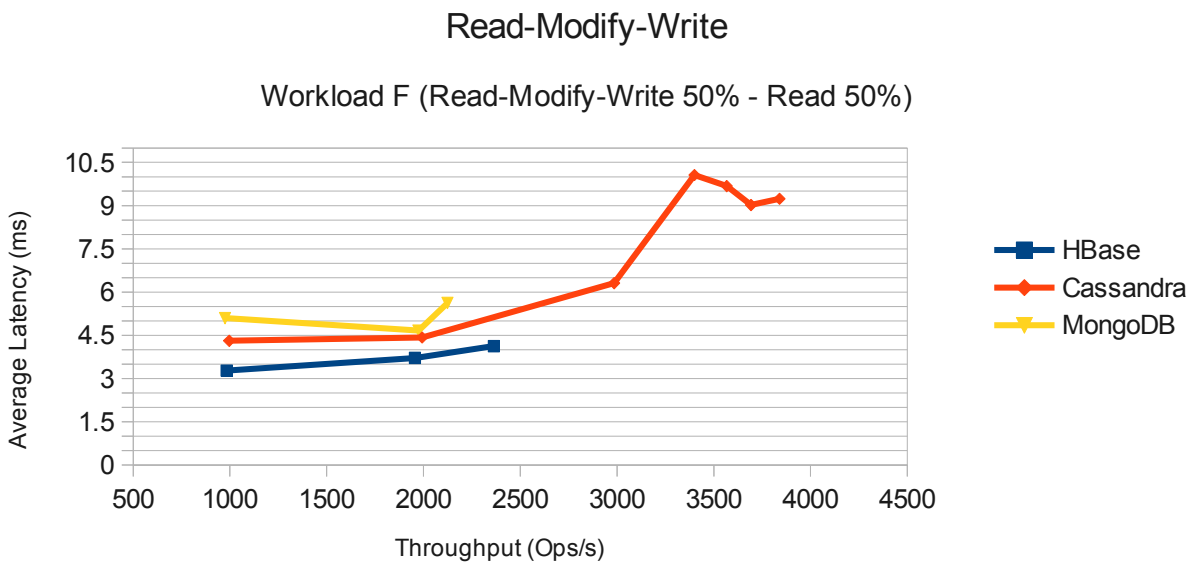


Σχήμα 6.11: Φορτίο D: Σύγκριση μέγιστων επιδόσεων με 1.000.000 και 1.800.000 καταχωρήσεις

