ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση Τεχνικών Αποθήκευσης και Αναζήτησης Πολυμεσικών Δεδομένων με χρήση NoSQL Βάσεων Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΛΗΡΟΔΕΤΗΣ ΠΕΤΡΟΣ

Επιβλέπουσα : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβριος 2016

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

Ανάλυση Τεχνικών Αποθήκευσης και Αναζήτησης Πολυμεσικών Δεδομένων με χρήση NoSQL Βάσεων Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΛΗΡΟΔΕΤΗΣ ΠΕΤΡΟΣ

Επιβλέπουσα : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Οκτωβρίου 2016.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....

Αθήνα, Οκτώβριος 2016

(Υπογραφή)

....

ΚΛΗΡΟΔΕΤΗΣ ΠΕΤΡΟΣ

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

Copyright © Κληροδέτης Πέτρος, 2016.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.
Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Ευχαριστίες

Η διπλωματική αυτή εκπονήθηκε στο Εργαστήριο Distributed Knowledge and Media Systems Group του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη της Καθηγήτριας Θεοδώρας Βαρβαρίγου.

Αρχικά, θα ήθελα να ευχαριστήσω την κ. Θεοδώρα Βαρβαρίγου, η οποία μου ανέθεσε αυτή την ενδιαφέρουσα εργασία, μέσα από την οποία ήρθα για πρώτη φορά σε επαφή με θέματα που βρίσκονται στο επίκεντρο των τεχνολογικών εξελίξεων.

Ιδιαίτερες ευχαριστίες θα ήθελα να απευθύνω στον υποψήφιο Διδάκτορα του Ε.Μ.Π., Βρεττό Μουλό, για την άριστη καθοδήγησή του και την πολύτιμη βοήθειά του σε οποιοδήποτε πρόβλημα αντιμετώπισα κατά τη διάρκεια εκπόνησης της εργασίας.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου και στα αγαπημένα μου πρόσωπα για όλη τη βοήθεια, υποστήριξη, εμπιστοσύνη και αντοχή που έδειξαν κατά τη διάρκεια όλων των σπουδών μου.

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, παρουσιάζοντας μεγαλύτερη ευελιξία, 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.

Στην παρούσα εργασία μελετάμε τρόπους αποδοτικής αποθήκευσης και αναζήτησης πολυμεσικών δεδομένων. Το μεγάλο μέγεθος και η πολύπλοκη δομή των πολυμεσικών δεδομένων, μας αποτρέπει να κάνουμε χρήση παραδοσιακών τεχνικών αποθήκευσης όπως είναι οι σχεσιακές βάσεις δεδομένων και έτσι αναζητήσαμε λύσεις στον τομέα των NoSQL βάσεων. Οι λύσεις που εξετάζουμε αφορούν δύο από τις δημοφιλέστερες βάσεις της νέας γενιάς, την MongoDB και την Apache Cassandra. Αρχικά, αναλύουμε την αρχιτεκτονική και τα χαρακτηριστικά των δύο βάσεων και στην συνέχεια μοντελοποιούμε τα δεδομένα μας κατάλληλα για την αποδοτική αποθήκευση σε αυτές. Έπειτα, πραγματοποιούμε πειραματική αξιολόγηση των δυνατοτήτων των δύο συστημάτων καταλήγοντας σε συγκριτικά συμπεράσματα ως προς τις λειτουργίες τους. Τέλος, για την κατανεμημένη επεξεργασία των δεδομένων κάνουμε χρήση του προγραμματιστικού μοντέλου Map Reduce και του κατανεμημένου περιβάλλοντος Apache Spark με στόχο την εξαγωγή χρήσιμης πληροφορίας από τα πολυμεσικά δεδομένα.

Λέξεις Κλειδιά:

Big Data, NoSQL, Κατανεμημένες ΒΔ, MongoDB, GridFS, Apache Cassandra, Ευρετήρια, MapReduce, Apache Spark, Mpeg-7

Abstract

The rapid increase in the volume of data in recent years has led to the development of new database systems. Traditional relational databases despite their widespread acceptance and variety of features they offer, are inefficient in cases of management of large data volumes. In order for management and useful information extraction from these data to be enabled, the new database systems differ from traditional, showing more flexibility, adjusting the data structure and sacrificing some functionality to ensure horizontal scaling and sharing of data in multiple nodes for parallel storage and processing.

Therefore, the scope of this thesis is the study of efficient storage and searching methods for multimedia data. Large size and complex structure of multimedia data prevent us from using traditional storage techniques such as relational databases and therefore alternative solutions in the NoSQL database field had to be searced. The solutions considered involve two of the most popular databases of the new generation, the MongoDB and Apache Cassandra. Firstly, we analyze the architecture and features of the two databases and then appropriate modeling of data for efficient storage is implemented. Secondly, we perform an experimental evaluation of the two systems and conclude to comparative results as far as their functionalities are concerned. Finally, the distributed data processing is based on the Map Reduce programming model and the distributed environment of Apache Spark for the extraction of useful information from the multimedia data.

Keywords:

Big Data, NoSQL, Distributed Databases, MongoDB, GridFS, Apache Cassandra, Indexes, MapReduce, Apache Spark, Mpeg-7

Πίνακας περιεχομένων

1 Big Data **1**

1.1 Ορισμός 2

1.2 Χαρακτηριστικά 2

1.3 Τεχνικές Διαχείρισης και Αξιοποίησης - Προκλήσεις 5

1.4 Υπολογιστικό Νέφος 8

1.5 Μη-σχεσιακές Βάσεις Δεδομένων 10

1.6 MapReduce 14

2 MPEG-7 .... **17**

3 Περιγραφή Προβλήματος **20**

4 Technical Components **23**

4.1 MongoDB 23

4.1.1 Μοντέλο Δεδομένων 24

4.1.2 Μοντέλο Ερωτημάτων 25

4.1.3 Ευρετήρια 26

4.1.4 Αρχιτεκτονική 30

4.1.5 Storage Engine 34

4.1.6 Aggregation 37

4.2 Apache Cassandra 40

4.2.1 Μοντέλο Δεδομένων 41

4.2.2 Μοντέλο Ερωτημάτων 44

4.2.3 Ευρετήρια 47

4.2.4 Αρχιτεκτονική 52

4.2.5 Storage Engine 56

4.3 Apache Spark 60

4.3.1 Το οικοσύστημα του Spark 60

4.3.2 Αρχιτεκτονική 62

4.3.3 Resilient Distributed Datasets (RDD) 64

5 Υλοποίηση και Αποτελέσματα Διπλωματικής 66

5.1 Μοντέλο Δεδομένων 66

5.2 Αναγνώσεις δεδομένων 70

5.3 Εγγραφές Δεδομένων 73

5.4 Αναζήτηση βάσει περιεχομένου 74

5.5 MapReduce 78

5.6 Συμπεράσματα 81

6 Βιβλιογραφία 83

Κατάλογος εικόνων

Εικόνα 1 Big Data Volume 3
Εικόνα 2 Big Data Web 60 seconds 4
Εικόνα 3 NoSQL vs RDBMS scalability 10
Εικόνα 4 Cap Theorem....12
Εικόνα 5 Map Reduce example....16
Εικόνα 6 Mpeg-7 applications....19
Εικόνα 7 Η δομή της MongoDB. 24
Εικόνα 8 Παράδειγμα ερωτήματος που χρησιμοποιεί ευρετήριο....27
Εικόνα 9 Ευρετήριο μονού πεδίου, αύξουσα σειρά 28
Εικόνα 10 Σύνθετο ευρετήριο με δύο πεδία....29
Εικόνα 11 Ευρετήριο πολλαπλών κλειδιών σε πίνακα με εμφολευμένα έγγραφα....29
Εικόνα 12 MongoDB cluster architecture 32
Εικόνα 13 MongoDB replica set 34
Εικόνα 14 WiredTiger Compression....36
Εικόνα 15 WiredTiger Caches (WiredTiger Cache and FS Cache) 37
Εικόνα 16 MongoDB aggregation pipeline example 38
Εικόνα 17 MongoDB map-reduce example 39
Εικόνα 18 Γραμμική κλιμάκωση στην Cassandra....40
Εικόνα 19 Column Family 42
Εικόνα 20 Δευτερεύον ευρετήριο user-by-country 47
Εικόνα 21 Token Ring....53
Εικόνα 22 Επίπεδα συνέπειας 56
Εικόνα 23 Cassandra, Μηχανισμοί στη διαδικασία εγγραφής....57
Εικόνα 24 Row cache and Key cache request flow....59
Εικόνα 25 Read request flow....59
Εικόνα 26 Το οικοσύστημα του Spark 62
Εικόνα 27 Αρχιτεκτονική του Spark 64

Εικόνα 28 Παράδειγμα mp7 (xml) 67

Εικόνα 29 Μετατροπή mp7 σε json 68

Εικόνα 30 Μετατροπή αρχείου mp7 σε table της Cassandra 69

Εικόνα 31 Παράδειγμα NewDescriptors 75

Κώδικας 1 Word count example 16

Κώδικας 2 mp7 Table 69

Κώδικας 3 Τύποι αντικειμένων για όλες τις σκηνές με aggregation pipeline 71

Κώδικας 4 Materialized View mp7_by_field 72

Κώδικας 5 Table αντικειμένων ανά αρχείο 75

Κώδικας 6 Δημιουργία ευρετηρίων MongoDB 76

Κώδικας 7 Δημιουργία ευρετηρίων Cassandra 77

Κώδικας 8 map_function.js 79

Κώδικας 9 reduce_function.js 79

Κώδικας 10 finalize_function.js 79

Κώδικας 11 Spark MapReduce (scala) 80

Διάγραμμα 1 Ασύγχρονη ανάκτηση αρχείων 71

Διάγραμμα 2 Διάβασμα ενός πεδίου (τύποι σχημάτων) απ’ όλες τις περιγραφές 73

Διάγραμμα 3 Εγγραφές Descriptors 73

Διάγραμμα 4 Ερωτήματα αναζήτησης βάσει χρώματος-όγκου με χρήση ευρετηρίων 77

Διάγραμμα 5 MapReduce στα ιστογράμματα χρώματος των σκηνών 80

1

Big Data

Με μια γρήγορη ματιά στο Google Trends διαπιστώνει κανείς ότι τα Big Data (BD) είναι ευρέως διαδεδομένα τα τελευταία χρόνια. Οι τελευταίες τεχνολογικές εξελίξεις κυρίως στον τομέα των επικοινωνιών και των ολοκληρωμένων κυκλωμάτων έχουν δώσει την δυνατότητα να δημιουργηθούν μηχανισμοί παρακολούθησης των λειτουργιών ενός οργανισμού σε πολύ λεπτομερές επίπεδο. Η λεπτομερής αυτή ψηφιοποίηση των διαδικασιών παραγωγής έχουν καταστήσει μεγάλους οργανισμούς αλλά και εταιρείες μικρού μεγέθους ικανούς να παράγουν τεράστιους όγκους δεδομένων με πολύ ταχείς ρυθμούς. Τα δεδομένα αυτά κρύβουν πολύτιμη γνώση καθώς η ανάλυση τους μπορεί να οδηγήσει σε σημαντικές βελτιστοποιήσεις της παραγωγής.

Δεν είναι όμως μόνο οι οργανισμοί που παράγουν τεράστιους όγκους δεδομένων. Ακόμη και σε μικρότερη κλίμακα οργάνωσης, στο επίπεδο του ατόμου, η παραγωγή δεδομένων είναι πρωτόγνωρη. Οι περισσότεροι άνθρωποι διαθέτουν έναν ψηφιακό εαυτό, ως προβολή των δραστηριοτήτων τους στα κοινωνικά δίκτυα. Οι κινητές συσκευές αποτελούν το πλέον προσφιλές μέσο, καθώς παρέχουν άμεση πρόσβαση στις εφαρμογές cloud.

Ο Eric Smidt ,εκτελεστικός πρόεδρος της Google, το 2010 είπε χαρακτηριστικά “Είχαν παραχθεί 5 exabytes δεδομένων από την αυγή του πολιτισμού έως το 2003, τώρα τόσα δεδομένα παράγουμε κάθε 2 μέρες” ενώ η IBM στην ιστοσελίδα της αναφέρει “Κάθε μέρα

δημιουργούμε 2,5 exabytes δεδομένων, τα οποία είναι τόσα πολλά, που το 90% των δεδομένων στον κόσμο σήμερα έχει δημιουργηθεί τα τελευταία δύο χρόνια μόνο” . Μεγάλα Δεδομένα δημιουργούνται συνεχώς γύρω μας και καταφθάνουν με μεγάλη ταχύτητα, όγκο και ποικιλία από πολλές πηγές. Αισθητήρες για συλλογή πληροφοριών περιβάλλοντος, δημοσιεύσεις σε σελίδες κοινωνικής δικτύωσης, ψηφιακές φωτογραφίες και βίντεο, αρχεία συναλλαγών, σήματα GPS κινητών τηλεφώνων είναι μερικές από αυτές. Αυτά τα δεδομένα είναι τα μεγάλα δεδομένα.

1.1 Ορισμός

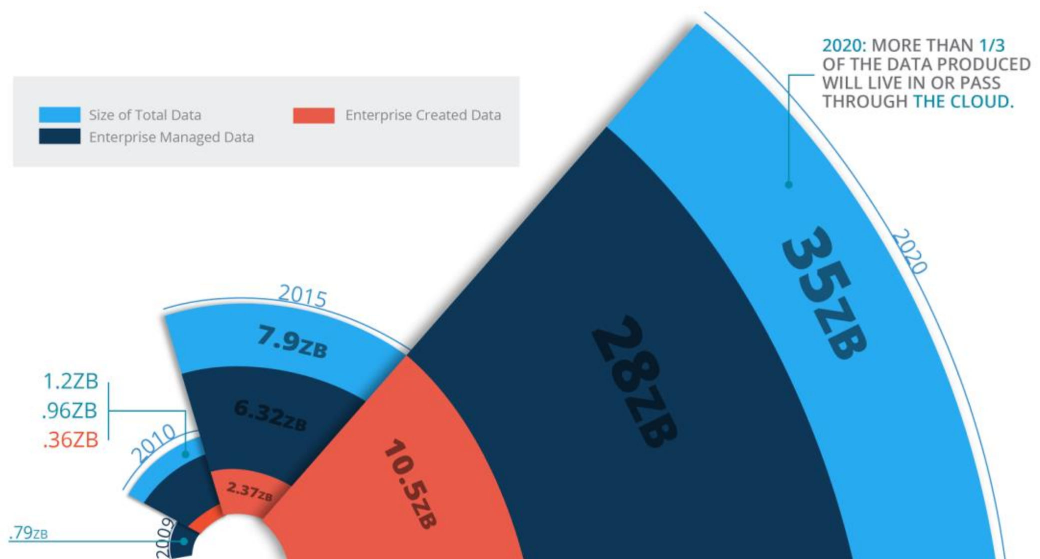
Η Wikipedia δίνει τον ακόλουθο ορισμό για τα Μεγάλα Δεδομένα: "Big Data είναι μια συλλογή από σύνολα δεδομένων τόσο μεγάλη και πολύπλοκη που είναι δύσκολο να επεξεργαστεί με τη χρήση των κλασικών βάσεων δεδομένων και των παραδοσιακών εφαρμογών επεξεργασίας δεδομένων".

1.2 Χαρακτηριστικά

Ο αναλυτής Doug Laney το 2001 αναφέρθηκε στα 3 βασικά χαρακτηριστικά των Μεγάλων Δεδομένων: Όγκος (Volume), Ταχύτητα (Velocity) και Ποικιλία (Variety) , γνωστά ως “The 3V’s of Big Data”. Έκτοτε έχουν προστεθεί και άλλα χαρακτηριστικά στην προσπάθεια για μια πιο περιεκτική περιγραφή της πολύπλοκης φύσης των Μεγάλων Δεδομένων, καταλήγοντας σε 7 χαρακτηριστικά “The 7V’s of Big Data”.

1. Όγκος (*Volume*)

   Όγκος είναι το πόσα δεδομένα έχουμε. Συνηθίζαμε να τα μετράμε σε Gigabytes και τώρα μετράμε σε Zettabytes (ZB) ή ακόμα και Yottabytes (YB). Στις μέρες μας είναι πολλοί οι παράγοντες που συμβάλλουν στην αύξηση του όγκου των δεδομένων. Υπάρχουν δεδομένα συναλλαγών αποθηκευμένα για χρόνια ενώ συλλέγονται και αδόμητα δεδομένα συνεχούς ροής από τα Social Media. Πολύ μεγάλη συμβολή στο συνεχώς διευρυνόμενο ψηφιακό κόσμο έχει το Διαδίκτυο των Πραγμάτων (IoT) με αισθητήρες σε όλο τον κόσμο, σε όλες τις συσκευές δημιουργώντας δεδομένα κάθε δευτερόλεπτο.



Εικόνα 1 Big Data Volume

1. Ταχύτητα (*Velocity*)

   Η ταχύτητα με την οποία τα δεδομένα δημιουργούνται, επεξεργάζονται και αναλύονται, αυξάνεται συνεχώς. Τα δεδομένα πλέον δημιουργούνται σε πραγματικό χρόνο και πρέπει να επεξεργαστούν και να αναλυθούν σε πραγματικό ή σχεδόν πραγματικό χρόνο για να έχουν αξία για τις επιχειρήσεις. Αρκεί να αναλογιστούμε ότι κάθε λεπτό: ανεβάζουμε 400 ώρες video στο Youtube, στέλνονται 205.6 εκατομμύρια emails, δημοσιεύονται 422.340 tweets και γίνονται περίπου 3.1 εκατομμύρια αναζητήσεις στο Google.

2. Ποικιλία (*Variety*)

   Η ποικιλία αναφέρεται περισσότερο στη διαχείριση της πολυπλοκότητας από τους πολλαπλούς τύπους δεδομένων συμπεριλαμβανομένων των δομημένων, ημι-δομημένων και μη δομημένων δεδομένων. Οι οργανισμοί θα πρέπει να ενσωματώσουν και να αναλύσουν δεδομένα από μια σύνθετη σειρά παραδοσιακών και μη πηγών πληροφόρησης εντός και εκτός της επιχείρησης. Στις μέρες μας το 90% των δεδομένων που δημιουργούνται είναι αδόμητα.

Εικόνα 2 Big Data Web 60 seconds

3. Μεταβλητότητα (*Variability*)

   Η ασυνέπεια της συλλογής δεδομένων μπορεί να παρεμποδίσει τις διαδικασίες για το χειρισμό και τη διαχείριση τους. Αυτό μπορεί να παρατηρείται σε εποχιακά δεδομένα ή ροές δεδομένων με περιοδικές κορυφώσεις , όπως όταν υπάρχει κάποια τάση στα social media. Επίσης, η μεταβλητότητα αφορά τα δεδομένα των οποίων η σημασία αλλάζει.

4. Αξιοπιστία (*Veracity*)

   Αφορά την ορθότητα και την ακρίβεια των δεδομένων μας. Παρά τους ισχυρισμούς αναφορικά με τη μεγάλη αξία των BD, τα δεδομένα είναι σχεδόν άχρηστα αν δεν είναι ακριβή. Αυτό επηρεάζει ιδιαίτερα τα προγράμματα, τα οποία είναι υπεύθυνα για

την αυτοματοποιημένη διαδικασία λήψης αποφάσεων και τους αλγόριθμους μηχανικής μάθησης. Τα αποτελέσματα αυτών των προγραμμάτων είναι τόσο αξιόπιστα όσο και τα δεδομένα που επεξεργάζονται.

5. Οπτικοποίηση (*Visualization*)

   Όταν τελειώσουμε με την επεξεργασία των δεδομένων και εξάγουμε τις κατάλληλες σχέσεις και συμπεράσματα χρειάζεται ένας οργανωμένος τρόπος παρουσίασης των αποτελεσμάτων, ώστε αυτά να είναι ευανάγνωστα και κατανοητά.

6. Αξία (*Value*)

   Τα BD θεωρείται ότι έχουν μεγάλη αξία τόσο για τις επιχειρήσεις όσο και την κοινωνία. Είναι πραγματικότητα ότι στις BD συλλογές μπορείς να ανακαλύψεις σχέσεις και να εξάγεις συμπεράσματα που θα ήταν αδύνατον να βρεις αν είχες μικρή συλλογή δεδομένων. Για παράδειγμα, στο παρελθόν αποθηκεύαμε πληροφορίες για κάθε ένα group καταναλωτών, ενώ, τώρα αποθηκεύονται δεδομένα για τον κάθε πελάτη ξεχωριστά. Όπως είναι κατανοητό, αυτό μας δίνει την δυνατότητα για μια πιο λεπτομερή ανάλυση που μας βοηθά να παράγουμε πιο χρήσιμα συμπεράσματα, δηλαδή χρειαζόμαστε πολλά δεδομένα ώστε να έχουμε σημαντικά ευρήματα.

1.3 Τεχνικές Διαχείρισης και Αξιοποίησης - Προκλήσεις

Τα χαρακτηριστικά των BD που αναφέρθηκαν, αποτελούν παράλληλα και τις προκλήσεις που αφορούν τα ίδια τα δεδομένα και την φύση τους. Οι προκλήσεις που θα εξετάσουμε σε αυτήν την ενότητα αφορούν τις τεχνικές που εφαρμόζονται στα BD από την σύλληψη τους έως τα τελικά αποτελέσματα.

1. Απόκτηση και Αποθήκευση (*Data Acquisition and Warehousing*)

   Η απόκτηση (*acquisition)* αφορά την διαδικασία συλλογής δεδομένων από ποικίλες πηγές, όπως η μέτρηση τιμών από αισθητήρες, η καταγραφή ήχου, βίντεο ή η εξαγωγή δεδομένων από τον ιστό.

   Η αποθήκευση (warehousing) αφορά τις υποδομές στις οποίες θα αποθηκευτούν τα δεδομένα με τρόπο τέτοιο ώστε να είναι ασφαλή, αξιόπιστα, εύκολα στην ανάκτηση και στη διαχείριση.

Η πολυπλοκότητα των BD και οι εκθετικά αυξανόμενες ανάγκες δημιούργησαν πρωτοφανή προβλήματα στην μηχανική των BD όπως η απόκτηση και η αποθήκευση τους (Wang & Wiebe, 2014). Τα κύρια εμπόδια στην ανάλυση BD προκύπτουν από την έλλειψη προέλευσης δεδομένων, γνώσεων και αποκλίσεις στην κλίμακα των δεδομένων που προέρχονται από τη συλλογή και την επεξεργασία τους. Αυτό περιορίζει περαιτέρω την ταχύτητα στην οποία μπορούν να συλληφθούν και να αποθηκευτούν τα δεδομένα και επηρεάζει την ικανότητα να αποσπάμε χρήσιμη πληροφορίας από αυτά (Chen & Zhang, 2014). Για να συλληφθούν σχετικές και πολύτιμες πληροφορίες , απαιτούνται ευφυή φίλτρα που να μπορούν να κρατήσουν την χρήσιμη πληροφορία και να πετάξουν αυτές που περιέχουν ασάφειες και ασυνέπειες. Για να επιτευχθεί αυτό, απαιτούνται αποτελεσματικοί αλγόριθμοι αναλυτικής για να διαχειριστούν τον τεράστιο όγκο δεδομένων συνεχούς ροής.

2. Καθαρισμός (*Data Cleansing*)

Ο καθαρισμός είναι η διαδικασία εντοπισμού φθαρμένων, ατελών, ανακριβών, άσχετων δεδομένων από μία βάση και στην συνέχεια της αντικατάστασης, διόρθωσης, ή διαγραφής των «βρώμικων» δεδομένων.

Δεδομένα έρχονται σε όλες τις μορφές και τα μεγέθη, ιδιαίτερα όταν προέρχονται από τον ιστό , σκοπός είναι να ανασχηματίσουμε σε ένα αξιόπιστο data set. Λόγω των έντονων, διαφορετικών, αλληλένδετων και αναξιόπιστων χαρακτηριστικών των δεδομένων, ο καθαρισμός τους αποτελεί μεγάλη πρόκληση.

3. Εξόρυξη (*Data Mining*)

Εξόρυξη δεδομένων (ή ανακάλυψη γνώσης από βάσεις δεδομένων) είναι η εξεύρεση μιας (ενδιαφέρουσας, αυτονόητης, μη προφανούς και πιθανόν χρήσιμης) πληροφορίας ή προτύπων (μοτίβων) από μεγάλες βάσεις δεδομένων με χρήση αλγορίθμων ομαδοποίησης ή κατηγοριοποίησης και των αρχών της στατιστικής, της τεχνητής νοημοσύνης, της μηχανικής μάθησης και των συστημάτων βάσεων δεδομένων. Στόχος της εξόρυξης δεδομένων είναι η πληροφορία που θα εξαχθεί και τα πρότυπα που θα προκύψουν να έχουν δομή κατανοητή προς τον άνθρωπο έτσι ώστε να τον βοηθήσουν να πάρει τις κατάλληλες αποφάσεις.

Η εξόρυξη δεν είναι εύκολη υπόθεση, καθώς οι αλγόριθμοι μπορούν να γίνουν πολύπλοκοι και τα δεδομένα δεν είναι πάντα διαθέσιμα σε ένα σημείο. Δυσκολίες αντιμετωπίζονται στη διαχείριση θορυβωδών ή ατελών δεδομένων και στη διαχείριση

πολύπλοκων τύπων δεδομένων. Πρόκληση αποτελεί η εξόρυξη πληροφορίας από ετερογενείς βάσεις όπου μπορεί τα δεδομένα να είναι δομημένα, ημι-αδόμητα, ή αδόμητα. Προκλήσεις υπάρχουν και στις επιδόσεις αφού απαιτείται υψηλή απόδοση και επεκτασιμότητα στους αλγόριθμους εξόρυξης όπως επίσης και αλγόριθμοι παράλληλοι και κατανεμημένοι.

4. Ενσωμάτωση - Ενοποίηση (*Data Integration*)

Η ενοποίηση αφορά τον συνδυασμό δεδομένων από διαφορετικές πηγές, που είναι αποθηκευμένα με ποικίλες τεχνολογίες, και παρέχει μια ενοποιημένη, ομοιόμορφη όψη αυτών των δεδομένων. Αυτή η διαδικασία γίνεται σημαντική σε μια ποικιλία καταστάσεων, που περιλαμβάνουν και εμπορικούς (όταν δύο παρόμοιες εταιρείες πρέπει να συγχωνεύσουν τις βάσεις δεδομένων τους) και επιστημονικούς τομείς (συνδυάζονται, για παράδειγμα, αποτελέσματα έρευνας από διαφορετικά αποθετήρια βιοπληροφορικής).

Η ενοποίηση δεδομένων εμφανίζεται με αυξανόμενη συχνότητα, καθώς αυξάνεται ο όγκος και η ανάγκη διαμοιρασμού των υπαρχόντων δεδομένων. Η μεγαλύτερη πρόκληση είναι η τεχνική υλοποίηση της ενοποίησης δεδομένων από ανόμοιες, συχνά ασυμβίβαστες πηγές.

5. Ανάλυση (*Data Analysis*)

Η ανάλυση των δεδομένων αφορά την εφαρμογή διάφορων εργαλείων και μεθόδων που αναπτύχθηκαν για να ρωτούν (query) τα δεδομένα και να βρίσκουν τις απαντήσεις σε αυτά. Χρησιμοποιείται σε επιχειρήσεις για την λήψη αποφάσεων και στις επιστήμες για την επιβεβαίωση ή την διάψευση θεωριών. Η ανάλυση (*data analysis*) διαφέρει από την εξόρυξη (*data mining*) ως προς τον σκοπό και την εστίαση της ανάλυσης. Η ανάλυση εστιάζει στην παραγωγή του συμπεράσματος βασιζόμενη αποκλειστικά σε ότι είναι ήδη γνωστό, ενώ η εξόρυξη δεδομένων επικεντρώνεται στην ανακάλυψη των προηγουμένως μη-γνωστών ιδιοτήτων στα υπάρχοντα δεδομένα.

Κάποιες από τις προκλήσεις που δημιουργούνται στην ανάλυση μεγάλων των δεδομένων λόγω των έντονων χαρακτηριστικών τους είναι

(i) Συσσώρευση θορύβου, «πλαστές» συσχετίσεις δεδομένων και συμπτωματική ομοιογένεια.
(ii) Μεγάλο υπολογιστικό κόστος και αλγοριθμική αστάθεια.

(iii) Συνήθως τα BD συλλέγονται από πολλαπλές πηγές σε διαφορετική χρονική στιγμή χρησιμοποιώντας διαφορετικές τεχνολογίες. Αυτό δημιουργεί θέματα ετερογένειας δεδομένων και πειραματικές παρεκκλίσεις που απαιτούν την ανάπτυξη πιο προσαρμοστικών και αποτελεσματικών διαδικασιών.

6. Ερμηνεία (*Data Interpretation*)

Η ερμηνεία είναι αρκετά όμοια με τη οπτικοποίηση των δεδομένων. Τα αποτελέσματα της ανάλυσης πρέπει να γίνουν κατανοητά στους χρήστες, υπεύθυνους για την λήψη αποφάσεων. Αυτοί πρέπει να ερμηνεύσουν τα ευρήματα και να εξάγουν από αυτά γνώση και αξία.

Η εκθετική αύξηση και ποικιλία των αδόμητων δεδομένων έχουν επηρεάσει έντονα τον τρόπο που οι άνθρωποι επεξεργάζονται και ερμηνεύουν τα δεδομένα.

1.4 Υπολογιστικό Νέφος

Ένας ενδιαφέρον τεχνολογικός κλάδος που σχετίζεται άμεσα με τα Big Data είναι αυτός του Cloud Computing ή αλλιώς Υπολογιστική Νέφους. Ουσιαστικά αφορά έναν εναλλακτικό τρόπο απόκτησης τεχνολογικών υπηρεσιών, υποδομών και εφαρμογών. Μέσω του Cloud Computing είναι δυνατή η απόκτηση των επιθυμητών υποδομών (π.χ. εξυπηρετητών και λογισμικού) μέσω του Internet ανάλογα με τη χρήση/κίνηση. Με πιο απλά λόγια, αφορά τον διαμοιρασμό υποδομών (υλικών και λογισμικού), ανάλογα με τις ανάγκες του εκάστοτε χρήστη (π.χ. εταιρίας ή ατόμου). Οι υποδομές αυτές είναι απομακρυσμένα εγκατεστημένες και ο πελάτης μπορεί να τις ενοικιάσει ανάλογα με τις ανάγκες του. Με αυτό τον τρόπο εξοικονομεί πόρους, ενώ ταυτόχρονα δεν ασχολείται και με την συντήρηση και απαρχαίωση του εξοπλισμού, που θα έπρεπε σε διαφορετική περίπτωση να αγοράσει, ή την αγορά ακριβού λογισμικού. Ένα ακόμη θετικό στοιχείο είναι η δυνατότητα ανά πάσα στιγμή για άμεση κλιμάκωση των υποδομών, όταν αυξάνονται οι ανάγκες του πελάτη. Ο πελάτης του Cloud Computing μπορεί επομένως να ενοικιάσει υπηρεσίες και υποδομές (εξυπηρετητές, αποθηκευτικό χώρο, κ.ο.κ.), ανάλογα με τις ανάγκες του χωρίς να διαθέτει εξειδικευμένο τμήμα πληροφορικής και χωρίς να απασχολείται με τα σχετιζόμενα θέματα. Ο τρόπος αυτός αποδεικνύεται να είναι οικονομικά αποδοτικός, ειδικά όταν οι απαιτήσεις μεγαλώνουν.

