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

# Τεχνικές Βελτιστοποίησης της Αποθήκευσης και Αναζήτησης Big Data χρησιμοποιώντας την Βάση Δεδομένων MongoDB

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Κρουσταλιός

**Επιβλέπων :** Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Αθήνα, Οκτώβριος 2015

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

# Τεχνικές Βελτιστοποίησης της Αποθήκευσης και Αναζήτησης Big Data χρησιμοποιώντας την Βάση Δεδομένων MongoDB

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Κρουσταλιός

**Επιβλέπων :** Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Οκτωβρίου 2015.

..... ..... .....

.....

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

# 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.

Θα ήθελα να ευχαριστήσω θερμά την κ. Θεοδώρα Βαρβαρίγου για την υποστήριξη και καθοδήγηση που μου προσέφερε, καθώς και για την ευκαιρία που μου έδωσε να ασχοληθώ με την ερευνητική περιοχή τον Big Data.

Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τον υποψήφιο διδάκτορα Βρεττό Μουλό για την πολύτιμη βοήθειά του καθόλη τη διάρκεια της εργασίας, όπου με τις στοχευμένες συμβουλές του συνέβαλλε στην εκπόνηση αυτής της διπλωματικής εργασίας.

# Περίληψη

Η μεγαλύτερη πρόκληση των σύγχρονων υπολογιστικών συστημάτων είναι αναμφισβήτητα η αποδοτική αποθήκευση και ανάκτηση πολύ μεγάλου όγκου δεδομένων. Η ανάγκη αυτή έκανε την εμφάνισή της τα τελευταία χρόνια λόγω της έκρηξης δεδομένων που παρατηρείται στο διαδίκτυο και αποκτά ολοένα και μεγαλύτερη σημασία λόγω του πολύ μεγάλου εύρους πληροφοριών που μπορούμε να αντλήσουμε.

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 με χρήση τεχνικών Big Data.

Η λύση που προτείνουμε, είναι η χρησιμοποίηση μιας NoSQL βάσης δεδομένων και συγκεκριμένα της document database MongoDB. Η συγκεκριμένη βάση δεδομένων, είναι η δημοφιλέστερη στον τομέα της, λόγω της πολύ καλής κλιμακοσιμότητας που προσφέρει, γεγονός υψίστης σημασίας όταν έχουμε να διαχειριστούμε μεγάλο όγκο δεδομένων. Τέλος, καταφέραμε να βελτιστοποιήσουμε την αναζήτηση και την επεξεργασία των δεδομένων μας, χρησιμοποιώντας κάποιες από τις πιο σύγχρονες τεχνικές στον τομέα τον big data, όπως είναι τα ευρετήρια, το Sharding, το Replication και το μοντέλο Map-Reduce.

## Λέξεις Κλειδιά

Big Data, Μη σχεσιακές βάσεις δεδομένων, document databases, MongoDB, GridFS, πολυμεσικά δεδομένα, μεταδεδομένα, ευρετήρια, Sharding, αντικατάσταση, Map-Reduce

# Abstract

Efficient Big Data storage and retrieval is undoubtedly the biggest challenge faced by modern computing systems. In the last few years, this necessity has become more obvious due to the huge data explosion that is currently taking place on the internet and it has become a critical issue because of the wide variety of information we can retrieve from all this data.

In this thesis, we study various ways of efficient storage and searching of multimedia data. The huge size and the complicated structure of multimedia data does not allow us to make use of tradition storing techniques such as relational database systems, therefore we applied some of the more modern Big Data techniques.

The approach presented in this thesis, regards, the use of a non relational (NoSQL) database and specifically the document database MongoDB. This is the most popular NoSQL database because of it's high scalability potential, which plays a vital role when we have to handle Big Data. Finally, we managed to optimize the retrieval and processing operations on our data, with the use of the most modern techniques in the area of Big Data, such as, Indexing, Sharding, Replication and the Map-reduce model.

## Keywords

Big Data, NoSQL databases, Non Relational Databases, document databases, MongoDB, GridFS, Multimedia, Metadata, Indexing, Sharding, Replication, Map-Reduce

# Περιεχόμενα

# Κατάλογος εικόνων

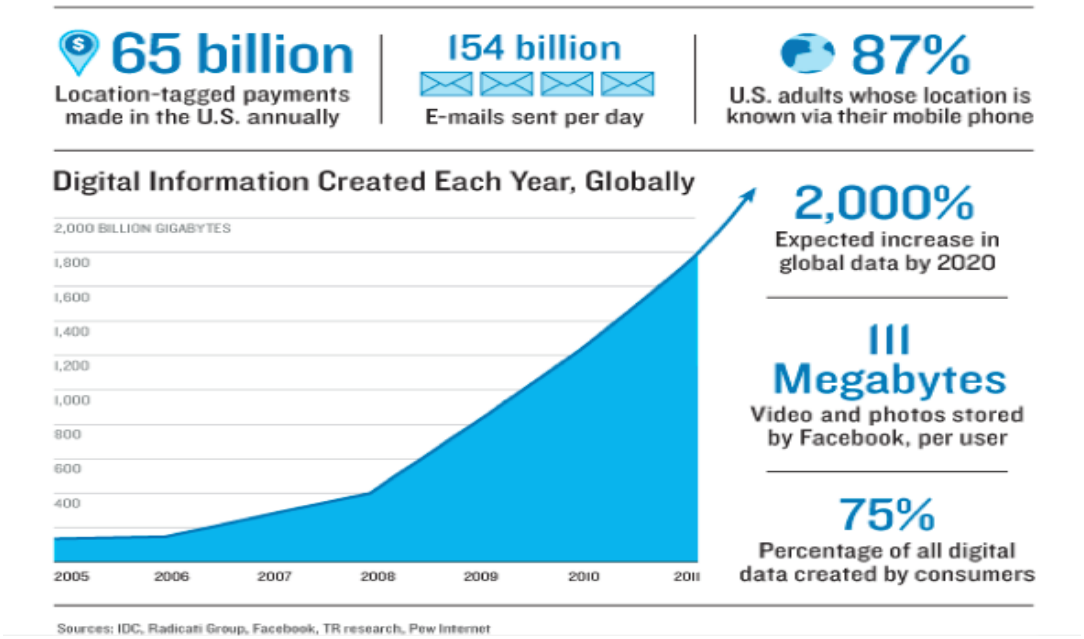
# Κεφάλαιο 1

# Big Data

## 1.1 Ορισμός

Ο λόγος για τον οποίο τα big data είναι ευρέως διαδεδομένα στις μέρες μας είναι γιατί όπως αναφέρει και ο εκτελεστικός πρόεδρος της Google, Eric Smidt "Κάθε δύο μέρες παράγουμε δεδομένα και πληροφορίες ίσες με αυτές που είχαν παραχθεί από την αρχή της ανθρωπότητας ως το 2003". Κάθε μέρα δημιουργούμε 2,5 τετράκις εκατομμύρια bytes δεδομένων, τα οποία είναι τόσα πολλά, που το 90% των δεδομένων στον κόσμο σήμερα έχει δημιουργηθεί τα τελευταία δύο χρόνια μόνο. Τα δεδομένα αυτά προέρχονται από πολλές πηγές, όπως αισθητήρες που χρησιμοποιούνται για τη συλλογή πληροφοριών του κλίματος, δημοσιεύσεις σε δικτυακούς τόπους κοινωνικών μέσων ενημέρωσης, ψηφιακές φωτογραφίες και βίντεο, τα αρχεία συναλλαγών, καθώς και τα σήματα GPS κινητού τηλεφώνου. Αυτά τα δεδομένα είναι τα μεγάλα δεδομένα.

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".




Εικόνα 1: Ο ρυθμός αύξησης των Big Data.

Οι προκλήσεις που περιλαμβάνουν είναι η σύλληψη, η επιμέλεια, η αποθήκευση, η αναζήτηση, η διανομή, η μεταφορά, η ανάλυση και η οπτικοποίηση των δεδομένων. Συνεπώς, αποτελεί επιτακτική ανάγκη η ύπαρξη ενός οργανωμένου τρόπου για την αποθήκευση και την επεξεργασία όλων αυτών των δεδομένων με σκοπό την απόκτηση πληροφοριών και τη δημιουργία αξίας από αυτές.

## 1.2 Χαρακτηριστικά των Big Data

Από το 2001, ο αναλυτής Doug Laney, αρθρογραφούσε σχετικά με την παρούσα επικρατούσα τάση των Big Data μιλώντας για τα 3 V που χαρακτηρίζουν τα Big Data: Volume, Velocity και Variety.

**Όγκος (Volume)**

Ο όγκος είναι το κυριότερο χαρακτηριστικό. Ειδικότερα, εάν ο όγκος των δεδομένων είναι της τάξης των μερικών gigabyte τότε πιθανώς το πρόβλημα δεν ανήκει στον τομέα των Big Data. Ο όγκος των δεδομένων είναι αυτός που καθιστά το πρόβλημα ακατάλληλο να διαχειριστεί από τις κλασικές σχεσιακές βάσεις δεδομένων (RDBMS). Στις μέρες μας είναι πολλοί οι παράγοντες που συμβάλλουν στην αύξηση του όγκου των δεδομένων. Υπάρχουν δεδομένα συναλλαγών αποθηκευμένα για χρόνια ενω συλλέγονται και αδόμητα δεδομένα συνεχούς ροής από τα Social Media. Επίσης, ο αριθμός δεδομένων που συλλέγονται από αισθητήρες (sensors) και την επικοινωνία μεταξύ μηχανών (machine-to-machine) συνεχώς αυξάνεται. Στο παρελθόν, ο υπερβολικός όγκος δεδομένων δημιουργούσε προβλήματα στην αποθήκευση. Αυτό επιλύεται με τη μείωση του κόστους αποθήκευσης, αλλά προκύπτουν νέα προβλήματα, όπως για παράδειγμα το πώς να διαχειριστούμε αποδοτικά τον μεγάλο όγκο δεδομένων και πώς μπορούμε μέσω τεχνολογιών analytics να αντλήσουμε πραγματική αξία από αυτά.

**Ταχύτητα (Velocity)**

Η ταχύτητα χωρίζεται σε τρεις επιμέρους κατηγορίες:

Δεδομένα σε κίνηση (data in-motion): Μία έννοια της ταχύτητας είναι να περιγράψει τα δεδομένα εν κινήσει. Τα δεδομένα "ρέουν" με πρωτοφανή ταχύτητα και πρέπει να τα αντιμετωπίζουμε έγκαιρα. Οι ετικέτες RFID, οι αισθητήρες και τα έξυπνα συστήματα, οδηγούν στην ανάγκη να αντιμετωπίσουμε χειμάρρους δεδομένων.

Διάρκεια ζωής της χρησιμότητας των δεδομένων: Μια δεύτερη διάσταση της ταχύτητας είναι το για πόσο καιρό τα δεδομένα που συλλέξαμε θα είναι χρήσιμα. Είναι μόνιμα χρήσιμα ή μήπως παλιώνουν γρήγορα και χάνουν το νόημα και τη σημασία τους; Η κατανόηση αυτής της διάστασης της ταχύτητας στα δεδομένα που

θα αποθηκευτούν θα είναι σημαντική για την απόρριψη των δεδομένων που δεν είναι πλέον χρήσιμα και μπορεί να μας παραπλανήσουν.

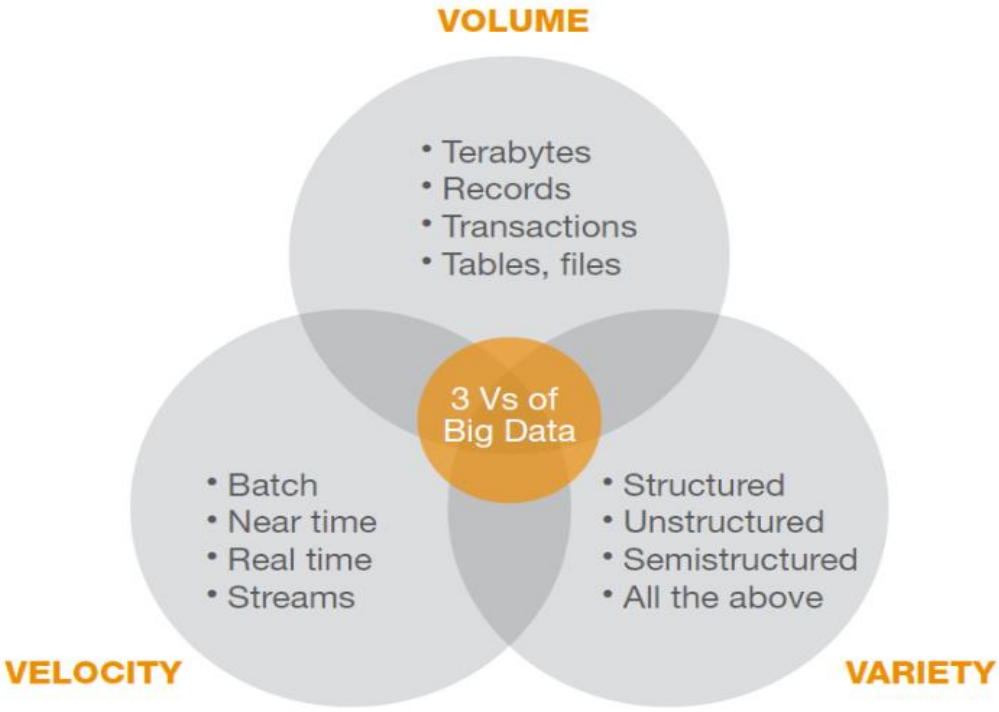
Δεδομένα σε πραγματικό χρόνο (Real Time Big Data): Η τρίτη διάσταση του προβλήματος, είναι η ταχύτητα με την οποία τα δεδομένα θα πρέπει να αποθηκευτούν και να ανακτηθούν. Τα δεδομένα ήταν πάντα εκεί, αλλά η ικανότητα να τα συλλέξουμε, να τα αναλύσουμε, και να τα επεξεργαστούμε σε (σχεδόν) πραγματικό χρόνο είναι πράγματι ένα ολοκαίνουριο χαρακτηριστικό της τεχνολογίας Big Data.

**Ποικιλία (Variety)**

Διαφορετικοί τύποι δεδομένων: Σήμερα τα δεδομένα έρχονται σε οποιαδήποτε πιθανή μορφή και έτσι οι σχεσιακές βάσεις δεδομένων δεν προσφέρονται για την αποθήκευση τους. Πλέον καλούμαστε να αποθηκεύσουμε αδόμητα έγγραφα κειμένου, email, video, ήχους, δεδομένα χρηματιστηριακών συναλλαγών και εμπορικών συναλλαγών. Η διαχείριση, η συγχώνευση και η διακυβέρνηση διαφορετικών ειδών δεδομένων είναι βασική δραστηριότητα των Big Data.

Δεδομένα από διαφορετικές πηγές: Ο όρος Variety χρησιμοποιείται και για να υποδείξει ότι τα δεδομένα προέρχονται από διαφορετικές πηγές. Για παράδειγμα, πλέον, τα δεδομένα μπορούν να προέρχονται τόσο από το εσωτερικό όσο και από το εξωτερικό περιβάλλον μιας εταιρείας.

Τέλος, υπάρχει η αλληλεπίδραση της ποικιλίας με τον όγκο. Τα αδόμητα δεδομένα αυξάνονται πολύ πιο γρήγορα από ότι τα δομημένα δεδομένα. Η εταιρεία Gartner που καταπιάνεται στον τομέα των Big Data, εκτιμά ότι ο όγκος των αδόμητων δεδομένων διπλασιάζεται κάθε τρεις μήνες.




Εικόνα 2: Τα 3 V’s των Big Data.

Στην δική μας περιγραφή των χαρακτηριστικών των Big Data, θα προσθέσουμε και άλλα 4 V’s ως χαρακτηριστικά. Συγκεκριμένα αυτά είναι:

**Αξία (Value)**

Τα Big Data θεωρείται ότι έχουν μεγάλη αξία τόσο για τις επιχειρήσεις όσο και την κοινωνία. Είναι πραγματικότητα ότι στις Big Data 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. Επίσης, όσον αφορά την αξία, τα Big Data τείνουν να έχουν χαμηλή σχέση αξίας – όγκου, δηλαδή χρειαζόμαστε πολλά δεδομένα ώστε να έχουμε σημαντικά ευρήματα.

**Μεταβλητότητα (Variability)**

Εκτός από τις αυξανόμενες ταχύτητες και την ποικιλία, οι ροές των δεδομένων μπορεί να είναι εξαιρετικά ασυνεπείς με περιοδικές κορυφώσεις. Αυτό συμβαίνει για παράδειγμα όταν υπάρχει κάποια τάση στα social media. Τα καθημερινά, τα εποχιακά και τα δεδομένα που πυροδοτούνται από διάφορα συμβάντα μπορεί να είναι δύσκολο να διαχειρισθούν, πόσο μάλλον όταν σε αυτά εμπλέκονται και αδόμητα δεδομένα. Επίσης, η μεταβλητότητα αφορά τα δεδομένα των οποίων η σημασία αλλάζει. Αυτό το πρόβλημα μας απασχολεί όταν τα δεδομένα μας είναι κείμενο ή κάποιες προτάσεις πάνω στις οποίες καλούμαστε να κάνουμε γλωσσική επεξεργασία.

**Αξιοπιστία (veracity)**

Veracity είναι η ορθότητα και η ακρίβεια των δεδομένων μας. Παρά τους ισχυρισμούς αναφορικά με τη μεγάλη αξία των Big Data, 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. Σε αυτό το σημείο είναι σημαντικό να κατανοήσουμε την "ακατάστατη" φύση των Big Data και το μεγάλο βαθμό θορύβου που αυτά εμπεριέχουν με αποτέλεσμα να χρειάζεται μεγάλη προεργασία ώστε να παράγουμε ένα αξιόπιστο και ακριβές σύνολο δεδομένων πριν καν ξεκινήσει η διαδικασία της ανάλυσης.

**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.

## 1.3 Περιπτώσεις χρήσης των Big Data

**1. Κατανόηση και ομαδοποίηση του καταναλωτικού κοινού**

Αυτή είναι μία από τις κυριότερες χρήσεις των Big Data. Τα εκμεταλλευόμαστε με σκοπό να κατανοήσουμε καλύτερα τους καταναλωτές και τις προτιμήσεις τους. Οι περισσότερες επιχειρήσεις επιδιώκουν να εμπλουτίσουν τα δεδομένα τους αντλώντας επιπλέον πληροφορίες από τα social media, τα browser logs και διάφορους αισθητήρες ώστε να αποκτήσουν μία πλήρη εικόνα για τον κάθε πελάτη τους. Η διαδικασία αυτή επιτρέπει στην επιχείρηση να δημιουργήσει μοντέλα με τα οποία θα προβλέπει τη ζήτηση.