6.5 Φορτίο F

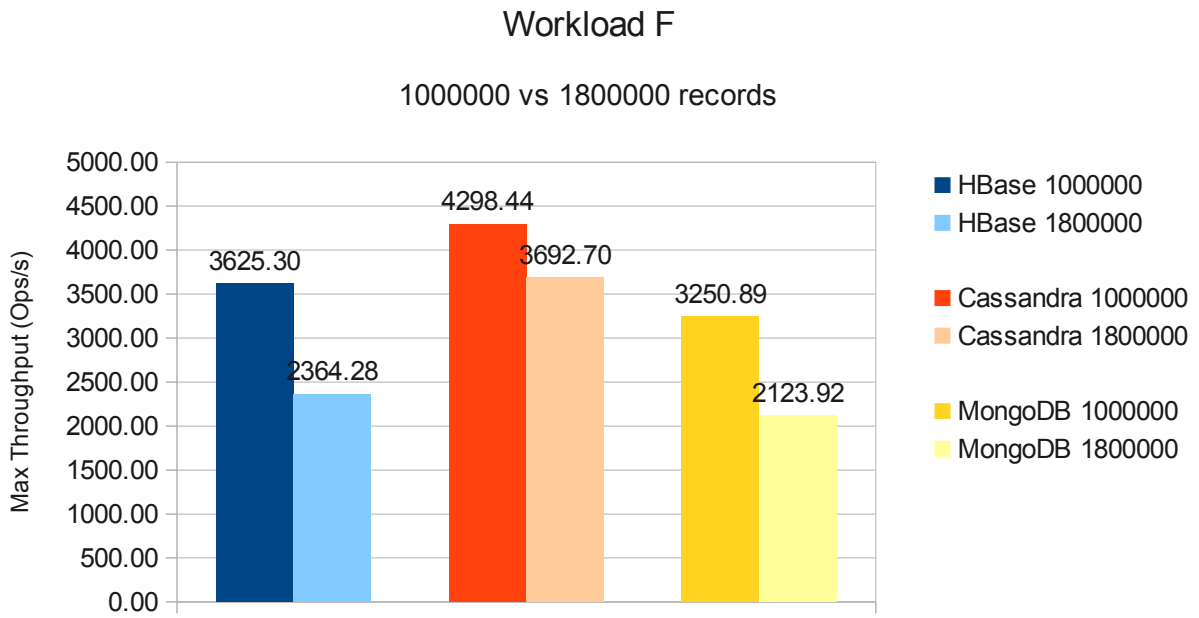
Το φορτίο F αποτελείται από 50% αναγνώσεις και 50% αναγνώσεις στη συνέχεια τροποποιήσεις και εγγραφές των δεδομένων πίσω στη βάση. Και στο φορτίο αυτό φαίνεται ότι βασικό στοιχείο για την επίτευξη υψηλών επιδόσεων και χαμηλών χρόνων απόκρισης είναι η αποδοτική λειτουργία των τεχνικών caching που χρησιμοποιούν τα συστήματα. Τα 4GB προσωρινής μνήμης που είχε ο κάθε κόμβος του συστήματος στη διάθεσή του ήταν πολύ λιγότερα από τις προτεινόμενες τιμές. Έτσι σε φορτία από κάπως πιο πολύπλοκα σενάρια χρήσης, τα συστήματα που επένδυαν περισσότερο στην προσπάθεια να βρίσκονται τα δεδομένα στη μνήμη δεν μπόρεσαν να επωφεληθούν από αυτή την προσπάθεια. Έτσι η MongoDB παρουσίασε στο φορτίο F μειωμένη απόδοση και παρόμοια συμπεριφορά με το φορτίο A και η HBase συνέχισε να παρουσιάζει μειωμένη απόδοση όπως στα φορτία B, C και D. Η Cassandra χωρίς να επιχειρεί να αποθηκεύσει δεδομένα στη μνήμη δεν παρουσίασε τόσο μεγάλη διακύμανση στην απόδοσή της όπως φαίνεται και στο σχήμα 6.14.





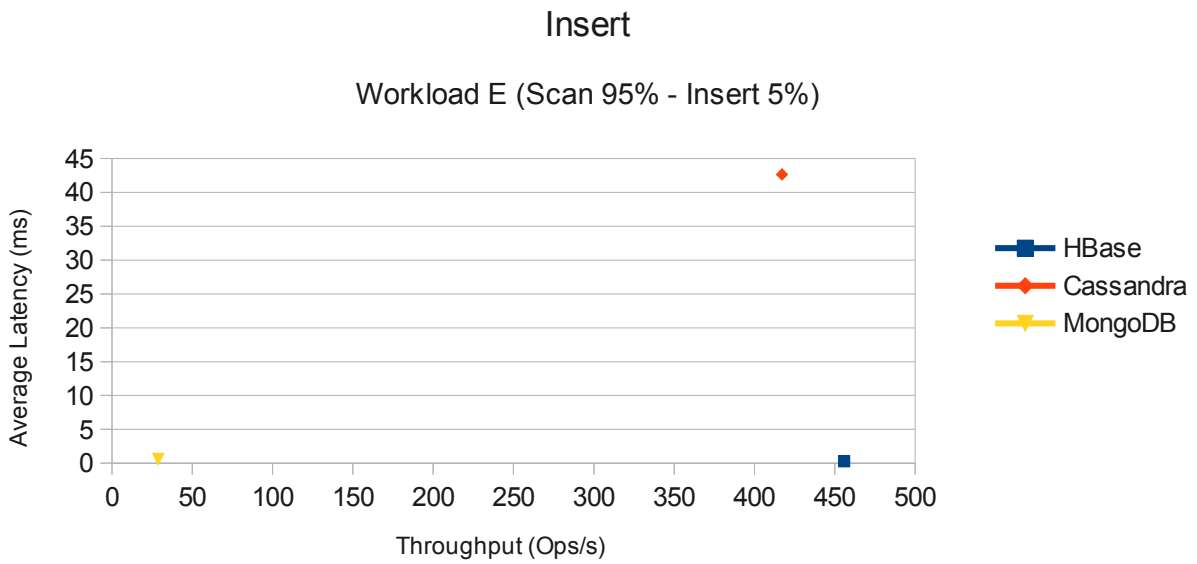
Σχήμα 6.13: Φορτίο F: Αναγνώσεις-Τροποποιήσεις-Εγγραφές 50%