Το Cloud Computing έρχεται να δώσει λύση στις ανάγκες των Big Data, μιας και απαιτούνται σημαντικά αυξημένες δυνατότητες και πόροι (π.χ. υπολογιστική ισχύ, αποθηκευτικό χώρο, κ.ά.). Οι πόροι που προσφέρονται μέσω του Cloud Computing επιτρέπουν την κατανομή του φόρτου εργασίας που απαιτούνται για τα Big Data, με τρόπο

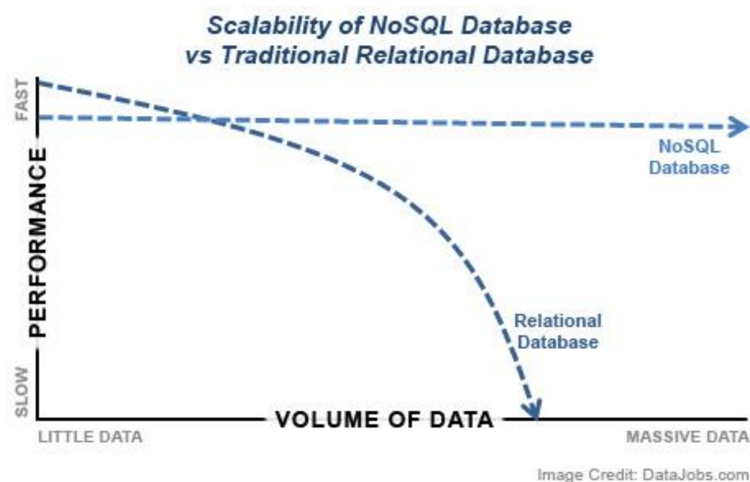
ώστε να είναι δυνατή η σε πραγματικό χρόνο επεξεργασία των τεράστιων όγκων των δεδομένων. Το Cloud Computing επιβάλει τη χρησιμοποίηση νέων μεθόδων εκτέλεσης των εργασιών επεξεργασίας των δεδομένων ή απόκρισης των συστημάτων σε περίπτωση σφαλμάτων, κ.ο.κ., λόγω του διαμοιρασμού των εργασιών που εκτελούνται. Οι τεχνολογίες του νέφους παρέχονται με διάφορους τρόπους και μεθόδους, με τα αναμενόμενα οικονομικά οφέλη για τους πελάτες τους. Τα τρία βασικά μοντέλα του Cloud Computing είναι τα ακόλουθα:

- Λογισμικό ως Υπηρεσία - Software-as-a-Service (SaaS): Οι επιθυμητές εφαρμογές παρέχονται με την μορφή υπηρεσίας και δεν απαιτείται η εγκατάστασή τους στους υπολογιστές του πελάτη. Ουσιαστικά πρόκειται για λογισμικό που χρησιμοποιείται μέσω του Internet και τρέχει σε κάποιο γνωστό φυλλομετρητή (web browser). Η εφαρμογή τρέχει σε υποδομές που ανήκουν στον πάροχο του SaaS και πωλούνται ως επί το πλείστον σε μηνιαία ή ετήσια βάση.
- Πλατφόρμα ως Υπηρεσία - Platform-as-a-Service (PaaS): Αναφέρεται σε μια πλατφόρμα ή περιβάλλον που παρέχεται στον πελάτη ως υπηρεσία προκειμένου να μπορεί να αναπτύξει και να διαχειριστεί δικές του εφαρμογές. Και σε αυτή την περίπτωση, ο πελάτης έχει πρόσβαση στις συγκεκριμένες εφαρμογές μέσω ενός απλού φυλλομετρητή. Η πλατφόρμα αποτελείται από το απαιτούμενο λογισμικό (π.χ. λειτουργικό σύστημα, άλλες ενδιάμεσες εφαρμογές, πρωτόκολλα επικοινωνίας, κ.ά.) που επιτρέπει στις εφαρμογές να τρέχουν στο νέφος. Τα θέματα ασφάλειας, διαχείρισης, κλιμάκωσης των απαιτήσεων, κ.λπ., σε σχέση με το PaaS απασχολούν μόνον τον πάροχο της υπηρεσίας και όχι τον πελάτη, ο οποίος πληρώνει μια συνδρομή ανάλογα με τις απαιτήσεις του.
- Υποδομή ως Υπηρεσία - Infrastructure-as-a-Service (IaaS): Το συγκεκριμένο μοντέλο παρέχει πρόσβαση σε υποδομές σε ένα εικονικό περιβάλλον μέσω του δικτύου (π.χ. μέσω του Internet). Οι υποδομές αφορούν για παράδειγμα, εξυπηρετητές, άλλο υλικό (π.χ. διάθεση bandwidth, IP διευθύνσεων), κ.ά. Οι πελάτες και πάλι πληρώνουν ανάλογα με τη χρήση και τους πόρους που χρησιμοποιούν ή ενοικιάζουν. Ένα σημαντικό πλεονέκτημα είναι πως υπάρχει η δυνατότητα κλιμάκωσης των υποδομών ανά πάσα στιγμή ανάλογα με τις ανάγκες του πελάτη. Οι υλικές υποδομές (π.χ. εξυπηρετητές) μπορεί να βρίσκονται σε διαφορετικά κέντρα, όπου ο πάροχος είναι υπεύθυνος για την συντήρηση και ορθή λειτουργία τους. Με αυτό τον τρόπο δεν απαιτούνται έξοδα συντήρησης και διαχείρισης των υποδομών από τον πελάτη του IaaS.

1.5 Μη-σχεσιακές Βάσεις Δεδομένων

Η προσέγγιση NoSQL (συντομογραφία του not only SQL) αναφέρεται σε μια τάξη συστημάτων διαχείρισης βάσεων δεδομένων τα οποία δεν χρησιμοποιούν την SQL ως γλώσσα ερωτημάτων και δεν ακολουθούν τους κανόνες της σχεσιακής σχεδίασης. Τα συστήματα αυτά είναι μοντελοποιημένα με διαφορετικό τρόπο από τις σχέσεις των πινάκων που χρησιμοποιούνται στις σχεσιακές βάσεις δεδομένων.

Οι NoSQL βάσεις χρησιμοποιούνται σε περιβάλλοντα στα οποία ο όγκος δεδομένων είναι πολύ μεγάλος και παρουσιάζονται προβλήματα απόδοσης λόγω των εγγενών περιορισμών της SQL. Τα εν λόγω συστήματα δεν συμμορφώνονται πιστά στους κανόνες ACID (Atomicity, Consistency, Integrity, Durability) των σχεσιακών βάσεων, έτσι ώστε να ανταποκρίνονται σε συνθήκες στις οποίες η απόδοση ενός ερωτήματος είναι πιο σημαντική από την απόλυτη συνέπεια των δεδομένων. Οι βάσεις NoSQL θυσιάζουν τις αυστηρές απαιτήσεις σε συνέπεια για υψηλές ταχύτητες και ελαστικότητα. Επιπλέον, σε αντίθεση με τις σχεσιακές βάσεις που είναι αυστηρά δομημένες, τα δεδομένα στις NoSQL δεν περιορίζονται εν γένει από κάποιο σχήμα. Ένα από τα σημαντικότερα χαρακτηριστικά τους είναι ότι παρέχουν υψηλή διαθεσιμότητα στα δεδομένα του συστήματος. Η φιλοσοφία των NoSQL εστιάζει στα κατανεμημένα συστήματα βάσεων, όπου αδόμητα δεδομένα αποθηκεύονται σε πολλαπλούς κόμβους. Αυτή η κατανεμημένη αρχιτεκτονική επιτρέπει την οριζόντια κλιμάκωση του συστήματος, δίνοντας την δυνατότητα να προστίθενται συνεχώς νέοι πόροι καθώς τα δεδομένα αυξάνονται, χωρίς επιβάρυνση στην απόδοση. Γίνεται αντιληπτό ότι οι υποδομές των NoSQL βάσεων είναι πολύ καλά προσαρμοσμένες στις μεγάλες ανάγκες των Big Data , και γι' αυτό και αποτελούν λύση στο χειρισμό των μεγαλύτερων data warehouses στο πλανήτη , όπως Google, Amazon, CIA.



Εικόνα 3 NoSQL vs RDBMS scalability

Τα ACID κριτήρια έχουν αντικατασταθεί από τα BASE κριτήρια:

- ***Basic Availability***

  Η βάση δεδομένων είναι διαθέσιμη κατά κύριο λόγο.

- ***Soft-state***

  Τα stores δεν είναι απαραίτητα συνεπή κατά την εγγραφή, ούτε τα διαφορετικά αντίγραφα παρουσιάζουν πάντα αμοιβαία συνέπεια.

- ***Eventual consistency***

  Οι εγγραφές παρουσιάζουν συνοχή σε κάποιο μεταγενέστερο σημείο (π.χ., οκνηρά , κατά την ανάγνωση).

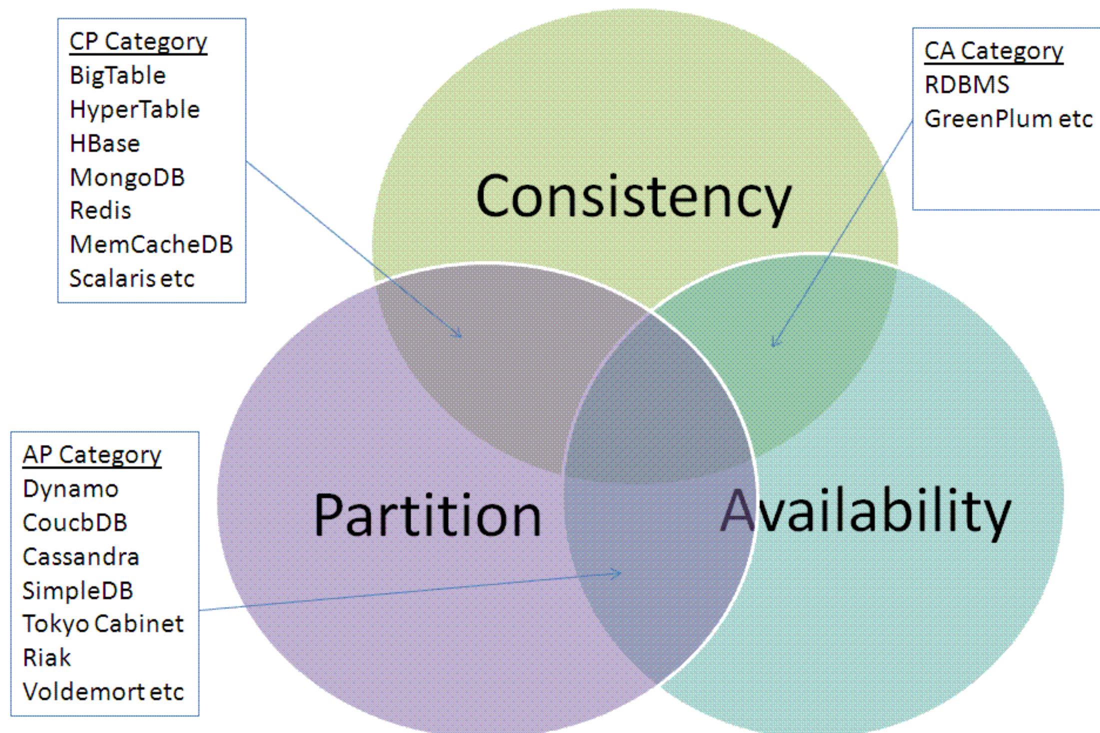
Το μοντέλο BASE εστιάζει περισσότερο στη διαθεσιμότητα, δεδομένου ότι αυτή είναι πιο σημαντική για την κλιμάκωση, και είναι λιγότερο αυστηρό στη διασφάλιση της συνέπεια από ότι το ACID.

Θεώρημα CAP

Σύμφωνα με το θεώρημα CAP (Eric Brewer, University of California, Berkeley) υπάρχουν τρία επιθυμητά χαρακτηριστικά για κάθε σύστημα δεδομένων που έχει αναπτυχθεί σε κατανεμημένο περιβάλλον.

- Συνέπεια (Consistency) : κάθε κόμβος στο σύστημα περιέχει τα ίδια δεδομένα.
- Διαθεσιμότητα (Availability): κάθε αίτηση σε ένα λειτουργικό κόμβο στο σύστημα επιστρέφει μια απάντηση.
- Ανοχή στο διαχωρισμό(Partition tolerance) : οι ιδιότητες του συστήματος (η συνέπεια και / ή η διαθεσιμότητα) έχουν ισχύ ακόμη και ακόμα και αν μεμονωμένα συστατικά του δεν είναι διαθέσιμα.

Το θεώρημα αναφέρει ότι κάθε κατανεμημένο σύστημα μπορεί να υποστηρίξει, το πολύ, δύο από αυτές τις ιδιότητες. Στην παραπάνω εικόνα φαίνονται τα πιο δημοφιλή προϊόντα λογισμικού NoSQL βάσεων τοποθετημένα στις τομές PA(Partition-Availability), CP(Consistency-Partition) και AC(Availability-Consistency).Γεγονός είναι ότι δεν υπάρχει μία και μοναδική επιλογή για την επίλυση προβλημάτων Big Data. Αντί αυτού υπάρχει πληθώρα διαφορετικών μοντέλων βάσεων δεδομένων καθένα από τα οποία είναι πιο ειδικό και πιο κατάλληλο για τη διαχείριση συγκεκριμένων ειδών προβλημάτων.



Εικόνα 4 Cap Theorem

Κατηγορίες Βάσεων NoSQL

- Βάσεις κλειδιού-τιμής (key-value)

  Χρησιμοποιείται ένας πίνακας κατακερματισμού με ένα μοναδικό κλειδί και ένα δείκτη στο αντίστοιχο στοιχείο δεδομένων. Οι βάσεις κλειδί-τιμή δεν απαιτούν σχήμα και προσφέρουν μεγάλη ευελιξία και επεκτασιμότητα ,δεν προσφέρουν τη δυνατότητα ατομικότητας, συνεκτικότητας, απομόνωσης, διατηρησιμότητα (ACID) και απαιτούν κάποια εργαλεία για την τοποθέτηση των δεδομένων, την αποφυγή διπλών αντιγράφων και την ανοχή σε σφάλματα καθώς όλα αυτά δεν ελέγχονται ρητά από την ίδια την τεχνολογία. Οι πιο γνωστές είναι η Memcached, η Dynamo και η Voldemort. Η S3 της Amazon χρησιμοποιεί Dynamo ως μηχανισμό αποθήκευσης ενώ η Riak είναι αρκετά διαδεδομένη key-value NoSQL βάση ελεύθερου λογισμικού και ανεκτική στα σφάλματα.

- Βάσεις αποθήκευσης κατά στήλες (columnar systems)

  Χρησιμοποιούνται για την αποθήκευση και επεξεργασία πολύ μεγάλων ποσοτήτων δεδομένων κατανεμημένων σε πολλά μηχανήματα. Οι σχεσιακές βάσεις είναι προσανατολισμένες κατά γραμμή καθώς τα δεδομένα σε κάθε γραμμή του πίνακα

αποθηκεύονται μαζί. Σε κατά στήλη προσανατολισμένες βάσεις τα δεδομένα αποθηκεύονται κατά μήκος των γραμμών. Είναι πολύ εύκολο να προσθέσουμε στήλες και μπορούν να προστεθούν σειρά-σειρά προσφέροντας μεγαλύτερη ευελιξία, απόδοση και επεκτασιμότητα. Όταν υπάρχει μεγάλος όγκος και ποικιλία δεδομένων είναι μια πολύ καλή λύση. Είναι ευπροσάρμοστη αφού το μόνο που κάνουμε είναι η προσθήκη κι άλλων στηλών. Το πιο σημαντικό παράδειγμα αυτής της κατηγορίας είναι το Big Table της Google όπου οι γραμμές ταυτίζονται με ένα κλειδί με τα δεδομένα να ταξινομούνται και να αποθηκεύονται από αυτό. Το Big Table αποτέλεσε τη βάση για κάποιες άλλες NoSQL βάσεις δεδομένων που βρήκαν απήχηση στη συνέχεια όπως η Cassandra που χρησιμοποιείται από το Facebook και τα HBase και Hypertable.

- Βάσεις εγγράφων (Document Databases)

  Είναι παρόμοιες με τις key-value αλλά βασίζονται σε έγγραφα που αποτελούν συλλογές από άλλες key-value συλλογές, υπάρχει δηλαδή εμφώλευση. Μία βάση εγγράφων απαιτεί από τα δεδομένα να αποθηκεύονται σε μια συγκεκριμένη μορφή, η οποία θα είναι κατανοητή από τη βάση. Η μορφή μπορεί να είναι XML, JSON, Binary JSON (BSON), ή οτιδήποτε άλλο η βάση δεδομένων μπορεί να αντιληφθεί. Τέτοιες βάσεις είναι συνήθως χρήσιμες όταν υπάρχουν αρκετές αναφορές κι αυτές παράγονται και συναρμολογούνται από στοιχεία που αλλάζουν συχνά. Οι πιο διαδεδομένες αυτής της κατηγορίας είναι η MongoDB και η CouchDB.

- Βάσεις γράφων (Graph Database)

  Η βασική δομή αποκαλείται και “σχέση κόμβων”. Η δομή αυτή είναι χρήσιμη όταν έχουμε να κάνουμε με διασυνδεδεμένα δεδομένα. Οι κόμβοι και οι σχέσεις υποστηρίζουν κάποιες ιδιότητες, δηλαδή ένα ζεύγος key-value όπου αποθηκεύονται τα δεδομένα. Η πλοήγηση στη βάση γίνεται ακολουθώντας τις σχέσεις. Αυτού του είδους η αποθήκευση και πλοήγηση στα συστήματα RDBMS δεν είναι δυνατή εξαιτίας της ακαμψίας της δομής των πινάκων και της αδυναμίας να ακολουθηθούν οι συνδέσεις μεταξύ των στοιχείων, όπου κι αν αυτές οδηγούν. Παραδείγματα τέτοιων βάσεων είναι το Neo4J, το Allegro και το Virtuoso.

1.6 MapReduce

Το MapReduce είναι ίσως το σημαντικότερο εργαλείο που έχουμε στη διάθεσή μας για την ανάλυση Big Data. Είναι ένα προγραμματιστικό μοντέλο, μαζί με τη σχετική υλοποίηση για δημιουργία και επεξεργασία πολύ μεγάλων datasets. Αναπτύχθηκε στη Google από τους Jeffrey Dean και Sanjay Ghemawat. Το κίνητρο για την ανάπτυξή του βρισκόταν στο μεγάλο αριθμό υπολογισμών που γίνονταν κάθε μέρα στη Google πάνω σε τεράστιο όγκο εισερχόμενων δεδομένων. Οι υπολογισμοί αυτοί συνήθως ήταν αρκετά ξεκάθαροι εννοιολογικά, όπως πχ. να βρεθεί το query με τις περισσότερες αναζητήσεις για κάποια μέρα ή να κρατιέται αρχείο με τις ιστοσελίδες που επισκέφτηκε ένας χρήστης μέχρι στιγμής. Λόγω του πολύ μεγάλου αριθμού χρηστών όμως ο όγκος των εισερχόμενων δεδομένων επέβαλλε τη χρησιμοποίηση κατανεμημένων συστημάτων με εκατοντάδες ή και χιλιάδες υπολογιστές ώστε να καταστεί δυνατό η επεξεργασία να ολοκληρωθεί σε λογικά χρονικά πλαίσια. Από τη στιγμή όμως που η επεξεργασία των δεδομένων γίνεται κατανεμημένα, έρχονται στο προσκήνιο πολλά άλλα θέματα, όπως το πώς θα γίνει διαμοιρασμός τους, πώς θα παραλληλοποιηθούν οι υπολογισμοί, πώς θα χειριστεί η ανοχή στα σφάλματα και η κατανομή του φόρτου κλπ.

Το MapReduce λοιπόν σχεδιάστηκε για να προσφέρει ένα επίπεδο αφαίρεσης μεταξύ των πραγματικών υπολογισμών πάνω στα δεδομένα που αποτελούν ευθύνη του προγραμματιστή, και των ζητημάτων που προκύπτουν από τη χρήση κατανεμημένων συστημάτων όπως αναφέραμε πιο πάνω. Είναι ένα απλό και συγχρόνως πολύ ισχυρό framework που επιτρέπει στον προγραμματιστή να εκτελεί τις απαιτούμενες εργασίες ως συναρτήσεις map και reduce. Στη συνέχεια, το framework διαμοιράζει τις εργασίες στο διαθέσιμο cluster, αναλαμβάνοντας έτσι τη διαχείριση όλων των ζητημάτων που θίξαμε προηγουμένως.

Για να δημιουργήσει μια MapReduce εργασία, ο προγραμματιστής πρέπει να ορίσει τις συναρτήσεις map και reduce. Η ιδέα αυτή προέρχεται από τις αντίστοιχες συναρτήσεις στις συναρτησιακές γλώσσες προγραμματισμού. Στο cluster τρέχουν πολλαπλά στιγμιότυπα αυτών των συναρτήσεων παράλληλα.

Η συνάρτηση map παίρνει ζευγάρια δεδομένων και επιστρέφει μια λίστα από ζευγάρια δεδομένων. Εφαρμόζεται παράλληλα σε κάθε κομμάτι που έχει διαχωριστεί η είσοδος, παράγοντας έτσι μια λίστα από ζευγάρια σε κάθε κλήση της.

`Map(k1,v1)` → `list(k2,v2)`

Έπειτα το MapReduce συγκεντρώνει όλα τα ζευγάρια από όλες τις λίστες, τις βάζει σε σειρά και τις ομαδοποιεί έτσι ώστε μια ομάδα να περιλαμβάνει όλες τις τιμές εξόδου για ένα συγκεκριμένο κλειδί. Η ομάδα αυτή αποτελεί είσοδο για τη συνάρτηση reduce, που θα τρέξει σε διαφορετικούς υπολογιστές. Δεν υπάρχει περιορισμός που να απαιτεί η έξοδος να έχει ίδιο τύπο με την είσοδο, είτε στο κλειδί, είτε στην τιμή. Η συνάρτηση reduce παίρνει ένα ζευγάρι που αποτελείται από κλειδί και λίστα τιμών και επιστρέψει μια λίστα από τιμές

`Reduce(k2, list (v2))` → `list(v3)`

Τα αποτελέσματα όλων των κλήσεων της reduce ομαδοποιούνται ως το τελικό αποτέλεσμα του MapReduce. Έτσι συνολικά το MapReduce έχει δεχτεί μια λίστα (κλειδί, τιμή) και έχει δώσει ως αποτέλεσμα μια λίστα από τιμές. Και εδώ βλέπουμε ότι οι τιμές εξόδου μπορούν να έχουν διαφορετικό τύπο από τις τιμές εισόδου. Προαιρετικά ο προγραμματιστής χρησιμοποιεί και μια τρίτη συνάρτηση, τη συνάρτηση combine. Η συνάρτηση combine είναι μια συνάρτηση τύπου reduce που τρέχει τοπικά στον υπολογιστή που έτρεξε τη συνάρτηση map. Η κλήση της γίνεται πριν τα αποτελέσματα της map φύγουν για τη reduce μέσω δικτύου, όσο είναι προσωρινά αποθηκευμένα στη μνήμη. Έτσι ένα ποσοστό του Reduce έχει γίνει πριν την μεταφορά δεδομένων στο δίκτυο, έχοντας ως αποτέλεσμα γρηγορότερη επεξεργασία και μείωση φόρτου δικτύου. Το υπολογιστικό μοντέλο που περιγράφηκε αποδεικνύεται ότι είναι Turing Complete. Έτσι οποιοδήποτε πρόβλημα έχει υπολογιστική λύση μπορεί να προσαρμοστεί στο μοντέλο Map και Reduce.

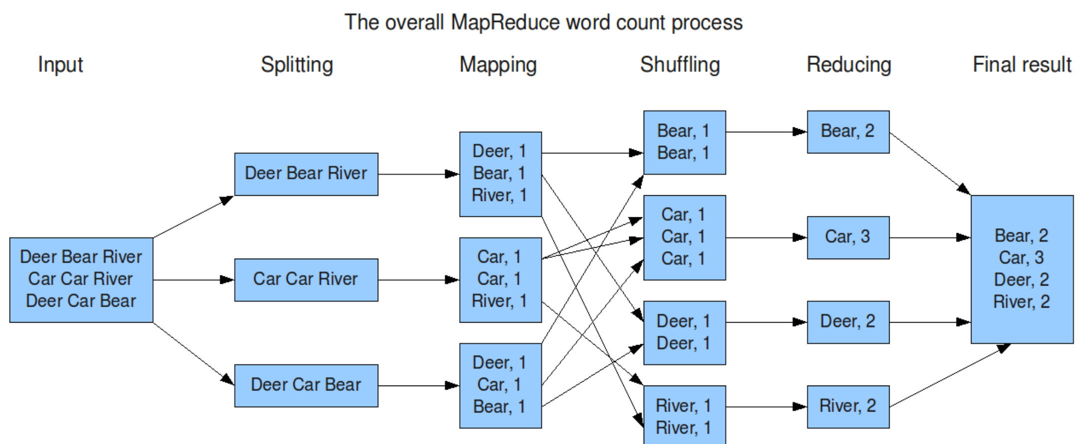
Το παρακάτω είναι το τυπικό παράδειγμα που μετράει την συχνότητα εμφάνισης λέξεων (WordCount) σε διάφορα έγγραφα σε ψευδοκώδικα.