**2. Κατανόηση και Βελτιστοποίηση Επιχειρησιακών Διαδικασιών**

Τα Big Data χρησιμοποιούνται όλο και περισσότερο για τη βελτίωση των επιχειρηματικών διαδικασιών. Μία επιχείρηση που διαθέτει καταστήματα λιανικής πώλησης μπορεί να διαχειριστεί βέλτιστα τα αποθέματα της με βάση τις προβλέψεις που δημιουργούνται από τα social media, τις αναζητήσεις στο διαδίκτυο ή ακόμα και από την πρόβλεψη του καιρού. Ειδικότερα, τα Big Data χρησιμοποιούνται στις μέρες μας κατά κόρων για να καταστήσουν πιο αποδοτική την αλυσίδα ανεφοδιασμού (supply chain) και για να βελτιστοποιήσουν τη διαδρομή παράδοσης που θα ακολουθηθεί ώστε να παραδοθεί κάποιο πακέτο (delivery route). Τέλος, αποτελεί γνωστή πρακτική για μεγάλες εταιρείες να χρησιμοποιούν τα Big Data ώστε να λειτουργεί πιο αποδοτικά το τμήμα ανθρώπινου δυναμικού και κυρίως για να καταστήσουν πιο αποτελεσματική και λιγότερο χρονοβόρα τη διαδικασία των προσλήψεων.

**3. Βελτίωση της καθημερινότητας και της αποδοτικότητας του ανθρώπου**

Τα Big Data δεν είναι μόνο για τις επιχειρήσεις και τις κυβερνήσεις. Όλοι οι πολίτες, μπορούν πλέον να επωφεληθούν από τα δεδομένα που δημιουργούνται από wearable συσκευές όπως τα έξυπνα ρολόγια ή έξυπνα βραχιόλια. Για παράδειγμα, κυκλοφορούν στην αγορά περιβραχιόνια, τα οποία έχουν τη δυνατότητα να

συλλέγουν δεδομένα σχετικά με την κατανάλωση θερμίδων μας, τα επίπεδα δραστηριότητας και τα μοτίβα του ύπνου μας.

**4. Βελτίωση της Περίθαλψης και της Δημόσιας Υγείας**

Η υπολογιστική δύναμη των Big Data μας επιτρέπει να αποκωδικοποιήσουμε το DNA μέσα σε λίγα λεπτά, και ευελπιστούμε ότι μελλοντικά θα μας επιτρέψει να κατανοήσουμε καλύτερα και να προβλέψουμε τα στάδια και την εξέλιξη των νοσημάτων. Τέλος, μας επιτρέπει να παρακολουθούμε και να προβλέπουμε τις εξελίξεις των επιδημιών. Δεδομένα από ιατρικά αρχεία σε συνδυασμό με τα social media μας δίνουν τη δυνατότητα να παρακολουθούμε τα κρούσματα γρίπης σε πραγματικό χρόνο, απλά ακούγοντας τι λέει ο κόσμος στις δημοσιεύσεις του, όπως για παράδειγμα το παρακάτω tweet: "Νιώθω χάλια σήμερα, είμαι στο κρεβάτι κρυωμένος".

**5. Βελτίωση των αθλητικών επιδόσεων**

Στα περισσότερα αθλήματα σήμερα χρησιμοποιούνται τεχνολογίες Big Data. Μας βοηθούν στην καταγραφή στατιστικών όπως για παράδειγμα πόσα μέτρα διάνυσε ο κάθε ποδοσφαιριστής κατά τη διάρκεια που αγωνίστηκε. Επίσης, υπάρχουν ειδικοί μηχανισμοί που ενσωματώνονται πάνω στον αθλητικό εξοπλισμό και βοηθάνε τον αθλητή να πετύχει καλύτερες επιδόσεις (π.χ. μπάλα μπάσκετ με αισθητήρα που μετρά την περιστροφή της μπάλας στο σουτ και σου υποδεικνύει πότε ήταν σωστή και πότε λάθος).

**6. Βελτίωση των επιστημών και της έρευνας**

Η επιστήμη και η έρευνα επηρεάζονται από τις νέες δυνατότητες που φέρνουν τα Big Data. Για παράδειγμα το κέντρο δεδομένων του CERN, που είναι ένα το εργαστήριο πυρηνικής φυσικής με το μεγαλύτερο και ισχυρότερο επιταχυντή σωματιδίων στον κόσμο έχει 65.000 επεξεργαστές και αναλύει 30 petabytes των δεδομένων.

**7. Βελτίωση της ασφάλειας και της επιβολής του νόμου**

Τα Big Data συμβάλλουν σε μεγάλο βαθμό στη βελτίωση της ασφάλειας και τη δυνατότητα επιβολής του νόμου. Διάφορες κυβερνητικές οργανώσεις (όπως π.χ. η αντιτρομοκρατική) τα χρησιμοποιεί για να αποτρέψει τρομοκρατικές ενέργειες ή για την ανίχνευση και την πρόληψη των επιθέσεων στον κυβερνοχώρο. Επίσης, η Αστυνομία χρησιμοποιεί Big Data για να πιάσει τους εγκληματίες ή ακόμα και για να προβλέψει κάποια εγκληματική δραστηριότητα. Τέλος, οι τράπεζες και άλλες εταιρείες πιστωτικών καρτών έχουν τη δυνατότητα να ανιχνεύσουν τις παράνομες συναλλαγές και να τις εμποδίσουν.

**8. Βελτιστοποίηση της λειτουργίας των πόλεων και των χωρών**

Τα Big data χρησιμοποιούνται για τη βελτίωση και την αποδοτικότερη λειτουργία των πόλεων. Για παράδειγμα, επιτρέπουν τη ρύθμιση της κυκλοφορίας με βάση πληροφορίες που αντλούν σε πραγματικό χρόνο σχετικά με την κυκλοφοριακή συμφόρηση. Ορισμένες πόλεις είναι σήμερα σε πειραματικό στάδιο, ώστε να εξελιχθούν σε αυτό που αποκαλούμε "έξυπνες" πόλεις, όπου οι δημόσιες συγκοινωνίες και η ενεργειακή απόδοση θα λειτουργούν βέλτιστα.

**9. Αυτοματοποίηση οικονομικών συναλλαγών**

Οι Συναλλαγές υψηλής συχνότητας (HFT) είναι μία περιοχή όπου τα Big Data χρησιμοποιούνται ευρέως. Υπάρχουν ειδικοί αλγόριθμοι που χρησιμοποιούνται για τη λήψη επενδυτικών αποφάσεων. Σήμερα, η πλειοψηφία των συναλλαγών στο χρηματιστήριο πραγματοποιείται μέσω αλγορίθμων που λαμβάνουν υπόψη όλο και περισσότερο τα δεδομένα των social media και των ειδησεογραφικών ιστοσελίδων και έπειτα αγοράζουν ή πωλούν μετοχές σε κλάσματα του δευτερολέπτου.


Εικόνα 3: Περιοχές χρήσης των Big Data.

# 1.4 Προκλήσεις στον τομέα των Big Data

**1. Έγγραφα με διαφορετική δομή ή μη πλήρως συμπληρωμένα**

Όταν οι άνθρωποι λαμβάνουν κάποια πληροφορία, έχουν τη δυνατότητα να την κατανοήσουν σε οποιαδήποτε μορφή και αν βρίσκεται. Τα συστήματα ηλεκτρονικών υπολογιστών λειτουργούν πολύ πιο αποτελεσματικά εάν τα στοιχεία που αποθηκεύουν έχουν όλα το ίδιο μέγεθος και δομή. Οπότε, αποτελεί μεγάλη πρόκληση η αποδοτική μετατροπή των δεδομένων ώστε να έχουν μια κοινή δομή αφού ειδικά τα Big Data χαρακτηρίζονται από την διαφοροποίηση των δεδομένων τους. Τέλος, εξίσου σημαντικό πρόβλημα υπάρχει όταν απουσιάζει η τιμή κάποιου πεδίου (π.χ. η ηλικία κάποιου χρήστη). Εάν θελήσουμε να αναλύσουμε τους χρήστες ως προς την ηλικία τους, τότε όχι μόνο πρέπει να εξαιρέσουμε τους χρήστες για τους οποίους δεν υπάρχει τιμή στο πεδίο αυτό αλλά επίσης δε θα λάβουμε και ακριβή αποτελέσματα.

**2. Όγκος δεδομένων**

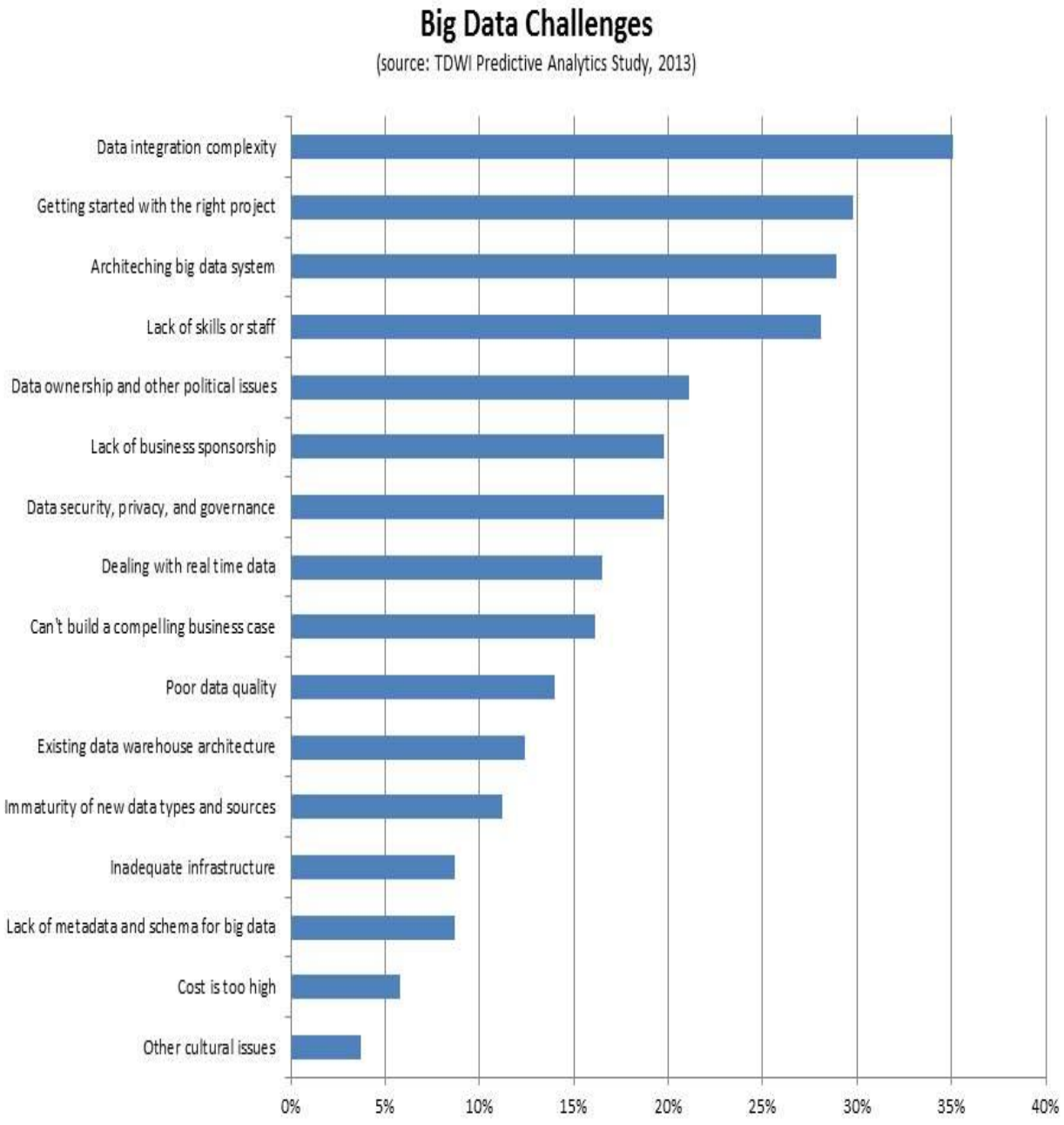
Η διαχείριση μεγάλου και ταχέως αυξανόμενου όγκου δεδομένων αποτελεί πρόκληση εδώ και πολλές δεκαετίες. Στο παρελθόν, η αύξηση της ταχύτητας των επεξεργαστών έλυνε εν μέρει το πρόβλημα. Στις μέρες μας, ο όγκος των δεδομένων αυξάνεται με δραματικούς ρυθμούς, ενώ η ταχύτητα των επεξεργαστών παραμένει σχεδόν σταθερή. Αυτές οι πρωτοφανείς αλλαγές, επιβάλλουν να επανεξετάσουμε τον τρόπο με τον οποίο σχεδιάζουμε και κατασκευάζουμε τα υπολογιστικά μέρη που είναι υπεύθυνα για την επεξεργασία δεδομένων.

**3. Απόκριση συστήματος**

Εκτός από την αύξηση του όγκου των δεδομένων, πολλές σύγχρονες εφαρμογές είναι αναγκαίο να παράγουν αποτέλεσμα πολύ γρήγορα. Αυτό συμβαίνει στις περιπτώσεις που τα αποτελέσματα πρέπει να αναλυθούν κατευθείαν, σχεδόν σε πραγματικό χρόνο, ώστε να γίνουν κάποιες ενέργειες. Για παράδειγμα, αν υπάρχει υποψία για παράνομη συναλλαγή μέσω πιστωτικής κάρτας, το σύστημα πρέπει σχεδόν σε πραγματικό χρόνο να ακυρώσει την κάρτα πριν καν η συναλλαγή προλάβει να ολοκληρωθεί. Ένα άλλο παράδειγμα έχει να κάνει με τη διαχείριση της κυκλοφοριακής συμφόρησης, όπου πάλι το σύστημα πρέπει να επεξεργαστεί πολλά δεδομένα άμεσα ώστε να προτείνει κάποια εναλλακτική διαδρομή.

## 4. Προστασία Προσωπικών Δεδομένων

Μία από τις σημαντικότερες πρακτικές που χρησιμοποιούμε για να αποθηκεύσουμε τα Big Data, είναι το sharding. Sharding είναι η κατανομή των δεδομένων σε πολλούς servers και η τοπική επεξεργασία σε αυτούς. Όμως, ένα σημαντικό μειονέκτημα αυτής της μεθόδου είναι ότι αυτοί οι servers πρέπει να έχουν τακτική επικοινωνία μεταξύ τους μέσω του διαδικτύου και έτσι είναι πολύ ευάλωτοι σε επιθέσεις από hackers ή κακόβουλο λογισμικό.




Εικόνα 4: Οι σημαντικότερες προκλήσεις που πρέπει να αντιμετωπίσουμε στον τομέα των Big Data.

# Κεφάλαιο 2

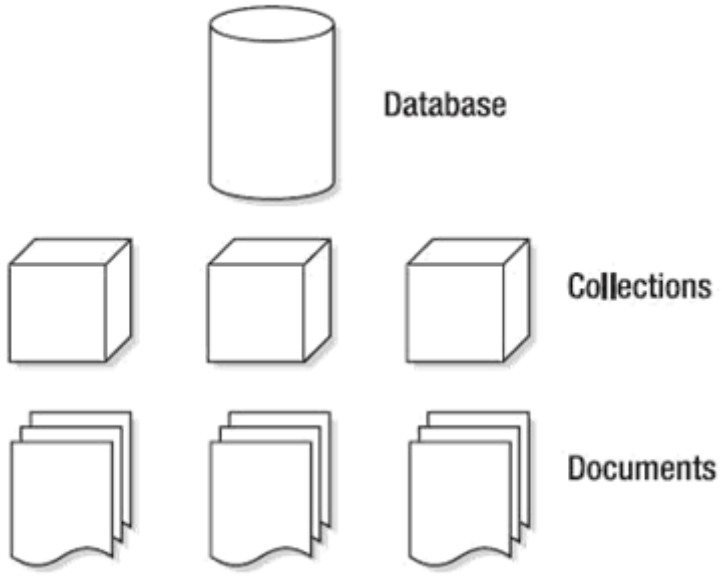
# Θεωρητική προσέγγιση των λειτουργιών της MongoDB

## 2.1 Εισαγωγή

Η ονομασία της βάσης δεδομένων MongoDB προήλθε από τη λέξη humongous που στα ελληνικά σημαίνει τεράστιος και είναι μια σχετικά νέα βάση δεδομένων, που δεν έχει την έννοια των πινάκων, των σχημάτων, της γλώσσας SQL και των συσχετίσεων (joins). Με λίγα λόγια, είναι μια πολύ διαφορετική βάση από τις συνηθισμένες σχεσιακές βάσεις δεδομένων.

Η ομάδα που σχεδίασε τη βάση αυτή, αποφάσισε να μη δημιουργήσει άλλη μια βάση δεδομένων, που θα έκανε τα πάντα. Αντίθετα ήθελε να δημιουργήσει μια βάση που θα έχει να κάνει με έγγραφα (documents), και όχι με γραμμές (rows), και θα ήταν πολύ γρήγορη, επεκτάσιμη, και εύκολη στη χρήση. Για να γίνει αυτό, η ομάδα έπρεπε να εγκαταλείψει ορισμένες λειτουργίες, πράγμα που σημαίνει ότι η MongoDB δεν είναι μια ιδανική υποψήφια βάση για όλες τις καταστάσεις. Για παράδειγμα, η απουσία της υποστήριξης συναλλαγών (transactions) σημαίνει ότι δεν θα χρησιμοποιήσουμε τη βάση MongoDB για λογιστικές εφαρμογές. Όμως η MongoDB, είναι πολύ καλή στην αποθήκευση σύνθετων δεδομένων. Η MongoDB, παρέχει μια πλούσια, προσανατολισμένη σε έγγραφα βάση, η οποία είναι σχεδιασμένη για να προσφέρει ταχύτητα και επεκτασιμότητα. Μπορεί να εκτελεστεί σχεδόν σε οποιοδήποτε λειτουργικό σύστημα χρειαστεί, όπως Windows, Linux και Mac.

Η εγκατάσταση της MongoDB φιλοξενεί έναν αριθμό βάσεων δεδομένων. Όπως φαίνεται και στην παρακάτω εικόνα, κάθε βάση δεδομένων από αυτές περιέχει ένα σύνολο από συλλογές (collections). Κάθε συλλογή περιέχει ένα σύνολο από έγγραφα. Τέλος, το κάθε έγγραφο είναι ένα σύνολο από ζεύγη κλειδιού-τιμής.