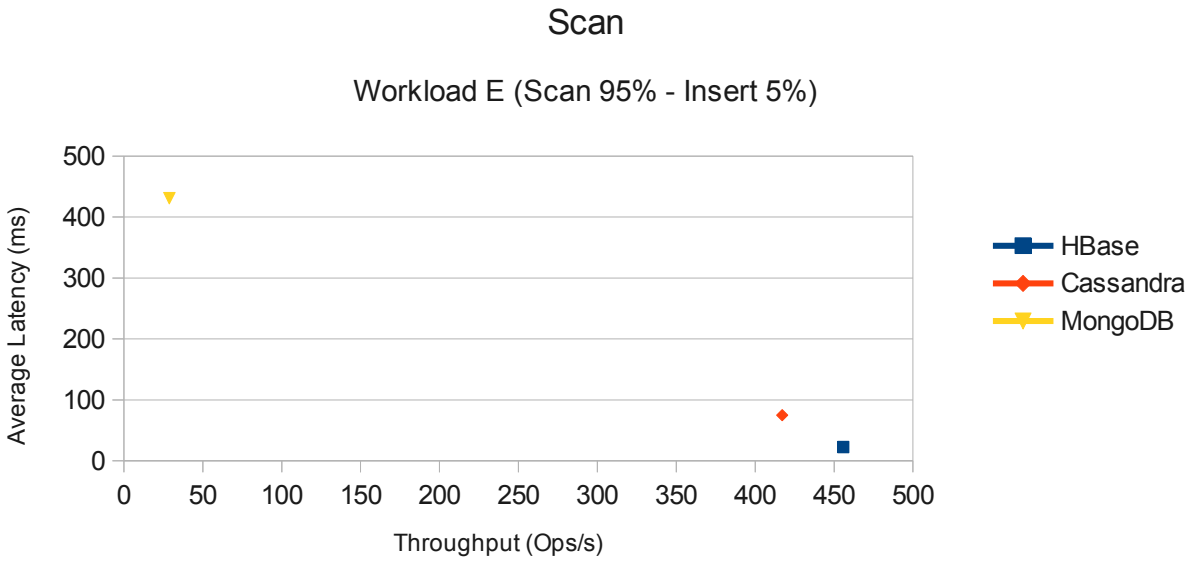
Η HBase παρουσίασε πτώση της απόδοσης κατά 53% όταν αυξήθηκε ο όγκος των δεδομένων στη βάση παρουσιάζοντας και σε αυτό το φορτίο προβληματική συμπεριφορά λόγω της περιορισμένης μνήμης. Η βάση MongoDB επηρεάστηκε και αυτή σημαντικά με πτώση κατά 53%. Η Cassandra που έκανε πιο συντηρητική χρήση των μηχανισμών caching παρουσίασε πτώση μόλις κατά 16.4%.