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
```

```
emit (word, sum)
```

Κώδικας 1 Word count example

Το κάθε έγγραφο χωρίζεται σε λέξεις και η κάθε λέξη έχει αρχικά μια συχνότητα ίση με 1. Η ίδια η λέξη είναι το κλειδί με βάση το οποίο θα ομαδοποιηθούν οι συχνότητες. Άρα, η έξοδος του Map είναι της μορφής ((word1, 1), (word2, 1), (word1, 1)), ενώ η είσοδος του Reduce για το κλειδί word1 θα είναι (word1, (1,1)). Η reduce αρκεί να αθροίσει τη λίστα από τους άσσους για να βρει τη συνολική συχνότητα της λέξης. Η χρήση της συνάρτησης reduce ως συνάρτηση combine στον παραπάνω κώδικα είναι πολύ συνηθισμένη. Τα αποτελέσματα της map θα συγκεντρωθούν τότε σε λίστες στην κύρια μνήμη αντί για τον δίσκο. Η έξοδος του κόμβου προς τη reduce θα είναι τότε (word, συχνότητα στο συγκεκριμένο κομμάτι της είσοδο).



Εικόνα 5 Map Reduce example

2

MPEG-7

Τα τελευταία χρόνια, η μεγάλη αύξηση του όγκου πολυμεσικών δεδομένων που καλούμαστε να διαχειριστούμε, συμβαδίζει με μια ανάλογη αύξηση της προσπάθειας για την ανάπτυξη τέτοιων τεχνολογιών. Ένα από τα κυριότερα σημεία που χρήζουν προτυποποίησης είναι η εξαγωγή χρήσιμης πληροφορίας από τα αρχεία πολυμέσων. Είναι πολύ σημαντικό να υπάρχει ένα κοινό πρωτόκολλο για τον ορισμό και τη χρησιμοποίηση πολυμεσικών δεδομένων, ώστε να διευκολύνεται η διαλειτουργικότητα, η κλιμακωσιμότητα και η ανεξαρτησία από την πλατφόρμα στην οποία τρέχει μια πολυμεσική εφαρμογή.

Αυτός είναι και ο κύριος στόχος του προτύπου MPEG-7. Το MPEG-7 διαφέρει από τα προγενέστερα πρότυπα (MPEG-1, MPEG-2 και MPEG-4) ως προς το ότι δεν είναι ένα πρότυπο κωδικοποίησης αρχείων βίντεο, και δεν προτίθεται να αντικαταστήσει κάποιο από αυτά. Η αποστολή του είναι να παρέχει τις κατάλληλες περιγραφές για τη μετάδοση και αποθήκευση πολυμεσικού περιεχομένου όπως εικόνες, ήχος και βίντεο. Αυτοί οι περιγραφείς μπορούν να χρησιμοποιηθούν για να συγκρίνουν, να φιλτράρουν ή να αναζητήσουν πολυμέσα αποκλειστικά με βάση το οπτικό περιεχόμενο του αρχείου. Για το λόγο αυτό, οι περιγραφείς μπορεί να περιέχουν πληροφορίες σχετικά με:

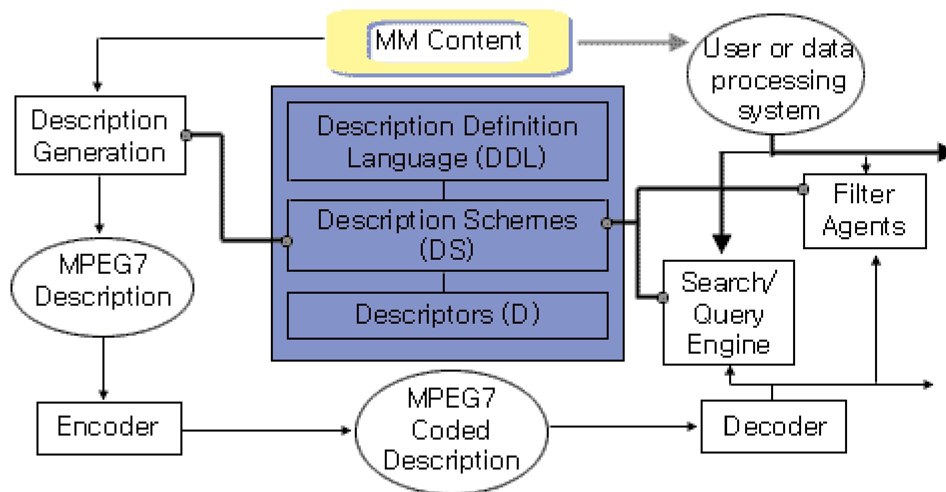
- Τα χαρακτηριστικά της δομής του περιεχομένου (μορφή αποθήκευσης, κωδικοποίηση κλπ.)
- Low-level χαρακτηριστικά όπως χρώμα, επιφάνεια, ήχος, μουσική κ.α.

- Χωρικά, χρονικά ή και χωροχρονικά στοιχεία του περιεχομένου όπως τμηματοποίηση μιας σκηνής σε περιοχές, εντοπισμός κίνησης μιας περιοχής κ.ά.
- Εννοιολογικές πληροφορίες σχετικά με το περιβάλλον που απεικονίζεται από το περιεχόμενο, όπως τον ορισμό αντικειμένων και γεγονότων, αλληλεπίδραση μεταξύ των αντικειμένων κ.ά.
- Πληροφορίες σχετικά με τη χρήση του περιεχομένου όπως πνευματικά δικαιώματα, ιστορικό χρήσης κλπ.
- Πληροφορίες που σχετίζονται με τη δημιουργία του περιεχομένου και τη διαδικασία παραγωγής του, όπως το όνομα του δημιουργού, τον τίτλο, ημερομηνία κλπ.

Οι περιγραφείς εφαρμόζονται στα δεδομένα ανεξάρτητα από το μέσο μετάδοσης και τον τρόπο κωδικοποίησης, η αναπαράστασή τους γίνεται με αντικειμενοστραφή τρόπο και μπορούν να επεκταθούν ανάλογα με τις ανάγκες τους χρήστη. Ορίζονται 4 εργαλεία, description tools, για την περιγραφή του περιεχομένου των αρχείων, οι Descriptors, τα Description Schemas, η Description Definition Language (DDL) και τα System tools.

- Descriptors (D): Ένας περιγραφές (Descriptor) αναπαριστά ένα στοιχειώδες χαρακτηριστικό και ορίζει τη σύνταξη και τη σημασιολογία μιας τέτοιας αναπαράστασης. Πιθανοί περιγραφείς είναι: ένα ιστόγραμμα χρώματος, οι timecodes για την αναπαράσταση χρονικής διάρκειας, ένα πεδίο κίνησης, το κείμενο ενός τίτλου κλπ.
- Description Schemes (DS): Ένα σχήμα περιγραφής (Description Schema), ορίζει τη δομή και τη σημασιολογία των συσχετίσεων μεταξύ των συστατικών τους, που μπορεί να είναι είτε descriptors είτε description schemes. Η δομή και η σύνταξη των Description Schemes ορίζεται από την DDL (Description Definition Language), η οποία σε αρκετές περιπτώσεις δίνει τη δυνατότητα δημιουργίας νέων σχημάτων περιγραφής.
- Description Definition Language (DDL) : Είναι μια γλώσσα βασισμένη στην XML και χρησιμοποιείται για να ορίζει δομικές σχέσεις μεταξύ των περιγραφέων. Επιτρέπει τη δημιουργία και την τροποποίηση των description schemes αλλά και τη δημιουργία νέων descriptors.
- System Tools: τα εργαλεία αυτά υποστηρίζουν το multiplexing των περιγραφών, το συγχρονισμό των περιγραφών με το συσχετιζόμενο περιεχόμενο, τη δυαδική αναπαράσταση για αποδοτική αποθήκευση και μετάδοση κλπ.

Ένα τυπικό σενάριο εφαρμογής υποδεικνύει πως οι MPEG-7 περιγραφείς εξάγονται από το περιεχόμενο του αρχείου. Στις περισσότερες περιπτώσεις, το πρότυπο MPEG-7 ορίζει μόνο εν μέρει πώς να γίνει η εξαγωγή αυτών των χαρακτηριστικών, ώστε να επιτρέπει όσο το δυνατόν μεγαλύτερη ευελιξία στις διάφορες εφαρμογές. Επιπλέον, αφήνεται ελεύθερο το πώς θα χρησιμοποιηθούν οι MPEG-7 περιγραφείς για επιπλέον επεξεργασία του περιεχομένου. Ωστόσο, ως διεθνές πρότυπο, η σύνταξη των περιγραφέων θα πρέπει να ικανοποιεί τις προδιαγραφές του MPEG-7. Αυτό εξασφαλίζει πως όλες οι εφαρμογές που είναι συμβατές με το MPEG-7 θα μπορούν ανταλλάσσουν δεδομένα.



Εικόνα 6 Mpeg-7 applications

3

Περιγραφή Προβλήματος

Όπως τονίσαμε σε προηγούμενα κεφάλαια, η αλματώδης αύξηση του όγκου των δεδομένων που καλούμαστε να διαχειριστούμε έχει οδηγήσει στην ανάγκη για εύρεση νέων, πιο αποδοτικών τεχνικών αποθήκευσης. Ειδικότερα στα Multimedia δεδομένα, όπως είναι τα video, οι εικόνες και οι ήχοι, το πρόβλημα αυτό εμφανίζεται με πολύ μεγάλη συχνότητα, καθώς το πολύ μεγάλο εύρος πληροφορίας που διατηρούν συνεπάγεται αντίστοιχες απαιτήσεις σε αποθηκευτικό χώρο και δυσχεραίνει την αναζήτηση.

Data Set

Στη διπλωματική αυτή μελετάται το πρόβλημα αποθήκευσης και αναζήτησης multimedia αρχείων και ειδικότερα 3D σκηνών. Αναλυτικότερα, τα δεδομένα που χρησιμοποιήσαμε στην παρούσα διπλωματική περιέχουν 3D (τρισδιάστατα) στρατιωτικά μοντέλα, όπως πολεμικά αεροσκάφη, πλοία, οχήματα, εξοπλισμός, στρατιώτες, σενάρια μάχης κλπ. Διατίθενται από την ερευνητική ομάδα SAVAGE (Scenario Authoring and Visualization for Advanced Graphical Environments) στον παρακάτω σύνδεσμο https://savage.nps.edu/Savage/. Τα δεδομένα αυτά είναι αποθηκευμένα με τη μορφή x3d αρχείων. Το x3d (eXtensible 3d) είναι η πρότυπη μορφή αρχείου για την αναπαράσταση τρισδιάστατων γραφικών. Επιτρέπει την κωδικοποίηση μιας σκηνής χρησιμοποιώντας XML σύνταξη, ή με δυαδική αναπαράσταση (binary formatting).

Για την εξαγωγή χρήσιμης και εύκολα διαχειρίσιμης πληροφορίας από x3d αρχεία είναι προτιμότερο να εξάγουμε την .mp7 (από το πρότυπο MPEG-7) περιγραφή τους. Χρησιμοποιήσαμε, λοιπόν, ένα software tool και εξάγαμε τα αντίστοιχα mp7 αρχεία από το αρχικό data set, σύμφωνα με μια επέκταση του προτύπου αυτού που δημιουργήθηκε στα πλαίσια του έργου iPromotion.

Αποθήκευση και αναζήτηση

Το κυρίως πρόβλημα που κληθήκαμε να αντιμετωπίσουμε ωστόσο ήταν η εύρεση ενός αποτελεσματικού τρόπου για την αποθήκευση αυτών των δεδομένων ώστε να μπορούμε να τα διατρέχουμε και να τα αναζητούμε γρήγορα. Παραδοσιακές τεχνικές αποθήκευσης των metadata πολυμεσικών αρχείων είναι

i. η αποθήκευση των XML αρχείων, είτε τοπικά στο file system είτε ως πεδίο text σε κάποια βάση δεδομένων,
ii. η μετατροπή τους σε πίνακες σε κάποιο RDBMS.

Οποιαδήποτε τεχνική περιλαμβάνει αποθήκευση σε σχεσιακή βάση ή τοπικό file system αντιμετωπίζει πρόβλημα στην κλιμάκωση των δεδομένων, στην αποθήκευση τους σε ένα κατανεμημένο cluster και στη διαχείρισή του. Στην συνέχεια αναλύονται προβλήματα που δημιουργούνται ανά περίπτωση.

Όσον αφορά την αναζήτηση σε XML αρχεία, αυτή γίνεται με χρήση του XQuery, μιας γλώσσας με σκοπό την υποβολή queries σε συλλογές XML αρχείων. Ωστόσο, το μεγάλο overhead που αναγκαστικά επιφέρει το xml parsing αλλά και η αργή διάσχιση του XML δέντρου , επιφέρουν μεγάλη αύξηση του χρόνου εκτέλεσης ενός query. Επιπλέον, με αυτόν τον τρόπο δεν υπάρχει η δυνατότητα δημιουργίας ευρετηρίων σε συγκεκριμένα πεδία των metadata, για την βελτιστοποίηση αντίστοιχων ερωτημάτων. Ως πεδία αναφέρουμε συγκεκριμένα μονοπάτια στην δενδρική μορφή του XML αρχείου των metadata.

Όσον αφορά την αναζήτηση σε RDBMS τώρα, γνωρίζουμε ήδη πως τα τυπικά σχεσιακά συστήματα βάσεων δεδομένων δυσκολεύονται να ανταποκριθούν στις τεράστιες αποθηκευτικές απαιτήσεις των Big Data. Ακόμη, η χρήση του σχεσιακού μοντέλου απαιτεί την ύπαρξη ενός αυστηρού schema για τα δεδομένα. Αυτό αποτελεί μεγάλο μειονέκτημα όταν πρόκειται να αποθηκευτούν MPEG-7 περιγραφές, όπου αν και υπάρχει η δόμηση σύμφωνα με τους κανόνες που ορίζει το πρότυπο, παρατηρούνται πολλές ελευθερίες στον τρόπο αναπαράστασης της πληροφορίας. Όμως οι σχεσιακές βάσεις δεδομένων δεν ενδείκνυνται για την αποθήκευση δεδομένων χωρίς αυστηρό schema. Στην προκειμένη

περίπτωση δηλαδή θα έπρεπε να κάνουμε χρήση πολύ μεγάλου αριθμού από null values για κάθε αρχείο ώστε να καλύψουμε όλους τους πιθανούς descriptors των MPEG-7. Αυτό φυσικά θα οδηγούσε σε πολλαπλάσιο τελικό μέγεθος της βάσης και επομένως πολύ χαμηλότερη απόδοση σε αποθηκευτικό χώρο αλλά και ταχύτητα εκτέλεσης των queries. Ακόμα, το query model των RDBMS θα απέδιδε πολύ χειρότερα από αυτό των NoSQL βάσεων, λόγω των πολλών πολύπλοκων joins των οποίων η εφαρμογή δεν είναι καθόλου αποδοτική σε μεγάλα data sets. Αυτό φυσικά κάνει απαγορευτική τη χρήση της μεθόδου αυτής σε περιβάλλοντα Big Data.

Για τους λόγους αυτούς λοιπόν επιλέξαμε να εξετάσουμε λύσεις στον τομέα των NoSQL βάσεων δεδομένων. Πιο συγκεκριμένα, χρησιμοποιήσαμε δύο διαφορετικές βάσεις, MongoDB και Cassandra, για να αποθηκεύσουμε τα δεδομένα μας με σκοπό την γρήγορη ανάκτησή τους και την αναζήτηση βάσει περιεχομένου. Η ανάκτηση όλων των metadata μια σκηνής σε μια σχεσιακή βάση, θα απαιτούσε joins σε όλα τα διαφορετικά tables που συνθέτουν την δενδρική μορφή του MPEG-7 αρχείου. Στις βάσεις αυτές η ανάκτηση γίνεται πιο αποδοτικά αφού όλες οι πληροφορίες ανά σκηνή βρίσκονται συγκεντρωμένες στο ίδιο document στην MongoDB(document-oriented) ή στο ίδιο partition στην Cassandra. Όμως, για την βελτιστοποίηση της απόδοσης σε άλλου τύπου ερωτήματα, χρησιμοποιήσαμε άλλες τεχνικές όπως ευρετηριοποίηση (indexing) και MapReduce.

Βελτιστοποίηση ερωτημάτων

Για την βέλτιστη αναζήτηση και τη σωστή ταξινόμηση της πληροφορίας, έτσι ώστε να μπορεί να γίνει ανάκτησή της όταν χρειαστεί θα πρέπει να υπάρξει κατάλληλη οργάνωση αλλά και ιεράρχηση. Για να επιτευχθούν τα παραπάνω χρησιμοποιούμε ευρετήρια ως μια αποτελεσματική τεχνική γρήγορης ανάκτησης πληροφορίας. Προς αυτήν την κατεύθυνση, θα δημιουργήσουμε δομές απαραίτητες για την βελτιστοποίηση ερωτημάτων, που αφορούν αναζήτηση βάσει περιεχομένου, όπως η εύρεση σκηνών που πληρούν ορισμένα κριτήρια. Επιπρόσθετα, θα εξετάσουμε την υποβολή δύσκολων ερωτημάτων, που αφορούν την ανάλυση του συνόλου των δεδομένων, μέσω του προγραμματιστικού μοντέλου MapReduce.

Τέλος, μέσα από την υλοποίηση του προβλήματος με χρήση και των δύο βάσεων, θα προσπαθήσουμε να εξάγουμε συγκριτικά συμπεράσματα που αφορούν τις λειτουργίες και τα χαρακτηριστικά των δύο βάσεων.

4

Technical Components

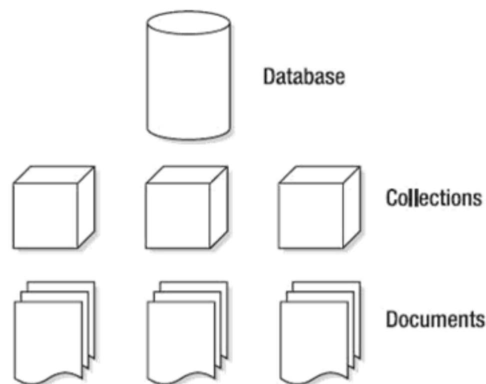
4.1 MongoDB

Η MongoDB (από το ‘‘humongous’’) είναι ένα διαπλατφορμικό σύστημα βάσης δεδομένων που βασίζεται σε έγγραφα (cross-platform document-oriented database system) και κατηγοριοποιείται ως NoSQL βάση δεδομένων. Η MongoDB δεν έχει την παραδοσιακή δομή μίας σχεσιακής βάσης δεδομένων με πίνακες, αλλά χρησιμοποιεί JSON-like έγγραφα με δυναμικά schemas, καθιστώντας την ενσωμάτωση των δεδομένων σε ορισμένους τύπους εφαρμογών ευκολότερη και ταχύτερη. Είναι γραμμένη σε C++ και είναι σχεδιασμένη να προσφέρει υψηλή απόδοση στις εφαρμογές, επεκτασιμότητα, υψηλή διαθεσιμότητα και δυνατότητα υποβολής σύνθετων ερωτημάτων. Για τα δεδομένα της εγγυάται τελική συνέπεια (eventual consistency).

Αρχικά, αναπτύχθηκε από την εταιρεία 10gen(πλέον ονομάζεται MongoDB Inc.) το 2007 ως συστατικό μιας πλατφόρμας PaaS. Στη συνέχεια, η εταιρεία στράφηκε σε ένα μοντέλο ανάπτυξης ανοιχτού κώδικα το 2009, με την 10gen να προσφέρει εμπορική υποστήριξη και άλλες υπηρεσίες. Από τότε, η MongoDB έχει υιοθετηθεί ως λογισμικό backend από μια σειρά σημαντικών ιστοσελίδων και υπηρεσιών, όπως οι Craigslist, eBay, Foursquare, SourceForge και The New York Times, μεταξύ άλλων. Η MongoDB είναι ίσως το πιο δημοφιλές NoSQL σύστημα βάσεων δεδομένων.

4.1.1 Μοντέλο Δεδομένων

Η MongoDB φιλοξενεί έναν αριθμό βάσεων δεδομένων. Όπως φαίνεται και στην παρακάτω εικόνα, κάθε βάση δεδομένων από αυτές περιέχει ένα σύνολο από συλλογές (collections) και κάθε συλλογή περιέχει ένα σύνολο από έγγραφα.



Εικόνα 7 Η δομή της MongoDB.

Κάθε έγγραφο είναι ένα σύνολο από ζεύγη κλειδιού-τιμής και έχουν δυναμικό σχήμα (dynamic schema). Αυτό σημαίνει ότι τα έγγραφα της ίδιας συλλογής δε χρειάζεται να έχουν το ίδιο σύνολο πεδίων ή την ίδια δομή, ενώ ακόμη, τα κοινά πεδία μίας συλλογής εγγράφων μπορούν να περιέχουν διαφορετικούς τύπους δεδομένων. Η MongoDB αποθηκεύει τα έγγραφα στο δίσκο σε φόρμα σειριοποίησης BSON. Το BSON είναι μία δυαδική αναπαράσταση των JSON εγγράφων, αλλά περιέχει πολλούς περισσότερους τύπους δεδομένων από το JSON. Το κέλυφος JavaScript mongo και οι οδηγοί γλώσσας MongoDB (MongoDB language drivers) κάνουν τη μετάφραση μεταξύ BSON και της αναπαράστασης των εγγράφων με βάση τη γλώσσα προγραμματισμού. Τα έγγραφα έχουν μέγιστο μέγεθος 16MB, ενώ για την αποθήκευση μεγαλύτερων δεδομένων σε μέγεθος, όπως εικόνες, χρησιμοποιείται το GridFS. Η τιμή ενός πεδίου μπορεί να είναι οποιουδήποτε τύπου δεδομένων BSON, συμπεριλαμβανομένων άλλων εγγράφων, πινάκων και πινάκων από έγγραφα. Για παράδειγμα, το έγγραφο παρακάτω περιέχει τιμές από ποικίλους τύπους.

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
```

```
                death: new Date('Jun 07, 1954'),
                contribs: [ "Turing machine", "Turing test",
"Turingery" ],
                views : NumberLong(1250000)
            }
```

Κάθε έγγραφο υποχρεωτικά πρέπει να διαθέτει ένα πεδίο όπου αποθηκεύεται το μοναδικό αναγνωριστικό _id του εγγράφου που χρησιμοποιείται για τον εντοπισμό του. Δημιουργείται αυτόματα ένα μοναδικό ευρετήριο για το πεδίο _id κατά τη δημιουργία της συλλογής. Επίσης είναι πάντα το πρώτο πεδίο στα έγγραφα και μπορεί να περιέχει κάθε τύπο BSON πλην των πινάκων.

Η MongoDB παρέχει αρκετές ελευθερίες στον τρόπο αποθήκευσης των δεδομένων. Εφόσον η βάση δομείται με schema-free τρόπο, δεν χρειάζεται τα έγγραφα μίας συλλογής να έχουν τα ίδια πεδία, την ίδια δομή και τους ίδιους τύπους δεδομένων και άρα μπορεί το καθένα να διαθέτει δικό του schema. Για την αποτελεσματικότερη εκτέλεση των εφαρμογών και για την επιλογή του κατάλληλου schema από τους σχεδιαστές πρέπει να λαμβάνονται υπόψη τα ερωτήματα που θα υποβάλλονται στη βάση, η συχνότητα των updates και ο ρυθμός αύξησης του μεγέθους των εγγράφων. Βλέπουμε πως η μοντελοποίηση πρέπει να γίνεται με βάση τη χρήση και το περιεχόμενο των βάσεων κι όχι με βάση τον τρόπο αποθήκευσής τους, όπως γίνεται στις σχεσιακές βάσεις δεδομένων όπου η εκτέλεση των ερωτημάτων βασίζεται στις συνενώσεις.

4.1.2 Μοντέλο Ερωτημάτων

Η MongoDB χρησιμοποιεί JSON-like αρχεία για να προσδιορίζει τις διάφορες παραμέτρους κατά την υποβολή ενός ερωτήματος, τα οποία στέλνονται στη βάση σαν BSON αντικείμενα μέσω του interactive shell ή του driver που χρησιμοποιεί η εφαρμογή. Τα ερωτήματα υποβάλλονται σε όλα τα έγγραφα μίας συλλογής κάθε φορά και συμπεριλαμβάνονται όλα τα ενσωματωμένα αντικείμενα και οι πίνακες που περιέχουν. Για την υποβολή ερωτημάτων σε έγγραφα περισσότερων συλλογών, θα πρέπει το ερώτημα να εφαρμόζεται σε όλες τις συλλογές που διαθέτουν τα ζητούμενα έγγραφα.

Η μέθοδος *db.collection.find()* χρησιμοποιείται για την ανάκτηση εγγράφων από μία συλλογή. Επιστρέφει όλα τα έγγραφα που ικανοποιούν τις συνθήκες που περιγράφονται από την 1η παράμετρο (query). Αν αυτή παραληφθεί επιστρέφονται όλα τα έγγραφα. Η 2η παράμετρος (projection) προσδιορίζει τα πεδία που θα επιστραφούν στα έγγραφα που ικανοποιούν το query. Αν παραληφθεί επιστρέφονται όλα τα πεδία. Πιο συγκεκριμένα, η εντολή find() δε γυρίζει τα έγγραφα που πληρούσαν το ερώτημά μας, αλλά έναν κέρσορα (cursor), ο οποίος δείχνει σε αυτά. Μπορούμε να εφαρμόσουμε τροποποιήσεις στον κέρσορα, ούτως ώστε να επιβάλλουμε όρια, παραβλέψεις και σειρά ταξινόμησης

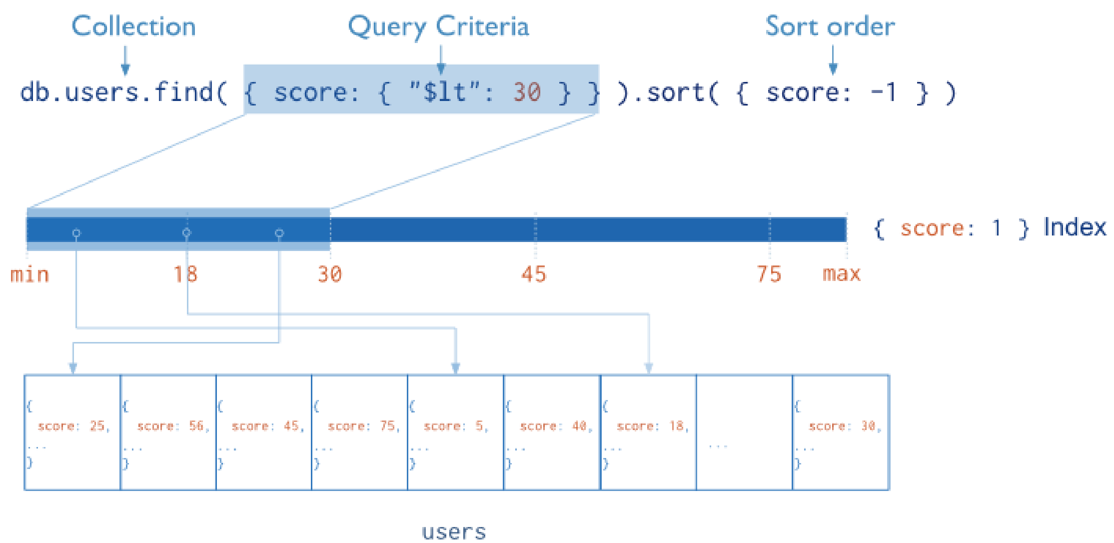
Το μοντέλο ερωτημάτων της mongoDB υποστηρίζει μεταξύ άλλων τα ακόλουθα χαρακτηριστικά:

- Υποβολή ερωτημάτων σε έγγραφα και σε ενσωματωμένα αντικείμενα.
- Υποβολή geospatial ερωτημάτων.
- Χρήση τελεστών σύγκρισης (, ≤, ≥, =).
- Χρήση λογικών τελεστών (and, or, not, nor).
- Χρήση τελεστών ικανοποίησης συνθηκών (equals, exists, in, mod, type, …).
- Χρήση συναρτήσεων ομαδοποίησης (count, sum, …).

4.1.3 Ευρετήρια

Με τη χρήση ευρετηρίων (indexes), τα ερωτήματα που θέτουμε στην MongoDB μπορούν να εκτελεστούν πολύ αποδοτικά. Όταν δεν υπάρχει κάποιο ευρετήριο που μπορεί να χρησιμοποιηθεί για το ερώτημα μας, τότε η MongoDB είναι αναγκασμένη να διατρέξει όλα τα έγγραφα της συλλογής, ώστε να επιστρέψει εκείνα τα έγγραφα που ικανοποιούν το περιορισμό που θέσαμε. Αντιθέτως, εάν υπάρχει κατάλληλο ευρετήριο που μπορεί να χρησιμοποιηθεί για ερώτημα μας, τότε μειώνεται δραματικά ο αριθμός των εγγράφων που θα επισκεφτεί η MongoDB ώστε να μας απαντήσει.

Τα ευρετήρια είναι ειδικές δομές δεδομένων (χρησιμοποιούν B-trees) που αποθηκεύουν ένα μικρό μέρος των δεδομένων μιας συλλογής με τέτοιο τρόπο ώστε να είναι πολύ εύκολο και γρήγορο να διασχιστούν. Το ευρετήριο αποθηκεύει τις τιμές ενός συγκεκριμένου πεδίου(ή συνόλου πεδίων) για όλα τα έγγραφα μιας συλλογής, ταξινομημένες με βάση την τιμή του πεδίου. Η ταξινόμηση αυτή είναι πολύ σημαντική, γιατί υποστηρίζει την αποδοτική εκτέλεση των ερωτημάτων ισότητας (equality queries) και των ερωτημάτων εύρους (range based queries). Τέλος, η MongoDB έχει τη δυνατότητα να επιστρέψει τα αποτελέσματα ταξινομημένα, απλά χρησιμοποιώντας τη σειρά με την οποία τα βρίσκει στο ευρετήριο.



Εικόνα 8 Παράδειγμα ερωτήματος που χρησιμοποιεί ευρετήριο

Query Plans

Η MongoDB χρησιμοποιεί έναν ειδικό μηχανισμό, ο οποίος ονομάζεται query optimizer (βελτιστοποιητής ερωτημάτων), για να επιλέξει το πιο αποτελεσματικό πλάνο (query plan) που μπορεί να ακολουθήσει για να απαντήσει σε ένα ερώτημα, με βάση τα διαθέσιμα ευρετήρια. Αποθηκεύει έπειτα το πλάνο στη cache για να το χρησιμοποιήσει κάθε φορά που τρέχει το συγκεκριμένο ερώτημα. Ο query optimizer αποθηκεύει τα query plans μόνο για τα ερωτήματα που έχουν παραπάνω από ένα δυνατό πλάνο.

Για κάθε ερώτημα, αναζητά στην cache των query plans για την εγγραφή που ταιριάζει με το ερώτημα. Αν δεν γίνει ‘match’ κάποια εγγραφή εκεί, δημιουργεί υποψήφια πλάνα προς εκτίμηση. Επιλέγει το νικητήριο πλάνο, φτιάχνει εγγραφή στη cache με το νικητήριο πλάνο και το χρησιμοποιεί για να πάρει τα αποτελέσματα. Αν υπάρχει αντίστοιχη εγγραφή στην cache, τότε δημιουργεί ένα πλάνο βάσει της εγγραφής αυτής και επανεκτιμά την απόδοση του

μέσω ενός μηχανισμού (replanning mechanism). Αυτός ο μηχανισμός θα πάρει την απόφαση για το αν θα κρατήσει το query plan και θα το χρησιμοποιήσει για τα αποτελέσματα ή αν θα αφαιρέσει την εγγραφή από την cache και θα ακολουθήσει την παραπάνω διαδικασία για την επιλογή νικητήριου πλάνου και την καταχώρησή του στη cache.

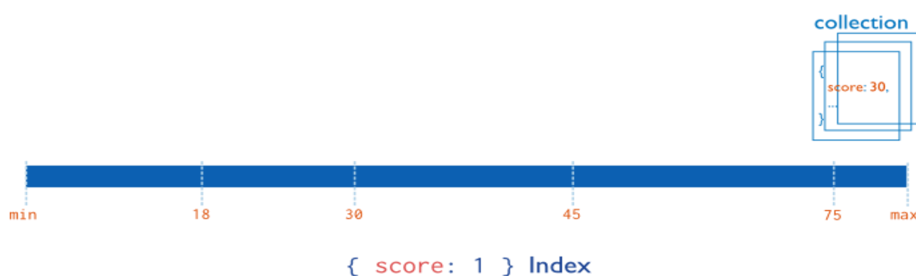
Επιπλέον, η MongoDB επιτρέπει την διασταύρωση(intersection) ευρετηρίων για την εκπλήρωση ερωτημάτων. Για ερωτήματα με σύνθετες συνθήκες(παραπάνω από ένα πεδίο), αν ένα ευρετήριο ικανοποιεί ένα μέρος των συνθηκών του query , και ένα άλλο ευρετήριο ικανοποιεί ένα άλλο μέρος των συνθηκών, τότε ο query optimizer μπορεί να χρησιμοποιήσει την διασταύρωση των δύο ευρετηρίων.

Τύποι ευρετηρίων

Η MongoDB προσφέρει μια πληθώρα διαφορετικών τύπων ευρετηρίων που υποστηρίζουν διάφορους τύπους δεδομένων και ερωτημάτων.

- Ευρετήριο μονού πεδίου (Single field index):

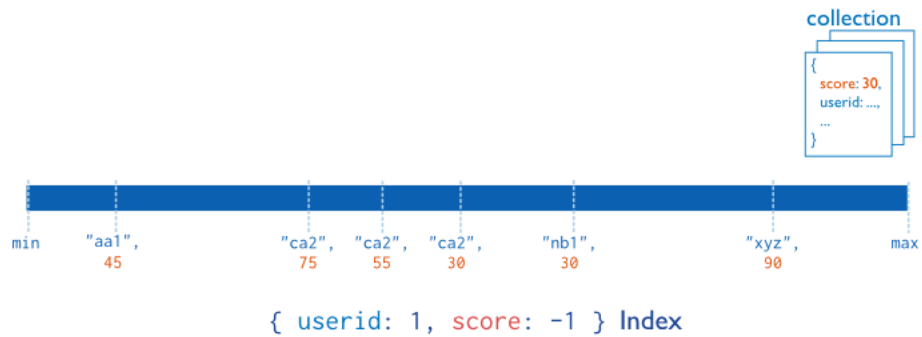
  Αυτό είναι το ευρετήριο που αφορά τις τιμές ενός και μόνο πεδίου των εγγράφων μιας συλλογής. Η σειρά ταξινόμησης δηλώνεται κάθε φορά κατά τη δημιουργία του ευρετηρίου απλά γράφοντας δίπλα από το όνομα του πεδίου τον αριθμό 1 ή -1 για αποθήκευση σε αύξουσα ή φθίνουσα σειρά αντίστοιχα.



Εικόνα 9 Ευρετήριο μονού πεδίου, αύξουσα σειρά

- Σύνθετο ευρετήριο (Compound index):

  Είναι η περίπτωση στην οποία μία και μοναδική δομή ευρετηρίου κρατά αναφορές για περισσότερα από ένα πεδία μιας συλλογής εγγράφων (το πολύ μέχρι 31 πεδία). Ένα παράδειγμα φαίνεται στην παρακάτω εικόνα.



Εικόνα 10 Σύνθετο ευρετήριο με δύο πεδία

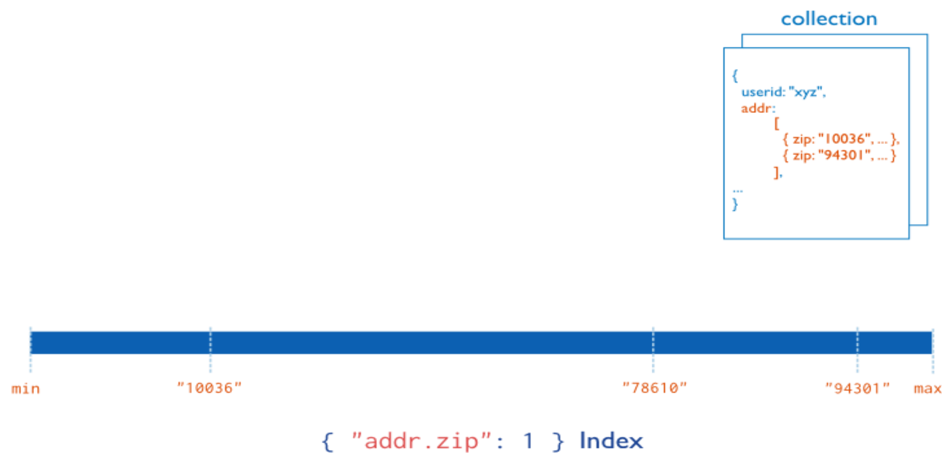
Μία σημαντική δυνατότητα που έχουν τα σύνθετα ευρετήρια είναι ότι η MongoDB μπορεί να τα χρησιμοποιήσει και σε περίπτωση που το ερώτημα δεν αφορά όλα τα πεδία του ευρετηρίου αλλά κάποιο από τα προθέματα του ευρετηρίου (index prefixes). Για παράδειγμα, το σύνθετο ευρετήριο

{ "item": 1, "location": 1, "stock": 1 } έχει τα εξής προθέματα

{ item: 1 } , { item: 1, location: 1 }.

- Ευρετήριο πολλαπλών κλειδιών(multikey index):

  Τα χρησιμοποιούμε όταν το πεδίο πάνω στο οποίο θέλουμε να φτιάξουμε ευρετήριο έχει για τιμή έναν πίνακα. Σε αυτή την περίπτωση, για κάθε μία από τις τιμές του πίνακα, η MongoDB δημιουργεί μια ξεχωριστή καταχώρηση στο ευρετήριο. Τα ευρετήρια αυτά μπορούν να κατασκευαστούν είτε όταν ο πίνακας περιέχει αλφαριθμητικές τιμές, είτε ακόμη και όταν περιέχει εμφωλευμένα έγγραφα (nested documents).



Εικόνα 11 Ευρετήριο πολλαπλών κλειδιών σε πίνακα με εμφολευμένα έγγραφα

- Επιπλέον τύποι ευρετηρίων που είναι διαθέσιμοι : Text, Geospatial, Hashed.

Ιδιότητες Ευρετηρίων

- Μοναδικά ευρετήρια (Unique Indexes)

  Η ιδιότητα αυτή υποχρεώνει την MongoDB να απορρίψει όλα τα έγγραφα που δεν έχουν μοναδική τιμή στο πεδίο για το οποίο έχει δημιουργηθεί το ευρετήριο.

- Μερικά ευρετήρια(Partial Indexes)

  Τα ευρετήρια με την ιδιότητα αυτή καταχωρούν μόνο τα έγγραφα μιας συλλογής , τα οποία ικανοποιούν ορισμένες συνθήκες.

- Αραιά ευρετήρια(Sparse Indexes)

  Τα ευρετήρια αυτά κάνουν καταχωρήσεις μόνο για έγγραφα που περιέχουν το πεδίο στο οποίο δημιουργήθηκε το ευρετήριο ακόμα και αν αυτό έχει τιμή null. Ένα μη-αραιό ευρετήριο περιέχει όλα τα έγγραφα μίας συλλογής αποθηκεύοντας την τιμή null για τα έγγραφα τα οποία δεν περιέχουν το πεδίο.