Εικόνα 5: Η δομή της MongoDB.

Τα έγγραφα 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.


Εικόνα 6: Παράδειγμα της δομής ενός εγγράφου στην MongoDB.

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. Στη δική μας περίπτωση η γλώσσα προγραμματισμού είναι η Java.

Η τιμή ενός πεδίου μπορεί να είναι οποιουδήποτε τύπου δεδομένων BSON, συμπεριλαμβανομένων άλλων εγγράφων, πινάκων και πινάκων από έγγραφα. Για παράδειγμα, το έγγραφο που φαίνεται στην εικόνα 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)
            }
```

Εικόνα 7: Έγγραφο με ποικίλους τύπους δεδομένων.

Εδώ βλέπουμε πως:

- Το πεδίο _id περιέχει ObjectId
- Το πεδίο name περιέχει ένα ενσωματωμένο έγγραφο που περιέχει τα πεδία first και last
- Τα πεδία birth και death περιέχουν τιμές τύπου Date
- Το contribs περιέχει έναν πίνακα από συμβολοσειρές (array of strings)
- Το views περιέχει μία τιμή τύπου NumberLong

Όσον αφορά το πεδίο _id, εξ' ορισμού η MongoDB δημιουργεί ένα μοναδικό ευρετήριο γι' αυτό κατά τη δημιουργία της συλλογής. Επίσης είναι πάντα το πρώτο πεδίο στα έγγραφα και μπορεί να περιέχει κάθε τύπο BSON πλην των πινάκων.

Ο χρήστης της MongoDB χρησιμοποιεί την τελεία για να έχει πρόσβαση στα στοιχεία ενός πίνακα και στα πεδία ενός ενσωματωμένου εγγράφου. Δηλαδή θα πρέπει να χρησιμοποιήσει κάτι τέτοιο:

'<array>.<index>' ή '<embedded document>.<field>'

για πίνακα και για ενσωματωμένο έγγραφο, αντίστοιχα. Συγκεκριμένα, στο παρακάτω παράδειγμα, για να έχουμε πρόσβαση στο πεδίο phone που περιέχει το τηλέφωνο του χρήστη '123xyz', θα γράφαμε contact.phone


Εικόνα 8: Έγγραφο με εμφολευμένο έγγραφο.

## 2.2 CRUD operations

CRUD operations είναι οι λειτουργίες που μπορεί να επιτελέσει κάποιος στη βάση δεδομένων MongoDB. Πιο συγκεκριμένα αυτές είναι οι εξής:

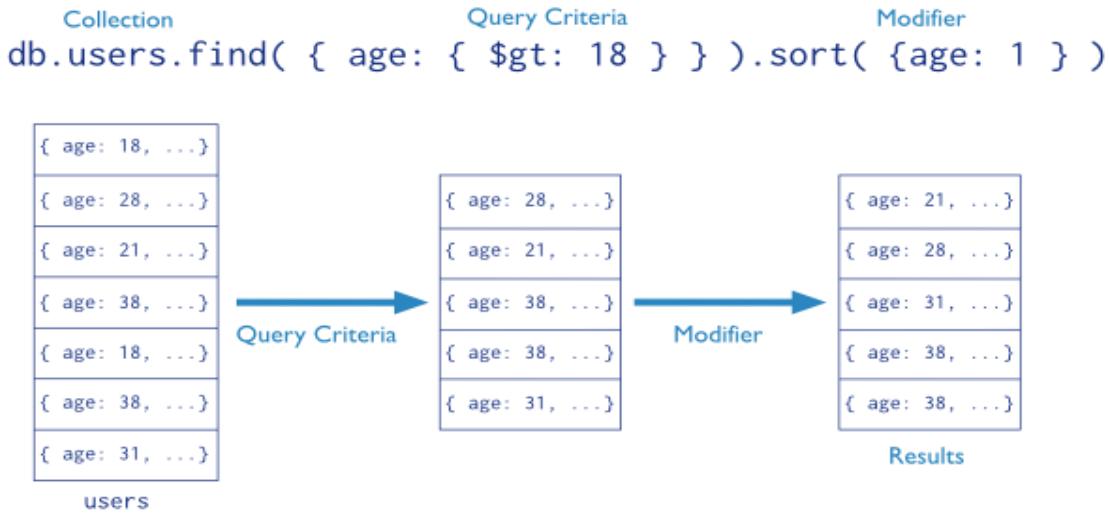
1. Create (δημιουργία), δηλαδή η εισαγωγή ενός εγγράφου στη βάση δεδομένων. Η λειτουργία αυτή εκτελείται στην MongoDB με την εντολή insert().
2. Read (ανάγνωση), δηλαδή όταν θέτω κάποιο ερώτημα (query) στη βάση δεδομένων προκειμένου να λάβω το αντίστοιχο αποτέλεσμα. Η λειτουργία αυτή, εκτελείται με την εντολή find().
3. Update (ενημέρωση), δηλαδή η τροποποίηση ή η προσθήκη ενός νέου πεδίου(field) σε κάποιο ήδη υπάρχον έγγραφο της βάσης δεδομένων. Η λειτουργία αυτή, εκτελείται με την ομώνυμη εντολή update().
4. Delete, δηλαδή η ολοκληρωτική διαγραφή κάποιου έγγραφου, εκτελώντας την εντολή delete().

Για ευκολία, ομαδοποιούμε αυτές τις τέσσερις λειτουργίες σε δύο επιμέρους ομάδες. Την Read operations (λειτουργίες ανάγνωσης) που αφορά την εντολή find() με την οποία ουσιαστικά διαβάζουμε κάποια από τα περιεχόμενα της βάσης και την ομάδα Write operations (λειτουργίες εγγραφής) που περιλαμβάνει τις εντολές insert(),update() kai delete() με τις οποίες τροποποιούμε το περιεχόμενο της βάσης.

## 2.2.1 Λειτουργίες ανάγνωσης

Για τα ερωτήματα (queries) η MongoDB παρέχει ένα σύνολο από τελεστές για να καθορίσουν πως η μέθοδος find(), η οποία στο mongo shell γράφεται db.collection.find(), θα επιστρέψει το κατάλληλο αποτέλεσμα. Συγκεκριμένα , η find() επιλέγει τα δεδομένα από μία συλλογή με βάση ένα έγγραφο προσδιορισμού ερωτημάτων που χρησιμοποιεί ένα συνδυασμό ακριβών ταιριασμάτων ισότητας και συνθηκών.

Ένα ερώτημα στοχεύει σε μία συγκεκριμένη συλλογή εγγράφων. Τα ερωτήματα καθορίζουν τα κριτήρια ή τις συνθήκες, τα οποία ή οι οποίες ορίζουν τα έγγραφα που η MongoDB θα επιστρέψει στο χρήστη. Μπορούμε να μεταβάλλουμε προαιρετικά τα ερωτήματα, ούτως ώστε να επιβάλλουμε όρια, παραβλέψεις και σειρά ταξινόμησης. Ένα παράδειγμα τέτοιου ερωτήματος φαίνεται στην εικόνα που ακολουθεί.




Εικόνα 9: Η εντολή αναζήτησης find().

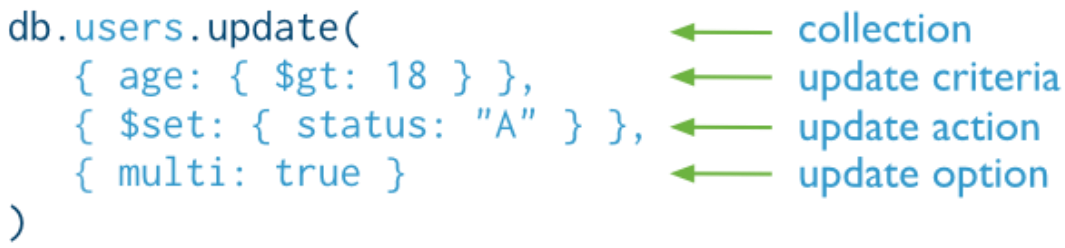
Πιο συγκεκριμένα, η εντολή find() δε γυρίζει τα έγγραφα που πληρούσαν το ερώτημά μας, αλλά έναν κέρσορα (cursor), ο οποίος δείχνει σε αυτά. Για να δημιουργήσουμε έναν κέρσορα στο mongo shell, θα γράφαμε την εντολή της παραπάνω εικόνας, ως εξής:

```
var myCursor = db.users.find( { age : { $gt : 18 } } ).sort( { age : 1} )
```

Για να έχουμε πρόσβαση στα έγγραφα, θα πρέπει να διατρέξουμε (iterate) τον κέρσορα αυτόν. Σε αντίθετη περίπτωση, όταν ο κέρσορας που επιστρέφει η MongoDB δεν έχει ανατεθεί σε κάποια μεταβλητή (όπως εδώ που θέσαμε την var myCursor), τότε ο κέρσορας διατρέχετε αυτόματα 20 φορές ώστε να εκτυπωθούν τα 20 πρώτα έγγραφα που έκαναν 'match' το query μας.

## 2.2.2 Λειτουργίες εγγραφής

Εκτός της ανάγνωσης των εγγράφων (μέσω της find() και των ερωτημάτων), μπορούμε και να τροποποιήσουμε τα δεδομένα. Οι λειτουργίες που επιτελούνται από τις εντολές insert, update και remove εκτελούνται ατομικά για ένα συγκεκριμένο έγγραφο (document level atomicity). Για την ενημέρωση και τη διαγραφή τα κριτήρια της επιλογής των εγγράφων μπορούν να καθοριστούν. Όσον αφορά την ενημέρωση, αυτή μπορεί είτε να αντικαταστήσει ολόκληρο κάποιο έγγραφο με κάποιο άλλο, είτε να τροποποιήσει κάποια από τα υπάρχοντα πεδία του εγγράφου, είτε να δημιουργήσει νέα πεδία σε αυτό, εάν δεν υπάρχουν ήδη. Για παράδειγμα, με την εντολή που φαίνεται στην εικόνα 10, αφού βρούμε όλα τα έγγραφα στη συλλογή users που έχουν στο πεδίο ηλικία (age) τιμή μεγαλύτερη από 18, θέτουμε στο πεδίο τους status την τιμή ‘’A’’.

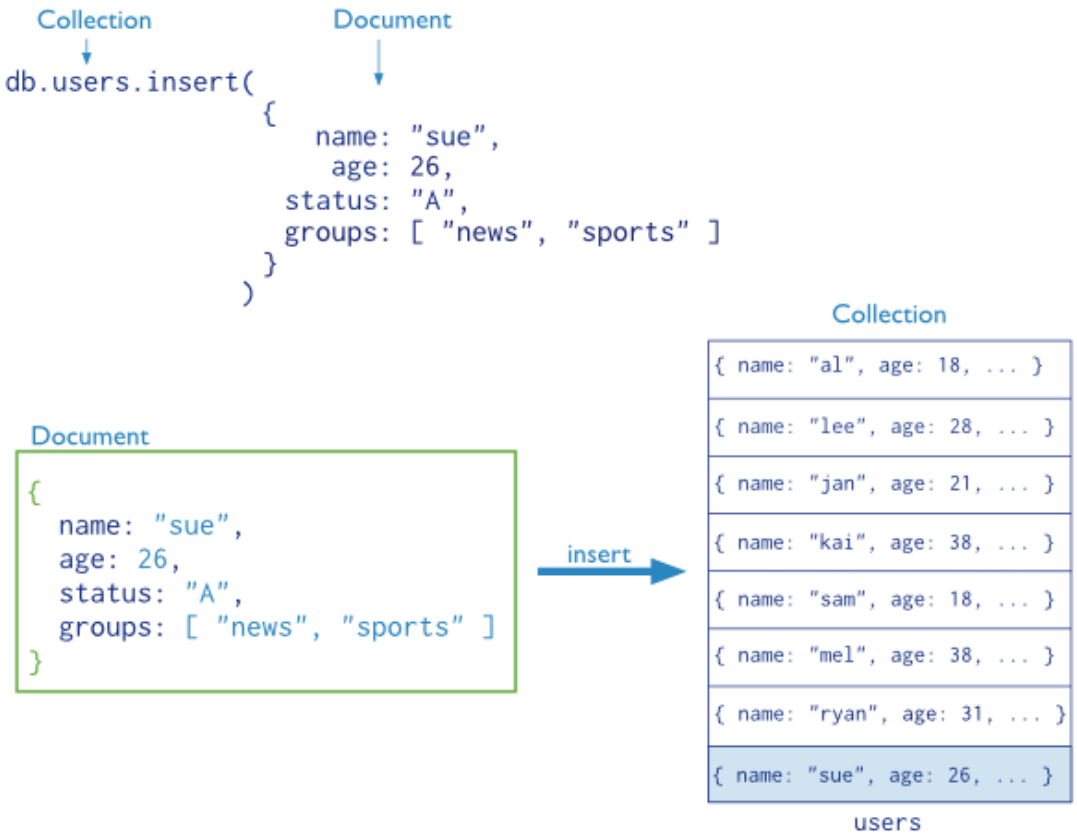



Εικόνα 10: Η μέθοδος update().

Παρόμοια, με την εντολή delete, μπορούμε να διαγράψουμε τα έγγραφα, τα οποία κάνουν ‘match’ τον περιορισμό μας. Για παράδειγμα, με την εντολή που φαίνεται στην παρακάτω εικόνα, διαγράφουμε όλα τα έγγραφα της συλλογής users που στο πεδίο ‘status’ έχουν τιμή ‘’D’’.


Εικόνα 11: Η μέθοδος remove().

Τέλος, για την εισαγωγή κάποιου έγγραφου χρησιμοποιούμε την εντολή insert(). Παράδειγμα της εισαγωγής δεδομένων σε μία συλλογή φαίνεται παρακάτω.



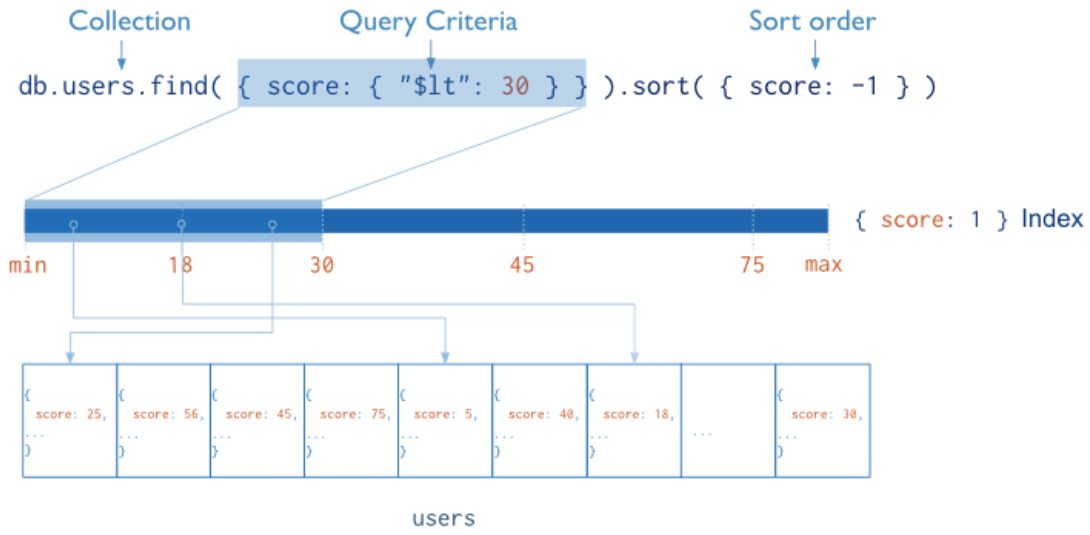

Εικόνα 12: Η μέθοδος insert().

Εάν εισάγουμε ένα έγγραφο χωρίς το πεδίο _id (όπως στο παραπάνω παράδειγμα), τότε η MongoDB προσθέτει αυτόματα το πεδίο αυτό και του θέτει μοναδική τιμή τύπου ObjectId. Σε περίπτωση που έχουμε δώσει τιμή στο πεδίο _id τότε αυτή θα πρέπει να είναι μοναδική για αυτή την collection, αλλιώς ενεργοποιείτε εξαίρεση (exception) και συγκεκριμένα η duplicate key exception.

## 2.3 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 κάνει full collection scan, δηλαδή να διατρέξει όλα τα έγγραφα της συλλογής, ώστε να επιστρέψει εκείνα τα έγγραφα που κάνουν 'match' στον 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 (και συγκεκριμένα είναι B-tree δομές δεδομένων) 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 για όλα τα έγγραφα μιας συλλογής, ταξινομημένες αλφαριθμητικά με βάση την ίδια την τιμή. 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 13: Παράδειγμα ερωτήματος που χρησιμοποιεί ένα ευρετήριο.

Τέλος, αξίζει να αναφέρουμε ότι η MongoDB βελτιστοποιεί αυτόματα τα ερωτήματα που της θέτουμε, ώστε να κάνει την αποτίμηση τους πιο αποδοτική. Αναλυτικότερα, υπάρχει ένας ειδικός μηχανισμός, ο οποίος ονομάζεται query optimizer (βελτιστοποιητής ερωτημάτων) και είναι υπεύθυνος για την επιλογή του αποδοτικότερου index για οποιοδήποτε ερώτημα θέσουμε. Για να το επιτύχει αυτό, τρέχει περιοδικά, διάφορες ομάδες ερωτήσεων (query plans) και επιλέγει για κάθε μία από αυτές το ευρετήριο, το οποίο είχε τον καλύτερο χρόνο απόκρισης. Τα αποτελέσματα αυτών των εμπειρικών μετρήσεων αποθηκεύονται στην κρυφή μνήμη του υπολογιστή και ενημερώνονται περιοδικά.

## 2.3.1 Τύποι ευρετηρίων

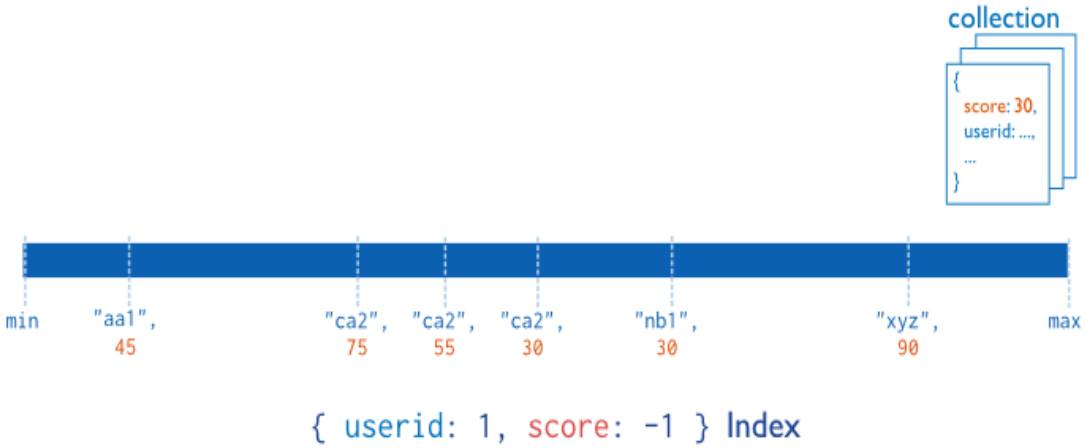
Η MongoDB προσφέρει μια πληθώρα διαφορετικών τύπων ευρετηρίων που υποστηρίζουν διάφορους τύπους δεδομένων και ερωτημάτων.

- Αρχικά, υπάρχει το Default _id ευρετήριο. Αυτό είναι το ευρετήριο πάνω στην τιμή "_id" που υπάρχει πάντα και δημιουργείται αυτόματα από την MongoDB. Το ευρετήριο _id είναι μοναδικό και αυτό είναι που αποτρέπει την εισαγωγή δύο εγγράφων, τα οποία έχουν την ίδια τιμή στο πεδίο _id.

- 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. Με τον αριθμό 1 οι τιμές αποθηκεύονται σε αύξουσα σειρά, ενώ με το -1 σε φθίνουσα σειρά.


Εικόνα 14: Ευρετήριο πάνω στο πεδίο score, σε αύξουσα ταξινόμηση.

- 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 15: Σύνθετο ευρετήριο στα πεδία userid και score.

Σε αυτό το σημείο πρέπει να τονίσουμε τη σημασία που έχει η επιλογή κατάλληλης σειράς ταξινόμησης (sort order). Στα ευρετήρια μονού πεδίου (single field indexes) αυτό δεν μας ενδιέφερε αφού ότι και να επιλέγαμε (είτε αύξουσα είτε φθίνουσα ταξινόμηση), η MongoDB μπορεί να διασχίσει το ευρετήριο και προς τις δύο κατευθύνσεις. Όμως, όσον αφορά τα σύνθετα ευρετήρια, η επιλογή αυτή έχει μεγάλη σημασία αφού καθορίζει εάν ένα ευρετήριο μπορεί να χρησιμοποιηθεί για να επιστρέψει τα αποτελέσματα σύμφωνα με την ταξινόμηση που ζητήσαμε.

Για παράδειγμα, θεωρήστε την events collection, η οποία περιέχει έγγραφα με τα πεδία username και date. Έστω, ότι επιλέγουμε να φτιάξουμε ένα σύνθετο ευρετήριο σε αυτή τη συλλογή, το οποίο θα ταξινομεί σε αύξουσα σειρά τις τιμές του username και σε φθίνουσα τις τιμές του πεδίου date. Η εντολή αυτή στο mongo shell θα γραφόταν ως εξής:

```
db.events.createIndex( { "username" : 1, "date" : -1 } )
```

Επομένως, μπορώ να χρησιμοποιήσω αυτό το ευρετήριο είτε στο ερώτημα 1, όπου το ευρετήριο θα διασχιστεί κανονικά, είτε στο ερώτημα 2, όπου το ευρετήριο θα διασχιστεί ανάποδα.

1) db.events.find().sort( { username: 1, date: -1 } )
2) db.events.find().sort( { username: -1, date: 1 } )

Όμως, το ευρετήριο που δημιουργήσαμε θα ήταν άχρηστο στις παρακάτω δύο ερωτήσεις:

1) db.events.find().sort( { username: 1, date: 1 } )
2) db.events.find().sort( { username: -1, date: -1 } )

Τέλος, 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, έστω ότι έχουμε το ακόλουθο σύνθετο ευρετήριο:

{ "item": 1, "location": 1, "stock": 1 }

Αυτό, έχει τα ακόλουθα προθέματα (prefixes):

{ item: 1 }

{ item: 1, location: 1 }

Οπότε, σύμφωνα με τα παραπάνω η MongoDB μπορεί να χρησιμοποιήσει το ευρετήριο αυτό σε ερωτήματα που εμπεριέχουν τα παρακάτω πεδία:

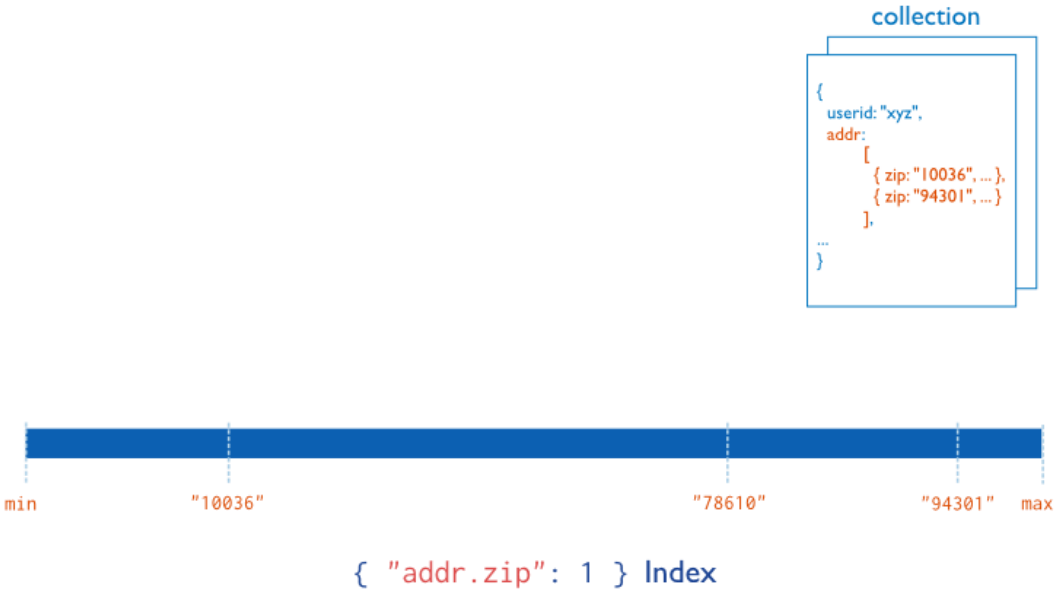
1) το πεδίο item.

2) το πεδίο item και το πεδίο location.

3) το πεδίο item, το πεδίο location και το πεδίο stock.

Όμως το παραπάνω ευρετήριο δεν μπορεί να χρησιμοποιηθεί σε όλες τις περιπτώσεις όπου τα ερωτήματα δεν εμπεριέχουν το πεδίο item.

- Η τελευταία κατηγορία ευρετηρίων είναι τα ευρετήρια πολλαπλών κλειδιών ή αλλιώς multikey indexes. 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 εμφολευμένα έγγραφα (nested documents).




Εικόνα 16: Ευρετήριο πολλαπλών κλειδιών σε πίνακα με εμφολευμένα έγγραφα.

## 2.3.2 Ιδιότητες ευρετηρίων

Μία σημαντική ιδιότητα είναι αυτή των μοναδικών ευρετηρίων (Unique indexes). Όταν το ευρετήριο μας έχει αυτήν την ιδιότητα, τότε η MongoDB υποχρεούται να απορρίψει όλα τα έγγραφα που δεν έχουν μοναδική τιμή στο πεδίο για το οποίο έχει δημιουργηθεί το ευρετήριο. Για παράδειγμα, για να δημιουργήσουμε ένα ευρετήριο στο πεδίο user_id της member collection με την ιδιότητα αυτή, απλά γράφουμε την παρακάτω εντολή στο mongo shell:

```
db.members.createIndex( { "user_id" : 1 } , { unique : true } )
```

Επίσης, μία πολύ σημαντική ιδιότητα την οποία θα χρησιμοποιήσουμε στη συνέχεια είναι αυτή του αραιού ευρετηρίου (sparse index). Ειδικότερα, τα ευρετήρια αυτά κάνουν καταχωρήσεις μόνο για έγγραφα που περιέχουν το πεδίο στο οποίο δημιουργήθηκε το ευρετήριο (indexed field) ακόμα και αν αυτό έχει τιμή null. Οπότε το ευρετήριο αυτό είναι “αραιό” αφού δεν περιέχει όλα τα έγγραφα μίας συλλογής. Αντίθετα, ένα μη-αραιό ευρετήριο (non-sparce index) περιέχει όλα τα έγγραφα μίας συλλογής αποθηκεύοντας την τιμή null για τα έγγραφα τα οποία δεν περιέχουν το indexed field. Για παράδειγμα, για να δημιουργήσουμε ένα ευρετήριο στο πεδίο zip_code της addresses collection με την ιδιότητα αυτή απλά γράφουμε την παρακάτω εντολή στο mongo shell:

```
db.addresses.createIndex( { "zip_code" : 1 } , { sparse : true } )
```

Τέλος, μία άλλη ενδιαφέρουσα ιδιότητα είναι αυτή του TTL ευρετηρίου. Αναλυτικότερα, 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. Τα δεδομένα με “ημερομηνία λήξης” είναι πολύ χρήσιμα για εφαρμογές όπου τα δεδομένα μου είναι machine generated (δημιουργήθηκαν από τον υπολογιστή) ή logs, τα οποία χρειάζονται να υπάρχουν στη βάση για κάποιο συγκεκριμένο χρονικό διάστημα και μετά είναι άχρηστα.

## 2.3.3 Διασταύρωση ευρετηρίων

Η MongoDB μπορεί να διασταυρώσει πολλά ευρετήρια (index intersection), ώστε να απαντήσει στα ερωτήματα μας. Για παράδειγμα, σκεφτείτε μια συλλογή orders που έχει δύο index, το { qty: 1 } και το { item: 1 }. Έστω τώρα ότι κάνουμε το παρακάτω ερώτημα:

```
db.orders.find( { item: "abc123", qty: { $gt: 15 } } )
```

Για να απαντήσει στο ερώτημα αυτό, η MongoDB θα μπορούσε να χρησιμοποιήσει την διασταύρωση των δύο αυτών ευρετηρίων. Για να δούμε εάν η βάση δεδομένων χρησιμοποίησε κάποια από τα ευρετήρια αλλά και ποια ακριβώς από αυτά, τρέχουμε την εντολή explain().

Η διασταύρωση ευρετηρίων μπορεί να γίνει και μεταξύ κάποιου ευρετηρίου και κάποιου προθέματος ενός σύνθετου ευρετηρίου. Για παράδειγμα, έστω ότι στη συλλογή orders έχουμε το ευρετήριο μονού πεδίου { qty: 1 } και το σύνθετο ευρετήριο {status: 1, ord_date: -1 }. Για να απαντήσει το παρακάτω ερώτημα, η MongoDB θα διασταυρώσει το ευρετήριο { qty: 1 } με το πρόθεμα ευρετηρίου { status: 1}.