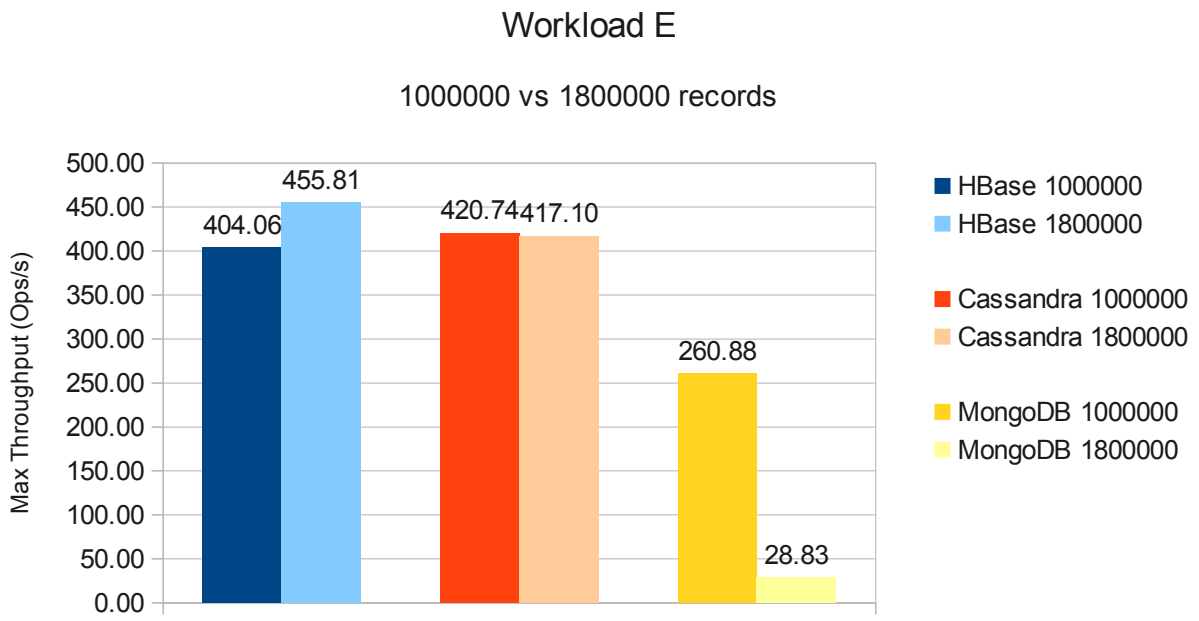
6.6 Φορτίο E

Στο φορτίο F ο YCSB Client πραγματοποιούσε αναζητήσεις δεδομένων σε ποσοστό 95% και εισαγωγές καινούριων δεδομένων σε ποσοστό 5%. Προκειμένου να πραγματοποιήσει τις αναζητήσεις επιλέγει κάποιο δεδομένο (χρησιμοποιώντας κατανομή Zipfian) και στη συνέχεια ζητά από τη βάση ένα εύρος από τα επόμενα δεδομένα σε σειρά. Το μέγεθος των αναζητήσεων καθορίζεται χρησιμοποιώντας Uniform κατανομή και ήταν μέχρι 100 καταχωρήσεις. Οι βάσεις HBase και Cassandra πήγαν αρκετά καλύτερα συγκριτικά με την MongoDB αφού τα δεδομένα τους είναι αποθηκευμένα σειριακά σε αρχεία και η ανάκληση ενός εύρους δεδομένων γινόταν σχετικά αποδοτικά ζητώντας από τον δίσκο ένα μέρος του αρχείου που περιέχει τα δεδομένα. Η MongoDB προκειμένου να φέρει τα ζητούμενα αρχεία πραγματοποιούσε μια σειρά από επαφές με τον δίσκο για κάθε αρχείο εφόσον αυτά δε βρίσκονταν εκείνη τη στιγμή στη μνήμη, γεγονός που έκανε την βάση να είναι λιγότερο αποδοτική σε αυτό το σενάριο χρήσης. Ένας ακόμη λόγος που η απόδοση της MongoDB ήταν μειωμένη σε αυτό το φορτίο, είναι ότι η αναζητήσεις στην βάση αυτή γίνονται με βάση τα πεδία των καταγραφών που είναι αποθηκευμένα μέσα στα αρχεία (αναγνωριστικό id) και επομένως το σύστημα δεν ανακτά τα δεδομένα με βάση έναν δείκτη όπως στα άλλα δύο συστήματα, αλλά πραγματοποιεί προσπέλασή των αρχείων προκειμένου να επιστρέψει αποτέλεσμα. Έτσι η MongoDB ήταν η βάση που επηρεάστηκε πιο σημαντικά από την αύξηση των καταγραφών στη βάση όπως φαίνεται στο σχήμα 6.17, φανερώνοντας ότι σημαντικό παράγοντα και σε αυτό το φορτίο παίζει η αποδοτικότητα των μηχανισμών caching και η ποσότητα της μνήμης που έχει το σύστημα στη διάθεσή του.