- TTL ευρετήρια(TimeToLive)

  Τα ευρετήρια αυτά είναι ειδικά single filed ευρετήρια, τα οποία μπορεί να χρησιμοποιήσει η MongoDB ώστε να σβήσει αυτόματα κάποια έγγραφα από μία συλλογή μετά από ένα προκαθορισμένο χρονικό διάστημα

4.1.4 Αρχιτεκτονική

Με τη MongoDB είναι δυνατή η ανάπτυξη κατανεμημένων εφαρμογών που τρέχουν σε clusters υπολογιστών, όπου παίζει σημαντικό ρόλο η οριζόντια κλιμακωσιμότητα , το υψηλό throughput και η υψηλή διαθεσιμότητα. Για αυτό πραγματοποιείται διαχωρισμός των δεδομένων (sharding), αυτόματη κατανομή του φόρτου εργασίας (cluster balancing) και διατήρηση αντιγράφων (replication) μεταξύ των κόμβων του cluster. Οι κόμβοι αυτοί διακρίνονται σε 3 κατηγορίες:

- Shards

  Ένας shard κόμβος είναι υπεύθυνος για την αποθήκευση ενός υποσυνόλου των sharded δεδομένων στο cluster. Μαζί όλοι τα shards κρατάνε ολόκληρο το σύνολο δεδομένων. Κάθε shard αποτελείται από ένα replica set, δηλαδή έναν ή περισσότερους κόμβους που διατηρούν αντίγραφα των ίδιων δεδομένων. Ο

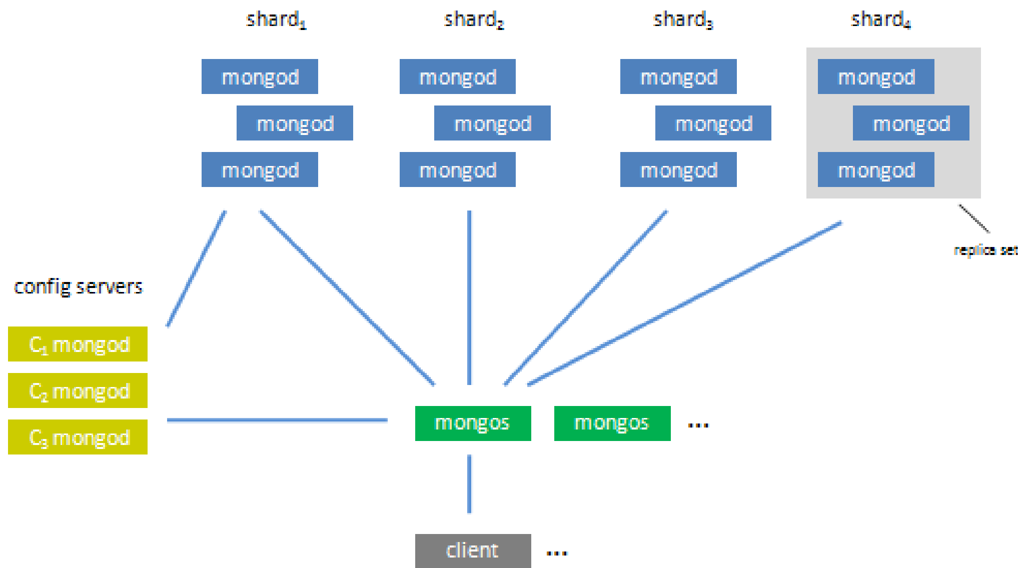
διαχωρισμός των δεδομένων και ο συγχρονισμός των δεδομένων στα replica sets αναλύονται παρακάτω.

- Query Routers

  Ο query router είναι το εξάρτημα της MongoDB, το οποίο είναι υπεύθυνο για τη δρομολόγηση των reads και των writes από τις εφαρμογές των πελατών στους κατάλληλους shards. Οι εφαρμογές των πελατών δεν έρχονται ποτέ σε άμεση επικοινωνία με τους shards για λόγους ασφάλειας και απόδοσης. Ο query router επεξεργάζεται το ερώτημα της εφαρμογής-πελάτη, δρομολογεί τις λειτουργίες που χρειάζονται στον κατάλληλο shard και έπειτα επιστρέφει τα αποτελέσματα. Ένα sharded cluster μπορεί να περιέχει πολλούς query routers, ώστε να διαμοιράζονται τα αιτήματα-ερωτήματα των πελατών και να υπάρχει πιο γρήγορη ανταπόκριση.

- Configuration Servers

  Οι configuration servers αποθηκεύουν μεταδεδομένα και πληροφορίες δρομολόγησης, που υποδεικνύουν ποια δεδομένα διατηρούνται στο κάθε shard. Ενημερώνονται από τους shard κόμβους για το ποια δεδομένα διαθέτουν. Οι routers ζητούν από τους configuration servers πληροφορίες για τη δρομολόγηση των ερωτημάτων που υποβάλλουν οι χρήστες. Έως την έκδοση 3.0 η MongoDB χρησιμοποιούσε ακριβώς 3 configuration servers για πλεονασμό και ασφάλεια των metadata του cluster. Στην έκδοση 3.2 και με χρήση του WiredTiger ως storage engine, επιτρέπεται η χρησιμοποίηση επιπλέον configuration servers ως replica sets(κάθε replica set έχει έως και 50 μέλη).



Εικόνα 12 MongoDB cluster architecture

Sharding

Η MongoDB σπάει τα δεδομένα στο επίπεδο της συλλογής. Για να σπάσουμε σε κομμάτια μια συλλογή χρειάζεται να επιλέξουμε ένα shard key. Το shard key μπορεί να είναι είτε ένα πεδίο στο οποίο υπάρχει single field index είτε περισσότερα του ενός πεδία στα οποία υπάρχει compound index. Το shard key, πρέπει να υπάρχει σε κάθε έγγραφο της συλλογής. Η MongoDB διαχωρίζει σε κομμάτια (chunks) μια συλλογή με βάση την τιμή του shard key, και έπειτα διανέμει τα κομμάτια αυτά στους shards του cluster.

Η MongoDB για να διαχωρίσει τις τιμές του shard key σε chunks χρησιμοποιεί είτε τη μέθοδο διαχωρισμού Ranged Sharding (διαίρεση με βάση το εύρος τιμών) είτε την μέθοδο Hashed Sharding (διαίρεση χρησιμοποιώντας συνάρτηση κατακερματισμού). Μία 3η μέθοδος ονομάζεται Tag Aware Sharding και επιτρέπει στον χρήστη να προσαρμόσει κατάλληλα τον διαχωρισμό βάζοντας μια ετικέτα στο κάθε διάστημα στο οποίο θέλει να χωρίσει το shard key. Είναι πολύ σημαντικό για την απόδοση του συστήματος, ο σχεδιαστής να επιλέξει κατάλληλη μέθοδο διαχωρισμού και κατάλληλο shard key.

Η MongoDB υποστηρίζει αυτόματο sharding, ισοκατανέμοντας τον όγκο των δεδομένων και το φόρτο εργασίας σε όλους τους κόμβους ενός cluster. Ο MongoDB balancer είναι μια διεργασία παρασκηνίου που παρακολουθεί τον αριθμό των chunks σε κάθε shard. Όταν διαπιστώσει ότι τα chunks μιας συλλογής δεν είναι ισοκατανεμημένα στους shards του cluster, τότε ο balancer μετακινεί ένα chunk κάθε φορά από τον shard που έχει τα

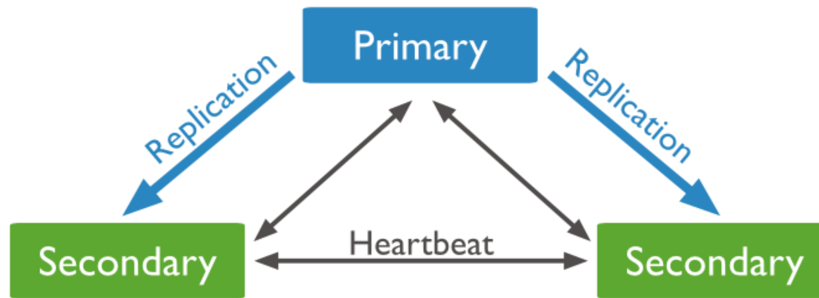
περισσότερα chunks αυτής της συλλογής προς τον shard με τα λιγότερα chunks, μέχρις ότου να υπάρχει ίση κατανομή στο cluster.

Replication

Η MongoDB υποστηρίζει την διατήρηση αντιγράφων (replica sets) προκειμένου να εξασφαλίζεται υψηλή διαθεσιμότητα και σταθερή λειτουργία με ανοχή σε αποτυχίες υλικού, κατατμήσεις δικτύου και σε ότι άλλες περιπτώσεις κάποιος κόμβος σταματά να εξυπηρετεί τις ανάγκες του συστήματος.

Τα Replica Sets αποτελούνται από έναν πρωτεύοντα κόμβο-εξυπηρετητή (primary node), ο οποίος αναλαμβάνει την εξυπηρέτηση των εγγραφών και των αναγνώσεων για τα δεδομένα που αποθηκεύει, και έναν ή περισσότερους δευτερεύοντες κόμβους- εξυπηρετητές (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 κόμβο, και είναι αποθηκευμένες στο αρχείο καταγραφής των ενεργειών του με χρονολογική σειρά (oplog). Η διαδικασία αυτή, πραγματοποιείται ασύγχρονα χωρίς να επηρεάζει τη λειτουργία του πρωτεύοντα κόμβου. 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 συνεπή.

Με χρήση replica sets διασφαλίζεται το αυτόματο failover σε περίπτωση σφάλματος. Οι κόμβοι ενός replica set χρησιμοποιούν heartbeats για τον εντοπισμό των κόμβων που έχουν αστοχήσει. Αν ο πρωτεύων κόμβος αποτύχει, τότε ξεκινά αυτόματα μία διαδικασία ψηφοφορίας ανάμεσα στους υπόλοιπους και εκλέγεται ένας νέος πρωτεύων από τους δευτερεύοντες κόμβους τους replica set. Στη 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. Όταν οι κόμβοι που έχουν αστοχήσει επανέλθουν, συγχρονίζονται με τον πρωτεύων και συνεχίζουν να λειτουργούν ως δευτερεύοντες.



Εικόνα 13 MongoDB replica set

4.1.5 Storage Engine

Ο μηχανισμός αποθήκευσης (storage engine) που χρησιμοποιούσε η MongoDB ως προεπιλογή πριν την έκδοση 3.2 είναι το ***MMAPv1***. Κεντρική ιδέα στο storage engine αυτό είναι τα memory-mapped files (αρχεία που το λειτουργικό σύστημα τοποθέτησε στην μνήμη με το mmap() system call). Συγκεκριμένα ο Virtual Memory Manager του λειτουργικού αντιστοιχίζει όλα τα αρχεία από το δίσκο στην εικονική μνήμη του συστήματος και αποφασίζει ποιά δεδομένα θα διατηρούνται στην ram και ποια στον δίσκο(LRU πολιτική). Η MongoDB γράφει και διαβάζει από την μνήμη και τα υπόλοιπα τα αναλαμβάνει το OS. Αρνητικά που παρουσιάζονται εδώ είναι ο θρυμματισμός του δίσκου, υψηλό read-ahead, έλλειψη συμπίεσης στη μνήμη. Επιπλέον, για να αποφύγει απώλεια δεδομένων και να επιτρέψει durability(αντοχή) των δεδομένων, οι λειτουργίες εγγραφής γράφονται στο journal. Το journal είναι ένας write-ahead logging buffer στην μνήμη ο οποίος γίνεται flushed στο δίσκο περιοδικά. Όταν το journal γραφεί στον δίσκο τα δεδομένα είναι ασφαλή. Επιπρόσθετα χαρακτηριστικά του είναι η χρησιμοποίηση preallocation και padding, καθώς και η εφαρμογή ελέγχου συγχρονισμού (concurrency control) σε επίπεδο συλλογής(collection).

Στην έκδοση 3.0 η MongoDB προσέθεσε ως pluggable μηχανή αποθήκευσης το **WiredTiger**(WT), αφότου αγόρασε την WiredTiger Inc. , εταιρεία που το δημιούργησε. Από την έκδοση 3.2 αποτελεί την προεπιλεγμένη storage engine. Τα βασικά χαρακτηριστικά του είναι τα εξής:

Document Level Concurrency

Το WT χρησιμοποιεί έλεγχο συγχρονισμού σε επίπεδο εγγράφων για τα writes. Ως αποτέλεσμα, πολλοί clients μπορούν να τροποποιήσουν διαφορετικά έγγραφα μιας συλλογής ταυτόχρονα. Για τις περισσότερες λειτουργίες διαβάσματος και εγγραφής, το WT

χρησιμοποιεί αισιόδοξο έλεγχο συγχρονισμού. Εφαρμόζει locks μόνο σε επίπεδο global, βάσης, συλλογής. Όταν το storage engine εντοπίσει συγκρούσεις(conflicts) μεταξύ δύο λειτουργιών , μία θα εγείρει σύγκρουση εγγραφής και η MongoDB θα επαναλάβει τη λειτουργία.

Μερικές λειτουργίες που εμπλέκουν πολλαπλές βάσεις, απαιτούν ακόμα global lock, ενώ κάποιες άλλες, όπως η διαγραφή μιας συλλογής, απαιτούν ακόμα κλείδωμα σε επίπεδο βάσης.

Snapshots and Checkpoints

Το WT χρησιμοποιεί MultiVersion Concurrency Control (MVCC). Μόλις ξεκινά μία λειτουργία, το WT παρέχει ένα στιγμιότυπο των δεδομένων στην συνδιαλλαγή. Το στιγμιότυπο αναπαριστά μία συνεπή όψη των δεδομένων που βρίσκονται στη μνήμη.

Όταν θα γράψει στον δίσκο, το WT θα γράψει όλα τα δεδομένα του στιγμιότυπου με συνεπή τρόπο στα αρχεία δεδομένων του δίσκου. Τα δεδομένα αυτά είναι πλέον «ασφαλή» (durable : θεωρούνται ασφαλή κάποια δεδομένα όταν δεν υπάρχει πιθανότητα να χαθούν από κάποια αποτυχία, κλείσιμο, επανεκκίνηση κάποιας λειτουργίας του server) και αποτελούν ένα checkpoint. Το checkpoint μπορεί να λειτουργήσει ως σημείο ανάκαμψης, αφού διασφαλίζει ότι όλα τα δεδομένα είναι συνεπή μέχρι το τελευταίο checkpoint. Το WT δημιουργεί checkpoints ανά 60 δευτερόλεπτα ή κάθε 2 GB δεδομένων journal.

Κατά την δημιουργία ενός νέου checkpoint, το προηγούμενο είναι ακόμη διαθέσιμο και έγκυο, έτσι ώστε αν η MongoDB τερματίσει ή αντιμετωπίσει σφάλμα στην εγγραφή του, να μπορεί να επανακάμψει από το τελευταίο έγκυρο checkpoint. Το νέο checkpoint γίνεται μόνιμο και διαθέσιμο, όταν τα metadata του WT ανανεωθούν και αναφέρονται σε αυτό. Όταν γίνει διαθέσιμο, τότε το WT ελευθερώνει τις σελίδες του παλιού checkpoint από την μνήμη.

Journal

Το journal είναι ένα write-ahead transaction log που χρησιμοποιείται σε συνδυασμό με τα checkpoints για να διασφαλίσουν αντοχή των δεδομένων(durability). Μετά από σφάλμα, η MongoDB μπορεί να επανακάμψει από το τελευταίο checkpoint. Για να επαναφέρει τις αλλαγές που έγιναν μετά από το checkpoint χρησιμοποιεί journaling.

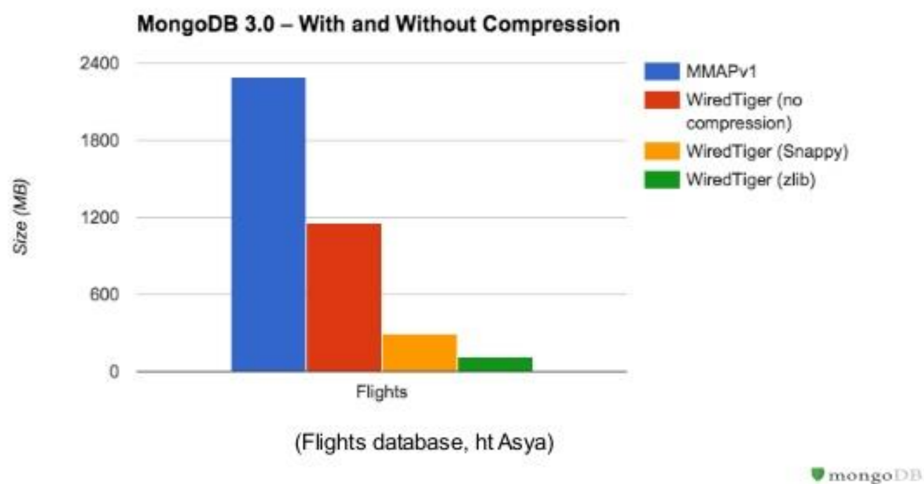
Για κάθε λειτουργία εγγραφής το WT δημιουργεί μια καταχώρηση στο journal, που περιλαμβάνει οποιαδήποτε εγγραφή προκλήθηκε εσωτερικά από την αρχική εγγραφή. Για παράδειγμα, ένα update μπορεί να προκαλέσει και εγγραφές σε ευρετήρια. Έτσι, η καταχώρηση στο journal περιλαμβάνει και το update και τις τροποποιήσεις των ευρετηρίων.

Οι καταχωρήσεις του journal γίνονται αρχικά buffered στην μνήμη και συγχρονίζονται στον δίσκο κάθε 50 ms ή κάθε φορά που συμβαίνει κάποιο από τα εξής:

i. Πραγματοποιείται ένα checkpoint
ii. Ζητείται ρητά από κάποια λειτουργίας εγγραφής να συγχρονίσει το WT τα αρχεία journal στον δίσκο.
iii. To journal έφτασε τα 100MB και το WT δημιουργεί καινούριο αρχείο journal. Τότε συγχρονίζει το παλιό.

Compression

Με το WT η MongoDB υποστηρίζει συμπίεση για όλες τις συλλογές και ευρετήρια. Η συμπίεση ελαχιστοποιεί τη ανάγκη σε αποθηκευτικούς πόρους χρησιμοποιώντας CPU πόρους. Υποστηρίζει διάφορους αλγόριθμους συμπίεσης. Ως προεπιλογή, συμπιέζει όλες τις collection και το journal με συμπίεση Snappy και όλα τα ευρετήρια με prefix συμπίεση. Διαθέσιμη είναι και η συμπίεση zlib, η οποία πετυχαίνει υψηλότερο λόγο συμπίεσης, ωστόσο η Snappy παρέχει καλύτερη ισορροπία μεταξύ λόγου συμπίεσης και κατανάλωσης CPU.

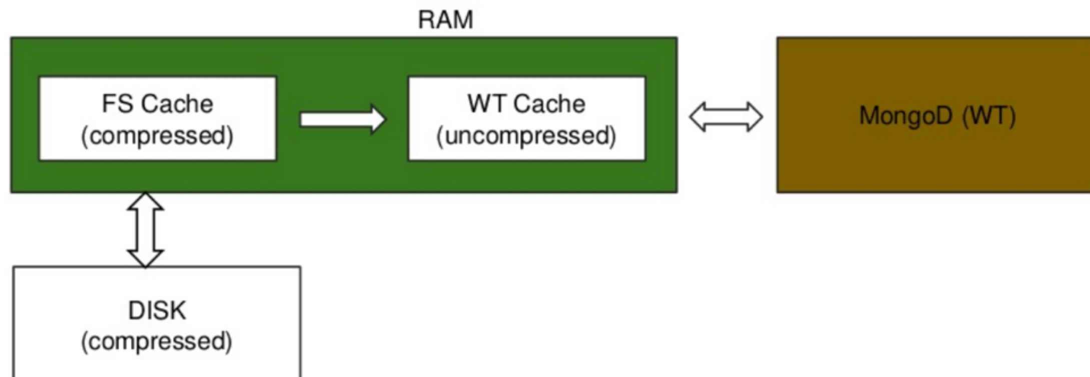


Εικόνα 14 WiredTiger Compression

Memory Use

Το WT μεγιστοποιεί την χρήση της διαθέσιμης μνήμης για να μειώσει τα I/O bottlenecks. Χρησιμοποιεί 2 caches : την WiredTiger cache και την cache του filesystem. Η WiredTiger cache αποθηκεύει ασυμπίεστα δεδομένα και έχει πολύ υψηλή απόδοση. Ως προεπιλογή,

καταλαμβάνει το μεγαλύτερο εκ των 1 GB , 60% της RAM μείον 1GB. Η filesystem cache του λειτουργικού συστήματος αποθηκεύει συμπιεσμένα δεδομένα και χρησιμοποιεί αυτόματα όλη την ελεύθερη μνήμη που δεν χρησιμοποιείται από την WiredTiger cache ή άλλες διεργασίες. Όταν τα δεδομένα δε βρίσκονται στην WiredTiger cache, τότε το WT θα τα αναζητήσει στην filesystem cache. Δεδομένα που βρίσκονται στη filesystem cache πρώτα θα αποσυμπιεστούν πριν μετακινηθούν στην WiredTiger cache.



Εικόνα 15 WiredTiger Caches (WiredTiger Cache and FS Cache)

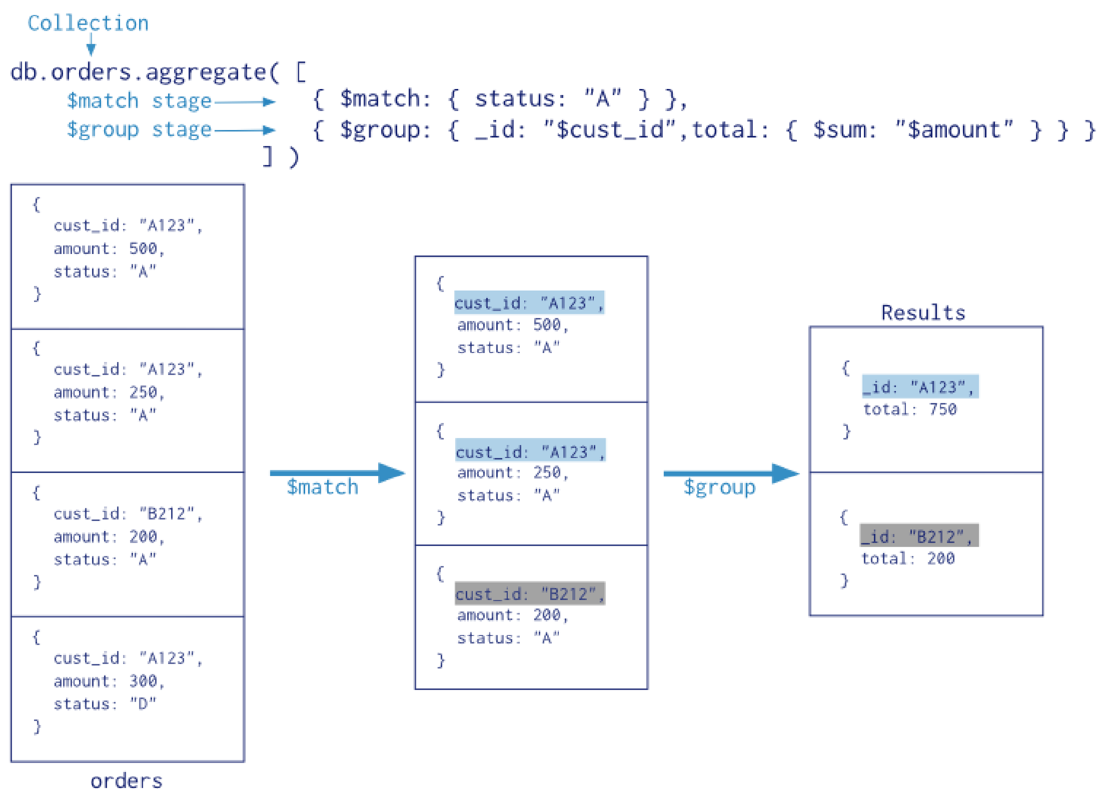
4.1.6 Aggregation

4.1.6.1 Aggregation Pipeline

Η πλατφόρμα συνάθροισης (aggregation framework) της MongoDB είναι μοντελοποιημένη στην λογική των σωληνώσεων επεξεργασίας δεδομένων (data processing pipelines). Τα έγγραφα ακολουθούν μια σωλήνωση πολλών σταδίων, η οποία τα μετατρέπει σε ένα συγκεντρωτικό αποτέλεσμα.

Τα πιο βασικά στάδια του pipeline παρέχουν φίλτρα, που λειτουργούν παρόμοια με τα queries, και μετασχηματισμούς εγγράφων που διαμορφώνουν το τελικό έγγραφο που επιστρέφεται ως αποτέλεσμα. Άλλες λειτουργίες του pipeline παρέχουν εργαλεία για ομαδοποίηση και ταξινόμηση εγγράφων με βάση συγκεκριμένο πεδίο ή πεδία, καθώς και εργαλεία για συνάθροιση των περιεχομένων από πίνακες, συμπεριλαμβανομένων των πινάκων με ενσωματωμένα έγγραφα. Επί πρόσθετα, διατίθενται διάφοροι τελεστές , όπως ο υπολογισμός μέσου όρου και η συνένωση string.

Το pipeline εκτελεί πολύ αποδοτικό data aggregation χρησιμοποιώντας έμφυτες λειτουργίες της MongoDB. Μπορεί να λειτουργήσει και σε κατανεμημένη συλλογή (sharded collection) και μπορεί να χρησιμοποιήσει ευρετήρια για να βελτιώσει την απόδοσή του κατά την διάρκεια κάποιων σταδίων.

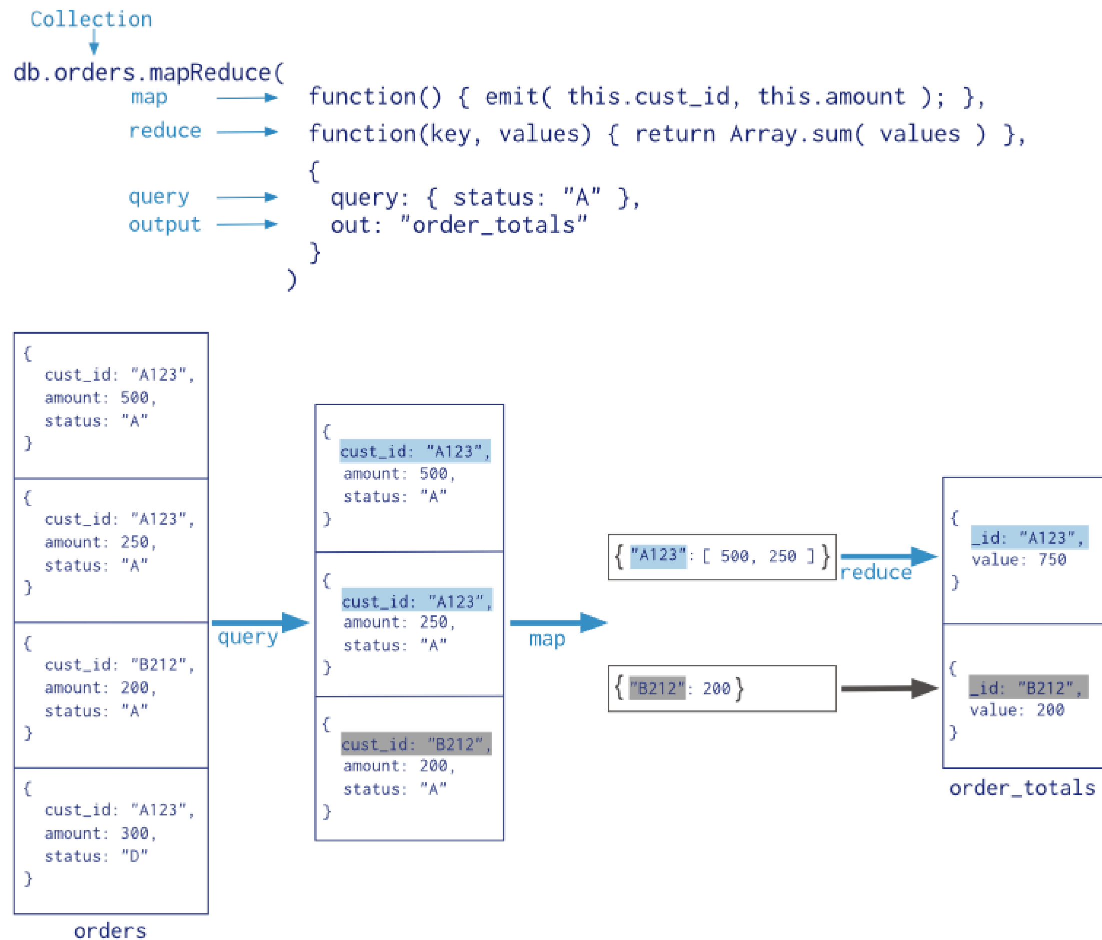


Εικόνα 16 MongoDB aggregation pipeline example

4.1.6.2 Map Reduce

Η MongoDB δίνει την δυνατότητα για εκτέλεση map-reduce λειτουργιών. Κατά την εκτέλεση, εφαρμόζεται η συνάρτηση map σε κάθε έγγραφο εισόδου(όσα έγγραφα ικανοποίησαν την συνθήκη του query) και παράγει ζεύγη key-value. Τα key-value ζεύγη ομαδοποιούνται με βάση το key και για κάθε key που έχει πολλαπλά values εφαρμόζεται η συνάρτηση reduce. Η reduce εφαρμόζεται ανά key , στα values που αντιστοιχούν σε αυτό και επιστρέφει μια τιμή. Η reduce δεν πρέπει να έχει πρόσβαση στη βάση. Η σειρά των values δεν πρέπει να επηρεάζει το αποτέλεσμα. Ο χρήστης μπορεί προαιρετικά να ορίσει μία ακόμα συνάρτηση, τη finalize function, η οποία εφαρμόζεται στα αποτελέσματα της reduce function

Όλες οι συναρτήσεις map-reduce στη MongoDB είναι Javascript. Πριν το στάδιο map μπορεί να εφαρμοστεί στην συλλογή εισόδου οποιαδήποτε ταξινόμηση και περιορισμός εγγράφων. Τα αποτελέσματα μπορούν να επιστραφούν σε έγγραφο ή να γραφτούν σε συλλογή. Το map-reduce μπορεί να εφαρμοστεί σε sharded collection αλλά και να γράψει τα αποτελέσματα σε sharded collection.