```
db.orders.find( { qty: { $gt: 10 } , status: "A" } )
```

Όπως παρατηρούμε, η δυνατότητα της διασταύρωσης ευρετηρίων μοιάζει πολύ με αυτό που προσφέρουν τα σύνθετα ευρετήρια. Η διαφορά είναι, όπως αναφέραμε ότι στα σύνθετα ευρετήρια διαδραματίζει σημαντικό ρόλο η σειρά με την οποία θα γράψουμε τα πεδία του ευρετηρίου (list order) κατά τη δημιουργία του, αλλά και η σειρά ταξινόμησης (sort order) που θα δώσουμε σε καθένα από αυτά.

Ειδικότερα, έστω ότι είχαμε το σύνθετο ευρετήριο { status: 1, ord_date: -1 }. Αυτό θα μπορούσε να απαντήσει στα παρακάτω δύο ερωτήματα:

1) 
```
db.orders.find( { status: { $in: ["A", "P" ] } } )
```
2) 
```
db.orders.find(
    {
            ord_date: { $gt: new Date("2014-02-01") },
            status: {$in:[ "P", "A" ] }
     }
    )
```

Όμως δεν θα μπορούσε να χρησιμοποιηθεί για τα δύο παρακάτω ερωτήματα, για τους λόγους που είχαμε εξηγήσει κατά τον ορισμό του σύνθετου ευρετηρίου.

1) db.orders.find( { ord_date: { $gt: new Date("2014-02-01") } } )

2) db.orders.find( { } ).sort( { ord_date: 1 } )

Αντίθετα, αν είχαμε τα μονού πεδίου ευρετήρια { status: 1 } και { ord_date: -1 } θα μπορούσαμε να χρησιμοποιήσουμε την διασταύρωση τους για να απαντήσουμε και στα τέσσερα ερωτήματα.

Σε αυτό το σημείο, εύλογα γεννάτε η απορία γιατί να χρησιμοποιούμε σύνθετα ευρετήρια αντί να φτιάχνουμε μονά ευρετήρια στο κάθε πεδίο και να αφήνουμε τη διασταύρωση των ευρετηρίων να κάνει την υπόλοιπη δουλειά. Η απάντηση είναι, πως αν ξέρουμε περίπου τη μορφή των ερωτημάτων μας, τότε είναι προτιμότερο να φτιάχνουμε σύνθετα ευρετήρια που μπορούν να τα απαντήσουν, καθώς είναι περισσότερο αποδοτικά σε σχέση με τη διασταύρωση ευρετηρίων που είναι μία ακριβή διαδικασία σε χρόνο και υπολογιστικούς πόρους. Ο τελευταίος ισχυρισμός μας, θα αποδειχθεί με τις μετέπειτα μετρήσεις που θα κάνουμε στη βάση δεδομένων μας.

Τέλος, όσον αφορά τη διασταύρωση ευρετηρίων σε ερωτήματα που περιέχουν τη συνάρτηση sort() πρέπει να τονίσουμε ότι αυτή είναι δυνατή αν και μόνο αν οι περιορισμοί του ερωτήματος περιέχουν τουλάχιστον ένα από τα πεδία του ευρετηρίου που θα επιλεχθεί για τη ταξινόμηση. Για να κατανοήσουμε το παραπάνω, ας δούμε ένα παράδειγμα.

Έστω ότι στη συλλογή orders έχουμε τα παρακάτω ευρετήρια:

{ qty: 1 }

{ status: 1, ord_date: -1 }

{ status: 1 }

{ ord_date: -1 }

Στο παρακάτω ερώτημα η MongoDB θα επιδίωκε να χρησιμοποιήσει διασταύρωση του ευρετηρίου { qty: 1 } και του { status: 1, ord_date: -1 }.

db.orders.find( { qty: { $gt: 10 } } ).sort( { status: 1 } )

Όμως, παρατηρούμε ότι κανένα πεδίο του ευρετηρίου { status: 1, ord_date: -1 } που χρειαζόμαστε για τη ταξινόμηση, δεν περιέχεται στο κυρίως ερώτημα. Για αυτόν τον λόγο είναι αδύνατη η διασταύρωση ευρετηρίων. Αντίθετα, στο παρακάτω ερώτημα, η διασταύρωση ευρετηρίων είναι εφικτή για τους λόγους που αναφέραμε.

db.orders.find( { qty: { $gt: 10 } , status: "A" } ).sort( { ord_date: -1 } )

## 2.4 Sharding

Shard στα αγγλικά σημαίνει θραύσμα. Ονόμασαν έτσι αυτή τη διαδικασία (sharding) ώστε να τονίσουν το “σπάσιμο” των δεδομένων σε πολλά μικρότερα κομμάτια. Συγκεκριμένα, το σπάσιμο των δεδομένων (θα το αναφέρουμε ως sharding στο εξής) είναι μία μέθοδος που μας επιτρέπει να αποθηκεύουμε τα δεδομένα μας σε πολλά ξεχωριστά μηχανήματα (servers). Η MongoDB, χρησιμοποιεί το sharding ώστε να μπορεί να υποστηρίξει εφαρμογές που διαχειρίζονται πολύ μεγάλο όγκο δεδομένων (large data set) και έχουν υψηλό εύρος ζώνης (high data transfer rate).

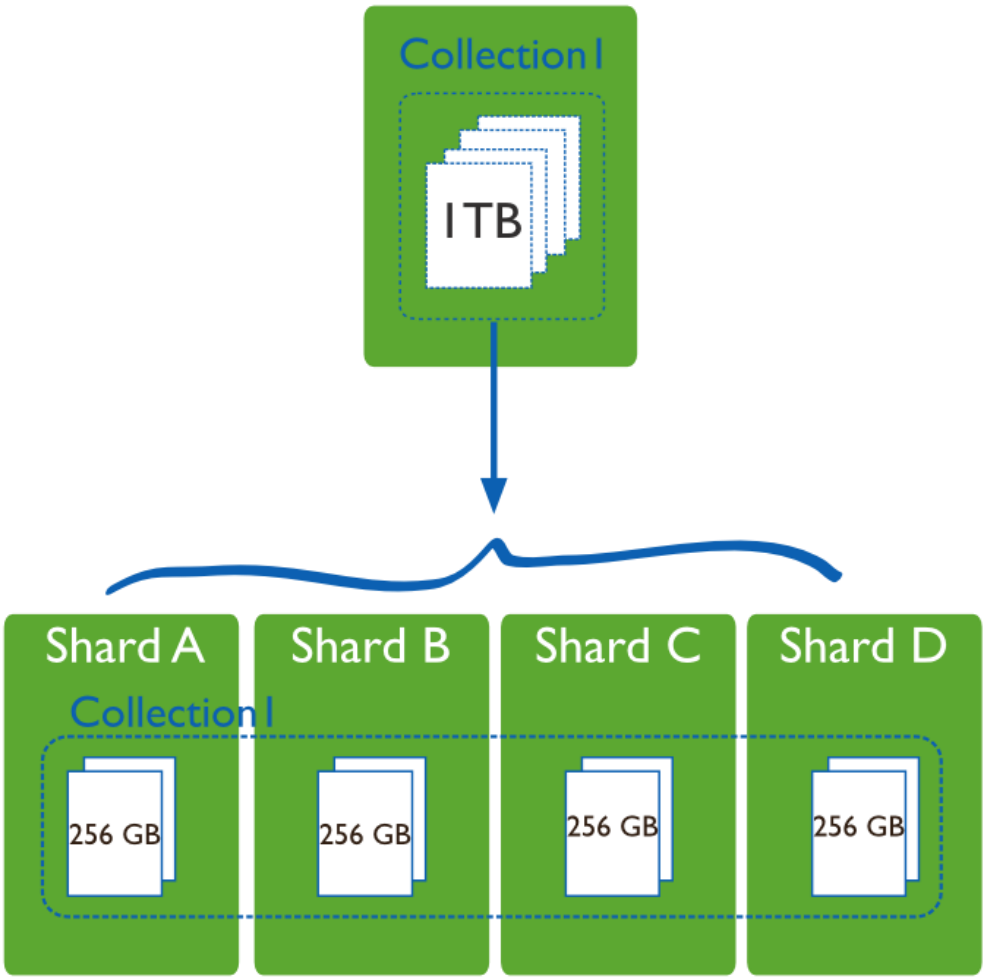
### 2.4.1 Σκοπός του Sharding

Οι βάσεις δεδομένων που αποθηκεύουν πολύ μεγάλο όγκο δεδομένων και έχουν υψηλό εύρος ζώνης, “ζορίζονται” όταν λειτουργούν μόνο σε ένα μηχάνημα (single server). Ο υψηλός ρυθμός ερωτημάτων μπορεί να εξαντλήσει την επεξεργαστική δύναμη του μηχανήματος. Επίσης, ο μεγάλος όγκος δεδομένων της βάσης πολλές φορές ξεπερνά τις αποθηκευτικές δυνατότητες ενός μηχανήματος. Τέλος, το σύνολο εργασίας (working set) μπορεί να ξεπερνά σε μέγεθος τη μνήμη ram του μηχανήματος και αυτό έχει ως αποτέλεσμα να λειτουργεί ακατάπαυστα ο σκληρός δίσκος κάτι που του προκαλεί μεγάλη φθορά με τον καιρό.

Υπάρχουν δύο βασικές προσεγγίσεις ώστε τα συστήματα βάσεων δεδομένων να αντιμετωπίσουν το πρόβλημα αυτό, η κατακόρυφη κλιμάκωση και το sharding.

- Η **κατακόρυφη κλιμάκωση ή vertical scaling** προσθέτει σε ένα μηχάνημα περισσότερους επεξεργαστές και περισσότερο αποθηκευτικό χώρο, ώστε να αυξήσει τις επιδόσεις του συστήματος αυτού. Όμως, η βελτίωση ενός υπολογιστικού συστήματος με αυτόν τον τρόπο έχει πολύ μεγαλύτερο κόστος συγκριτικά με μικρότερα συστήματα, ενώ επίσης αυξάνεται σημαντικά η πολυπλοκότητα του. Ως αποτέλεσμα, υπάρχει ένα συγκεκριμένο “ταβάνι” μέχρι το οποίο μπορούμε να βελτιώσουμε ένα υπολογιστικό σύστημα, ώστε να επιτύχουμε κατακόρυφη κλιμάκωση.

- Το **sharding ή horizontal scaling**, αντιθέτως, χωρίζει το σύνολο των δεδομένων σε μικρότερα επιμέρους κομμάτια δεδομένων και τα διανέμει σε πολλά διαφορετικά μηχανήματα. Το κάθε ένα από αυτά τα μηχανήματα που στο εξής θα αποκαλούμε shard, είναι μία ανεξάρτητη βάση δεδομένων. Όμως οι shards στο σύνολό τους, αποτελούν θεωρητικά μία και μοναδική ενιαία βάση δεδομένων.




Εικόνα 17: Μια απλή περιγραφή του sharding.

Στην παραπανω εικόνα, βλέπουμε ένα παράδειγμα του sharding, στο οποίο μία συλλογή δεδομένων μεγέθους 1TB, έχει 'σπάσει' σε τέσσερα ίσα κομμάτια και έχει αποθηκευτεί σε ένα σύμπλεγμα (cluster) που αποτελείται από τέσσερις shards.

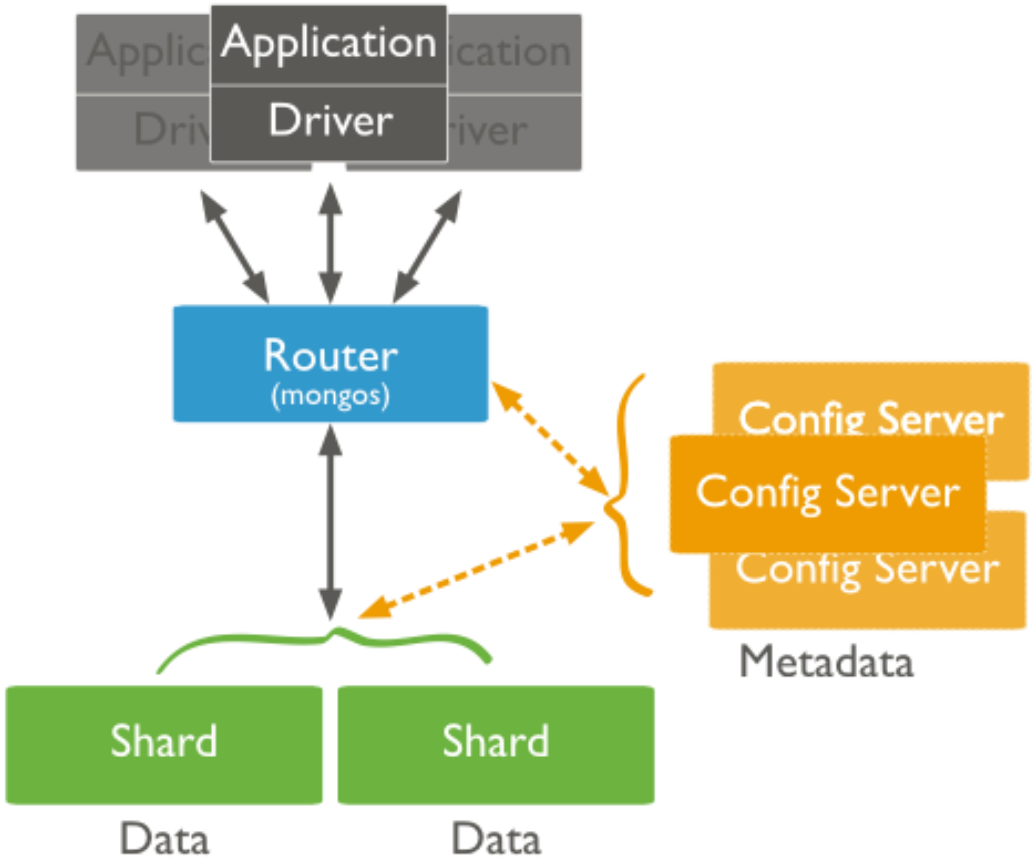
Με την τεχνική του sharding, καταφέρνουμε να μειώσουμε τον αριθμό των λειτουργιών/εργασιών που επιτελεί ο κάθε shard, δηλαδή ο κάθε ένας από τους servers ενός συμπλέγματος (cluster). Αυτό συμβαίνει διότι ο κάθε shard είναι υπεύθυνος για την επεξεργασία μόνο των δεδομένων που έχει ο ίδιος. Για παράδειγμα, αν θέλουμε να εισάγουμε δεδομένα σε κάποιο έγγραφο, τότε η εφαρμογή χρειάζεται να έχει πρόσβαση μόνο στον shard που περιέχει το συγκεκριμένο έγγραφο.

Τέλος, με το sharding μειώνεται το πλήθος των δεδομένων που χρειάζεται να αποθηκεύσει ο κάθε server. Ο κάθε shard αποθηκεύει όλο και λιγότερα δεδομένα καθώς το cluster μεγαλώνει. Για παράδειγμα, αν μία βάση δεδομένων έχει μέγεθος 1 TB και έχουμε 4 shards (όπως στην εικόνα 17), τότε ο καθένας θα αποθηκεύσει 256GB. Εάν είχαμε ένα cluster με 40 shards, τότε ο καθένας από αυτούς θα χρειαζόταν να αποθηκεύσει μόνο 25GB δεδομένων.

Συνεπώς, ένα cluster μπορεί να επεκτείνει την επεξεργαστική του ισχύ, αλλά και τις αποθηκευτικές του δυνατότητες ‘οριζόντια’, δηλαδή χωρίς να υπάρχει ταβάνι που να το περιορίζει.

## 2.4.2 Η αρχιτεκτονική του sharded cluster

Ένα sharded cluster στην MongoDB, έχει την παρακάτω αρχιτεκτονική.




Εικόνα 18: Η αρχιτεκτονική του MongoDB sharded cluster.

Ένα sharded cluster αποτελείται από τα παρακάτω μέρη.

- Δυο ή περισσότερους **shards**. Όπως αναφέραμε, shard είναι το ‘εξάρτημα΄ της MongoDB, το οποίο είναι υπεύθυνο για την αποθήκευση των δεδομένων.

- Έναν ή περισσότερους **query routers** (δρομολογητές ερωτημάτων). 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. Τέλος, μια εφαρμογή-πελάτης στέλνει τα ερωτήματά της σε έναν query server, όμως ένας query server μπορεί να εξυπηρετεί πολλούς πελάτες.

- Τρεις ακριβώς **configuration servers,** οι οποίοι αποθηκεύουν τα metadata (μεταδεδομένα) του cluster. Τα μεταδεδομένα είναι σαν τον χάρτη του cluster, δηλαδή μας πληροφορούν σε ποιον shard βρίσκεται το κάθε "κομμάτι" δεδομένων. Αυτά είναι τα δεδομένα τα οποία χρησιμοποιεί ο query server, ώστε να προωθεί τα ερωτήματα του πελάτη στους κατάλληλους shards. Οι τρεις configuration servers, κρατάνε αντίγραφα των ίδιων μεταδεδομένων. Αυτό γίνεται για να υπάρχει διαθεσιμότητα (availability) ακόμα και όταν ένας ή δυο από τους τρεις δεν λειτουργούν. Σε διαφορετική περίπτωση, εάν υπήρχε μόνο ένας configuration server και έβγαινε εκτός λειτουργίας, τότε το cluster θα ήταν μη λειτουργικό, αφού δεν θα γνωρίζαμε σε ποιον shard βρίσκεται το κάθε κομμάτι δεδομένων.

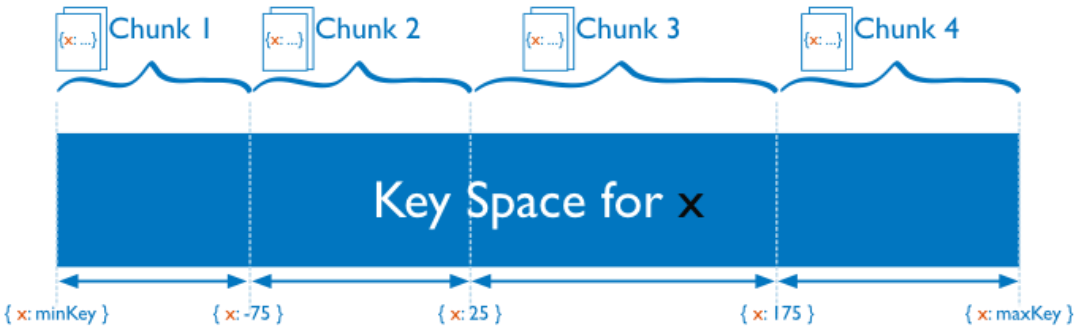
## 2.4.3 Διαχωρισμός των δεδομένων

Η MongoDB σπάει τα δεδομένα στο επίπεδο της συλλογής. Αυτό σημαίνει ότι μια συλλογή μπορεί να σπάσει σε πολλές μικρότερες ομάδες εγγράφων αλλά ποτέ ένα έγγραφο δεν θα σπάσει ώστε να αποθηκευτεί σε διαφορετικούς shards. 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 μοναδικό ευρετήριο (single field index) είτε περισσότερα του ενός πεδία για τα οποία υπάρχει σύνθετο ευρετήριο (compound index). Επίσης, το shard key, πρέπει να υπάρχει σε κάθε έγγραφο της συλλογής και αυτό μας το εξασφαλίζει το ευρετήριο, που όπως είπαμε, πάντα πρέπει να υφίσταται για το shard key.

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 range based partitioning (διαίρεση με βάση το εύρος τιμών) είτε την

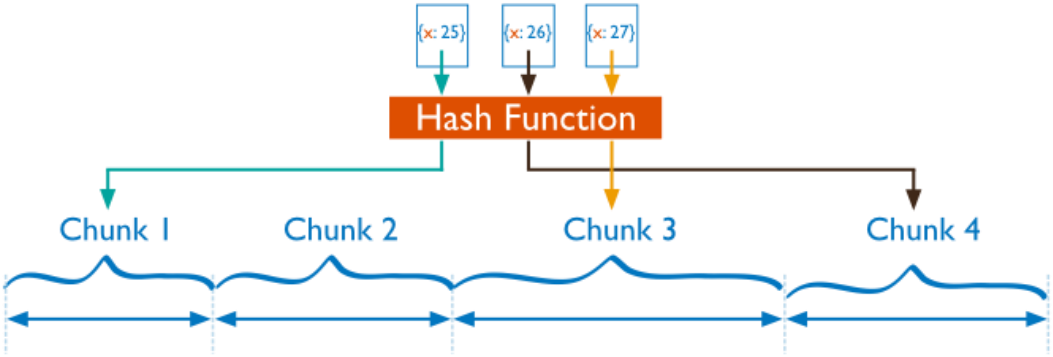
μέθοδο hashed based partitioning (διαίρεση χρησιμοποιώντας τη συνάρτηση κατακερματισμού).

- **Range based partitioning**: Έστω ότι το shard key που επιλέξαμε λαμβάνει ακέραιες τιμές. Εάν σκεφτούμε τον άξονα x'x ο οποίος λαμβάνει τιμές από το μείον άπειρο έως το συν άπειρο, τότε κάθε τιμή του shard key είναι βέβαιο πως θα υπάρχει σε κάποιο σημείο αυτού του άξονα. Αυτό που κάνει η MongoDB σε αυτή τη μέθοδο διαχωρισμού, είναι να σπάει τον άξονα αυτόν σε ίσα μη επικαλυπτόμενα διαστήματα τα οποία αποκαλούμε κομμάτια ή chunks. Οπότε το κάθε chunk περιέχει ένα εύρος τιμών του shard key, και έχει κάποια ελάχιστη τιμή και κάποια μέγιστη τιμή. Με αυτόν τον τρόπο, κρατώντας στα metadata μόνο τις δυο αυτές τιμές για το κάθε chunk, είναι πολύ εύκολο να εντοπίσουμε που βρίσκεται το κάθε έγγραφο. Επίσης, εάν αναζητούμε δύο έγγραφα με παραπλήσιες τιμές στο shard key, τότε είναι πολύ πιθανό να βρίσκονται στο ίδιο chunk και συνεπώς και στον ίδιο shard.




Εικόνα 19: Απεικόνιση του range based partitioning.

- **Hashed base partitioning**: Σε αυτή τη μέθοδο, η MongoDB, περνάει την τιμή shard key από την hash function (συνάρτηση κατακερματισμού) και χρησιμοποιεί το αποτέλεσμα που λαμβάνει ώστε να φτιάξει τα κατάλληλα chunks. Κατά συνέπεια, δύο έγγραφα με παραπλήσιες τιμές στο πεδίο που επιλέχτηκε για shard key κατά πάσα πιθανότητα θα βρίσκονται σε διαφορετικά chunks. Με αυτόν τον τρόπο διασφαλίζουμε μια πιο τυχαία κατανομή των δεδομένων της συλλογής μας στο sharded cluster.




Εικόνα 20: Απεικόνιση του hashed based partitioning.

Όσον αφορά την απόδοση, το range base partitioning είναι πολύ καλό όταν το ερώτημα μας είναι ερώτημα εύρους. Για παράδειγμα, ένα ερώτημα εύρους είναι το εξής: "Επέστρεψε όλους τους πελάτες που γεννήθηκαν από το 1980 έως το 1990". Σε αυτήν την περίπτωση, αν το ερώτημα εύρους αφορά το shard key (στο παραπάνω παράδειγμα θα έπρεπε να είχαμε θέσει για shard key την ημερομηνία γέννησης) τότε ο query router μπορεί πολύ εύκολα να υπολογίσει ποια chunks υπερκαλύπτουν το εύρος αυτό και να δρομολογήσει το ερώτημα μόνο στους shards που περιέχουν αυτά τα chunks. Όμως, πολλές φορές, αυτή η μέθοδος οδηγεί σε ανομοιογενή διαχωρισμό των δεδομένων (δηλαδή το μέγεθος τον chunks διαφέρει) κάτι που επιδρά αρνητικά στην απόδοση του sharded cluster.

Αντιθέτως, το hash based partitioning διασφαλίζει τον ομοιογενή διαχωρισμό των δεδομένων με το μειονέκτημα όμως ότι δεν υποστηρίζει πολύ αποδοτικά τα ερωτήματα εύρους. Με τη μέθοδο αυτή γίνεται τυχαία ο διαμοιρασμός των δεδομένων στα διαφορά chunks αρά και στους shards. Ως αποτέλεσμα, ο query server δεν θα μπορεί να επιλέξει κάποιους συγκεκριμένους shards για να απαντήσει το ερώτημα εύρους του πελάτη, αλλά, κατά πασά πιθανότητα θα πρέπει να προωθήσει το αίτημα σε όλους τους shards του cluster, ώστε να λάβει το αποτέλεσμα.

Τέλος, όσον αφορά τον διαχωρισμό των δεδομένων υπάρχει και μια τρίτη μέθοδος την οποία μπορεί να προσαρμόσει ο χρήστης κατάλληλα στη δική του περίπτωση χρήσης βάζοντας μια ετικέτα στο κάθε διάστημα στο οποίο θέλει να χωρίσει το shard key. Η μέθοδος αυτή ονομάζεται **Tag Aware Sharding**. Συγκεκριμένα, ο διαχειριστής της βάσης δεδομένων αφού δημιουργήσει ετικέτες και συσχετίσει την καθεμία με ένα συγκεκριμένο εύρος τιμών του shard key, αναθέτει την κάθε ετικέτα και σε κάποιον shard. Κατόπιν, η MongoDB φροντίζει να μετακινήσει το κάθε chunk στον κατάλληλο shard server, ώστε τα δεδομένα να βρίσκονται εκεί που προστάζουν οι ετικέτες.

Εκτός από την επιλογή της κατάλληλης μεθόδου για τον διαχωρισμό των δεδομένων, εξίσου, και ίσως περισσότερο σημαντική είναι η επιλογή του κατάλληλου shard key.

Κατά την επιλογή του shard key, τρεις είναι οι σημαντικότερες ιδιότητες που πρέπει να έχει σε κάποιο βαθμό.

- **Πληθικότητα (cardinality)**: Στην MongoDB η πληθικότητα έχει να κάνει με την ικανότητα που έχει το σύστημα να διασπάει τα δεδομένα στα διάφορα chunks. Για να καταλάβουμε τη σημασία της πληθικότητας ας δούμε ένα παράδειγμα. Έστω μια συλλογή "βιβλίο_διευθύνσεων'' η οποία περιέχει τα στοιχεία κάποιων πολιτών. Εάν επιλέξουμε ως shard key για τη συλλογή αυτή το πεδίο "πόλη" τότε θα έχουμε χαμηλή πληθικότητα, αφού όλα τα έγγραφα που θα έχουν την ίδια τιμή στο πεδίο αυτό (και θα είναι πολλά, μιας και μόνο στην Αθήνα διαμένει ο μισός πληθυσμός της χώρας), θα πρέπει να αποθηκευτούν στο ίδιο chunk ακόμα και αν αυτό ξεπερνά σε μέγεθος το ανώτερο επιτρεπτό. Αυτό θα είναι μεγάλο "αγκάθι" για την απόδοση του sharded cluster, καθώς ο διαμοιρασμός των δεδομένων στα διάφορα chunks θα είναι άνισος, γεγονός, το οποίο επιβαρύνει τους shards που θα κληθούν να αποθηκεύσουν τα μεγαλύτερα από αυτά. Εάν τώρα, επιλέξουμε για shard key τον "δήμο" τότε θα έχουμε πολύ καλύτερη (μεγαλύτερη) πληθικότητα εκτός εάν τα δεδομένα μας αφορούν μια συγκεκριμένη γεωγραφική περιοχή όπως π.χ. τα νότια προάστια, οπότε και πάλι έχουμε πρόβλημα αφού τα δεδομένα μας θα χωριστούν σε 10-15 τεράστια chunks που θα είναι δύσκολο να διαχειρισθούν. Τέλος, εάν επιλέξουμε για shard key τον "αριθμό τηλεφώνου" τότε θα έχουμε την καλύτερη δυνατή πληθικότητα αφού το shard key θα έχει ξεχωριστή τιμή για κάθε έγγραφο και έτσι η MongoDB θα μπορεί να δημιουργήσει όσα chunks χρειαστούν.

- **Κλιμάκωση των λειτουργιών εγγραφής (Write scaling)**: Μερικά shard keys δίνουν τη δυνατότητα σε μία εφαρμογή να αξιοποιήσει τις αυξημένες δυνατότητες που έχει ένα cluster αναφορικά με την εγγραφή δεδομένων, ενώ άλλα όχι. Για παράδειγμα, ας πάρουμε μία συλλογή για την οποία επιλέγουμε ως shard key το πεδίο _id, που είναι τύπου objectId. Όπως έχουμε αναφέρει, η MongoDB δημιουργεί αυτόματα objectId τιμές για κάθε νέο έγγραφο, ώστε να προσδώσει σε αυτό ένα μοναδικό αναγνωριστικό. Όμως, η τιμή του objectId αναπαριστά ουσιαστικά μία σφραγίδα χρόνου (time stamp), το οποίο σημαίνει ότι αυξάνεται με ένα συνεχή και προβλεπόμενο ρυθμό. Παρ' όλο που η τιμή αυτή έχει μεγάλη πληθικότητα, όπως και οποιαδήποτε ημερομηνία ή άλλος αριθμός που αυξάνεται μονοτονικά, αν την επιλέξουμε για shard key, τότε όλες οι λειτουργίες εισαγωγής δεδομένων θα αποθηκεύουν τα δεδομένα σε ένα μοναδικό chunk και συνεπώς σε ένα και μοναδικό shard. Ως αποτέλεσμα, η ικανότητα εγγραφής δεδομένων στο σκληρό δίσκο που έχει ο συγκεκριμένος shard θα καθορίσει την αποδοτικότητα της εγγραφής δεδομένων ολόκληρου του cluster. Γενικά, είναι καλό να επιλέγουμε shard keys που έχουν και υψηλή πληθικότητα και διαμοιράζουν ισόποσα στο cluster όλες τις λειτουργίες εγγραφής που απαιτούνται. Καταληκτικά, για να έχουμε

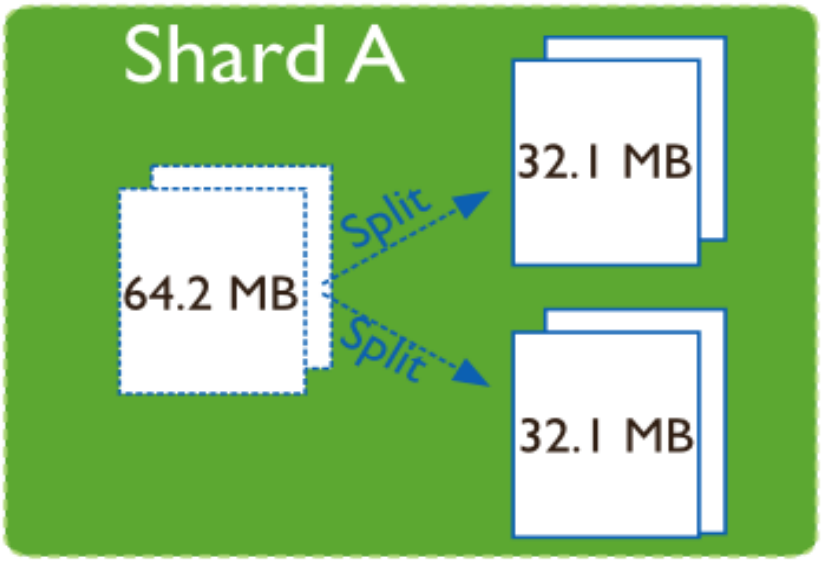
το βέλτιστο write scaling πρέπει να επιλέγουμε shard keys που παρουσιάζουν κάποιο βαθμό τυχαιότητας. Όπως, για παράδειγμα, τα hashed shard keys που έχουν τη δυνατότητα να διαχωρίζουν έγγραφα με παραπλήσιο shard key και να τα καταχωρούν σε διαφορετικά chunks και άρα και σε διαφορετικούς shards. Παρ' όλα αυτά, αυτό αντιτίθεται στη τρίτη ιδιότητα που χρειάζεται να έχει το shard key, δηλαδή την απομόνωση των ερωτημάτων.