Οι βάσεις Cassandra και HBase δεν επηρεάστηκαν από την αλλαγή του όγκου των δεδομένων. Η HBase μάλιστα παρουσίασε και αύξηση της επίδοσης στης κατά 11%. Η MongoDB για τους λόγους που αναφέρθηκαν παραπάνω παρουσίασε σημαντική πτώση όταν αυξήθηκαν τα δεδομένα της βάσης και η μνήμη δεν επαρκούσε ώστε να είναι η πλειοψηφία των δεδομένων αποθηκευμένα σε αυτή. Η μείωση της απόδοσης ήταν της τάξης του 88%



Σχήμα 6.17: Φορτίο Ε: Σύγκριση μέγιστων επιδόσεων με 1.000.000 και 1.800.000 καταχωρήσεις

Κεφάλαιο 7
Συμπεράσματα

Συμπερασματικά, το σύστημα MongoDB λειτουργούσε ιδιαίτερα αποδοτικά σε περιπτώσεις οπού οι αναγνώσεις αποτελούσαν το μεγαλύτερο μέρος του φορτίου χάρη στην χρήση των memory-maped αρχείων που χρησιμοποιούσε. Το σύστημα αυτό έδειξε ιδιαίτερη αδυναμία στο φορτίο που πραγματοποιούσε αναζητήσεις αφού ήταν αναγκαία η επικοινωνία με τον δίσκο αρκετές φορές προκειμένου να ανακληθούν τα απαραίτητα αρχεία με τα δεδομένα, τα οποία στη συνέχεια διαβάζονταν προκειμένου να πραγματοποιηθεί η λειτουργία αυτή. Η Cassandra κάνοντας στην προκαθορισμένη λειτουργία της caching μόνο των θέσεων των δεδομένων στον δίσκο δεν επηρεαζόταν σημαντικά από τις διάφορες αναλογίες των ενεργειών που εκτελούσε ο YCSB Client και διατηρούσε μία ικανοποιητική και σταθερή συμπεριφορά σε όλες τις περιπτώσεις. Σε αυτό συντέλεσε και η επιλογή για χαλαρωμένη συνέπεια αφού η κάθε ενέργεια ανάγνωσης είχε τρεις κόμβους διαθέσιμους για να την εξυπηρετήσουν και έτσι η κατανομή του φορτίου ήταν πιο ομοιογενής στο σύστημα. Τέλος η HBase με την επιλογή να αποθηκεύει τα προσωρινά δεδομένα on-heap σε συνδυασμό με την περιορισμένη μνήμη RAM έδειξε όχι μόνο να μην μπορεί να επωφεληθεί από τη χρήση των μηχανισμών caching αλλά αυτοί να επιβαρύνουν και τη συνολική λειτουργία του συστήματος σε κάποιες περιπτώσεις.

Τα συστήματα που χρησιμοποιήθηκαν βασίζουν σε μεγάλο βαθμό την καλή απόδοσή τους στην χρήση των μηχανισμών caching των δεδομένων. Επομένως το σύστημα που είχαμε στη διάθεσή μας με την περιορισμένη μνήμη RAM δεν βοήθησε ώστε αυτά να αναδείξουν τις πραγματικές τους δυνατότητες. Παρόλα αυτά το πείραμα που πραγματοποιήθηκε κάτω από αυτές τις συνθήκες βοήθησε στο να γίνουν αισθητές οι διαφορές που προκαλούσαν οι διάφοροι εσωτερικοί μηχανισμοί των συστημάτων αυτών. Κατά το στήσιμο και την εκτέλεση των πειραμάτων συναντήσαμε πολλές δυνατότητες παραμετροποίησης των συστημάτων και των μηχανισμών αυτών που θα μπορούσαν να χρησιμοποιηθούν ώστε τα συστήματα να αποδίδουν καλύτερα σε κάθε σενάριο χρήσης. Όμως παρά τον πειρασμό για περισσότερες δοκιμές με τροποποιήσεις των προκαθορισμένων παραμέτρων, δεν τις πραγματοποιήσαμε αφού κάτι τέτοιο θα ήταν εκτός του στόχου της παρούσας εργασίας και τα αποτελέσματα θα αντικατόπτριζαν τις δυνατότητές μας για παραμετροποίηση.