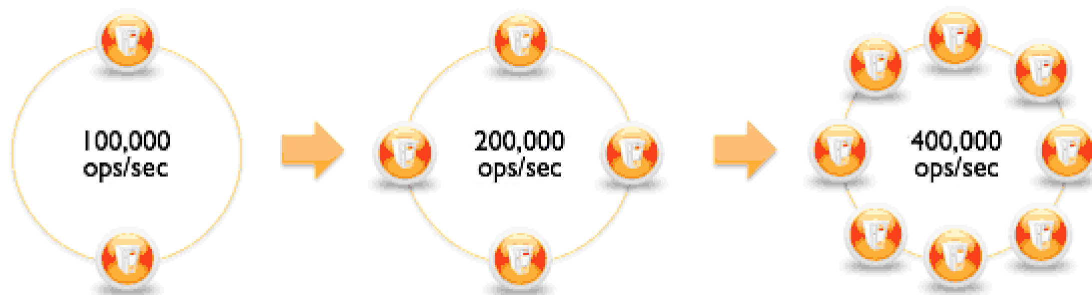


Εικόνα 17 MongoDB map-reduce example

4.2 Apache Cassandra

Η Cassandra είναι μία βάση η οποία συνδυάζει την κατανεμημένη αρχιτεκτονική του Dynamo της Amazon χρησιμοποιώντας μία δομή δεδομένων αντίστοιχη με αυτή του BigTable της Google. 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 για να διαχειρίζεται τεράστιες ποσότητες δεδομένων σε πολλαπλά datacenters καθώς και στο cloud με ασύγχρονη αντιγραφή επιτρέποντας υψηλή απόδοση και χαμηλή καθυστέρηση για τους χρήστες.. Τα βασικά χαρακτηριστικά που προσφέρει είναι τα εξής:

- Ομοιογενώς κατανεμημένο σύστημα, που αποτελείται από ισότιμους κόμβους, χωρίς Single Point of Failure.
- Υψηλή διαθεσιμότητα του συστήματος με ανοχή στις κατατμήσεις και τις αποτυχίες, με αυτόματη διαχείριση τέτοιων περιπτώσεων.
- Δυνατότητες προσαρμόσιμης συνέπειας στα δεδομένα.
- Πολύ υψηλός ρυθμός διεκπεραίωσης εγγραφών.
- Αλληλεπίδραση με τη βάση μέσω μίας γλώσσας που ονομάζεται CQL(Cassandra Query Language) και έχει παρόμοια σύνταξη με την SQL για ευκολία στην εκμάθηση και την χρήση. Το παλιό Thrift API έχει αντικατασταθεί πλήρως και δεν υποστηρίζεται καθόλου από τις τελευταίες εκδόσεις της Cassandra.
- Η Cassandra παρέχει δυνατότητα γραμμικής κλιμάκωσης. Η δυναμική του συστήματος μπορεί πολύ εύκολα να αυξηθεί απλά προσθέτοντας νέους κόμβους στο δίκτυο. Για παράδειγμα, αν 2 κόμβοι μπορούν να διαχειρίζονται 100,000 λειτουργίες ανά δευτερόλεπτο, 4 κόμβοι μπορούν να υποστηρίξουν 200,000 λειτουργίες ανά δευτερόλεπτο, ενά 8 κόμβοι 400,000 λειτουργίες ανά δευτερόλεπτο.



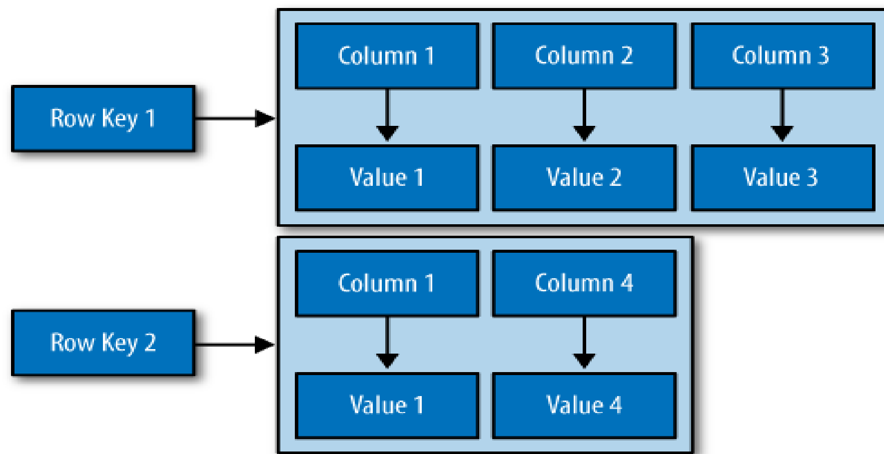
Εικόνα 18 Γραμμική κλιμάκωση στην Cassandra

Χρησιμοποιείται σε πάνω από 1500 εταιρίες ανάμεσα στις οποίες οι εξής: Facebook, Twitter, Cisco, Hulu, Rackspace, Digg, Cloudkick, Reddit, CERN, eBay και Instagram.

4.2.1 Μοντέλο Δεδομένων

Το μοντέλο που χρησιμοποιείται είναι ένα υβριδικό μεταξύ ενός key-value και column oriented μοντέλου συστήματος διαχείρισης. Στην Cassandra, ο χώρος που περιέχει τα δεδομένα ονομάζεται keyspace. Είναι το περίβλημα των δεδομένων, αντίστοιχο του database στα RDBMS, και συνήθως ορίζεται ένα ανά εφαρμογή. Μέσα στο keyspace βρίσκονται οι οικογένειες στηλών (column families) οι οποίες είναι το βασικό αντικείμενο διαχείρισης των δεδομένων και είναι το αντίστοιχο ενός πίνακα στις σχεσιακές βάσεις, με τη διαφορά ότι το σχήμα είναι δυναμικό και εύκαμπτο. Αντίθετα απ' ότι έχουμε συνηθίσει στις σχεσιακές βάσεις δεδομένων, με την Cassandra δε χρειάζεται να γνωρίζουμε εκ των προτέρων όλες τις στήλες που χρειάζεται η εφαρμογή μας (flexible schema). Οι επιπλέον στήλες και τα μεταδεδομένα τους μπορούν να προστεθούν στην εφαρμογή μας εφόσον χρειάζονται χωρίς να επιβαρυνθεί το σύστημά μας με downtime. Κάθε γραμμή μιας οικογένειας στηλών προσδιορίζονται από ένα κλειδί και περιέχει κάποιο αριθμό στηλών. Δε χρειάζεται κάθε γραμμή να έχει τον ίδιο αριθμό στηλών με τις υπόλοιπες. Χώρος καταλαμβάνεται μόνο για τις στήλες εκείνες που έχουν δεδομένα. Κάθε στήλη προσδιορίζεται με τη σειρά της από ένα μοναδικό κλειδί και περιέχει κάποια τιμή , μια χρονοσφραγίδα εγγραφής και μπορεί να προστεθεί και TTL(TimeToLive).

Δεν μπορούν να γίνουν ενώσεις (joins) ή υποερωτήματα (subqueries), αντίθετα δίνεται έμφαση στην αποκανονικοποίησης (denormalization). Είναι κλασική τεχνική στην Cassandra να χρησιμοποιείται ακόμα και μία column family(table, όπως ονομάζεται στην CQL) ανά ερώτημα που χρησιμοποιούμε. Σε κάθε table κάποιο μέρος των ίδιων δεδομένων επαναλαμβάνονται. Με αυτόν τον τρόπο η Cassandra εκμεταλλεύεται τα πολύ γρήγορα writes που επιτυγχάνει, και θυσιάζει αποθηκευτικό χώρο για υψηλές επιδόσεις ανάγνωσης.



Εικόνα 19 Column Family

Για τον ορισμό μιας column family(table) η μόνη πληροφορία σχήματος που είναι απαραίτητη είναι το πρωτεύον κλειδί(primary key) και ο τύπος του. Το primary key είναι μοναδικό για κάθε γραμμή και αποτελείται από 1 ή περισσότερες στήλες. Το πρώτο κομμάτι του primary key είναι το partition key(1 ή περισσότερες στήλες). Το partition key είναι υπεύθυνο για τον διαμοιρασμό των δεδομένων στους κόμβους του cluster. Είναι πολύ σημαντικό στα κατανεμημένα συστήματα αφού καθορίζει την τοπικότητα των δεδομένων (data locality). Γραμμές με το ίδιο partition key(λέμε ότι αποτελούν ένα partition) αποθηκεύονται σε έναν μοναδικό κόμβο(και στους replicas του).

Σε ένα απλό παράδειγμα, το primary key αποτελείται μόνο από το partition key.

```
CREATE TABLE example (
    key text,
    data1 int,
    data2 timestamp,
    PRIMARY KEY (key)
);
```

Στην περίπτωση που το primary key είναι σύνθετο, όλες οι στήλες μετά το partition key ονομάζονται clustering columns και αποτελούν το clustering key. Αυτό είναι υπεύθυνο για την ταξινόμηση των δεδομένων εντός του κάθε partition. Ένα παράδειγμα table στην γενικότερη μορφή είναι το εξής:

```
CREATE TABLE example (
    partitionKey1 text,
    partitionKey2 text,
    clusterKey1 text,
    clusterKey2 text,
    normalField1 text,
    normalField2 text,
    PRIMARY KEY ( (partitionKey1, partitionKey2),
        clusterKey1, clusterKey2)
    );
```

Σε πίνακες που χρησιμοποιούν clustering columns, κάποιες μη-clustering στήλες μπορούν να δηλωθούν ως static. Οι στατικές στήλες είναι κοινές για κάθε εγγραφή με κοινό partition key και αποθηκεύονται μία φορά ανά partition.

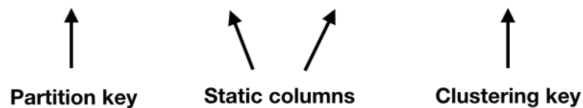
Ακολουθεί ένα χαρακτηριστικό παράδειγμα που δείχνει πως αποθηκεύονται εσωτερικά οι εγγραφές ενός table με clustering και static columns.

```
CREATE TABLE book (
    author_id text,
    birthday timestamp static,
    firstname text static,
    book_id bigint,
    book_date timestamp,
    book_title text,
    book_price int,
    PRIMARY KEY((author_id), book_id)
);
```

Κάθε συγγραφέας έχει γράψει πολλά βιβλία. Partition key είναι το author_id, και clustering το book_id. Οι γραμμές θα διαμοιραστούν βάσει του συγγραφέα σε όλους τους κόμβους. Όλα τα βιβλία του ίδιου συγγραφέα αποθηκεύονται μαζί στον ίδιο κόμβο και είναι ταξινομημένα σε αύξουσα σειρά με βάση το το book_id.

Η CQL θα παρουσιάσει τα περιεχόμενα του table ως εξής :

author_id	birthday	firstname	book_id	book_date	book_title	book_price
aaa	1/1/1970	alice	1	12/12/2014	AAA	10
aaa	1/1/1970	alice	2	17/12/2014	BBB	20
aaa	1/1/1970	alice	3	19/12/2014	CCC	30
bbb	1/1/1984	bob	1	13/12/2014	DDD	40
bbb	1/1/1984	bob	2	18/12/2014	EEE	50



Ωστόσο, εσωτερικά η Cassandra αποθηκεύει αυτά τα δεδομένα σε μία γραμμή ανά συγγραφέα. Το partition key γίνεται row key, και όλα τα υπόλοιπα δεδομένα, πληροφορίες συγγραφέα και όλα τα βιβλία, αποθηκεύονται στην ίδια γραμμή(partition).

<table>
<tr><td rowspan="2">aaa</td><td>birthday</td><td>firstname</td><td>1:book_date</td><td>1:book_price</td><td>1:book_title</td><td>2:book_date</td><td>2:book_price</td><td>...</td></tr>
<tr><td>1/1/1970</td><td>alice</td><td>12/12/2014</td><td>10</td><td>AAA</td><td>17/12/2014</td><td>20</td><td>...</td></tr>
<tr><td rowspan="2">bbb</td><td>birthday</td><td>firstname</td><td>1:book_date</td><td>1:book_price</td><td>1:book_title</td><td>2:book_date</td><td>2:book_price</td><td>2:book_title</td></tr>
<tr><td>1/1/1984</td><td>bob</td><td>13/12/2014</td><td>40</td><td>DDD</td><td>18/12/2014</td><td>50</td><td>EEE</td></tr>
</table>

Οι στατικές στήλες αποθηκεύονται μία φορά στην αρχή του partition. Το clustering key (book_id) χρησιμοποιείται στο όνομα των στηλών. Τα βιβλία είναι αποθηκευμένα ταξινομημένα με βάσει το clustering key.

Αυτές οι γραμμές συχνά αποκαλούνται “wide rows”.

4.2.2 Μοντέλο Ερωτημάτων

Η εγκατάσταση της Cassandra περιλαμβάνει το εργαλείο cqlsh, ένα πρόγραμμα γραμμής εντολών γραμμένο σε python που χρησιμοποιείται για την εκτέλεση Cassandra Query Language (CQL) εντολών. Παρόμοια με την SQL, η CQL χρησιμοποιεί μια εντολή SELECT για την ανάκτηση δεδομένων από την βάση.

```
SELECT select_expression
FROM [ keyspace_name. ] table_name
[ WHERE relation [ AND relation ] [ . . . ] ]
[ ORDER BY clustering_column [ { ASC | DESC } ]
[ LIMIT n ]
[ ALLOW FILTERING ]
```

Επιλέγεται ποιες στήλες θα επιστραφούν και από ποιό table. Υπάρχει δυνατότητα περιορισμού των συνολικών γραμμών που θα επιστραφούν ή των γραμμών ανά partition. Φυσικά η πρόταση “Where” καθορίζει ποιες είναι οι συνθήκες που πρέπει να ικανοποιεί μια γραμμή για να συμπεριληφθεί στα αποτελέσματα που θα επιστραφούν. Επίσης, εφόσον προσδιοριστεί ένα συγκεκριμένο partition στην where πρόταση , μπορούν τα ταξινομηθούν οι γραμμές βάσει της 1ης clustering στήλης σε αύξουσα ή φθίνουσα σειρά. Τέλος, επιτρέπεται η χρήση built-in συγκεντρωτικών(aggregate) συναρτήσεων - όπως max, avg, count - και άλλων ορισμένων από τον χρήστη συναρτήσεων και συγκεντρωτικών συναρτήσεων.

Where restrictions

Οι στήλες που αποτελούν το partition key υποστηρίζουν τους τελεστές = και IN (δεν προτείνεται) . Η Cassandra απαιτεί να υπάρχουν περιορισμοί είτε σε όλα τα partition keys είτε σε κανένα, εκτός αν το query χρησιμοποιεί δευτερεύον ευρετήριο. Ο λόγος που η Cassandra χρειάζεται όλα τα partitions keys είναι για να μπορέσει να υπολογίσει την hash τιμή που θα της επιτρέψει να εντοπίσει ποιος κόμβος περιέχει το partition αυτό. Η χρήση τελεστών εύρους (> , >= , < , <=) στα partition keys επιτρέπεται μόνο με την χρήση ByteOrderedPartitioner και της συνάρτησης token() σε όλα τα partition keys. Αν δεν περιοριστούν καθόλου, τότε απαιτείται να προστεθεί ALLOW FILTERING στο τέλος του query(αναλύεται παρακάτω).

Οποιαδήποτε clustering column μπορεί να περιοριστεί με = ή IN , χωρίς να απαιτείται filtering, αν έχουν περιοριστεί όλες οι προηγούμενες στήλες που αποτελούν το Primary key. Στην τελευταία(ως προς την σειρά της στο Primary key) clustering column που περιορίζεται , μπορεί να χρησιμοποιηθεί, εκτός από = και IN, οποιοσδήποτε τελεστής εύρους. Από την έκδοση Cassandra 3.6 και μετά, μπορείς να προσθέσεις ALLOW FILTERING για να φιλτράρεις οποιαδήποτε non-indexed cluster στήλη.

Δημιουργώντας δευτερεύον ευρετήριο σε κάποια στήλη, μπορώ να εκτελέσω απευθείας query σε αυτό χρησιμοποιώντας μόνο την ισότητα , ή τους περιορισμούς CONTAINS / CONTAINS KEY για στήλες τύπου συλλογής (map, set, list). Ερωτήματα σε δευτερεύοντα ευρετήρια , επιτρέπουν να περιορίσουμε περαιτέρω τα αποτελέσματα με filtering, χρησιμοποιώντας περιορισμούς =, >, >=, <=, <, CONTAINS, CONTAINS KEY σε non-indexed στήλες. Τέλος, να αναφέρουμε ότι μια ερώτηση ευρετηρίου θα απευθυνθεί σε όλους τους κόμβους για να εξετάσει το κομμάτι του ευρετηρίου που περιέχει ο καθένας. Για αυτό, είναι αποδοτικό να προσδιορίσουμε το partition key σε ερωτήματα που χρησιμοποιούν ευρετήριο για να γνωρίζει σε ποιο κόμβο βρίσκεται το κομμάτι του ευρετηρίου που χρειάζεται να αναζητήσει.

Allow Filtering

Η Cassandra σε κάποια ερωτήματα απαντάει με το εξής μήνυμα λάθους, γνωρίζοντας ότι ίσως να μη μπορεί να εκτελεστεί αποτελεσματικά:

```
Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING.
```

Για να εκτελέσει ένα τέτοιο ερώτημα η Cassandra πρέπει να ανακτήσει δεδομένα και έπειτα να τα διατρέξει, βγάζοντας εκτός (filtering out) τις γραμμές που δεν ικανοποιούν έναν συγκεκριμένο περιορισμό. Η απόδοση του ερωτήματος είναι απρόβλεπτη αφού δε γνωρίζει πόσα δεδομένα θα παραλείψει για κάθε γραμμή που επιστρέφει ως αποτέλεσμα. Στην χειρότερη περίπτωση μπορεί να σαρώσει όλα τα δεδομένα για να επιστρέψει μηδέν γραμμές! Αυτό έρχεται σε αντίθεση με τις λειτουργίες χωρίς “ALLOW FILTERING”, στις οποίες τα δεδομένα που διαβάζονται κλιμακώνονται γραμμικά με το πλήθος των δεδομένων που επιστρέφονται.

Αυτή η διαδικασία είναι αποδεκτή αν επιστρέφονται τα περισσότερα απ τα δεδομένα που ανακτώνται, και τα παραλειπόμενα δεδομένα δε κοστίζουν πολύ. Τότε μπορούμε να προσθέσουμε “ALLOW FILTERING” , αλλιώς θα πρέπει να σκεφτούμε την δημιουργία κάποιου ευρετηρίου.

4.2.3 *Ευρετήρια*

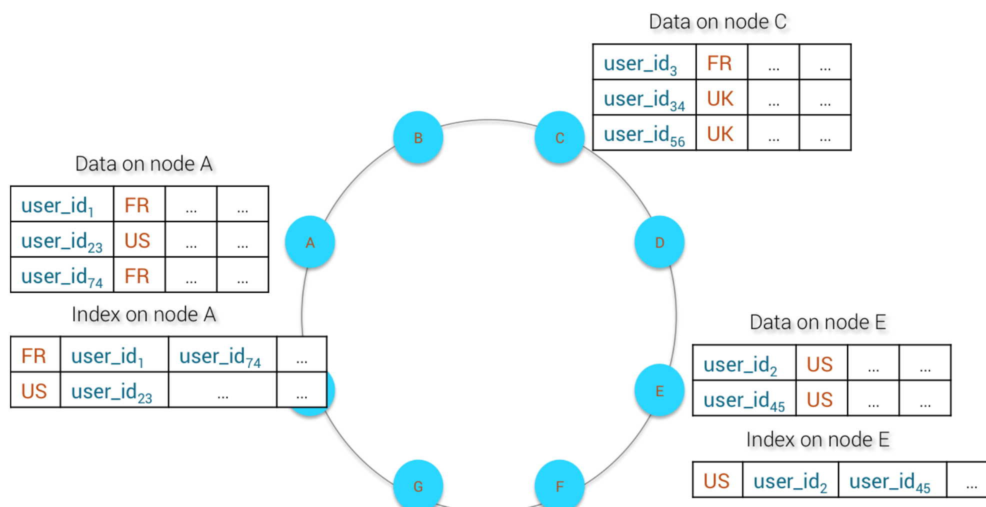
Secondary Indexes

Τα δευτερεύοντα ευρετήρια (secondary indexes) χρησιμοποιούνται για την εκτέλεση ερωτημάτων πάνω σε στήλες, στις οποίες δεν είναι κανονικά εφικτό, όπως οι στήλες εκτός του primary key. Για παράδειγμα, έστω ένα table με χρήστες που περιλαμβάνει πληροφορίες όμως το mail των χρηστών. Το user ID είναι το primary key και για να ανακτήσουμε το email ενός συγκεκριμένου χρήστη, θα το αναζητήσουμε μέσω του ID του. Για την εκτέλεση του αντίστροφου ερωτήματος – δοσμένου ενός email , βρες το user ID – απαιτείται δευτερεύον ευρετήριο στο email.

Είναι εύκολα στην δημιουργία και ανανεώνονται αυτόματα από το σύστημα. Ωστόσο υπάρχουν περιπτώσεις που πρέπει να αποφεύγεται η χρήση τους:

- Υψηλής πληθικότητας ευρετήρια. Για παράδειγμα ένα ευρετήριο user-by-email, όπως το περιγράψαμε παραπάνω, είναι πολύ κακή επιλογή. Κάθε email είναι μοναδικό, επομένως το ευρετήριο θα έχει τόσες διακριτές τιμές όσες και οι χρήστες.
- Πολύ χαμηλής πληθικότητας ευρετήρια, όπως στήλες τύπου boolean, ή το φύλο κάποιο χρήστη. Σε κάθε κόμβο, όλος ο πληθυσμός των χρηστών θα διαμοιραστεί σε μόνο 2 partitions για το index(male & female).
- Ευρετήρια σε στήλη με συχνές ενημερώσεις ή διαγραφές.

Τα secondary indexes είναι τοπικά. Αυτό σημαίνει ότι σε κάθε κόμβο, δημιουργείται ευρετήριο για τα δεδομένα που βρίσκονται εκεί. Στην εικόνα φαίνεται ένα παράδειγμα με πίνακα χρηστών και ευρετήριο στην στήλη “country”.



Εικόνα 20 Δευτερεύον ευρετήριο user-by-country

Οι τεχνικοί λόγοι για την συνύπαρξη των δεδομένων ευρετηρίου και των βασικών δεδομένων στον ίδιο κόμβο είναι:

- Μείωση της καθυστέρησης των ανανεώσεων και της πιθανότητας αστοχίας του ευρετηρίου.
- Αποφυγή αυθαίρετα μεγάλων partition στα δεδομένα ευρετηρίου. Για παράδειγμα, αν όλοι οι χρήστες της χώρας "US" ήταν στο ίδιο partition, ένας κόμβος θα επιβαρυνόταν πολύ.

Τα secondary indexes θα πρέπει ωστόσο να χρησιμοποιούνται με σύνεση, καθώς μπορεί να επηρεάσουν πολύ αρνητικά την απόδοση των ερωτημάτων. Για παράδειγμα, ας υποθέσουμε ότι έχουμε ένα ring 10 κόμβων στους οποίους έχει διαμοιραστεί ο πίνακας των users με βάση το user_id και υπάρχει ευρετήριο στη στήλη email. Διατυπώνοντας query βάσει του ID (primary key), μπορεί να βρεθεί αμέσως ποιος κόμβος περιέχει την εγγραφή για αυτόν τον χρήση. Ένα query, μία ανάγνωση από δίσκο. Ωστόσο, ένα query βάσει του email (indexed value) θα αναγκάσει κάθε μηχάνημα να αναζητήσει στις δικές του εγγραφές. . Ένα query, 10 αναγνώσεις από δίσκο. Κλιμακώνοντας είτε των αριθμό των χρηστών, είτε τα μηχανήματα στο ring, η απόδοση πέφτει δραματικά. Λύση του προβλήματος αυτού, αποτελούν τα πρωτεύοντα ευρετήρια(primary indexes) και τα materialized views.

Τέλος, επειδή τα δευτερεύοντα ευρετήρια είναι υλοποιημένα να διαμοιράζονται στον cluster με αυτόν τον τρόπο, λειτουργούν καλύτερα όταν η Cassandra μπορεί να περιορίσει των αριθμό των κόμβων τους οποίους πρέπει να ρωτήσει. Αυτό επιτυγχάνεται όταν το query περιορίζει το partition key, είτε με μοναδική τιμή (`WHERE partition = xxx`), είτε με λίστα από τιμές(`WHERE partition IN (aaa, bbb, ccc)`) , είτε με εύρος τιμών (`WHERE token(partition) ≥ xxx AND token(partition) ≤ yyy`). Όσο πιο στενός ο περιορισμός, τόσο καλύτερη απόδοση.

Primary Indexes

Αναζητήσεις μέσω του primary key είναι εξαιρετικά γρήγορες στην Cassandra. Για να το εκμεταλλευτούμε αυτό, αποκανονικοποιούμε (denormalize) τα δεδομένα σε πολλαπλά tables και χτίζουμε τα δικά μας ευρετήρια-tables. Για παράδειγμα από τον πίνακα users, μπορούμε να δημιουργήσουμε πρωτεύοντα ευρετήρια ως εξής:

```
CREATE TABLE users(
     user_id bigint,
     email text,
     age int,
     firstname text,
     lastname text,
     country text,
     ...
     PRIMARY KEY(user_id)
);

CREATE TABLE users_by_age .. PRIMARY KEY(age,user_id)
CREATE TABLE users_by_name ..PRIMARY KEY((firstname,lastname), user_id)
CREATE TABLE users_by_email .. PRIMARY KEY(email)
```

Τα ευρετήρια αυτά δεν είναι τοπικά σε κάθε κόμβο αλλά κατανέμουν τα δεδομένα τους, όπως και κάθε table, σε όλο το cluster χρησιμοποιώντας τον partitioner (e.g. *Murmur3Partitioner* , ως προεπιλογή). Έτσι, για να αναζητήσουμε π.χ όλους τους users με ηλικία 28, δεν χρειάζεται να ρωτήσουμε όλους τους κόμβους, αλλά μόνο αυτόν που είναι υπεύθυνος για το συγκεκριμένο partition. Ένα query, μία ανάγνωση. Εκτελούν τα ερωτήματα, επομένως, πολύ αποδοτικά.

Δεν προτείνονται για δεδομένα με χαμηλή πληθικότητα, όπως η στήλη "country", γιατί δημιουργούνται μεγάλα partitions που επιβαρύνουν μεμονωμένα κάποιον κόμβο και δημιουργούνται hotspots.

Το πρόβλημα εδώ είναι ότι απαιτείται κώδικας από την πλευρά του client, για κάθε εφαρμογή ξεχωριστά, για να διαχειριστεί την ενημέρωση των ευρετηρίων σε κάθε λειτουργία εγγραφής (insert, update, delete) στο αρχικό table. Πρέπει να εγγυηθεί τελική συνέπεια και να λάβει υπόψη τυχόν αποτυχίες. Για να συμβούν αυτά, για κάθε εντολή εγγραφής γράφονται logged batches(συνδυάζει πολλαπλές εντολές σε μία λογική λειτουργία, εξασφαλίζει ότι αν το batch είναι επιτυχές, τότε όλες οι εντολές επέτυχαν). Τα batches δημιουργούν καθυστερήσεις στο δίκτυο και επιβραδύνουν τις εγγραφές.

Η βασικότερη αιτία του write latency είναι τα read-before-write. Όταν εκτελείται ενημέρωση σε μια γραμμή του αρχικού table, απαιτείται τώρα να διαβαστεί η συγκεκριμένη γραμμή, για

να ανακτήσουμε τις στήλες που χρειάζονται για την εφαρμογή των αλλαγών στα tables-ευρετήρια.

Materialized Views (MV)

Στην έκδοση 3.0, η Cassandra εισήγαγε τα materialized views(MV). Ένα MV είναι ένα table που δημιουργείται από τα δεδομένα ενός άλλου table, με νέο primary key και νέες ιδιότητες. Τα MV διαχειρίζονται την από-κανονικοποίηση (denormalization) αυτόματα στην πλευρά του server, αφαιρώντας από τον client την ευθύνη να διατηρεί τα denormalized tables (τα primary indexes) συγχρονισμένα με τα βασικά tables. Ένα MV αυτόματα λαμβάνει τις ενημερώσεις από το αρχικό table.

Οι περιορισμοί στην δημιουργία ενός MV είναι:

- Όλες οι στήλες που αποτελούν το primary key του αρχικού table, πρέπει να είναι μέρος του primary key του MV.
- Μόνο μία νέα στήλη μπορεί να προστεθεί στο primary key του MV. Στήλες static δεν επιτρέπονται.
- Δε μπορούμε να φιλτράρουμε τις γραμμές που θα αντιγραφούν από το αρχικό table βάσει κάποια συνθήκης. Μπορούμε μόνο να παραλείψουμε γραμμές όπου κάποιες στήλες είναι NULL.

Τα MV είναι και αυτά σχεδιασμένα για υψηλής πληθικότητας δεδομένα. Επιπλέον, μειώνουν την απόδοση των writes , σε σχέση με τα απλά table. Για την ενημέρωση των MV η Cassandra εκτελεί ένα επιπλέον read-before-write και κάνει έλεγχο συνέπειας στα replicas. Παρουσιάζεται συνοπτικά η ακολουθία των λειτουργιών στο cluster, όταν δεδομένα εισάγονται/ανανεώνονται/διαγράφονται (mutation) στο αρχικό (base) table:

1. O coordinator στέλνει το mutation σε όλα τα base replicas και περιμένει για επιβεβαιώσεις.
2. Κάθε replica αποκτά τοπικό lock στο συγκεκριμένο partition το οποίο θα ανανεωθεί στο base table.
3. Κάθε replica διαβάζει τοπικά το partition από το base table.
4. Κάθε replica δημιουργεί ένα BatchLog με τις εντολές προς εκτέλεση στα views και εκτελεί ασύγχρονα το BatchLog στα view replicas
5. Κάθε replica εφαρμόζει το mutation τοπικά στο base table.

6. Κάθε replica απελευθερώνει το τοπικό lock στο partition και στέλνει επιβεβαίωση στον coordinator.
7. O coordinator αν λάβει τόσες επιβεβαιώσεις όσες ορίζει το Consistency Level, ενημερώνει τον client ότι το mutation έγινε επιτυχώς.

SSTable attached secondary indexes (SASI)

Από την Cassandra 3.4 και μετά, παρουσιάστηκαν τα SSTable Attached Secondary Indexes (SASI), ευρετήρια που βελτιώνουν την εκτέλεση των υπαρχόντων δευτερευόντων ευρετηρίων με καλύτερες αποδόσεις, για τις περιπτώσεις ερωτημάτων που μέχρι παλιότερα απαιτούσαν τη χρήση του ALLOW FILTERING. Το SASI απαιτεί σημαντικά λιγότερους πόρους, χρησιμοποιώντας λιγότερη μνήμη, δίσκο και CPU και υποστηρίζει ερωτήσεις σε strings για προθέματα και substrings που περιέχονται, αντίστοιχα με SQL υλοποίηση του LIKE = "foo%" ή LIKE = "%foo%". Επιπροσθέτως, υποστηρίζει SPARSE ευρετήρια για τη βελτίωση της απόδοσης ερωτημάτων εύρους σε μεγάλους και πυκνούς αριθμούς , όπως για παράδειγμα δεδομένα χρονοσειρών.

Το SASI υποστηρίζει όλα τα ερωτήματα που υποστηρίζονται ήδη από την CQL, καθώς και τον τελεστή LIKE χρησιμοποιώντας PREFIX, CONTAINS και SPARSE. Αν γίνεται χρήση του ALLOW FILTERING, υποστηρίζει επίσης ερωτήματα με πολλαπλά κατηγορήματα με τη χρήση του AND. Οι επιπτώσεις του φιλτραρίσματος δεν γίνονται αντιληπτές, αφού δεν εκτελείται φιλτράρισμα ακόμα και αν χρησιμοποιείται το ALLOW FILTERING.

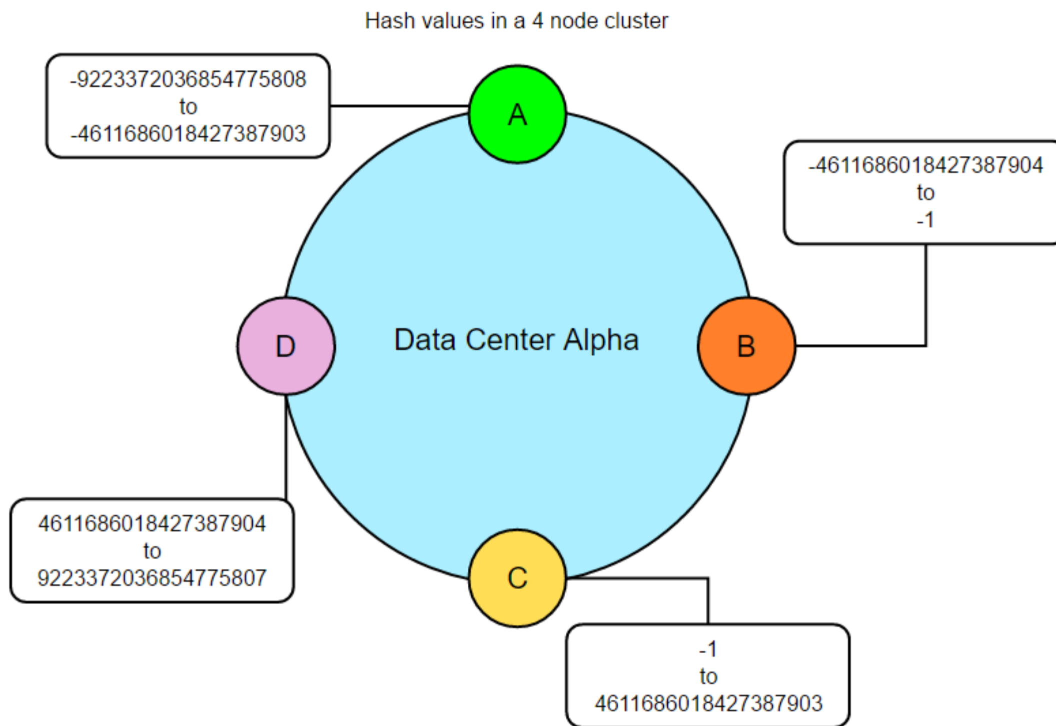
Τα SASI υλοποιούνται χρησιμοποιώντας B+ trees αντιστοιχιζόμενα στην μνήμη (memory mapped), μια αποτελεσματική δομή δεδομένων για ευρετήρια. Τα B+ trees επιτρέπουν στα ερωτήματα εύρους να εκτελούνται γρήγορα. Για κάθε SSTable δημιουργείται ένα SASI ευρετήριο και ακολουθεί τον κύκλο ζωής του. Όποτε ένα SSTable δημιουργείται στον δίσκο, ένα αντίστοιχο αρχείο SASI ευρετηρίου δημιουργείται. Πιο συγκεκριμένα, όταν μια αλλαγή δεδομένων πραγματοποιείται σε έναν κόμβο, πρώτα καταχωρείται στο CommitLog και μετά γράφεται στην μνήμη, στο MemTable. Την ίδια στιγμή, η αλλαγή ευρετηριοποιείται σε μια SASI δομή ευρετηρίου στην μνήμη (IndexMemtable). Αργότερα όταν τα MemTables θα γίνουν flushed στον δίσκο, θα δημιουργηθεί για το SASI ένα αρχείο OnDiskIndex για κάθε SSTable. Όταν πραγματοποιείται compaction(συγχώνευση) στα SSTables, τα αρχεία OnDiskIndex θα ακολουθήσουν τον κύκλο του compaction και θα συγχωνευτούν σε ένα μεγάλο OnDiskIndex αρχείο.

Προς το παρόν, το SASI δεν υποστηρίζει συλλογές. Θα πρέπει να δημιουργηθούν κλασικά δευτερεύοντα ευρετήρια στις συλλογές. Static στήλες υποστηρίζονται από την Cassandra 3.6 και μετά.

4.2.4 Αρχιτεκτονική

Ο σχεδιασμός της Cassandra βασίστηκε στην φιλοσοφία του ότι αστοχίες συστήματος/υλικού πάντα συμβαίνουν, και έτσι με βάση τη λογική αυτή σχεδιάστηκε ως ένα **peer-to-peer** κατανεμημένο σύστημα. Όλοι οι κόμβοι που το αποτελούν είναι ισότιμοι. Για να προστεθεί ένας καινούριος κόμβος στο cluster αρκεί να έχει κάποια πληροφορία για το πού βρίσκεται ένας άλλος κόμβος του συστήματος. 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). Οι αιτήσεις εγγραφής/ανάγνωσης ενός χρήστη (client) μπορούν να σταλούν σε οποιονδήποτε κόμβο του cluster. Όταν ο χρήστης συνδεθεί σε έναν κόμβο με ένα αίτημα, ο κόμβος λειτουργεί ως συντονιστής (**coordinator**) για το συγκεκριμένο αίτημα μεταξύ της εφαρμογής του χρήστη και των κόμβων που έχουν τα δεδομένα προς ζήτηση. Εκείνος καθορίζει σε ποιους κόμβους θα ανακατευθύνει το αίτημα ανάλογα με την κατανομή των δεδομένων και τον τρόπο που έχει στηθεί το cluster.

Τα δεδομένα είναι κατανεμημένα σε όλους τους κόμβους του cluster και αντιγράφονται και μοιράζονται αυτόματα και με διαφάνεια, ώστε να διασφαλίζεται υψηλή διαθεσιμότητα και ανοχή σε τυχόν βλάβες/αστοχίες του συστήματος. Για να αποφασίσει η Cassandra σε ποιο κόμβο θα τοποθετήσει μια γραμμή, εφαρμόζει στο partition key μια συνεπή συνάρτηση κατακερματισμού (consistent hashing) και παράγει μία τιμή (αναφέρεται ως token). Η ιδέα του consistent hashing χρησιμοποιείται για να ελαχιστοποιήσει την αναδιοργάνωση των δεδομένων όταν προστίθεται ή αφαιρείται ένας κόμβος. Η Cassandra διαμοιράζει το συνολικό εύρος των token σε όλους τους κόμβους του cluster. Κάθε κόμβος είναι υπεύθυνος για την αποθήκευση ενός εύρους τιμών token. Έτσι δημιουργείται ένα **token ring** όπως φαίνεται στην παρακάτω εικόνα, γι αυτό και το cluster στην Cassandra αναφέρεται συχνά ως δακτύλιος.



Εικόνα 21 Token Ring

Η συνάρτηση που χρησιμοποιείται για την παραγωγή των tokens ονομάζεται **partitioner**. Η Cassandra διαθέτει τους εξής partitioners:

- Murmur3Partitioner (προεπιλογή): κατανέμει ομοιόμορφα τα δεδομένα στο cluster χρησιμοποιώντας την *MurmurHash* συνάρτηση κατακερματισμού.
- RandomPartitioner: κατανέμει ομοιόμορφα τα δεδομένα στο cluster χρησιμοποιώντας την *MD5* κρυπτογραφική συνάρτηση κατακερματισμού.
- ByteOrderedPartitioner: δεν χρησιμοποιεί hash αλλά διαμοιράζει λεξικογραφικά τις γραμμές στους κόμβους βάσει των bytes του κλειδιού. Δεν προτείνεται η χρήση του και περιλαμβάνεται για συμβατότητα με προηγούμενες εκδόσεις.

Gossip

Όλοι οι κόμβοι μέσα στο cluster ανταλλάσουν μεταξύ τους πληροφορίες για την κατάσταση και την τοποθεσία τους μέσα στο cluster μέσω ενός πρωτοκόλλου το οποίο ονομάζεται Gossip. Το gossip είναι ένα peer-to-peer πρωτόκολλο επικοινωνίας κατά το οποίο οι κόμβοι περιοδικά ανταλλάσουν πληροφορίες για την κατάσταση τους και την κατάσταση των άλλων κόμβων για τους οποίους γνωρίζουν. Στην Cassandra η διαδικασία αυτή εκτελείται κάθε δευτερόλεπτο και ανταλλάσει μηνύματα με έως τρεις άλλους κόμβους μέσα στο σύστημα. Οι κόμβοι ανταλλάσουν πληροφορίες για τον εαυτό τους και για όλους τους κόμβους για τους

οποίους ενημερώθηκαν και με αυτό τον τρόπο πολύ γρήγορα όλοι οι κόμβοι μαθαίνουν για την κατάσταση που βρίσκονται όλοι οι υπόλοιποι κόμβοι του cluster. Ένα μήνυμα που ανταλλάσουν δύο κόμβοι μεταξύ τους συνδέεται με μία έκδοση έτσι ώστε παλιότερες εκδόσεις να μην λαμβάνονται υπόψη.

Replication

Η Cassandra αποθηκεύει αντίγραφα των δεδομένων σε πολλαπλούς κόμβους (replicas) , έτσι ώστε να διασφαλιστεί η αξιοπιστία του συστήματος και η ανοχή του σε σφάλματα. Ο συνολικός αριθμός των αντιγράφων που διατηρούνται στους κόμβους μέσα στο cluster καθορίζεται από τον παράγοντα αντιγραφής (replication factor) κατά την δημιουργία ενός keyspace στο σύστημα. Αν η παράμετρος replication factor οριστεί σε 1 αυτό σημαίνει πως υπάρχει μόνο ένα αντίγραφο των δεδομένων σε ένα κόμβο, αν οριστεί σε 2 τότε αυτό σημαίνει πως διατηρούνται δύο αντίγραφα σε δύο κόμβους και ούτω καθεξής. Όλα τα αντίγραφα είναι το ίδιο σημαντικά και δεν υπάρχει πρωταρχικό αντίγραφο μέσα στο cluster. Δύο είναι οι διαθέσιμες στρατηγικές αντιγραφής:

- Η απλή στρατηγική (Simple strategy).
- Η στρατηγική τοπολογίας δικτύου (Network topology strategy).

Η απλή στρατηγική χρησιμοποιείται μόνο όταν έχουμε ένα μοναδικό data center και ένα rack. Έχοντας οριστεί replication factor μεγαλύτερο από ένα, το πρώτο αντίγραφο τοποθετείται στον κόμβο που καθορίζεται από τον partitioner και κάθε επιπλέον αντίγραφο τοποθετείται στους επόμενους κόμβους με την φορά του ρολογιού μέσα στον δακτύλιο των κόμβων χωρίς να λαμβάνεται υπόψη η τοπολογία(η τοποθεσία του rack ή του data center).

Η στρατηγική τοπολογίας δικτύου χρησιμοποιείται όταν έχεις (ή σκοπεύεις να έχεις) το cluster κατανεμημένο σε πολλαπλά data centers. Η στρατηγική αυτή καθορίζει το πόσα αντίγραφα θα διατηρούνται σε κάθε data center και τοποθετεί τα αντίγραφα των δεδομένων σε διαφορετικά racks σε ένα data center. Η στρατηγική αυτή προσπαθεί να τοποθετήσει τα αντίγραφα σε διαφορετικά racks γιατί συνήθως οι κόμβοι οι οποίοι βρίσκονται στο ίδιο rack τίθενται μαζί εκτός λειτουργίας αν παρουσιαστούν προβλήματα στην ηλεκτροδότηση ή προβλήματα στο δίκτυο επικοινωνίας.

Σε ποιο data center και σε ποιο rack ανήκει ο εκάστοτε κόμβος, καθορίζεται από τα **snitches**. Αυτά είναι μηχανισμός που συλλέγει πληροφορίες για την τοπολογία του δικτύου και ενημερώνει την Cassandra έτσι ώστε να κατανείμει τα αντίγραφα σε data centers και racks, καθώς και να δρομολογεί τα αιτήματα αποτελεσματικά.

Ρυθμιζόμενη Συνέπεια

Η Cassandra είναι ένα AP (availability & partition tolerance) σύστημα, σύμφωνα με το θεώρημα CAP, που προσφέρει τελική συνέπεια (eventual consistency). Η Cassandra επεκτείνει αυτόν τον όρο προσφέροντας ρυθμιζόμενη συνέπεια, αφού επιτρέπει στον χρήστη να ορίσει το επίπεδο συνέπειας (Consistency Level) για κάθε λειτουργία που επιτελεί. Αυτό επιτρέπει στην Cassandra να λειτουργεί περισσότερο ως CP (consistency & partition tolerance) ή ως AP σύστημα, ανάλογα με τις απαιτήσεις της εφαρμογής.

Πιο συγκεκριμένα, για ένα αίτημα εγγραφής ο συντονιστής, με βάση το partition key και την στρατηγική αντιγραφής, θα προωθήσει το αίτημα σε όλα τα replicas που είναι υπεύθυνα για την συγκεκριμένη γραμμή προς εγγραφή. Ανεξαρτήτως του επιπέδου συνέπειας που όρισε ο χρήστης, όλα τα replicas εφόσον είναι διαθέσιμα θα πάρουν το αίτημα εγγραφής. Το επίπεδο συνέπειας στην εγγραφή καθορίζει πόσοι κόμβοι πρέπει να απαντήσουν ότι η εγγραφή ολοκληρώθηκε επιτυχώς για να θεωρηθεί το αίτημα εγγραφής επιτυχές.

Κατά την ανάγνωση, ο συντονιστής περιμένει απάντηση από τόσους κόμβους όσους ορίζει το επίπεδο συνέπειας. Αν οι κόμβοι αυτοί έχουν διαφορετικές εκδοχές της ζητούμενης γραμμής, θα επιστραφεί στον χρήστη η πιο πρόσφατη εκδοχή (βάσει της χρονοσφραγίδας). Πιο αναλυτικά, ο συντονιστής στέλνει το αίτημα ανάγνωσης σε έναν κόμβο (το πιο κοντινό replica). Έπειτα θα στείλει αιτήματα συνοπτικής ανάγνωσης (digest read) σε τόσους κόμβους όσους ορίζει το επίπεδο συνέπειας(σε αυτούς που θεωρεί ότι θα απαντήσουν πιο γρήγορα). Αν αυτό είναι ONE, δεν επικοινωνεί με άλλους replicas. Αν είναι QUORUM, τότε πρέπει συνολικά να καλέσει *(replication_factor / 2) + 1* replica-κόμβους. Σε αντίθεση με το πρώτο full-data read, όλα τα υπόλοιπα είναι συνοπτικές αναγνώσεις. Περιλαμβάνουν MD5 hash τιμές για την ζητούμενη γραμμή και χρονοσφραγίδες. Τα hash όλων των αιτημάτων, συμπεριλαμβανομένου και του πρώτου, θα συγκριθούν. Αν δεν ταυτιστούν, το replica με την πιο πρόσφατη εκδοχή θα πρέπει να στείλει (όχι digest) τα δεδομένα. Οι out-of-date κόμβοι θα ενημερωθούν για να επιλυθούν οι ασυνέπειες. Περιστασιακά, στέλνεται digest αίτημα σε όλα τα replicas ανεξαρτήτως του επιπέδου συνέπειας. Αυτό εξαρτάται από την παράμετρο *read_repair_chance* του table. Αυτή η διαδικασία επίλυσης ασυνεπειών ονομάζεται **read repair**.

Εικόνα 22 Επίπεδα συνέπειας

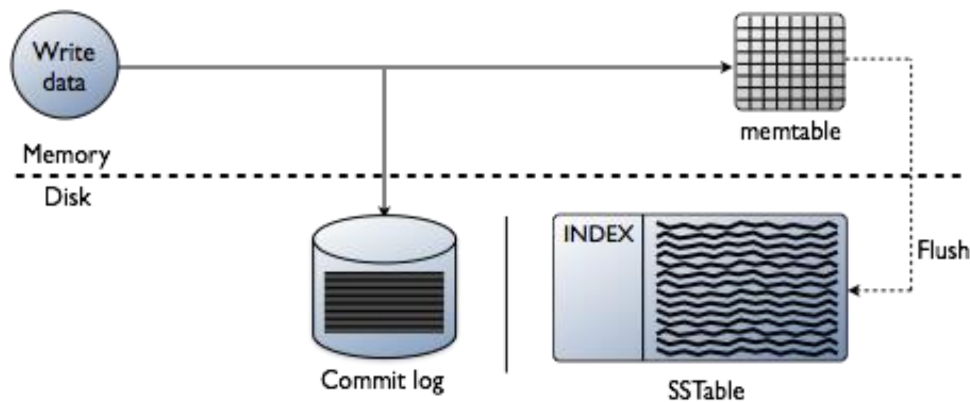
4.2.5 Storage Engine

Οι διάφοροι μηχανισμοί που χρησιμοποιεί η Cassandra βασικό κριτήριο έχουν την καλύτερη δυνατή επίδοση του συστήματος και την αποδοτικότερη χρησιμοποίηση των διαθέσιμων αποθηκευτικών μέσων.

Όταν πραγματοποιείται μια εγγραφή:

- Αρχικά, η Cassandra την καταγράφει στο **commit log**. Στο commit log καταγράφονται (append-only) όλες οι εγγραφές, για όλα τα column family, που πραγματοποιούνται σε έναν κόμβο και περιοδικά συγχρονίζεται με τον δίσκο. Χρησιμοποιείται για να ικανοποιούνται οι εγγυήσεις της μονιμότητας (durability) στα δεδομένα της βάσης αφού οι συγκεκριμένες καταχωρήσεις μπορούν να χρησιμοποιηθούν για επαναφορά των δεδομένων ακόμη και σε περίπτωση απώλειας ρεύματος σε κάποιο κόμβο.
- Στη συνέχεια τα δεδομένα γράφονται σε μια δομή εντός της μνήμης, το **memtable** , το οποίο λειτουργεί σαν write-back cache για τα δεδομένα των partitions, τα οποία η Cassandra μπορεί να αναζητήσει βάσει κλειδιού. Κάθε memtable αντιστοιχεί σε ένα table. Όταν τα περιεχόμενα του memtable ξεπεράσουν ένα (ρυθμιζόμενο) όριο ή όταν το commit log ξεπεράσει το συνολικό μέγεθος που έχει οριστεί, τα δεδομένα περνάνε στον δίσκο.
- Όταν ένα memtable γίνεται flushed στο δίσκο, αποθηκεύονται τα δεδομένα σε μια άλλη δομή που ονομάζεται **SStable** (Shorten String table). Επίσης, δημιουργείται το **partition index**, μια δομή που αντιστοιχίζει τα partition keys με την θέση τους στον

δίσκο. Οι αντίστοιχες εγγραφές στο commit log διαγράφονται. Κάθε SSTable αντιστοιχεί σε ένα table. Τα SSTables είναι immutable, δεν επανεγγράφουν τα δεδομένα τους. Όλες οι εγγραφές στον δίσκο γίνονται αξιοποιώντας την ταχύτητα σειριακών εγγραφών του δίσκου (sequential).



Εικόνα 23 Cassandra, Μηχανισμοί στη διαδικασία εγγραφής

Η Cassandra χρησιμοποιεί immutable SSTables. Αντί να επανεγγράφει τις υπάρχοντες γραμμές με inserts και updates, γράφει νέες εκδοχές των γραμμών αυτών με διαφορετική χρονοσφραγίδα σε νέα SSTables. Με το πέρασμα του χρόνου, μπορεί να υπάρχουν διάφορες εκδοχές μιας γραμμής σε διαφορετικά SSTables, όπου καθεμία έχει μοναδικό σύνολο στηλών και διαφορετική χρονοσφραγίδα. Έτσι για την ανάκτηση ολόκληρης της γραμμής απαιτείται πρόσβαση σε πολλά SSTables. Τα deletes τα χειρίζεται επίσης σαν εγγραφές. Δεν αφαιρεί τα δεδομένα, αλλά τα στιγματίζει με tombstones, μια σήμανση που ορίζει ότι προορίζονται για διαγραφή.

Compaction

Η Cassandra περιοδικά συγχωνεύει μια συλλογή από SSTables σε ένα μεγάλο SSTable. Η διαδικασία αυτή συλλέγει όλες τις εκδοχές για κάθε γραμμή και διαμορφώνει μια ολοκληρωμένη γραμμή, με τα πιο πρόσφατα δεδομένα για κάθε της στήλη(μέσω των χρονοσφραγίδων). Αυτή η συγχώνευση γίνεται αποδοτικά αφού τα δεδομένα στο SSTable είναι αποθηκευμένα ταξινομημένα βάσει του partition key, και δεν χρησιμοποιούνται τυχαία I/O. Οι παλιές εκδοχές των γραμμών και στιγματισμένες με tombstone γραμμές μένουν στα παλιά SSTables, τα οποία θα διαγραφούν όταν οι τρέχοντες αναγνώσεις τελειώσουν. Αναγνώσεις στο νέο SSTable μπορούν να γίνουν ακόμα και αν δεν έχει ολοκληρωθεί η εγγραφή του. Όταν ολοκληρωθεί, απελευθερώνει χώρο στον δίσκο και βελτιώνει την απόδοση των αναγνώσεων. Η Cassandra υποστηρίζει διαφορετικές στρατηγικές compaction,

που καθορίζουν ποιά SSTables θα επιλεγούν για συγχώνευση και πώς οι ταξινομημένες γραμμές θα ταξινομηθούν στο νέο SSTable. Κάθε στρατηγική έχει τα δικά της προτερήματα και η κατάλληλη επιλογή εξαρτάται από την εφαρμογή.

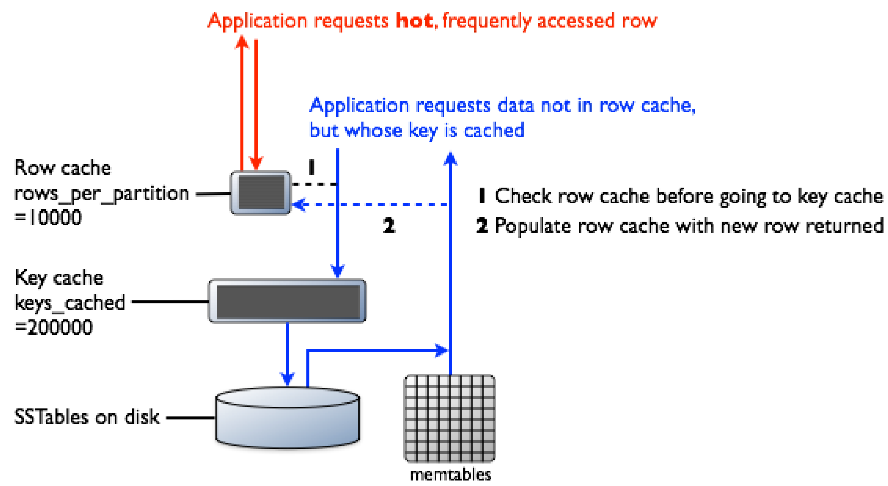
Compression

Η συμπίεση μεγιστοποιεί τις δυνατότητες αποθήκευσης μειώνοντας τον όγκο των δεδομένων και τα I/O. Η Cassandra συμπιέζει τα δεδομένα στα SSTables. Αυτό δεν επιβαρύνει την απόδοση των εγγραφών, αφού όπως είπαμε τα SSTables είναι immutable. Έτσι δε χρειάζεται να γίνει αποσυμπίεση των δεδομένων, επανεγγραφή και συμπίεση ξανά όπως γινόταν στις παραδοσιακές σχεσιακές βάσεις. Τα SSTables συμπιέζονται μία φορά κατά την εγγραφή τους στον δίσκο. Εγγραφές σε συμπιεσμένα tables, παρουσιάζουν έως 10% βελτίωση. Από την Cassandra 2.2 και μετά, επιτρέπεται συμπίεση του commit log που παρουσιάζει βελτίωση 6-12% στις εγγραφές. Η Cassandra υποστηρίζει τους αλγόριθμους συμπίεσης LZ4Compressor (προεπιλογή), SnappyCompressor, DeflateCompressor.

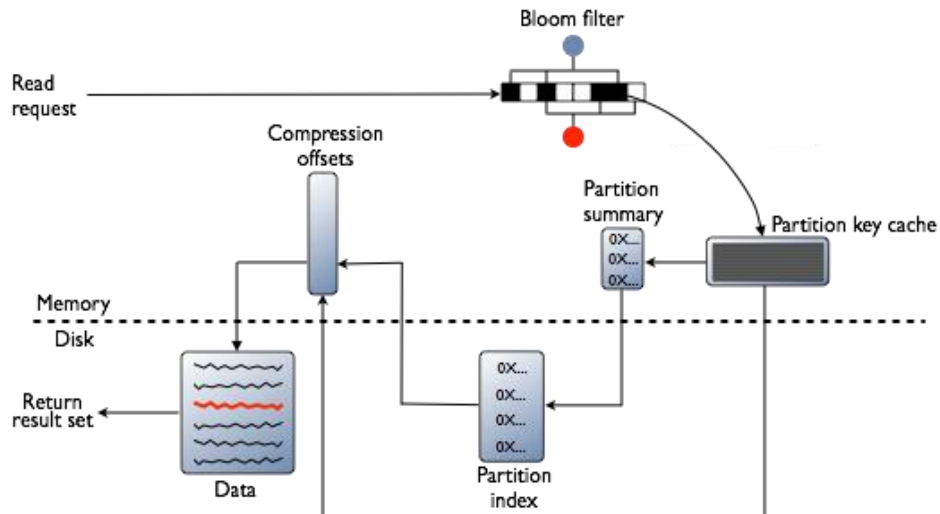
Για να ικανοποιήσει ένα read η Cassandra:

- Πρώτα ελέγχει αν το memtable περιέχει τα ζητούμενα δεδομένα. Αν ναι, τότε διαβάζονται και συγχωνεύονται αργότερα με τα δεδομένα από τα SSTables.
- Αν είναι ενεργοποιημένη, ελέγχει την **row cache**, η οποία αποθηκεύει ένα υποσύνολο των δεδομένων των SSTables στην μνήμη. Όταν γεμίσει χρησιμοποιεί LRU (least recently used) πολιτική αντικατάστασης για να επανακτήσει μνήμη. Αν ικανοποιήσει μια ανάγνωση, αποφεύγονται 2 αναζητήσεις στον δίσκο.
- Χρησιμοποιεί τα **bloom filters** τα οποία είναι ένας γρήγορος μηχανισμός για να απορριφθούν τα SSTables που σίγουρα δεν περιέχουν τα ζητούμενα δεδομένα. Βρίσκονται στην μνήμη και υπάρχει ένα για κάθε SSTable.
- Αν είναι ενεργοποιημένη, ελέγχει την **partition key cache**, η οποία αποθηκεύει ένα μέρος του partition index στην μνήμη. Καταλαμβάνει λίγο χώρο και με κάθε "hit" αποφεύγεται μία πρόσβαση στον δίσκο(το επόμενο βήμα δεν πραγματοποιείται).
- Απευθύνεται στο **partition summary**, μια δομή που αποθηκεύει δειγματοληπτικά κάποια primary keys και την θέση τους στον δίσκο στο primary index. Έτσι, μπορεί μετά να εκτελέσει μια πιο γρήγορη αναζήτηση του primary key που χρειάζεται από το **primary index** στον δίσκο.
- Στη συνέχεια αναζητά την τοποθεσία του τμήματος των συμπιεσμένων δεδομένων από ένα άλλο ευρετήριο που βρίσκεται στη μνήμη (**compression offsets**). Τα

συμπιεσμένα δεδομένα ανακτούνται από το/α SSTable(s), και επιστρέφεται το αποτέλεσμα στον χρήστη.



Εικόνα 24 Row cache and Key cache request flow



Εικόνα 25 Read request flow

4.3 Apache Spark

Το Apache Spark είναι μια υπολογιστική πλατφόρμα ειδικά σχεδιασμένη για cluster υπολογιστών (cluster computing platform) υλοποιημένη στην γλώσσα προγραμματισμού Scala. Είναι κατασκευασμένη ώστε να υποστηρίζει κατανεμημένες εφαρμογές γενικού σκοπού που βασίζονται εν γένει στην επεξεργασία μεγάλου όγκου δεδομένων, με μεγάλο βαθμό αποδοτικότητας και ταχύτητας.

Σε ένα γενικότερο πλαίσιο το Spark είναι προέκταση του προγραμματιστικού μοντέλου MapReduce, με τη κύρια διαφορά ότι υποστηρίζει περισσότερους τύπους υπολογισμών, όπως διαδραστικά ερωτήματα (interactive queries) και επεξεργασία δεδομένων συνεχούς ροής (streaming data processing). Μία ακόμα διαφορά σε σχέση με τις εργασίες (jobs) που μπορούν να ανατεθούν στο Spark σε σχέση με τις υλοποιήσεις του MapReduce σε άλλα συστήματα, είναι η ικανότητα που παρέχει για αποθήκευση δεδομένων στην μνήμη του κάθε κόμβου στο cluster κατά τη διάρκεια της εκτέλεσης του job. Αυτή η τελευταία ιδιότητα περιγράφεται ως caching και είναι ίσως ένα από τα πιο σημαντικά χαρακτηριστικά που εισάγει το Spark στην ανάπτυξη κατανεμημένων συστημάτων και του δίνει προβάδισμα στη ταχύτητα έναντι του Hadoop MapReduce πετυχαίνοντας μάλιστα μέχρι και 100 φορές καλύτερη χρονική απόδοση.

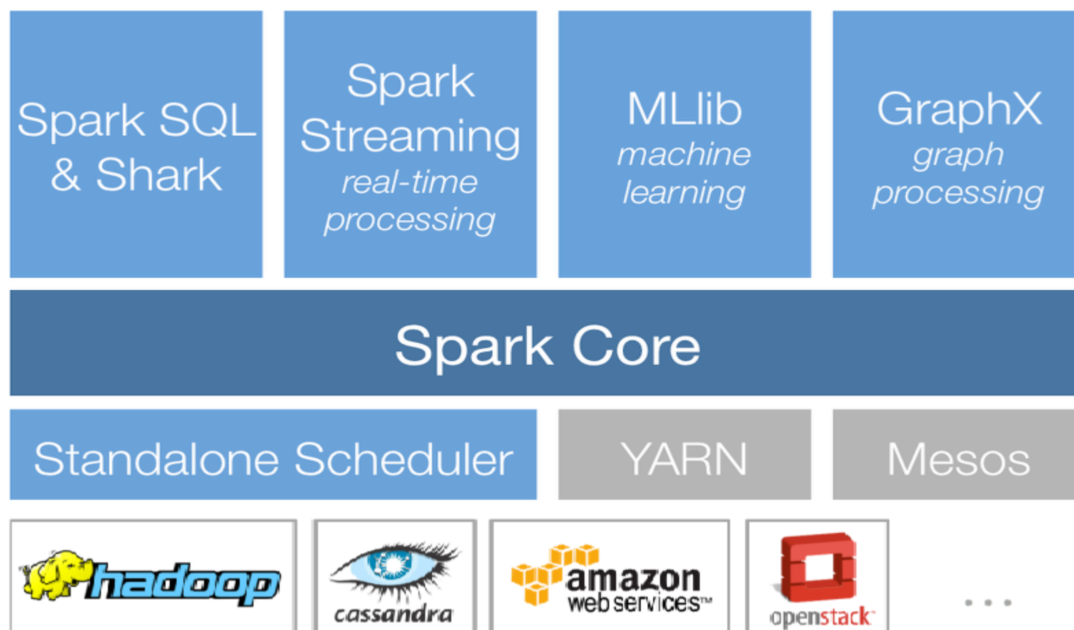
Το Spark όπως προαναφέραμε είναι υλοποιημένο εξ’ολοκλήρου στη γλώσσα προγραμματισμού Scala και προσφέρει APIs σε Scala, Python και Java για την ανάπτυξη εφαρμογών καθώς και πλούσιες ενσωματωμένες βιβλιοθήκες. Η Scala αποτελεί μία γλώσσα προγραμματισμού γενικού σκοπού με ισχυρά στοιχεία συναρτησιακού αλλά και αντικειμενοστραφούς προγραμματισμού. Η Scala χρησιμοποιεί το JVM περιβάλλον για την εκτέλεση των προγραμμάτων της, ενώ είναι σχεδιασμένη έτσι ώστε να μπορεί να εισάγει και να χρησιμοποιεί βιβλιοθήκες της Java.

4.3.1 Το οικοσύστημα του Spark

Επειδή ο πυρήνας του Spark είναι γρήγορος και γενικού σκοπού, υποστηρίζει και άλλα στοιχεία υψηλότερου επιπέδου, εξειδικευμένα για ένα συγκεκριμένο είδος επεξεργασίας. Αυτά τα στοιχεία είναι σχεδιασμένα να λειτουργούν στενά με τον πυρήνα, και μπορούν να χρησιμοποιηθούν σαν βιβλιοθήκες κατά την ανάπτυξη ενός προγράμματος.