Βιβλιογραφία

[1] Digital Universe Infographic.IDC (Dec,2012)(Accessed: Sep-2014)
[Online]:http://www.emc.com/infographics/digital-universe-business-infographic.htm

[2] Presto: Interacting with petabytes of data at Facebook.Lydia Chan (Nov,2013)(Accessed: Sep-2014)
[Online]:https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920

[3] CERN Computing (Accessed: Sep-2014)
[Online]:http://home.web.cern.ch/about/computing

[4] E.F.Codd, (1970, Jun)A relational model of data for large shared data banks
[Online]:http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[5] E.A.Brewer, (2000, Jul)Towards robust distributed systems
[Online]:http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[6] List of nosql databases (Accessed: Sep-2014)
[Online]:http://nosql-database.org

[7] N.Lynch, S.Gilbert,(2002, Nov)Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services
[Online]:http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

[8] D.Abadi,(2010, Apr)Problems with cap, and yahoo's little known nosql system
[Online]:http://dbmsmusings.blogspot.gr/2010/04/problems-with-cap-and-yahoos-little.html

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber,(2006)Bigtable: A Distributed Storage System for Structured Data
[Online]:http://static.googleusercontent.com/media/research.google.com/el//archive/bigtable-osdi06.pdf

[10]: Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, The Google file system, 2003

[11] Patrick O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth O'Neil,(1996, Jun)The log-structured merge-tree (LSM-tree)
[Online]:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.44.2782&rep=rep1&type=pdf

[12] B-tree. (Accessed: Sep-2014)
[Online]:http://en.wikipedia.org/wiki/B-tree

[13] HBase Documentation: Data Model. (Accessed: Sep-2014)
[Online]:http://hbase.apache.org/book/datamodel.html

[14] HBase Documentation: Physical View. (Accessed: Sep-2014)
[Online]:http://hbase.apache.org/book/physical.view.html

[15] HBase ACID Semantics. (Accessed: Sep-2014)
[Online]:http://hbase.apache.org/acid-semantics.html

[16] ACID in HBase . (Accessed: Sep-2014)
[Online]:http://hadoop-hbase.blogspot.gr/2012/03/acid-in-hbase.html

[17]: Hastorun Deniz, Jampani Madan, Kakulapati Gunavardhan, Pilchin Alex, Sivasubramaniaz Swaminathan, Vosshall Peter, Vogels Werner, Dynamo: Amazon's highly available key-value store, 2007

[18] The Apache Cassandra Project. (Accessed: Sep-2014)
[Online]:http://cassandra.apache.org/

[19] Datastax Documentation: About transactions and concurrency control. (Accessed: Sep-2014)

[Online]:http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_about_transactions_c.html

[20] Who Uses MongoDB?. (Accessed: Sep-2014)
[Online]:http://www.mongodb.com/who-uses-mongodb

[21] Journaling Mechanics. (Accessed: Sep-2014)
[Online]:http://docs.mongodb.org/v2.4/core/journaling/

[22] MongoDB: Write Concern. (Accessed: Sep-2014)
[Online]:http://docs.mongodb.org/manual/core/write-concern/

[23] GitHub: Yahoo! Cloud Serving Benchmark. (Accessed: Sep-2014)
[Online]:https://github.com/brianfrankcooper/YCSB

[24] Zipf's Law. (Accessed: Sep-2014)
[Online]:http://en.wikipedia.org/wiki/Zipf%27s_law

[25] BonFIRE Project. (Accessed: Sep-2014)
[Online]:http://www.bonfire-project.eu/

[26] Why does HBase care about /etc/hosts?. (Accessed: Sep-2014)
[Online]:http://devving.com/?p=414

[27] BlockCache 101.Nick Dimiduk (Accessed: Sep-2014)
[Online]:http://www.n10k.com/blog/blockcache-101/