Τα στοιχεία που αποτελούν το οικοσύστημα του Spark είναι:

- *Spark Core*: Είναι ο πυρήνας του Spark και περιέχει τις βασικές λειτουργίες του. Περιλαμβάνει τα απαραίτητα στοιχεία για τον χρονοπρογραμματισμό και την δρομολόγηση εργασιών, διαχείριση μνήμης, αποκατάστασης βλαβών, αλληλεπίδρασης με το σύστημα αποθήκευσης, και άλλα. Το core είναι επίσης αυτό που παρέχει τις διεπαφές για τα δομικά προγραμματιστικά στοιχεία του συστήματος όπως είναι για παράδειγμα τα Resilient Distributed Datasets (RDD) που θα αναλύσουμε αργότερα.
- *Spark SQL*: Παρέχει υποστήριξη ερωτημάτων SQL καθώς και γλώσσας παραπλήσιας της SQL που προσφέρει το Apache Hive και παρέχει αφαιρετικούς τύπους δεδομένων που υποστηρίζουν δομημένα και ημι-δομημένα δεδομένα.
- *Spark Streaming*: Αξιοποιεί την ικανότητα του Spark Core για γρήγορη δρομολόγηση ώστε να εκτελέσει ανάλυση ροής δεδομένων (streaming analytics). Χωρίζει τα δεδομένα σε μίνι σωρούς και εκτελεί μετασχηματισμούς RDD σε αυτούς τους μικρούς σωρούς δεδομένων.
- *MLlib*: Είναι μια βιβλιοθήκη που διαθέτει συλλογή αλγορίθμων μηχανικής μάθησης υλοποιημένων με προσανατολισμό την κατανεμημένη εφαρμογή τους. Εξαιτίας του ότι μεγάλο μέρος της αρχιτεκτονικής του Spark είναι βασισμένη στην κατανεμημένη μνήμη είναι έως και 9 φορές γρηγορότερο απ’ το βασισμένο στο δίσκο Apache Mahout.
- *GraphX*: Είναι μια βιβλιοθήκη για το χειρισμό γράφων και την παράλληλη εκτέλεση υπολογισμών σε γράφους. Παρέχει ένα API για την έκφραση ενός υπολογισμού γράφου που μπορεί να μοντελοποιηθεί στην αφαίρεση Pregel και μάλιστα βελτιστοποιημένης εκτέλεσης.



Εικόνα 26 Το οικοσύστημα του Spark

Το Spark απαιτεί ένα διαχειριστή του cluster και ένα κατανεμημένο σύστημα αποθήκευσης. Ως διαχειριστή του cluster το Spark υποστηρίζει τους Standalone (ενσωματωμένη υλοποίηση που παρέχει το spark), Hadoop Yarn και Apache Mesos. Ως κατανεμημένο σύστημα αποθήκευσης υποστηρίζεται μια ευρεία γκάμα συστημάτων, συμπεριλαμβανομένων των Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, HBase, Hive, OpenStack Swift, Amazon S3, Kudu ή άλλων προσαρμοσμένων λύσεων.

4.3.2 Αρχιτεκτονική

Η αρχιτεκτονική του Spark ακολουθεί το αρχιτεκτονικό μοντέλο master/slave. Πιο ειδικά στη ορολογία του Spark υπάρχουν οι οντότητες του master και των workers. Όταν εκκινείται ένα Spark cluster μία διεργασία master εκτελείται στον κόμβο από όπου δίνεται η εντολή εκκίνησης, ενώ διεργασίες workers εκτελούνται στους κόμβους slaves. Εν γένει μια εφαρμογή που ξεκινάει να εκτελείται στο Spark χρειάζεται υπολογιστικούς πόρους (κύρια μνήμη και πυρήνες cpu). Την εποπτεία των πόρων του cluster την αναλαμβάνει ο διαχειριστής του cluster (Standalone, Yarn, Mesos). Οι διεργασίες του master και των worker αποτελούν κομμάτια του resouce manager και εκκινούν έχοντας υπό την επίβλεψή τους ένα απόθεμα πόρων του κόμβου στον οποίον εκτελούνται. Εκτός αυτού διατηρείται συνεχής επικοινωνία μεταξύ των κόμβων workers και του κόμβου master με στόχο τον εντοπισμό

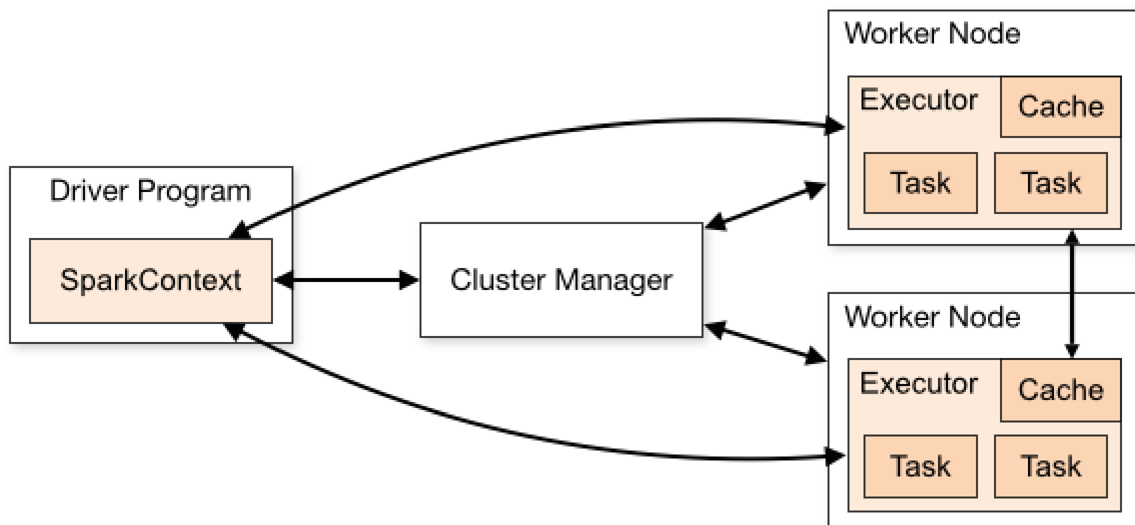
τυχόν προβλημάτων. Τέλος οι διεργασίες αυτές εγκαθιστούν servers σε κάθε κόμβο ώστε να παρέχουν χρήσιμες πληροφορίες της λειτουργίας τους μέσω του πρωτοκόλλου http.

Οι δομικές μονάδες που συνιστούν μία εφαρμογή Spark κατά τη διάρκεια εκτέλεσής της είναι αυτές του driver και των executors:

O driver είναι η διεργασία όπου βρίσκεται η εφαρμογή του πελάτη. Σε αυτή υπάρχει το αντικείμενο SparkContext που αντιπροσωπεύει και ενθυλακώνει όλες τις επιλεγμένες ρυθμίσεις και παραμέτρους για ένα συγκεκριμένο job. Όταν ξεκινά να εκτελείται η εφαρμογή, ευθύνη του driver είναι να μετατρέψει τον κώδικα σε tasks τα οποία θα μπορέσουν να ανατεθούν στους executors που βρίσκονται διαθέσιμοι στους workers του cluster. Αυτή η διαδικασία ξεκινά με τον driver πρώτα να καταστρώνει ένα λογικό πλάνο εκτέλεσης του job (logical plan), το οποίο έχει την μορφή κατευθυνόμενου ακυκλικού γραφήματος, (DAG) των διαδικασιών που πρέπει να τελεστούν. Έπειτα, μεταφράζει το λογικό πλάνο σε φυσικό πλάνο επιχειρώντας διάφορες βελτιστοποιήσεις στον τρόπο εκτέλεσης που μπορεί να πραγματοποιήσει. Στην συνέχεια, ακολουθεί η δρομολόγηση των tasks στους executors ώστε να ξεκινήσει το κυρίως σώμα του υπολογισμού. Ο driver κάνει έξυπνες επιλογές, αναθέτοντας τα tasks όσο είναι δυνατό στους κόμβους οι οποίοι έχουν τοπικά τα δεδομένα τα οποία θα χρειαστεί το εν λόγω task. Αυτά μπορεί να αφορούν είτε δεδομένα αποθηκευμένα στο τοπικό δίσκο του κόμβου και θα προσπελαστούν για πρώτη φορά είτε ακόμα και δεδομένα που έχουν αποθηκευτεί στη cache του κόμβου από προγενέστερο task που εκτελέστηκε εκεί. Σε οποιαδήποτε από τις δυο παραπάνω περιπτώσεις ο driver θα προτιμήσει πάντα το συγκεκριμένο κόμβο που πληρεί τα κριτήρια αυτά. Αυτό όπως είναι φυσικό δεν είναι πάντοτε εφικτό, αφού μπορεί ο προτιμητέος κόμβος να είναι απασχολημένος την συγκεκριμένη στιγμή με την εκτέλεση ενός άλλου task. To spark παρέχει και σχετική ρύθμιση που ορίζει πόσο μπορεί να αναμένει ο driver μέχρι να αποδεσμευτεί ένας κόμβος ώστε τελικά να του αναθέσει το task.

Οι executors είναι java διεργασίες που εκκινούν οι workers σε κόμβους του cluster. Εκκινούνται στην αρχή κάθε job και η διάρκεια ζωή τους είναι όση και η διάρκεια ζωής του job. Οι executors είναι αυτοί που αναλαμβάνουν να κομμάτι του υπολογισμού που χρειάζεται για το job. Αυτό γίνεται με την ανάθεση ανεξάρτητων μονάδων υπολογισμού (tasks) σε κάθε executor. Τα αποτελέσματα της δουλειάς τους διαφέρουν ανάλογα με το είδος του task. Αυτά είτε ανακοινώνονται πίσω στο driver που παρακολουθεί την διαδικασία εκτέλεσης είτε αποθηκεύονται στο δίσκο ή στη cache ως ενδιάμεσα αποτελέσματα που θα καταναλωθούν από μελλοντικά task. Τέλος είναι σημαντικό να τονίσουμε ότι η ανεξαρτησία των task ως

υπολογιστικής μονάδας συνεπάγεται ότι όταν για κάποιο λόγο υπάρξει αποτυχία ολοκλήρωσης ενός task αυτό δεν επηρεάζει την εξέλιξη της εκτέλεσης του job παρά μόνο στο γεγονός ότι θα πρέπει να επανακινηθεί ένα πανομοιότυπο task.



Εικόνα 27 Αρχιτεκτονική του Spark

4.3.3 Resilient Distributed Datasets (RDD)

Ένα resilient distributed dataset είναι μια αμετάβλητη (immutable) κατανεμημένη συλλογή αντικειμένων. Ουσιαστικά είναι μια δομή που καλύπτει με ένα επίπεδο αφαιρετικότητας (abstraction) τα προς επεξεργασία δεδομένα που χρησιμοποιούνται σε jobs του spark. Το σύνολο των αντικειμένων μπορεί να αποτελείται από οποιοδήποτε τύπο δεδομένο υποστηρίζουν οι γλώσσες Python, Scala και Java είτε ακόμα και κλάσεις οριζόμενες από τον χρήστη.

Όλα τα jobs που υποβάλλονται στο Spark ξεκινούν ορίζοντας ένα RDD από μία πηγή δεδομένων (π.χ HDFS , Cassandra) ή από ένα ήδη υπάρχον RDD. Από τη στιγμή που ορίζεται ένα RDD αυτό πλέον ορίζει μια απεικόνιση των πραγματικών δεδομένων από τα οποία δημιουργήθηκε και διασπάται σε κομμάτια (partitions) που μπορούν να υπολογιστούν σε ξεχωριστούς κόμβους του cluster. Τα RDDs υποστηρίζουν δύο τύπους λειτουργιών : μετασχηματισμούς (transformations), οι οποίοι δημιουργούν ένα νέο σύνολο δεδομένων RDD από ένα υπάρχον, και πράξεις (actions), που επιστρέφουν τιμές στο πρόγραμμα driver μετά την εκτέλεση επεξεργασίας σε ένα σύνολο δεδομένων. Οι μετασχηματισμοί λειτουργούν με

οκνηρή αποτίμηση, δεν εκτελείται επομένως κάποιο job έως ότου κάποιο action το «πυροδοτήσει».

Ένα σημαντικό πλεονέκτημα που παρουσιάζουν τα RDD είναι ο τρόπος που μοντελοποιούνται από το Spark ώστε να παρέχουν μηχανισμούς ανάκτησής τους σε περίπτωση προβλήματος. Συγκεκριμένα για κάθε RDD ανά πάσα στιγμή αποθηκεύεται η σειρά των μετασχηματισμών (lineage) που ακολουθήθηκαν για να κατασκευαστεί καθώς και πληροφορίες για την πηγή των δεδομένων που χρησιμοποιήθηκε για να κατασκευαστεί το συγκεκριμένο RDD. Τέλος μία από τις πιο σημαντικές ικανότητες του Spark είναι η διατήρηση (persisting) ή προσωρινή αποθήκευση (caching) ενός συνόλου δεδομένων στη μνήμη. Όταν διατηρείται ένα RDD, κάθε κόμβος κρατά αποθηκευμένα όλα τα partitions του συγκεκριμένου RDD στη μνήμη και τα επαναχρησιμοποιεί εκτελώντας άλλες διεργασίες σε αυτό το σύνολο δεδομένων. Αυτό επιτρέπει μελλοντικές ενέργειες να είναι πολύ πιο γρήγορες, συχνά περισσότερο από δέκα φορές. Την πρώτη φορά που θα εκτελεστεί ένα action, θα διατηρηθεί στη μνήμη των κόμβων.

5

Υλοποίηση και Αποτελέσματα Διπλωματικής

Αρχικά, αναφέρουμε τις εκδόσεις όλων των εργαλείων και των driver που χρησιμοποιήθηκαν:

- Java 8
- MongoDB 3.2.9
- mongodb java driver 3.3.0
- Cassandra 3.7.0 , CQL 3.4.2
- cassandra java driver 3.0.0
- Scala 2.11.8
- Spark 2.0.0
- spark-cassandra connector v2.0.0-M2

5.1 Μοντέλο Δεδομένων

Όπως αναφέραμε στην περιγραφή του προβλήματος, το σύνολο δεδομένων που θα μας απασχολήσει αποτελείται από αρχεία 3D σκηνών και οι mp7 περιγραφές τους. Κάποιες απ' τις σκηνές αυτές ξεπερνούν τα 50MB. Στην MongoDB το ανώτερο επιτρεπτό μέγεθος ενός εγγράφου είναι 16MB. Στην Cassandra δεν υπάρχει όριο στο μέγεθος ενός partition, αλλά μόνο στον αριθμό των στηλών του (2^{31}). Το value κάθε στήλης όμως , παρόλο που έχει θεωρητικό όριο στα 2GB, δεν προτείνεται (από το επίσημο documentation) να ξεπερνάει το

1MB. Η λύση είναι να διαχωρίσουμε τα αρχεία σε μικρότερα κομμάτια (chunks) και να τα αποθηκεύσουμε με τέτοιο τρόπο ώστε να μπορούμε να τα ανακτήσουμε και να ανακατασκευάσουμε το αρχικό αρχείο.

Η Cassandra δεν παρέχει τέτοιο feature. Θα έπρεπε να διασπαστεί κάθε αρχείο είτε σε περισσότερες στήλες είτε σε wide-row partition με clustering στήλη. Ακόμη, είναι αποδοτικό μεγάλα αρχεία να διαχωρίζονται σε περισσότερα partitions ώστε να αυτά να αποθηκεύονται κατανεμημένα και να επιτρέπουμε έτσι την ανάγνωση από πολλαπλούς servers ταυτόχρονα. Τέτοια projects στην Cassandra δεν θα μας απασχολήσουν στην διπλωματική αυτή. Έχουν υλοποιηθεί από διάφορες εταιρείες, όπως το Astyanax της Netflix και το CFS (Cassandra File System) στην εμπορική έκδοση της Cassandra (η οποία λέγεται “DataStax Enteprise”) που παρέχεται από την DataStax.

Η MongoDB υποστηρίζει τέτοιες λειτουργίες μέσω του GridFS,το οποίο είναι απαραίτητο για αποθήκευση και ανάκτηση αρχείων που ξεπερνούν τα 16MB. Το GridFS διαχωρίζει τα αρχεία σε πολλά κομμάτια (chunks) των 255KB και αποθηκεύει κάθε chunk ως ξεχωριστό έγγραφο. Σε κάθε ένα chunk προσθέτει πεδίο file_id με τιμή, το αναγνωριστικό _id του αρχικού εγγράφου. Χρησιμοποιεί 2 συλλογές, μία για την αποθήκευση των chunks (chunks collection) και μία για την αποθήκευση διαφόρων metadata των αρχείων (files collection). Εμείς αποθηκεύουμε τα binaries αρχεία των 3D σκηνών στην chunks collection. Τα binaries, ωστόσο, δε θα μας απασχολήσουν. Όλα τα ερωτήματα και οι λειτουργίες που θα εκτελέσουμε αφορούν τις mp7 περιγραφές τους, οι οποίες αποθηκεύονται ξεχωριστά.

Mpeg-7 Περιγραφές

Όσον αφορά την MongoDB, αποθηκεύσαμε τις mp7 περιγραφές στην files collection του GridFS. Σε αυτήν την συλλογή κάθε έγγραφο αφορά ένα αρχείο, μία 3D σκηνή. Εμείς μετατρέψαμε κάθε mp7 περιγραφή από xml σε json και την αποθηκεύσαμε κάτω από πεδίο metadata ως εμφωλευμένο έγγραφο. Η μετατροπή αυτή είναι απλή και γρήγορη διαδικασία λόγω των ομοιοτήτων των δύο μορφών. Παρατίθεται τμήμα ενός mp7 στην αρχική xml μορφή του και ακολούθως σε json .

```
...
  <DescriptorCollection id="Geometry_w9ab1b3b9b1b1" name="Geometry_Lower_fuselage">
        <Descriptor xsi:type="BoundingBox3DType">
                 <BoundingBox3DSize BoxWidth="11.1" BoxHeight="11.1" BoxDepth="11.1"/>
                 <BoundingBox3DCenter BoxCenterW="4.5" BoxCenterH="4.5" BoxCenterD="4.5"/>
        </Descriptor>
        <Descriptor ...
    </DescriptorCollection>
...
```

Εικόνα 28 Παράδειγμα mp7 (xml)

```
"DescriptorCollection":{
   "id" : "Geometry_w9ab1b3b9b1b1",
   "name": "Geometry_Lower_fuselage",
   "Descriptor": [
      {
       "type": "BoundingBox3DType",
       "BoundingBox3DSize": {
          "BoxWidth": 11.1 , "BoxHeight":  11.1 , "BoxDepth":  11.1
       },
       "BoundingBox3DCenter": {
          "BoxCenterW":  4.5 , "BoxCenterH":  4.5 , "BoxCenterD":  4.5
       }
     },
   ...
  ]
}
```

Εικόνα 29 Μετατροπή mp7 σε json

Στην Cassandra τα δεδομένα μας δεν είναι free-schema, όπως στην MongoDB, αλλά έχουν ένα ευέλικτο σχήμα το οποίο ορίζεται κατά την δημιουργία του Table. Ο πλήρης ορισμός του mpeg-7 σχήματος αποτελεί το κυριότερο πρόβλημα στην περίπτωσή μας. Μπορεί να μην έχουμε φυσικά το πρόβλημα των null values (που αναφέραμε στις SQL βάσεις) αλλά ο ορισμός όλων των διαφορετικών descriptors και οποιουδήποτε πεδίου εμφανίζεται στο σχήμα είναι αδύνατος. Επιπρόσθετο πρόβλημα αποτελεί η ιεραρχική δομή των δεδομένων μας. Κάθε πληροφορία είναι εμφωλευμένη σε πολλαπλά επίπεδα, κάτι το οποίο διαχειρίζεται εξαιρετικά η MongoDB με τα json έγγραφα. Στην Cassandra για την δημιουργία σχήματος με json-like δεδομένα χρησιμοποιούνται συνήθως τύποι ορισμένοι από τον χρήστη (user defined types - UDT) οι οποίοι εισήχθησαν στην έκδοση 2.1 και μπορούν να περιλαμβάνουν ως πεδία οποιοδήποτε τύπο, συλλογή ή άλλα UDTs. Το πρόβλημα εδώ είναι οι *frozen* τύποι όταν έχουμε παραπάνω από 2 επίπεδα φωλιάσματος. Μια frozen τιμή σειριοποιεί πολλαπλές συνιστώσες σε μία μοναδική τιμή. Οι non-frozen τύποι επιτρέπουν ενημερώσεις σε συγκεκριμένα πεδία. Η Cassandra χειρίζεται την τιμή των frozen τύπων σαν blob. Ολόκληρη η τιμή πρέπει να επανεγγραφεί. Η Cassandra από την έκδοση 3.6 επιτρέπει non-frozen UDT στους ορισμούς των tables μόνο εφόσον δεν περιέχουν συλλογές. Επιπλέον κάθε UDT που συμμετέχει στον ορισμό κάποιου άλλου UDT πρέπει να είναι frozen. Επομένως με τον ορισμό σχήματος των εμφωλευμένων δεδομένων μας με UDTs δε θα μπορούσαμε να ενημερώσουμε ξεχωριστά πεδία. Δηλαδή στην περίπτωση μας π.χ. να τροποποιήσουμε τους Descriptors ή να προσθέσουμε νέους. Βεβαίως αυτό δεν αποτελεί πρόβλημα αν θεωρήσουμε ότι τα δεδομένα μας είναι immutable, και κάθε περιγραφή mp7 γράφεται μία φορά κατά την εισαγωγή της στην βάση.

Για να αποφύγουμε τα παραπάνω προβλήματα, φτιάξαμε ένα γενικότερο σχήμα το οποίο διατηρεί όλες τις πληροφορίες ενός XML αρχείου. Φτιάξαμε ένα πρόγραμμα για την

εισαγωγή των mp7 περιγραφών στη βάση, το οποίο βασίζεται στη χρήση parser για την διάσχιση του xml αρχείου. Συγκεκριμένα, εκτελεί μια αναδρομική pre-order διάσχιση σε όλους τους κόμβους του δένδρου του xml. Εισάγει στην βάση κάθε πληροφορία είτε αυτή βρίσκεται σε κόμβο-φύλλο περικλειόμενη από κάποιο tag, είτε ως attribute. Κάθε πληροφορία που εισάγεται χαρακτηρίζεται από ένα όνομα πεδίου , το οποίο είναι το όνομα του tag ή του attribute, μια τιμή πεδίου και το μονοπάτι στο xml δέντρο από το root element έως τον κόμβο της πληροφορίας. Υπάρχουν σημεία στα μονοπάτια, όπου εμφανίζονται στη σειρά πολλά tags με το ίδιο όνομα, όπως το "Descriptor" στο προηγούμενο παράδειγμα, το οποίο μετατράπηκε σε json array. Έτσι, για κάθε πληροφορία χρειάζεται και η θέση της για κάθε tag στο μονοπάτι της που εμφανίζεται πολλές φορές. Τέλος, το partition key που χρησιμοποιήθηκε είναι το όνομα του αρχείου.

```
CREATE TABLE mp7 (
    filename text,
    path text,
    field_name text,
    positions frozen<map<text, int>>,
    field_value text,
    PRIMARY KEY (filename, path, field_name, positions)
) WITH CLUSTERING ORDER BY (path ASC, field_name ASC, positions ASC)
```

Κώδικας 2 mp7 Table

Το παραπάνω παράδειγμα στην CQL εμφανίζεται ως εξής :

filename	path	field_name	positions	field_value
exampleFile	...DescriptionCollection.	id	{}	Geometry_w9ab1b3b9b1b1
exampleFile	...DescriptionCollection.	name	{}	Geometry_w9ab1b3b9b1b1
exampleFile	...DescriptionCollection.Descriptor.	type	{'Descriptor': 0}	BoundingBox3DType
exampleFile	...DescriptionCollection.Descriptor.BoundingBox3DCenter.	BoxCenterD	{'Descriptor': 0}	4.5
exampleFile	...DescriptionCollection.Descriptor.BoundingBox3DCenter.	BoxCenterH	{'Descriptor': 0}	4.5
exampleFile	...DescriptionCollection.Descriptor.BoundingBox3DCenter.	BoxCenterW	{'Descriptor': 0}	4.5
exampleFile	...DescriptionCollection.Descriptor.BoundingBox3DSize.	BoxDepth	{'Descriptor': 0}	11.1
exampleFile	...DescriptionCollection.Descriptor.BoundingBox3DSize.	BoxHeight	{'Descriptor': 0}	11.1
exampleFile	...DescriptionCollection.Descriptor.BoundingBox3DSize.	BoxWidth	{'Descriptor': 0}	11.1

Εικόνα 30 Μετατροπή αρχείου mp7 σε table της Cassandra

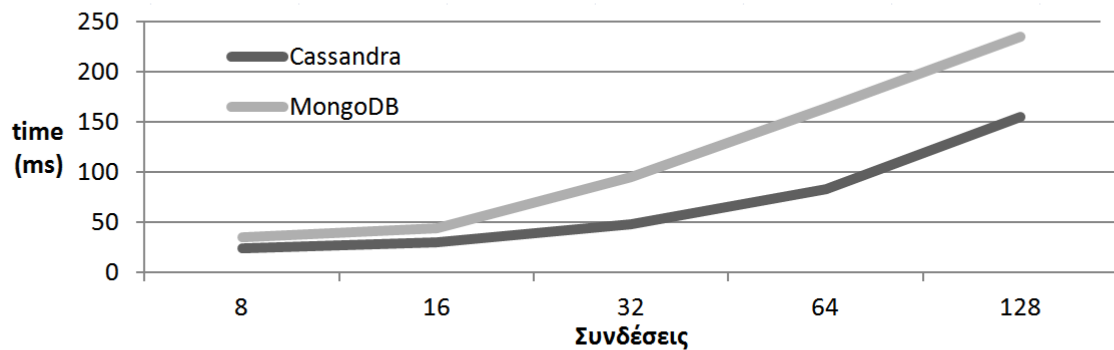
Όμως, η εσωτερική αναπαράσταση των δεδομένων, όπως αναφέραμε και σε προηγούμενο κεφάλαιο, είναι λίγο διαφορετική. Όλες οι γραμμές με κοινό partition key αποθηκεύονται μαζί , και οι clustering στήλες διαμορφώνουν στην ουσία το όνομα των στηλών. Το partition μοιάζει ως εξής :

exampleFile	...DescriptionCollection : id : {}	...DescriptionCollection : name : {}	...
	Geometry_w9ab1b3b9b1b1	Geometry_w9ab1b3b9b1b1	...

5.2 *Αναγνώσεις δεδομένων*

Αρχικά να αναφέρουμε ότι και οι δύο βάσεις χρησιμοποιούν διαθέσιμη RAM ως cache για τα δεδομένα τους. Στην cassandra η row cache είναι βέβαια απενεργοποιημένη, αλλά το λειτουργικό σύστημα κάνει map σελίδες (pages) των SSTables στην filesystem cache. Η Cassandra δε παρουσίασε έντονα φαινόμενα cold start. Αντίθετα, η MongoDB - η οποία καταλαμβάνει όλη την διαθέσιμη μνήμη χρησιμοποιώντας την WiredTiger cache και την filesystem cache - στην πρώτη της αναφορά σε κάποια collection προσπαθεί να μεταφέρει στην μνήμη τα δεδομένα της συλλογής αλλά και τα ευρετήρια που χρησιμοποιεί. Αυτό είναι ιδιαίτερα χρήσιμο στην δικιά μας περίπτωση όπου το working set χωράει στην μνήμη. Έτσι, πριν την εκτέλεση των λειτουργιών που παρουσιάζονται στο κεφάλαιο αυτό, εκτελούμε κάποια ερωτήματα για την τοποθέτηση δεδομένων και ευρετηρίων στην cache (η διαδικασία αυτή ονομάζεται “warming up”). Ο χρόνος του warming up δε συνυπολογίζεται.

Για την προσομοίωση της λειτουργίας ενός server που εξυπηρετεί αναγνώσεις δεδομένων από πολλαπλούς clients, γράψαμε ένα πρόγραμμα (java) το οποίο εκτελεί ασύγχρονες αναγνώσεις διαφορετικών αρχείων mp7. Κάθε ασύγχρονη εκτέλεση δημιουργεί μια νέα σύνδεση, στέλνει το αίτημα στην βάση και επιστρέφει στο client πριν την ανάκτηση των δεδομένων. Με αυτό τον τρόπο η εκτέλεση κάποιου ερωτήματος δε μπλοκάρει την εκτέλεση των επόμενων. Κάθε αίτημα αφορά ένα mp7 αρχείο. Και οι δύο βάσεις παρέχουν API για την πραγματοποίηση ασύγχρονων λειτουργιών. Για την απόκτηση των αποτελεσμάτων χρησιμοποιήσαμε Callback συναρτήσεις για την διαχείριση της απάντησης που επιστρέφει η βάση σε περιπτώσεις επιτυχίας ή αποτυχίας του αιτήματος. Μετρήσαμε τον χρόνο μέχρι την επιστροφή και της τελευταίας απάντησης για 8, 16, 32, 64, 128 διαφορετικά αρχεία. Όπως βλέπουμε στα αποτελέσματα η Cassandra στην συγκεκριμένη λειτουργία παρουσιάζει λίγο καλύτερες επιδόσεις.



Διάγραμμα 1 Ασύγχρονη ανάκτηση αρχείων

Ας θεωρήσουμε τώρα την περίπτωση όπου χρειαζόμαστε να διαβάσουμε κάποια συγκεκριμένη πληροφορία απ' όλες τις mp7 περιγραφές. Για παράδειγμα, στο ερώτημά μας θέλουμε τους τύπους σχήματος (Extrusion, Cylinder, Box, Sphere κλπ) όλων των αντικειμένων για κάθε σκηνή. Ενώ στην προηγούμενη περίπτωση μετράγαμε μόνο τον χρόνο ανάκτησης των αρχείων, τώρα μετράμε τον χρόνο μέχρι την πρόσβαση στην συγκεκριμένη πληροφορία. Η πληροφορία αυτή στην MongoDB είναι εμφωλευμένη σε πολλά επίπεδα και για την πρόσβαση σε αυτήν πρέπει, μετά την ανάκτηση των εγγράφων, να χρησιμοποιήσουμε κάποιο parser και να τα διασχίσουμε. Χρησιμοποιούμε "*projecting*" ώστε να μην μεταφέρεται στον client ολόκληρο το κάθε έγγραφο, αλλά μόνο τα επιλεγμένα πεδία, τα οποία βέβαια εξακολουθούν να είναι εμφωλευμένα. Δεν απαλλασσόμαστε από την διαδικασία του parsing και της διάσχισης, αλλά την επιταχύνουμε και επιπλέον μειώνουμε την κίνηση στο δίκτυο, γεγονός το οποίο σε ένα μεγάλο cluster αποτελεί μεγάλο κέρδος. Παρόλα αυτά το parsing και η διάσχιση σε βάθος κάθε εγγράφου της συλλογής ξεχωριστά είναι πολύ χρονοβόρα διαδικασία, η οποία στο παράδειγμά μας αποτελούσε το 70% του συνολικού χρόνου. Για να ξεπεράσουμε αυτό το πρόβλημα χρησιμοποιούμε το *aggregation pipeline* με στάδια $match και $project. Με το aggregation pipeline καταφέρνουμε να επιστρέψουμε τα αποτελέσματα σε μορφή την οποία την διαμορφώνουμε εμείς. Η εντολή που χρησιμοποιούμε είναι η εξής:

```
db.getCollection('Savage.files').aggregate([{$match:{
"metadata.Mpeg7.Description.MultimediaContent.StructuredCollection.Collection.Descriptor
Collection.Descriptor.Geometry3D.ObjectType": { $exists: true } }},
 {$project: {
name:"$filename" ,
objects:"$metadata.Mpeg7.Description.MultimediaContent.StructuredCollection.Collection.
DescriptorCollection.Descriptor.Geometry3D.ObjectType"} } ])
```

Κώδικας 3 Τύποι αντικειμένων για όλες τις σκηνές με aggregation pipeline

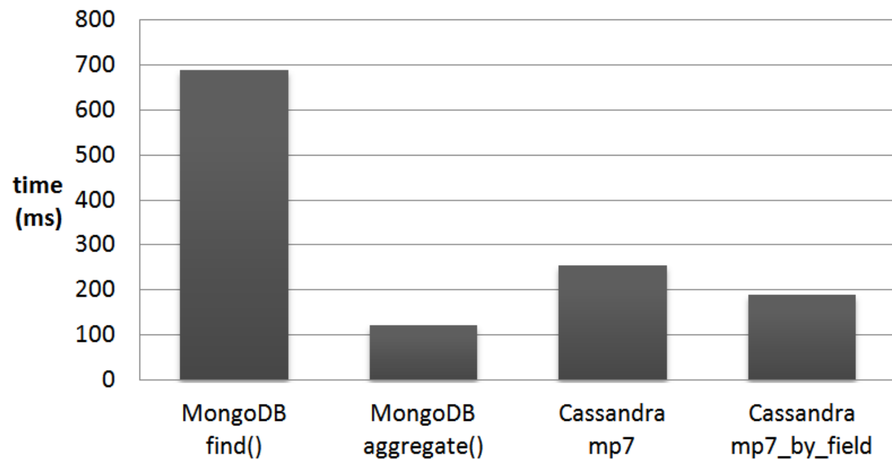
Όσον αφορά την Cassandra, η διατύπωση ενός τέτοιου ερωτήματος στο mp7 table απαιτεί χρήση *FILTERING*. Για να το αποφύγουμε αυτό και να βελτιώσουμε την απόδοση δημιουργήσαμε το παρακάτω Materialized View.

```
CREATE MATERIALIZED VIEW mp7_by_field AS
    SELECT *  FROM mp7
    WHERE path IS NOT NULL AND field_name IS NOT NULL
    AND positions IS NOT NULL AND filename IS NOT NULL
    PRIMARY KEY ((path, field_name), filename, positions)
    WITH CLUSTERING ORDER BY (filename ASC, positions ASC)
```

Κώδικας 4 Materialized View mp7_by_field

Με αυτόν τον τρόπο αποφεύγουμε την αναζήτηση πολλών partition στον δίσκο αλλά και την αναζήτηση των ζητούμενων γραμμών εντός του κάθε partition. Επίσης, όπως εξηγήθηκε στην θεωρία, σε ένα πολύ μεγάλο cluster με την χρήση MV επιτρέπεται στον coordinator να απευθυνθεί σε έναν συγκεκριμένο κόμβο (ή στα replicas) για την εξυπηρέτηση του ερωτήματος και δεν χρειάζεται να ρωτήσουμε κάθε κόμβο στο cluster. Ωστόσο, τέτοιες λειτουργίες συνήθως στόχο έχουν την περαιτέρω επεξεργασία των δεδομένων και παραγωγή αποτελεσμάτων. Σε αυτές τις περιπτώσεις η μεταφορά πάρα πολλών rows πάνω στο δίκτυο αποτελεί πρόβλημα , όπως και η επιβάρυνση ενός κόμβου για την επεξεργασία όλων των δεδομένων. Αυτές οι περιπτώσεις απαιτούν την χρήση ενός εργαλείου για την μαζική και κατανεμημένη επεξεργασία των δεδομένων , όπως το Spark. Παράδειγμα τέτοιο θα εξετάσουμε στην τελευταία ενότητα του κεφαλαίου.

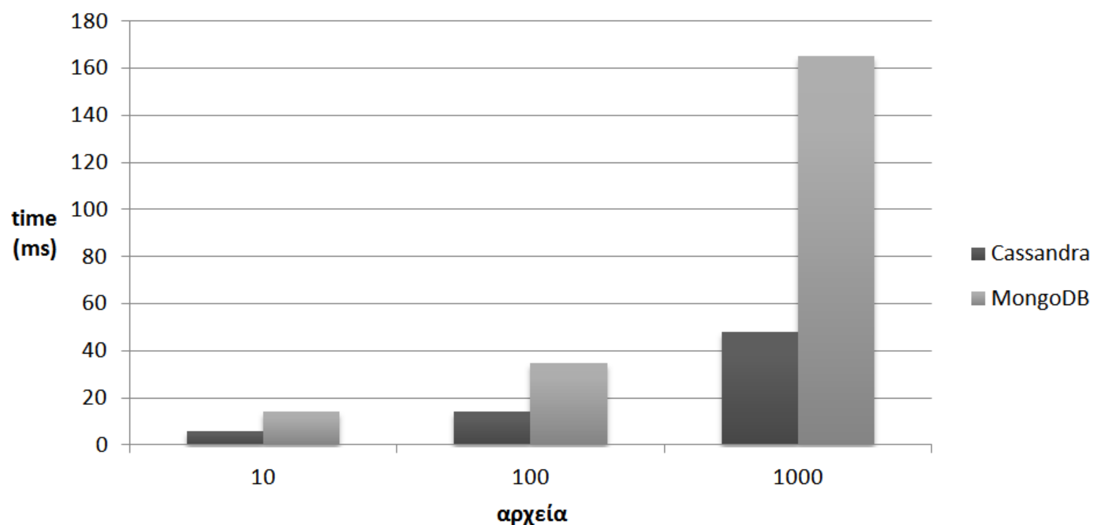
Εκτελέσαμε το ερώτημα και στις 2 βάσεις και παρουσιάζουμε τα αποτελέσματα στο επόμενο διάγραμμα:



Διάγραμμα 2 Διάβασμα ενός πεδίου (τύποι σχημάτων) απ' όλες τις περιγραφές

5.3 Εγγραφές Δεδομένων

Σε αυτήν την ενότητα θα εξετάσουμε την εισαγωγή ενός νέου *descriptor* στις *mpeg-7* περιγραφές. Εκτελούμε σε ένα πρόγραμμα (java) εγγραφές νέων descriptors για 10, 100 και 1000 αρχεία. Στην MongoDB χρησιμοποιήσαμε εντολή για bulk writes. Στην Cassandra τα καλύτερα αποτελέσματα παρατηρήθηκαν με χρήση *unlogged* batches. Σε ένα μεγάλο cluster θα ήταν αποδοτικότερη η ασύγχρονη εκτέλεση ξεχωριστά, είτε κάθε εγγραφής είτε μικρών batches. Η Cassandra πετυχαίνει πολύ καλύτερους χρόνους εγγραφής και καλύτερη κλιμάκωση όσο αυξάνεται ο όγκος των δεδομένων. Τα αποτελέσματα παρουσιάζονται στο επόμενο διάγραμμα:



Διάγραμμα 3 Εγγραφές Descriptors

5.4 Αναζήτηση βάσει περιεχομένου

Όπως αναφέραμε, τα mp7 αρχεία που διαχειριζόμαστε περιλαμβάνουν μεταδεδομένα για 3D αντικείμενα (σκηνές). Αυτά τα μεταδεδομένα μπορούν να χρησιμοποιηθούν για την αναζήτηση μιας σκηνής βάσει του περιεχομένου της ή την εύρεση ενός συνόλου σκηνών που πληρούν κάποια κριτήρια.

Κάθε σκηνή αποτελείται από αντικείμενα, καθένα από τα οποία έχει τις δικές του ιδιότητες όπως τύπος αντικειμένου, χρώμα , διαστάσεις, θέση κλπ. Εμείς αντλήσαμε πληροφορίες για το χρώμα και τον όγκο των αντικειμένων και τις τοποθετήσαμε με τέτοιο τρόπο ώστε να είναι δυνατή η δημιουργία αποδοτικών ευρετηρίων σε αυτές. Συγκεκριμένα, στο μονοπάτι *Mpeg7.Description.1.MultimediaContent.StructuredCollection.Collection[id=Geometries].DescriptorCollection.* βρίσκονται συλλογές από Descriptors. Κάθε συλλογή αντιστοιχεί σε ένα αντικείμενο και περιλαμβάνει, μεταξύ άλλων, κάποιον Descriptor με τις διαστάσεις του και κάποιον με το χρώμα του. Από τις διαστάσεις υπολογίσαμε αρχικά τον όγκο κάθε αντικειμένου και στην πορεία το ποσοστό του συνολικού όγκου της σκηνής που καταλαμβάνει το αντικείμενο. Τα χρώματα αναπαρίστανται σε RGB μορφή. Εμείς χωρίσαμε όλο το χρωματικό φάσμα σε 512 ίσα διαστήματα, όπου το πρώτο αντιστοιχεί στις τιμές R:[0…32) , G:[0… 32), B:[0…32), το δεύτερο στις τιμές R:[32…64), G:[0… 32), B:[0…32) κ.ο.κ μέχρι το 512ο : R:[224…255], G:[224…255], B:[224…255]. Επομένως ο όγκος παίρνει τιμές από 0.0 έως 1.0 και το χρώμα παίρνει ακέραιες τιμές από 0 έως 511.

Και στις δύο βάσεις εκτελέστηκε η παραπάνω διαδικασία στα δεδομένα τους. Στη MongoDB αποθηκεύσαμε τα αντικείμενα σε πίνακα που αντιστοιχεί στο πεδίο-κλειδί "NewDescriptors", το οποίο προσθέσαμε κάτω από το key "metadata" μαζί με τις mp7 περιγραφές. Κάθε στοιχείου του πίνακα είναι ένα εμφωλευμένο έγγραφο με πεδία για το χρωματικό κελί και τον όγκο ενός αντικειμένου, όπως φαίνεται στην παρακάτω εικόνα.

```
"NewDescriptors" : [
            {
                "color" : 438,
                "volume" : 0.680346524641658
            },
            {
                "color" : 438,
                "volume" : 0.0135196036146881
            },
            {
                "color" : 438,
                "volume" : 0.000276344048124021
            },
            {
                "color" : 511,
                "volume" : 5.28939779612383e-005
            },
            {
                "color" : 511,
                "volume" : 5.4648114985463e-005
            }
            ...
        ]
```

Εικόνα 31 Παράδειγμα NewDescriptors

Στην Cassandra δημιουργούμε το παρακάτω table, το οποίο έχει partition key το όνομα του αρχείου και clustering στήλες το χρώμα, τον όγκο και ένα μοναδικό αναγνωριστικό για τον διαχωρισμό των αντικειμένων του ίδιου αρχείου (το id υπάρχει ως πεδίο για κάθε αντικείμενο στην mp7 περιγραφή). Η επιλογή των clustering στηλών εξαρτάται από τα ερωτήματα που περιμένουμε διατυπωθούν στη βάση. Για παράδειγμα στην περίπτωση που ζητάμε τα X μεγαλύτερα αντικείμενα από μια σκηνή, το κατάλληλο primary key είναι (filename,volume,id) και χρησιμοποιούμε *LIMIT X* στο ερώτημα μας. Αντίστοιχα, με τον παρακάτω ορισμό απαντώνται πολύ αποδοτικά ερωτήματα όπως: «επέστρεψε τα X πιο σκούρα αντικείμενα μιας σκηνής», ή «τα μεγαλύτερα αντικείμενα συγκεκριμένης απόχρωσης». Ερωτήματα τέτοιου τύπου απαντώνται πολύ πιο αποδοτικά στην Cassandra, αφού αποθηκεύει διατεταγμένα τις γραμμές, και δεν θα τα εξετάσουμε περαιτέρω.

```
CREATE TABLE objects_by_filename (
    filename text,
    color int,
    volume double,
    id text,
    PRIMARY KEY (filename, color, volume, id)
) WITH CLUSTERING ORDER BY (color ASC, volume DSC, id ASC)
```

Κώδικας 5 Table αντικειμένων ανά αρχείο

Τα ερωτήματα που θα εξετάσουμε εδώ έχουν την ακόλουθη μορφή.

Βρες ποιές σκηνές (αρχεία) περιέχουν τουλάχιστον ένα αντικείμενο με χρώμα :

- ενός συγκεκριμένου χρωματικού κελιού. *(color equality)*
- σε ένα εύρος αποχρώσεων (χρωματικών κελιών). *(color range)*
- σε ένα εύρος αποχρώσεων και καταλαμβάνουν τουλάχιστον (ή το πολύ) ένα ποσοστό όγκου της συνολικής σκηνής. *(color range – volume range)*

Δευτερεύοντα ευρετήρια

Για αποδοτική απάντηση στα παραπάνω ερωτήματα δημιουργήσαμε ευρετήρια στα ζητούμενα πεδία. Στην MongoDB δημιουργήσαμε ένα ευρετήριο μονού πεδίου στο χρώμα και ένα σύνθετο (compound) ευρετήριο στο χρώμα και τον όγκο. Το σύνθετο ευρετήριο μπορεί να χρησιμοποιηθεί και για τα ερωτήματα που αφορούν μόνο το χρώμα αφού περιλαμβάνει το πρόθεμα {color : 1}, όμως το ευρετήριο μονού πεδίου πετυχαίνει λίγο καλύτερους χρόνους αφού εξετάζει λιγότερα κλειδιά στο ευρετήριο. Για τα ερωτήματα που αφορούν και τα δύο πεδία, το σύνθετο ευρετήριο είναι πιο αποδοτικό από την διασταύρωση δύο ευρετηρίων μονού πεδίου στο χρώμα και στον όγκο. Τα ευρετήρια που δημιουργήθηκαν είναι προφανώς *multikey* αφού κάθε σκηνή έχει πολλά αντικείμενα, καθώς και *sparse* (αραιά) έτσι ώστε να μην καταχωρούνται έγγραφα που δεν περιέχουν τα ζητούμενα πεδία.

```
db.getCollection('Savage.files')
.createIndex( {"metadata.NewDescriptors.color" : 1} , { sparse : true } );

db.getCollection('Savage.files')
   .createIndex( {"metadata.NewDescriptors.color" : 1,
    "metadata.NewDescriptors.volume" : 1 } , { sparse : true } );
```

Κώδικας 6 Δημιουργία ευρετηρίων MongoDB

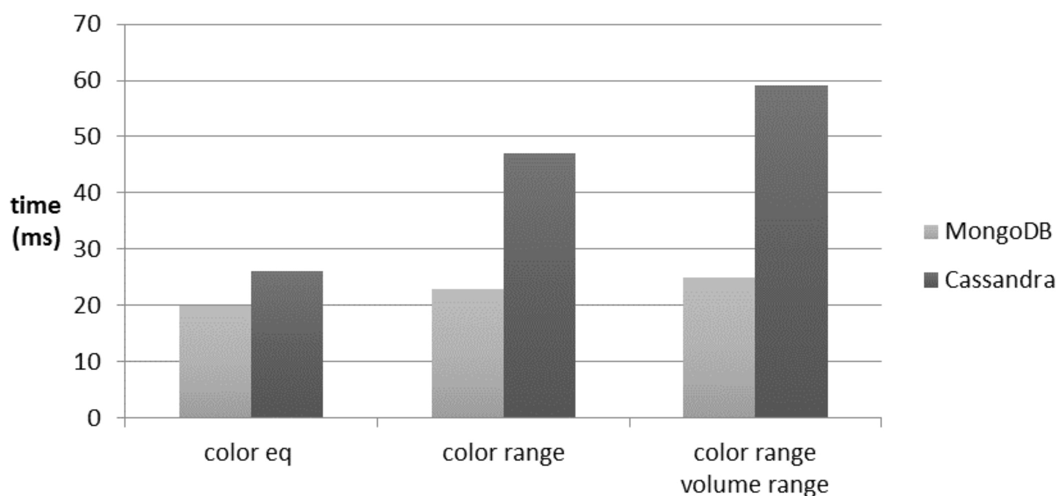
Στην Cassandra τα (απλά) δευτερεύοντα ευρετήρια που διαθέτει υποστηρίζουν μόνο έλεγχο ισότητας και δεν ήταν αποδοτικά στα ερωτήματα που υποβάλαμε. Επιλέξαμε τα νέα SASI ευρετήρια που περιγράφηκαν σε προηγούμενο κεφάλαιο, τα οποία χρησιμοποιώντας B+ trees υποστηρίζουν και ερωτήματα εύρους. Η επιλογή του τύπου των SASI ευρετηρίων εξαρτάται από το είδος των τιμών και την πληθικότητα τους. Για την στήλη color επιλέξαμε *Prefix* (default) ευρετήριο αφού δεν περιέχει πάρα πολλές τιμές (512 διακριτές τιμές) και κάθε χρωματικό κελί αντιστοιχεί σε πολλά αντικείμενα από πολλά αρχεία. Αντίθετα, η στήλη

volume παρουσιάζει μεγάλη πληθικότητα έχοντας double τιμές με πολλά δεκαδικά ψηφία. Για τέτοια δεδομένα ο κατάλληλος τύπος SASI είναι *Sparse.*

```
CREATE CUSTOM INDEX objects_color_prefix_idx ON objects_by_filename (color)
USING 'org.apache.cassandra.index.sasi.SASIIndex';

CREATE CUSTOM INDEX objects_volume_sparse_idx ON objects_by_filename(volume)
USING 'org.apache.cassandra.index.sasi.SASIIndex'
WITH OPTIONS = {'mode': 'SPARSE'};
```

Κώδικας 7 Δημιουργία ευρετηρίων Cassandra

Εκτελέσαμε διαδοχικά από 20 ερωτήματα για κάθε μία από τις περιπτώσεις που αναφέραμε προηγουμένως και παρουσιάζουμε τα αποτελέσματα στο επόμενο διάγραμμα. Η MongoDB έχει καλύτερη απόδοση σε ερωτήματα δευτερευόντων ευρετηρίων και παρουσιάζει μεγαλύτερη ευελιξία στην μεταβολή των ερωτημάτων.



Διάγραμμα 4 Ερωτήματα αναζήτησης βάσει χρώματος-όγκου με χρήση ευρετηρίων

5.5 MapReduce

Στην ενότητα αυτή θα εξετάσουμε την απόδοση των συστημάτων μας με ένα ερώτημα το οποίο να έχει μεγάλες απαιτήσεις σε υπολογιστικούς πόρους, αλλά παράλληλα να έχει και αρκετά μεγάλη σημασία για ένα multimedia dataset. Το ερώτημα που επιλέξαμε αφορά το πόσο «κυρίαρχο» είναι κάθε χρώμα κατά μέσο όρο όταν εμφανίζεται σε μια σκηνή. Πρώτα υπολογίζουμε το ιστόγραμμα χρώματος για κάθε σκηνή και έπειτα υπολογίζουμε τον (ποσοστιαίο) όγκο που καταλαμβάνει ένα χρώμα κατά μ.ό. όταν εμφανίζεται σε μια σκηνή. Θα απαντήσουμε στο ερώτημα αυτό μέσω του προγραμματιστικού μοντέλου MapReduce.

Στην MongoDB θα χρησιμοποιήσουμε το MapReduce framework που διαθέτει, το οποίο έχει περιγραφεί σε προηγούμενο κεφάλαιο. Η Cassandra δεν διαθέτει ενσωματωμένο framework για mapreduce λειτουργίες, αλλά αποτελεί ιδανική βάση για ενσωμάτωση σε Hadoop και Spark παρουσιάζοντας εξαιρετική κλιμάκωση. Εμείς χρησιμοποιήσαμε το Spark, το οποίο αποτελεί βέλτιστη λύση για υψηλή απόδοση. Για την σύνδεση τους χρησιμοποιήθηκε ο Spark Cassandra Connector της Datastax για Scala , ο οποίος επιτρέπει την έκθεση των πινάκων (tables) της Cassandra σε Spark RDDs και το αντίστροφο, όπως επίσης και την εκτέλεση ερωτημάτων CQL στις εφαρμογές του Spark.

Ως προς την υλοποίηση, αρχικά υπολογίζουμε για κάθε σκηνή το ιστόγραμμά της, δηλαδή τον όγκο που καταλαμβάνει κάθε χρωματικό κελί στη σκηνή συνολικά, από όλα τα αντικείμενά της. Έπειτα, αντιστοιχίζουμε (map) κάθε χρωματικό κελί με τον όγκο που υπολογίστηκε καθώς και έναν άσσο που χρησιμοποιείται για την εύρεση των συνολικών εμφανίσεων κάθε χρώματος σε σκηνές. Κατά την φάση του reduce παίρνουμε ομαδοποιημένες ανά κλειδί όλες τις τιμές που αφορούν ένα χρώμα και υπολογίζουμε δύο αθροίσματα: ένα για τον συνολικό όγκο που αντιστοιχεί σε κάθε κελί και ένα με το πλήθος των εμφανίσεων του. Τέλος, στη φάση finalized υπολογίζεται ο ζητούμενος μέσος όρος διαιρώντας τον όγκο με το πλήθος εμφανίσεων. Η φάση του finalize είναι απαραίτητη για τον υπολογισμό μ.ο και ο υπολογισμός αυτός δεν πρέπει να διεξαχθεί στο reduce αφού δεν αποτελεί προσεταιριστική διαδικασία.

Ακολουθούν οι συναρτήσεις σε Javascript που χρησιμοποιήθηκαν για την υλοποίηση σε MongoDB:

```
function map(){
        var sceneHistogram = {};
        var sceneObjects = this.metadata.NewDescriptors;
        for(var j = 0; j < sceneObjects.length; j++){
            var color = sceneObjects[j].color;
            var volume = sceneObjects[j].volume;
            if (isNaN(volume)) volume = 0;
            if (color in sceneHistogram){
                sceneHistogram[color] += volume
            } else{
                sceneHistogram[color] = volume
            }
        }
        for(var cell in sceneHistogram){
            var value = {
                    volume: sceneHistogram[cell],
                    count:1
            };
            emit(cell,value);
        }
}
```

Κώδικας 8 map_function.js

```
function reduce(key,values){
    var reducedVal = { volume: 0, count: 0 };

    values.forEach(function(value){
        reducedVal.volume += value.volume;
        reducedVal.count += value.count;
    });

    return reducedVal;
}
```

Κώδικας 9 reduce_function.js

```
function finalize(key, reducedVal) {
            reducedVal.avg = reducedVal.volume/reducedVal.count;
            return reducedVal;
}
```

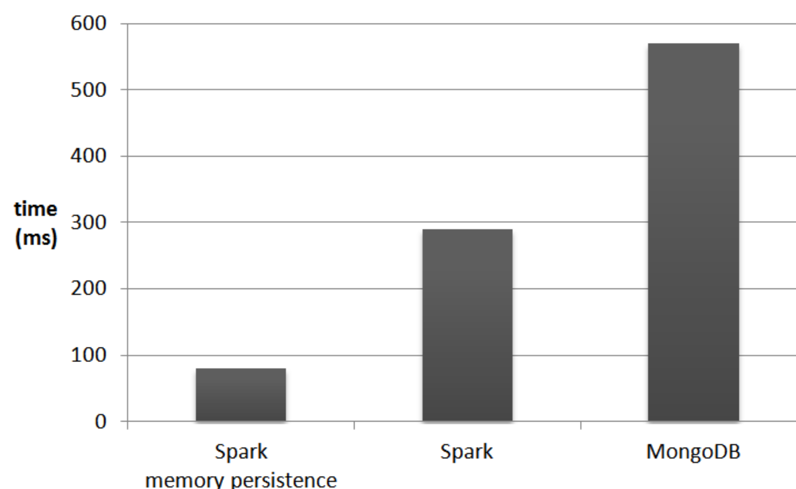
Κώδικας 10 finalize_function.js

Ο ίδιος αλγόριθμος εκτελέστηκε χρησιμοποιώντας transformations και actions του Spark στο RDD με τα δεδομένα μας. Ακολουθεί ο κώδικας σε Scala:

```
sc.stop
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"localhost")
val sc = new SparkContext(conf)

val objects_rdd = sc.cassandraTable[(String,Int,Double)]("diplomatikh",
"objects_by_filename").select("filename","color","volume")
                    //.cache()

val rdd = objects_rdd.keyBy(row => row._1).spanByKey

rdd.flatMap(x=>(x._2.groupBy(_._2)
   .map { case (k,v) => (k,v.map(_._3))})
   .map { case (k,v) => (k,(v.sum,1))}})
   .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
   .map { case (k,v) => (k,v._1/v._2)}
   .collect    //.foreach(println)
```

Κώδικας 11 Spark MapReduce (scala)

Η πρώτη εκτέλεση είναι φάση *warm up* και για τα δύο συστήματα και δεν λαμβάνεται υπόψη. Οι επόμενες εκτελέσεις στην MongoDB χρειάστηκαν περίπου 510-620 ms για να ολοκληρωθούν. Στο Spark διήρκησαν 275-310 ms. Στην συνέχεια ζητήσαμε ρητά την διατήρηση του RDD με το table των αντικειμένων της Cassandra στην μνήμη, μέσω της εντολής cache() η οποία είναι σε σχόλιο στον παραπάνω κώδικα. Μετά την πρώτη εκτέλεση τα δεδομένα διατηρούνται στην μνήμη (*memory persistence*) και οι επόμενες εκτελέσεις διαρκούν 70-90 ms.



Διάγραμμα 5 MapReduce στα ιστογράμματα χρώματος των σκηνών

5.6 Συμπεράσματα

Μέσω της πειραματική αξιολόγηση των βάσεων αλλά και της συνολικής ενασχόλησης με τα δύο αυτά συστήματα καταλήξαμε σε ορισμένα συγκριτικά συμπεράσματα που συνοψίζονται σε αυτήν την ενότητα.

1. ***Μοντέλο Δεδομένων***

   Η MongoDB διαθέτει ένα πλούσιο και εκφραστικό μοντέλο δεδομένων. Τα αντικείμενά της μπορούν να έχουν πεδία και εμφωλευμένα άλλα αντικείμενα σε πολλαπλά επίπεδα χωρίς περιορισμούς. Το μοντέλο αυτό είναι «αντικειμενοστραφές» και μπορεί εύκολα να αναπαραστήσει ιεραρχίες. Η Cassandra, από την άλλη, δεν έχει "ελεύθερο" αλλά "ευέλικτο" σχήμα. Τα tables δομούνται με γραμμές και στήλες όπου κάθε στήλη πρέπει να χει συγκεκριμένο τύπο που ορίζεται στην δημιουργία. Η MongoDB δεν απαιτεί ορισμό σχήματος, όπως κάνει η Cassandra στις νεώτερες εκδόσεις (με CQL).

2. ***Υψηλή διαθεσιμότητα***

   Η MongoDB διαθέτει αρχιτεκτονική master-slaves. Αν ένας master κόμβος βγεί εκτός, ένας εκ των slaves θα εκλεχθεί ως master. Αυτή η διαδικασία γίνεται αυτόματα αλλά κρατάει μερικά δευτερόλεπτα. Κατά την διάρκεια εκλογής το replica set είναι εκτός και δεν δέχεται εγγραφές. Η Cassandra υποστηρίζει peer to peer αρχιτεκτονική. Η απώλεια ενός μοναδικού κόμβου δεν επηρεάζει την ικανότητα του cluster να δέχεται εγγραφές. Έτσι επιτυγχάνεται 100% uptime στα writes.

3. ***Δευτερεύοντα Ευρετήρια***

   Τα δευτερεύοντα ευρετήρια της MongoDB είναι ευέλικτα και πολύ αποδοτικά. Επιτρέπεται η ευρετηριοποίηση οποιαδήποτε πεδίου ενός αντικειμένου, ακόμα και εμφωλευμένου. Τα νέα SASI ευρετήρια της Cassandra είναι σαφώς πολύ πιο αποδοτικά από τα default, πολύ περιορισμένα, ευρετήρια που διαθέτει, αλλά δεν παρουσίασαν επιδόσεις ανάλογες με αυτές των ευρετηρίων της MongoDB. Η Cassandra πραγματοποιεί πολύ αποδοτικά ερωτήματα βάσει του πρωτεύοντος κλειδιού. Έτσι, είναι συνηθισμένη τεχνική να αποφεύγεται η δημιουργία δευτερευόντων ευρετηρίων είτε μέσω της χρήσης clustering στηλών είτε μέσω της

δημιουργίας νέου table που επαναλαμβάνει τα δεδομένα (denormalization) του αρχικού και έχει ως πρωτεύον κλειδί το ζητούμενο πεδίο (για αναζητήσεις ισότητας).

4. ***Κλιμακωσιμότητα εγγραφών***

   Η Cassandra όπως είδαμε εκτελεί μαζικές εγγραφές πολύ γρήγορα. Επιπλέον λόγω του peer-to-peer μοντέλου της μπορεί να δεχθεί εγγραφές σε οποιονδήποτε server. Στην ουσία η κλιμακωσιμότητα των εγγραφών περιορίζεται μονάχα από τον αριθμό των servers που υπάρχουν στο cluster. Όσο μεγαλύτερο το cluster , τόσο καλύτερα κλιμακώνεται. Αντίθετα, στην MongoDB μόνο ο primary κόμβος δέχεται εγγραφές και αυτό περιορίζει πολύ την κλιμακωσιμότητα των writes.

5. ***Aggregation Framework***

   Η MongoDB διαθέτει ένα ενσωματωμένο Aggregation Framework στο οποίο τρέχει ένα ETL pipeline για τον μετασχηματισμό των δεδομένων στην βάση. Αυτό είναι ένα πολύ χρήσιμο εργαλείο που χρησιμοποιήσαμε και στις προηγούμενες ενότητες. Η Cassandra δεν διαθέτει ενσωματωμένο aggragation framework και χρησιμοποιούνται συνήθως εργαλεία όπως το Haddop ή το Spark.

Εν κατακλείδι, για την συγκεκριμένη εφαρμογή προτείνουμε ως λύση την MongoDB κυρίως λόγω της ευκολίας να διαχειριστεί το πολύπλοκο σχήμα των δεδομένων μας και να αναπαραστήσει την ιεραρχία των πεδίων των mpeg-7 περιγραφών. Παρότι παρατηρήθηκαν πολύ υψηλές αποδόσεις εκ μέρους της Cassandra σε διάφορα λειτουργίες του συστήματος, τα πολλαπλά επίπεδα φωλιάσματος των δεδομένων μας και η απαίτηση της βάσης για ορισμό σχημάτος δεδομένων οδήγησαν σε προβλήματα κατά την εκτέλεση των ερωτημάτων και είχαν αρνητική επιρροή στην απόδοσή της. Επιπλέον λόγος αποτελούν τα αποδοτικά ερωτήματα εύρους τιμών της MongoDB σε δευτερεύοντα ευρετήρια, τα οποία αποτελούν την πιο βασική λειτουργία του συστήματος.

6

Βιβλιογραφία

[1] Russom, P. (2011). big data analytics. TDWI Best Practices Report, 4 th Quarter 2011

[2] Lakshman, A., & Malik, P. (2010). Cassandra—A decentralized structured storage system. Operating systems review, 44(2), 35

[3] Padhy, R. P., Patra, M. R., & Satapathy, S. C. (2011). RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database_s‖. International Journal of Advanced Engineering Science and Technologies, 11(1), 15-30.

[4] Hecht, R., & Jablonski, S. (2011, December). NoSQL evaluation: A use case oriented survey. In Cloud and Service Computing (CSC), 2011 International Conference on (pp. 336- 341). IEEE.

[5] https://academy.datastax.com/demos/brief-introduction-apache-cassandra

[6] http://cassandra.apache.org/

[7] https://www.mongodb.org/

[8] Sugam Sharma et al., International Journal of Big Data Intelligence 05/2015

[9] Moniruzzaman, Hossain, International Journal of Database Theory and Application Vol. 6, No. 4. 2013

[10] http://docs.datastax.com/en/cassandra/3.x/

[11] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[12] https://spark.apache.org/

[13] http://mongodb.github.io/mongo-java-driver/3.3/driver-async/getting-started/

[14] Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., J. Franklin M., Shenker S., & Stoica I. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proc. of NSDI 2012 at USENIX, pp. 15–28

[15] http://www.scala-lang.org/

[16] S. Gilbert and N. A. Lynch, "Perspectives on the CAP Theorem"

[17] R. Lämmel, "Google's MapReduce Programming Model — Revisited"

[18] http://dataconomy.com/seven-vs-big-data/

[19] https://dzone.com/articles/introduction-apache-cassandras

[20] http://www.datastax.com/dev/blog/new-in-cassandra-3-0-materialized-views

[21] https://docs.datastax.com/en/cql/3.3/

[22] http://www.planetcassandra.org/blog/the-most-important-thing-to-know-in-cassandra-data-modeling-the-primary-key/

[23] http://www.planetcassandra.org/blog/cassandra-native-secondary-index-deep-dive/

[24] http://www.doanduyhai.com/blog/?p=1930

[25] D. Tsoumakos, N. Papailiou, I. Giannakopoulos, N. Koziris "PANIC: Modeling Application Performance over Virtualized Resources"

[26] Big Data: A Revolution That Will Transform How We Live, Work, and Think Paperback, by Viktor Mayer-Schönberger , Kenneth Cukier