- **Απομόνωση ερωτημάτων (Query isolation)**: Γενικότερα, τα γρηγορότερα ερωτήματα είναι αυτά τα οποία οι routers servers προωθούν σε ένα και μόνο shard. Για τα ερωτήματα, τα οποία δεν περιέχουν το shard key οι query servers είναι υποχρεωμένοι να επικοινωνήσουν με όλους τους shards και να περιμένουν την απάντηση τους πριν επιστρέψουν στην εφαρμογή το τελικό αποτέλεσμα. Γι' αυτό το λόγο, πρέπει να επιλέγουμε για shard key κάποιο πεδίο, το οποίο τις περισσότερες φορές να εμπεριέχεται στα ερωτήματα του πελάτη, ώστε οι query servers να μπορούν να το προωθούν σε ένα μόνο shard ή τουλάχιστον σε έναν περιορισμένο αριθμό από shards. Εάν αυτό το shard key έχει χαμηλή πληθικότητα τότε μπορούμε να του προσθέσουμε και ένα δεύτερο πεδίο και να το κάνουμε σύνθετο shard key, ώστε τα δεδομένα να είναι περισσότερο διαχωρίσιμα.

### 2.4.4 Διατήρηση ισορροπίας στον διαμοιρασμό των δεδομένων

Η προσθήκη νέων δεδομένων ή νέων servers (shards) στο cluster, προκαλεί ανισορροπία στην κατανομή των δεδομένων. Συγκεκριμένα, έχει σαν αποτέλεσμα ένας shard να περιέχει πολύ περισσότερα chunks από έναν άλλο, ή ένα chunk να έχει πολύ μεγαλύτερο μέγεθος σε σχέση με τα υπόλοιπα. Η MongoDB εξασφαλίζει την "ισορροπία" στο cluster μέσω δύο διαδικασιών που τρέχουν συνεχώς στο παρασκήνιο (background) της βάσης δεδομένων. Η μία είναι η διαδικασία του "σπασίματος" (splitting) και η άλλη είναι η διαδικασία ισορροπιστής (balancer).

Το **splitting,** είναι η διαδικασία που αποτρέπει τα chunks από το να αποκτήσουν πολύ μεγάλο μέγεθος. Όταν ένα chunk ξεπεράσει κάποιο προκαθορισμένο μέγεθος, τότε η MongoDB διαιρεί (split) αυτό το chunk στη μέση. Όπως γίνεται κατανοητό, οι λειτουργίες insert και update είναι αυτές που μπορούν να "πυροδοτήσουν" μια τέτοια διαίρεση. Η διαδικασία του splitting είναι πολύ αποδοτική, μιας και δεν μετακινεί τα δεδομένα αλλά ούτε επηρεάζει καθόλου τους shards. Το μόνο που χρειάζεται είναι μια μικρή αλλαγή στα μεταδεδομένα του cluster.



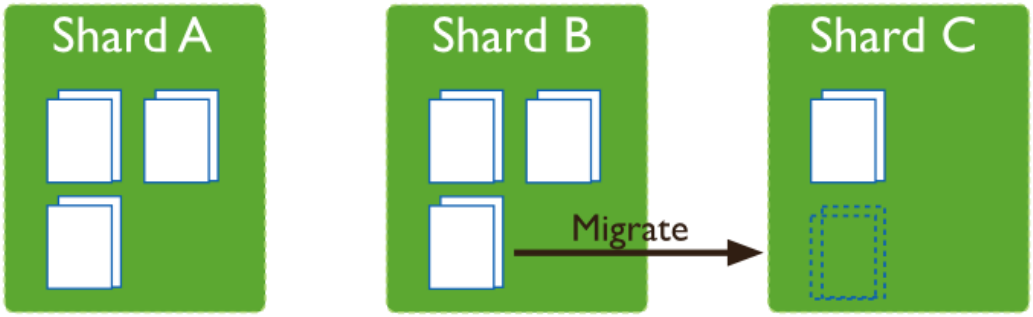

Εικόνα 21: Απεικόνιση του splitting.

Το προεπιλεγμένο μέγιστο μέγεθος ενός chunk είναι 64MB. Από εκεί και πέρα, ο διαχειριστής της MongoDB, αν επιθυμεί, μπορεί είτε να το αυξήσει είτε να το μειώσει. Όταν έχουμε μικρά chunks, τότε πετυχαίνουμε πιο ίση κατανομή των δεδομένων στο cluster με το κόστος όμως ότι θα γίνονται πιο συχνά μετακινήσεις των chunks από τον ένα shard στον άλλο. Αντίθετα, όταν έχουμε μεγάλα chunks, θα γίνονται λιγότερο συχνά splits και άρα η μετακίνηση των δεδομένων από shard σε shard γίνεται με μικρότερη συχνότητα. Βέβαια, στην τελευταία περίπτωση υπάρχει πιο άνιση κατανομή των δεδομένων στο cluster αφού, για παράδειγμα, δύο shards μπορεί να περιέχουν τον ίδιο αριθμό chunks, αλλά ο ένας να περιέχει πολύ μεγαλύτερο όγκο δεδομένων από τον άλλο.

Η άλλη διαδικασία που φροντίζει για την ισορροπία στην κατανομή των δεδομένων είναι ο **balancer**. Ο balancer είναι υπεύθυνος για τη μετακίνηση των chunks από έναν shard σε κάποιον άλλο. Συγκεκριμένα, όταν διαπιστωθεί 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 (αυτής της συλλογής), μέχρις ότου να υπάρχει ίση κατανομή των chunks στο cluster, για αυτή τη συγκεκριμένη συλλογή.

Για παράδειγμα, εάν μια συλλογή users έχει 100 chunks στον shard 1 και 50 στον shard 2, τότε ο balancer θα μετακινεί chunks (που αφορούν τη συλλογή users) από τον shard 1 στον shard 2, έως ότου επέλθει ισορροπία στην κατανομή των chunks της συλλογής users. Τη διαδικασία του balancing μπορεί να την εκκινήσει οποιοσδήποτε από τους query routers. Κατά τη διάρκεια της μεταφοράς των chunks, αποστέλλονται στο shard-λήπτη όλα τα έγγραφα από το chunk του shard-δότη. Στη συνέχεια, ο shard-λήπτης, λαμβάνει και εφαρμόζει στα έγγραφα αυτά όλες τις αλλαγές που

πραγματοποιήθηκαν κατά τη διάρκεια αυτής της μεταφοράς. Τέλος, ενημερώνονται τα metadata στους configuration servers που αφορούν την τοποθεσία του συγκεκριμένου chunk μέσα στο cluster. Εάν υπάρξει κάποιο σφάλμα κατά τη διάρκεια της παραπάνω διαδικασίας, τότε ο balancer ακυρώνει τη μεταφορά και αφήνει το chunk αμετάβλητο εκεί που βρισκόταν. Για αυτόν ακριβώς το λόγο, η MongoDB σβήνει το chunk από τον shard-δότη, μόνο όταν η μετακίνηση του έχει ολοκληρωθεί επιτυχώς.




Εικόνα 22: Μετακίνηση ενός chunk από τον shard B στον C.

Όπως προηγουμένως, μπορούσαμε να καθορίσουμε το μέγιστο μέγεθος ενός chunk πέρα από το οποίο θα γίνεται split, έτσι και στην περίπτωση του balancer, ο διαχειριστής της βάσης δεδομένων μπορεί να ρυθμίσει "τι" ορίζουμε ως ανισορροπία στην κατανομή των chunks μιας συλλογής. Ειδικότερα, μπορεί να ορίσει τη διαφορά που θα πρέπει να υπάρχει στον αριθμό των chunks (μιας συλλογής) ανάμεσα σε δυο shards ώστε να ξεκινήσει η διαδικασία του balancing. Φυσικά αυτή η διαφορά έχει άμεση σχέση και με το πλήθος των chunks που απαρτίζουν τη συλλογή.

Για παράδειγμα, αν μια συλλογή αποτελείται από 100 chunks και ορίσουμε εμείς ως όριο για να ξεκινήσει το balancing να υπάρχει διαφορά έστω και δύο chunks ανάμεσα σε δύο shards, τότε για κάθε δύο με τρία split που θα γίνονται θα πρέπει να ξεκινά ολόκληρη η διαδικασία του balancing η οποία είναι χρονοβόρα και κοστίζει σε υπολογιστικούς πόρους. Τα προεπιλεγμένα όρια που έχει η MongoDB για το balancing σε σχέση και με τον αριθμό των chunks της συλλογής είναι τα εξής:

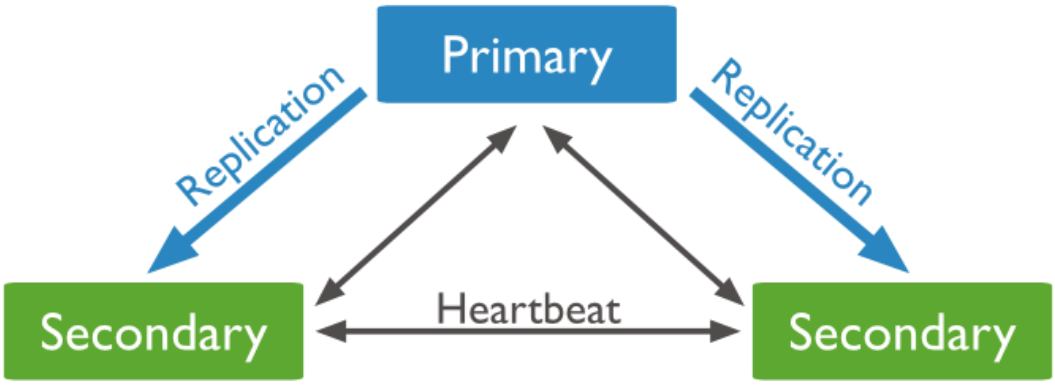
| Αριθμός Chunks συλλογής | Διαφορά στον αριθμό των chunks |
|---|---|
| Λιγότερα απο 20 | 2 |
| 20-79 | 4 |
| 80 και περισσότερα | 8 |

Όπως είπαμε, ανισορροπία στην κατανομή των δεδομένων υπάρχει είτε όταν δημιουργούμε το cluster, όπου ολόκληρη η συλλογή βρίσκεται σε έναν από τους shards, είτε όταν γίνουν κάποια splits. Εκτός από αυτές τις δυο περιπτώσεις, ανισορροπία στα δεδομένα του cluster μπορεί να δημιουργήσει και η προσθήκη ή η αφαίρεση ενός shard από το cluster. Συγκεκριμένα, όταν προσθέσουμε έναν νέο shard, τότε αυτός δεν περιέχει κανένα chunk, οπότε αυτόματα γίνεται shard-λήπτης και ξεκινάει ο balancer να του μεταφέρει chunks. Αντίθετα, για να αφαιρέσουμε ένα shard από το cluster πρέπει πρώτα να μεταφέρουμε όλα τα chunks που περιέχει στους άλλους shards, οπότε και πάλι ενεργοποιείται ο balancer.

## 2.5 Replication

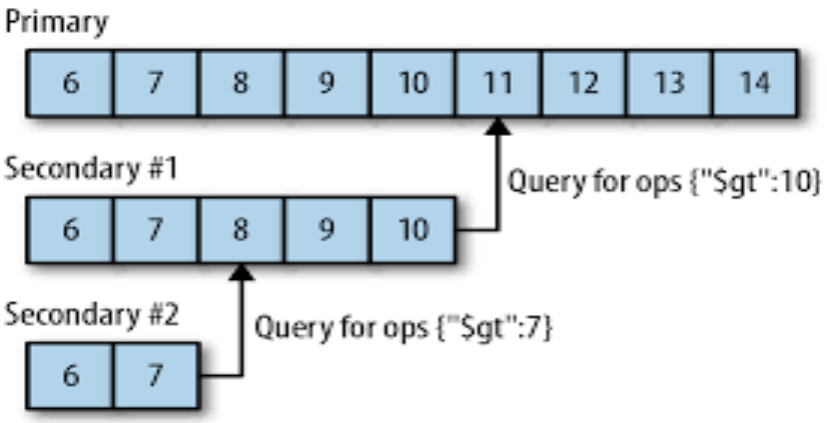
Ένα άλλο χρήσιμο χαρακτηριστικό της MongoDB είναι η αντικατάσταση (replication). Η αντικατάσταση στην ουσία είναι η διαδικασία συγχρονισμού των δεδομένων μέσω πολλαπλών εξυπηρετητών. Η διαδικασία αυτή προσφέρει επάρκεια και αυξάνει τη διαθεσιμότητα των δεδομένων. Η φύλαξη πολλαπλών αντιγράφων των δεδομένων σε μια σειρά από διαφορετικούς εξυπηρετητές, εγγυάται την αδιάκοπη λειτουργία της βάσης δεδομένων ακόμα και μετά την κατάρρευση ενός εξυπηρετητή λόγω αστοχίας υλικού του υπολογιστή. Επίσης, διευκολύνει την ανάκτηση τυχών χαμένων δεδομένων αλλά και τη διατήρηση αντιγράφων. Σε ορισμένες περιπτώσεις η αντικατάσταση μπορεί να χρησιμοποιηθεί, ώστε να αυξηθεί η αποδοτικότητα των λειτουργιών ανάγνωσης του cluster, αφού οι πελάτες (clients) αποκτούν την ικανότητα να στέλνουν διαδικασίες ανάγνωσης σε διάφορους εξυπηρετητές. Παράλληλα, η διατήρηση αντιγράφων σε διαφορετικά κέντρα δεδομένων (data centers) αυξάνει την τοπικότητα (locality) και τη διαθεσιμότητα των δεδομένων κυρίως σε κατανεμημένες εφαρμογές.

Όσον αφορά την MongoDB, η αντικατάσταση γίνεται μέσω των Replica Sets. 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.




Εικόνα 23: Το Replica set.

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 (oplog). Οι δευτερεύοντες κόμβοι εκτελούν τις ενέργειες του πρωτεύοντος με την ίδια χρονολογική σειρά. Η διαδικασία αυτή, πραγματοποιείται ασύγχρονα, δηλαδή με μικρή χρονική καθυστέρηση χωρίς όμως να επηρεάζει τη λειτουργία του κυρίαρχου 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. Η διαδικασία της ασύγχρονης επανάληψης των ενεργειών φαίνεται πιο παραστατικά στην εικόνα 24, όπου κάθε κόμβος συγχρονίζει τα δεδομένα του από κάποιον άλλο, ο οποίος έχει πιο πολλές εντολές αποθηκευμένες στο αρχείο καταγραφής ενεργειών. Επίσης, φαίνεται ότι κάποιος δευτερεύων κόμβος μπορεί να συγχρονίζει τα δεδομένα του από κάποιον άλλο δευτερεύοντα κατανέμοντας το φορτίο αυτής της διαδικασίας στους επιμέρους κόμβους και απαλλάσσοντας τον πρωτεύοντα από αυτό.

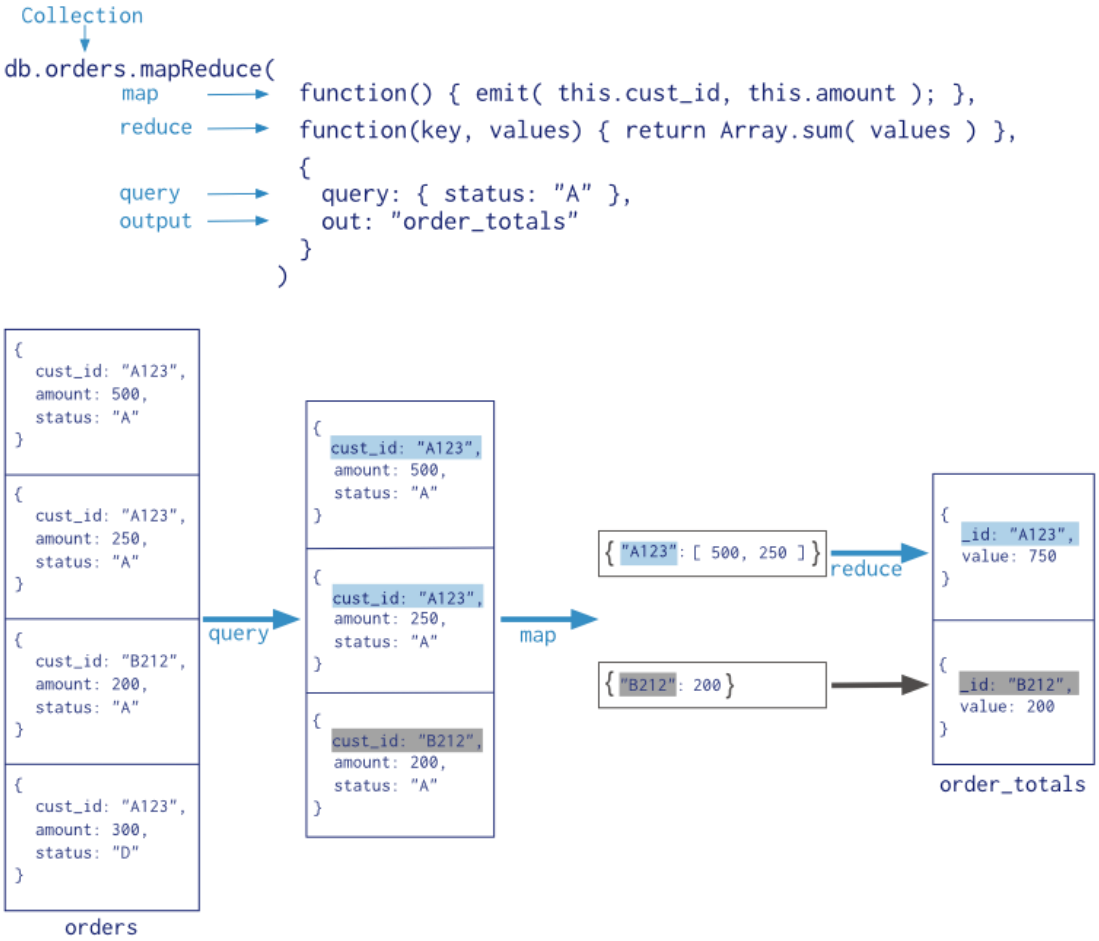



Εικόνα 24: Η διαδικασία συγχρονισμού τον κόμβων του Replica set.

Όλοι οι κόμβοι που περιλαμβάνονται σε ένα τέτοιο σύνολο, ανταγωνίζονται προκειμένου να γίνουν οι κυρίαρχοι κόμβοι του συνόλου. Σε περίπτωση που ο πρωτεύων κόμβος σταματήσει να είναι διαθέσιμος, τότε ξεκινά αυτόματα μία διαδικασία ψηφοφορίας ανάμεσα στους υπόλοιπους και εκλέγεται καινούριος κυρίαρχος κόμβος, ο οποίος αναλαμβάνει στη συνέχεια την εξυπηρέτηση των εντολών στα δεδομένα που έχει το Replica Set. Στη ψηφοφορία αυτή, προκειμένου κάποιος κόμβος να γίνει πρωτεύων, θα πρέπει να εξασφαλίσει ψήφους από τουλάχιστον τους μισούς κόμβους του συνόλου. Αυτό γίνεται για να αποκλείεται η πιθανότητα τα μέλη του συνόλου να χωριστούν σε δύο τμήματα λόγω κάποιας βλάβης δικτύου που ενώνει τις δύο τοποθεσίες, εκλέγοντας στη συνέχεια το κάθε τμήμα τον δικό του πρωτεύοντα κόμβο. 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.

## 2.6 Map-Reduce

Το Map-Reduce είναι η πιο διαδεδομένη μέθοδος επεξεργασίας των Big Data. Όπως αναφέραμε, η αποθήκευση των Big Data γίνεται με κατανεμημένο τρόπο, δηλαδή δεν αποθηκεύονται σε ένα μόνο υπολογιστικό σύστημα, αλλά σε πολλούς διαφορετικούς servers λόγω του μεγάλου όγκου τους. Αυτό ακριβώς το στοιχείο εκμεταλλεύεται το Map-Reduce, που επιτρέπει πρώτα την τοπική επεξεργασία των δεδομένων στον server που βρίσκονται και ύστερα τη λήψη συγκεντρωτικών αποτελεσμάτων για ολόκληρη τη βάση δεδομένων. Ειδικότερα, η μέθοδος Map-Reduce, αποτελείται από δυο επιμέρους στάδια, το map και το reduce. Το στάδιο map εφαρμόζεται στα έγγραφα της συλλογής τα οποία κάνανε “match” στο ερώτημα μας, και παράγει για κάθε ένα από αυτά ένας ζεύγος τιμών key:value. Έπειτα, το στάδιο reduce ομαδοποιεί τα ζεύγη τιμών ως προς το κλειδί (key) και παρουσιάζει τα συγκεντρωτικά αποτελέσματα. Στην παρακάτω εικόνα φαίνεται ένα παράδειγμα της εκτέλεσης της μεθόδου Map-Reduce στην MongoDB.




Εικόνα 25: Παράδειγμα Map-reduce στην MongoDB.

Αναλυτικότερα, η MongoDB βρίσκει ποια έγγραφα έχουν στο πεδίο status την τιμή “A” και εφαρμόζει σε αυτά τη συνάρτηση map. Η συνάρτηση map, για κάθε έγγραφο στο οποίο εφαρμόζεται παράγει ένα ζεύγος τιμών key:value, όπου για key θέτει το cust_id και για value την τιμή του πεδίου amount. Έπειτα η reduce ομαδοποιεί τις τούπλες αυτές ως προς το cust_id και επιστρέφει ως αποτέλεσμα το συνολικό ποσό για κάθε πελάτη. Τέλος, η MongoDB, χρησιμοποιεί Javascript συναρτήσεις για τη λειτουργία του map και του reduce, οι οποίες παρέχουν μεγάλη ευελιξία στον προγραμματιστή.

# Κεφάλαιο 3

# Υλοποίηση και αποτελέσματα διπλωματικής

## 3.1 Data set

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.

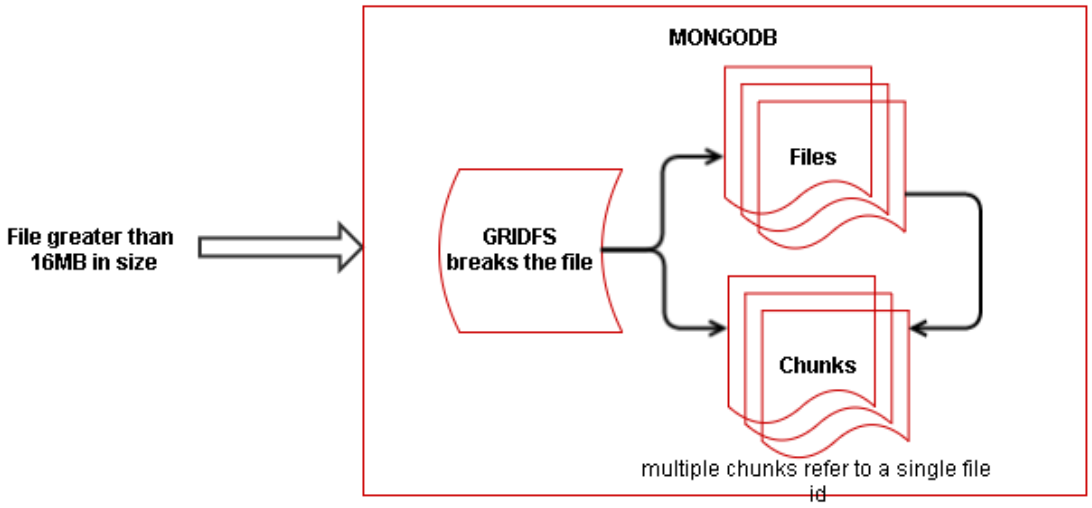
Στη διπλωματική αυτή μελετάται το πρόβλημα αποθήκευσης και αναζήτησης multimedia αρχείων και ειδικότερα 3D σκηνών. Αναλυτικότερα, τα δεδομένα που χρησιμοποιήσαμε στην παρούσα διπλωματική περιέχουν τα 3D (τρισδιάστατα) μοντέλα διαφόρων αεροσκαφών και ελικοπτέρων της πολεμικής αεροπορίας και είναι αποθηκευμένα με τη μορφή x3d αρχείων. Τα αρχεία αυτά είναι διαθέσιμα δωρεάν στο διαδίκτυο. Στην βιβλιογραφία υπάρχει ο σύνδεσμος του site στο οποίο βρίσκονται. 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.

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. Η χρησιμοποίηση κάποιας κλασικής SQL βάσης δεδομένων θα ήταν σχεδόν αδύνατο να αποθηκεύσει τέτοιου είδους δεδομένα, αλλά και να μπορούσε δεν θα είχε ικανοποιητικό αποτέλεσμα αναφορικά με την αναζήτηση και την επεξεργασία τους. Για αυτό το λόγο χρησιμοποιήσαμε την document database MongoDB.

## 3.2 Data model

Όπως γνωρίζουμε η MongoDB είναι ιδανική για την αποθήκευση αρχείων, τα οποία έχουν ανομοιογενή μεγέθη και δομές. Στη βάση αυτή, το ανώτερο επιτρεπτό μέγεθος ενός εγγράφου είναι τα 16Mb. Όμως, στα δεδομένα μας υπάρχουν σκηνές που έχουν μέγεθος πολύ μεγαλύτερο από αυτό το όριο, με τη μεγαλύτερη να είναι 183,4Mb. Για αυτές τις περιπτώσεις, η MongoDB έχει ένα μοντέλο δεδομένων (data model), το οποίο ονομάζεται GridFS.

Το **GridFS** παρέχει αφενός την απαραίτητη για εμάς λειτουργικότητα που είναι η αποθήκευση αρχείων μεγέθους μεγαλύτερου των 16Mb, αλλά επιπλέον έχει το χαρακτηριστικό πως χρησιμοποιεί δύο συλλογές για την αποθήκευση κάθε αρχείου. Η μία είναι για την αποθήκευση των chunks (chunks collection) που απαρτίζουν το αρχείο (3D σκηνή) και η άλλη (files collection) για την αποθήκευση των metadata για το κάθε αρχείο.




Εικόνα 26: Το μοντέλο δεδομένων GridFS.

Αναλυτικότερα, κάθε json document της **files collection** αντιπροσωπεύει ένα αρχείο, δηλαδή μία 3D σκηνή. Μέσα σε αυτό το json document, κάτω από το πεδίο metadata υπάρχει ένα εμφολευμένο έγγραφο (emeded document), το οποίο περιέχει την mp7 περιγραφή του αντίστοιχου αρχείου (σκηνής), η οποία μπορεί να αποθηκευτεί σε ένα έγγραφο, αφού έχει μέγεθος πολύ μικρότερο από 16Mb. Όπως γνωρίζουμε από τα προηγούμενα κεφάλαια, κάθε json document περιέχει ένα μοναδικό αναγνωριστικό, το πεδίο _id. Οπότε κάθε αρχείο (σκηνή) μπορούμε να το προσδιορίζουμε από αυτό το μοναδικό αναγνωριστικό.

Από την άλλη, όσων αφορά την **chunk collection**, η MongoDB καταφέρνει να αποθηκεύει αρχεία μεγαλύτερα των 16MB με το να τα σπάει σε πολλά κομμάτια (chunks) των 256KB. Ύστερα προσθέτει στο κάθε ένα από αυτά το πεδίο file_id, με τιμή, την τιμή του αναγνωριστικού ( _id ) του μεγάλου σε μέγεθος αρχείου, που απαρτίζουν τα chunks αυτά.

Με αυτόν τον τρόπο, επιτυγχάνουμε:

- Πρώτον, να γνωρίζουμε ανά πάσα στιγμή πόσα αλλά και ποια chunks συνθέτουν κάποιο αντικείμενο, απλά κοιτώντας ποια από αυτά έχουν κοινό file_id.
- Και δεύτερον, να τα συνδέουμε άμεσα με την mp7 περιγραφή της σκηνής που απαρτίζουν απλά κοιτώντας ποιο έγγραφο της files collection έχει για _id ίδια τιμή με το πεδίο file_id όλων των παραπάνω chunks.

Η παραπάνω λειτουργικότητα του GridFS θα αποδειχθεί πολύ σημαντική, καθώς το κύριο σενάριο πάνω στο οποίο θα χρησιμοποιηθούν τα δεδομένα μας είναι για context based retrieval.

Στην παρούσα διπλωματική θα αναλύσουμε την απόδοση της MongoDB στην αποθήκευση, ανάκτηση και επεξεργασία των δεδομένων αυτών και θα την βελτιστοποιήσουμε χρησιμοποιώντας κάποιες από τις πιο σύγχρονες τεχνικές που υπάρχουν στον τομέα των Big Data, όπως είναι το sharding, το indexing και το replication. Τέλος, θα ερευνήσουμε την αποδοτικότητα της μεθόδου Map-Reduce υλοποιώντας με αυτή ένα ερώτημα, το οποίο θα βρίσκει το ιστόγραμμα χρώματος των σκηνών.

## 3.3 Στήσιμο βάσης δεδομένων

Στα πλαίσια αυτής της διπλωματικής, μας δόθηκαν τρεις servers, ο κάθε ένας από τους οποίους είχε εγκατεστημένο το λειτουργικό σύστημα Linux 64-bit και συγκεκριμένα την έκδοση Linux Mint 17.2 Rafaela (ubuntu edition). Σε κάθε ένα από αυτούς τους servers ακολουθήσαμε τα επόμενα βήματα, ώστε να εγκαταστήσουμε τις εφαρμογές που θα χρειαστούν για αυτή τη διπλωματική.

1. Εγκατάσταση της βάσης δεδομένων MongoDB. Κατεβάζουμε τα απαραίτητα αρχεία από το site της MongoDB το οποίο υπάρχει στην βιβλιογραφία και εγκαθιστούμε την MongoDB version 2.6.10. Μετά την εγκατάσταση, εκκινούμε την MongoDB, απλά τρέχοντας την εφαρμογή mongod.exe. Η mongod είναι η κύρια διαδικασία (primary daemon process) της MongoDB, η

οποία εκτελείται συνεχώς στο παρασκήνιο του server και είναι αυτή που υποστηρίζει τη λειτουργία της βάσης δεδομένων. Για να τρέξουμε την εφαρμογή mongod.exe, πληκτρολογούμε στο terminal την ακόλουθη εντολή:

```
sudo service monogdb start
```

Με αυτόν τον τρόπο εκκινούμε ένα instance της MongoDB στον localhost:27017, όπου localhost είναι η ip του server στον οποίο βρισκόμαστε και ο αριθμός 27017 είναι η port στην οποία πρέπει να συνδεθούμε για να επικοινωνήσουμε με τη βάση δεδομένων. Τέλος, μπορούμε να συνδεθούμε στο MongoDB Shell απλά πληκτρολογώντας την εντολή 'mongo' στο terminal. Επίσης, καλό είναι να γνωρίζουμε που βρίσκεται τι στον sever μας. Αναλυτικότερα, τα αρχεία της βάσης δεδομένων αποθηκεύονται στην τοποθεσία /var/lib/mongodb, τα εκτελέσιμα αρχεία (binaries) βρίσκονται στο /usr/bin και το configuration file της MongoDB βρίσκεται στο /etc με όνομα mongodb.config.

2. Εγκατάσταση του Robomongo. Το πρόγραμμα Robomongo είναι ένα gui tool που επιτελεί τις ίδιες ακριβώς λειτουργίες με το mongo shell μόνο που είναι πιο εύκολο στη χρήση και μας δίνει μία καλύτερη εποπτεία της βάσης δεδομένων.

3. Εγκατάσταση του openJDK (version IcedTea 2.5.6) και συγκεκριμένα της java version 1.7.0_79.

4. Εγκατάσταση του Apache Maven 3.0.5. Το Maven είναι ένα build automation tool που χρησιμοποιείται κυρίως σε java projects.

5. Τέλος, για Java IDE εγκαταστήσαμε και χρησιμοποιήσαμε το Intellij Idea 14.1.

Έπειτα από όλη αυτή τη διαδικασία, έπρεπε να αντιγράψουμε τη βάση δεδομένων που θα χρησιμοποιήσουμε σε έναν από τους τρεις διαθέσιμους servers μας. Η βάση δεδομένων, όπως αναφέραμε, υπάρχει έτοιμη στη μορφή GridFS από το project i-promotion. Γνωρίζοντας τη διεύθυνση ip του server (από το project i-promotion) και την port στην οποία τρέχει η MongoDB instance που περιέχει τη βάση δεδομένων μας, τρέξαμε την ακόλουθη εντολή στο shell ενός από τους δικούς μας servers:

```
db.copyDatabase(<from_db>, <to_db>, <from_host_ip:port>)
```

Όπου έχει <from_db> γράψαμε το όνομα που έχει η βάση δεδομένων που θέλαμε να αντιγράψουμε, στο <to_db> γράψαμε το όνομα που θέλουμε να έχει η βάση δεδομένων όταν αντιγραφεί στον server μας, και τέλος, στο <from_host_ip:port > συμπληρώσαμε την ip και την port του server από τον οποίο θα πάρουμε τη βάση δεδομένων.

# 3.4 Ευρετηρία

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, τα ευρετήρια είναι ειδικές δομές δεδομένων που αποθηκεύουν ένα μικρό μέρος των δεδομένων μιας συλλογής με τέτοιο τρόπο, ώστε να είναι πολύ εύκολο και γρήγορο να διασχιστούν. Όσον αφορά τη βάση δεδομένων μας, έχουμε πει ότι αυτή αποτελείται από δύο συλλογές, την file collection και την chunk collection. Τα έγγραφα της chunk collection, είναι όπως το παρακάτω:

```
{
   "_id" : ObjectId("545a8c531d8e0bdcdffd705f"),
   "files_id" : ObjectId("545a8c531d8e0bdcdffd705e"),
   "n" : 0,
   "data" : { "$binary" : "Μεγάλο_string_απο_διάφορους_χαρακτήρες”}
}
```

Επομένως, δεν υπάρχει κάποιο πεδίο, στο οποίο θα χρησίμευε να φτιάξουμε ευρετήριο, αφού κανένα από αυτά τα πεδία δεν πρόκειται να υπάρχει στα ερωτήματα που θα θέτει ο χρήστης στη βάση. Αντιθέτως, η file collection που αποθηκεύει τα metadata της κάθε σκηνής περιέχει χρήσιμα πεδία, όπως, για παράδειγμα, το χρώμα των αντικειμένων από τα οποία αποτελείται μια σκηνή.

Αναλυτικότερα, το κάθε έγγραφο (σκηνή) περιέχει πολλά αντικείμενα, κάθε ένα από τα οποία έχει το δικό του όγκο και χρώμα, οπότε, θα χρειαστεί να δημιουργήσουμε Multikey ευρετήρια. Όπως αναφέραμε στο προηγούμενο κεφάλαιο, τα ευρετήρια αυτά έχουν τη δυνατότητα για ένα έγγραφο να μπορούν να πραγματοποιήσουν πολλές καταχωρήσεις. Στη δική μας περίπτωση, σε κάθε έγγραφο (σκηνή), θα κάνουν μια καταχώρηση (στο ευρετήριο) για κάθε ένα αντικείμενο που περιέχει αυτή η σκηνή. Η καταχώρηση αυτή θα αφορά τα πεδία color και volume του συγκεκριμένου αντικειμένου. Το πρόβλημα εδώ είναι ότι η δομή των metadata που προέκυψε από το πρότυπο mpeg-7 δεν βολεύει για να δημιουργήσουμε multikey ευρετήρια που αφορούν το χρώμα και τον όγκο. Αυτό συμβαίνει γιατί το πεδίο color του **i** αντικειμένου βρίσκεται στο μονοπάτι:

metadata.Mpeg7.Description.1.MultimediaContent.StructuredCollection.Collection.1.DescriptorCollection.**i**.Descriptor.1.Geometry3D.DominantColor3D.Value.color

ενώ ο όγκος του ίδιου αντικειμένου **i** βρίσκεται στο μονοπάτι:

metadata.Mpeg7.Description.1.MultimediaContent.StructuredCollection.Collection.1.DescriptorCollection.**i**.Descriptor.0.BoundingBox3DSize.volume

Δηλαδή ο DescriptionCollection είναι ο πίνακας που περιέχει για κάθε ένα αντικείμενο της σκηνής ένα εμφολευμένο έγγραφο, το οποίο περιέχει τα πεδία color και volume. Όμως, από το προηγούμενο κεφάλαιο**,** γνωρίζουμε ότι μπορούμε να δημιουργήσουμε multikey ευρετήριο μόνο όταν το πεδίο πάνω στο οποίο θέλουμε να φτιάξουμε το ευρετήριο έχει για τιμή έναν πίνακα ο οποίος περιέχει άμεσα τις τιμές, οι οποίες πρόκειται να καταχωρηθούν. Με τη λέξη άμεσα εννοούμε ότι ο πίνακας αυτός θα πρέπει να περιέχει απευθείας τα πεδία με τις τιμές του όγκου και του χρώματος των αντικειμένων της σκηνής. Στη δική μας περίπτωση, ενώ υπάρχει ο πίνακας που περιέχει το χρώμα και τον όγκο των αντικειμένων της σκηνής, δεν περιέχει μέσα του απευθείας τις τιμές αυτές, αλλά έχει το εμφολευμένο έγγραφο Descriptor, το οποίο πρέπει να διασχίσουμε κάθε φορά.

Για να επιλύσουμε αυτό το πρόβλημα φτιάξαμε ένα νέο πίνακα NewDescriptors, ο οποίος περιέχει για τιμές τούπλες (tuples) της μορφής { color: 357 , volume: 0.63 } και τον τοποθετήσαμε κάτω από το πεδίο metadata. Επομένως, αν για παράδειγμα κάποιος θέλει να βρει τη τιμή του χρώματος του 7$^{ου}$ αντικειμένου της σκηνής, χρειάζεται απλά να διασχίσει τη διαδρομή:

metadata.NewDescriptors.7.color

Αναλυτικότερα το color λαμβάνει τιμές από το 1 έως το 512. Η τιμή 1 αφορά τα χρώματα που η RGB τιμή τους βρίσκεται στο διάστημα R:{0,32} , G:{0,32} και B {0,32}, ενώ η τιμή 2 τα χρώματα με R:{32,64} , G:{0,32} και B {0,32} κ.ο.κ μέχρι το 512ο : R:[224, 256), G:[224, 256), B:[224, 256). Με αυτόν τον τρόπο χωρίζουμε όλο το χρωματικό φάσμα σε 512 διαστήματα γνωρίζοντας με καλή ακρίβεια, ποια απόχρωση αντιπροσωπεύει το κάθε διάστημα. Τέλος, ο όγκος λαμβάνει τιμές από 0.0 έως 1.0 και αντιπροσωπεύει το ποσοστό του όγκου της συνολικής σκηνής, που καταλαμβάνει το συγκεκριμένο αντικείμενο.

Για τη δημιουργία και την τοποθέτηση του πίνακα NewDescriptors σε αυτό το σημείο γράψαμε το πρόγραμμα **metatroph.java** το οποίο βρίσκεται στο παράτημα. Συνοπτικά, αυτό που κάναμε ήταν αρχικά να διασχίσουμε με τον json parser το κάθε έγγραφο της συλλογής, ώστε να πάρουμε τις τιμές color και volume των αντικειμένων του. Έπειτα, για κάθε ένα έγγραφο αποθηκεύσαμε τις τιμές αυτές στον πίνακα NewDescriptors που έχει τύπο BasicDBList. Τέλος, εφαρμόζουμε για κάθε έγγραφο μία εντολή update με την οποία αποθηκεύουμε τον πίνακα αυτό στο σημείο metadata.NewDescriptors.

Ο χρόνος εκτέλεσης αυτού του προγράμματος που φτιάχνει νέους descriptors κατάλληλους για τη δημιουργία multikey index είναι περίπου 5.5 δευτερόλεπτα. Ο χρόνος αυτός δεν λαμβάνεται υπόψιν στις μετρήσεις της αποδοτικότητας των διάφορων ευρετηρίων που θα παρουσιαστούν στη συνέχεια, αφού είναι σταθερός και το πρόγραμμα αυτό αρκεί να εκτελεστεί μονάχα μία φορά από τον administrator της βάσης δεδομένων, ώστε ύστερα να μπορούν να δημιουργηθούν όλα τα απαραίτητα ευρετήρια που αφορούν το χρώμα και τον όγκο.

Τέλος, εκτός από multikey, τα ευρετήρια που θα δημιουργήσουμε χρειάζεται να έχουν την ιδιότητα sparse, να είναι δηλαδή sparsed indexes (αραιά ευρετήρια).

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, η ιδιότητα αυτή μας δίνει τη δυνατότητα να αποκλείσουμε όσα έγγραφα δεν περιέχουν το πεδίο στο οποίο πρόκειται να δημιουργήσουμε ευρετήριο. Στη δική μας περίπτωση, αυτό είναι πολύ χρήσιμο, γιατί η files collection περιέχει αρκετά έγγραφα που δεν περιέχουν κάποια σκηνή, αλλά περιέχουν άλλου είδους metadata, τα οποία δημιουργήθηκαν από το πρότυπο mpeg-7. Επομένως, κατά τη δημιουργία του ευρετηρίου η MongoDB θα προσπερνά τα έγγραφα, τα οποία δεν έχουν τα πεδία color και volume συνεχίζοντας κανονικά την υπόλοιπη διαδικασία.

Στη συνέχεια θα παρουσιάσουμε τα αποτελέσματα σχετικά με την αποδοτικότητα των διαφόρων ειδών index για δύο περιπτώσεις. Πρώτον, στην περίπτωση που το ερώτημα του χρήστη εμπεριέχει ένα πεδίο (μόνο το color) και δεύτερον στην περίπτωση που περιλαμβάνει δύο πεδία (το color και το volume).

## 3.4.1 Απόδοση ευρετηρίων για ερώτημα που περιέχει ένα πεδίο.

Το ερώτημα που επιλέξαμε να τρέξουμε στη βάση είναι το εξής:

```
db.Savage.files.find( { "metadata.NewDescriptors.color":  { $gt : 500 } }
).explain("executionStats");
```

Η Savage.files είναι η συλλογή που περιέχει τα metadata και στην οποία θέλουμε να εκτελέσουμε το ερώτημα. Περιγραφικά, το ερώτημα είναι το εξής:

“Βρες ποια από τα έγγραφα της συλλογής Savage.files περιέχουν τουλάχιστον ένα αντικείμενο, το οποίο έχει χρώμα πάνω από 500 και επέστρεψέ τα.”

Η μέθοδος explain("executionStats") που έχουμε χρησιμοποιήσει μας δείχνει πρώτον πόσα έγγραφα χρειάστηκε να διασχίσει (‘σκανάρει’) η MongoDB, ώστε να απαντήσει στο ερώτημα μας, και δεύτερον πόσα έγγραφα από τη συλλογή μας, όντως ταίριαξαν ( 'matched' ) με τον περιορισμό του χρήστη και επιστράφηκαν ως αποτέλεσμα. Στη συγκεκριμένη περίπτωση όπου το ερώτημα του χρήστη περιέχει μόνο το πεδίο color, κάναμε μετρήσεις για τρεις περιπτώσεις:

1. **Χωρίς να φτιάξουμε κανένα ευρετήριο**. Στη συγκεκριμένη περίπτωση η MongoDB χρειάστηκε να σκανάρει όλα τα έγγραφα της συλλογής (δηλαδή 1326), ώστε να επιστρέψει τελικά 192 έγγραφα που ταίριαξαν με το ερώτημά μας.

2. **Με μονού πεδίου ευρετήριο στο πεδίο color**. Συγκεκριμένα δημιουργήσαμε το ευρετήριο με την εξής εντολή:

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

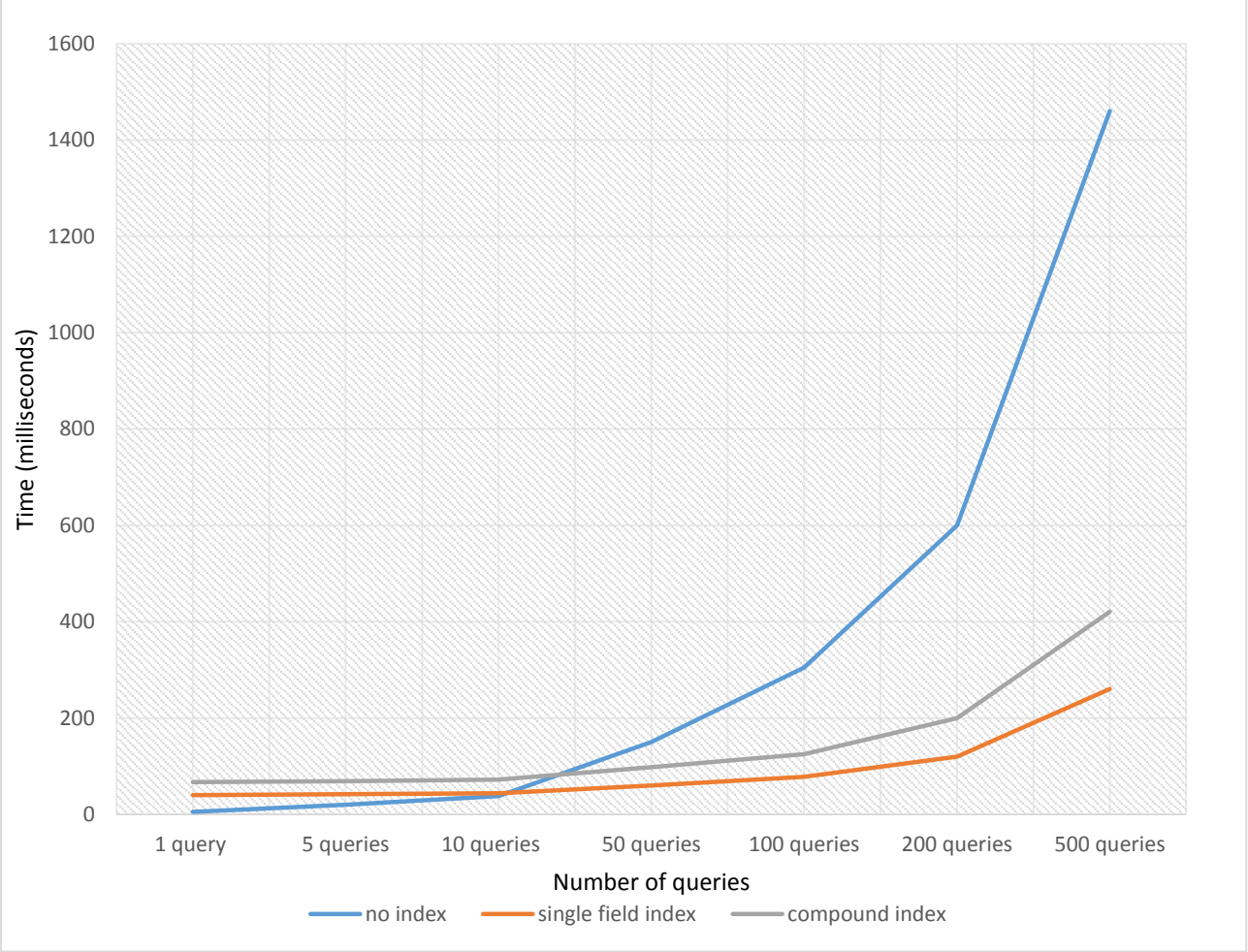
   Σε αυτή την περίπτωση η MongoDB χρειάστηκε να σκανάρει 195 έγγραφα από τη συλλογή, ώστε να επιστρέψει τελικά τα 192 έγγραφα που ταίριαξαν με το ερώτημα μας.

3. **Με σύνθετο ευρετήριο στα πεδία color και volume**, που δημιουργήσαμε ως εξής:

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

   Σε αυτήν την περίπτωση, όπου έχουμε σύνθετο ευρετήριο αλλά το ερώτημά μας, αφορά μόνο ένα πεδίο θα δούμε την αποδοτικότητα που έχουν τα προθέματα (prefixes). Συγκεκριμένα, η MongoDB χρειάστηκε να σκανάρει 422 έγγραφα της συλλογής, ώστε να επιστρέψει τελικά τα 192 έγγραφα που ταίριαξαν με το ερώτημά μας.

Στο παρακάτω γράφημα, φαίνονται οι χρόνοι σε millisecond που έκανε η εντολή για να εκτελεστεί στην κάθε μία περίπτωση. Συγκεκριμένα φτιάξαμε ένα javascript script το οποίο για την κάθε μία από τις τρεις περιπτώσεις τρέχει το παραπάνω ερώτημα 1, 5, 10, 50, 100, 200 και 500 φορές. Στους χρόνους αυτούς συμπεριλαμβάνεται και η δημιουργία του αντίστοιχου index.




Εικόνα 27: Απόδοση ευρετηρίων όταν το ερώτημα περιέχει ένα πεδίο.

Αρχικά, παρατηρούμε ότι όταν ο αριθμός των ερωτημάτων είναι μικρός (από 1 έως 10) η MongoDB επιστρέφει πιο γρήγορα τα αποτελέσματα στην περίπτωση που δεν έχουμε φτιάξει κάποιο ευρετήριο. Αυτό συμβαίνει διότι στις άλλες δύο περιπτώσεις η δημιουργία ευρετηρίου καθυστερεί για λίγα millisecond την έναρξη των ερωτημάτων. Από τα 50 ερωτήματα και πάνω (όπου ο χρόνος δημιουργίας του ευρετηρίου είναι μικρός σε σχέση με το χρόνο που χρειάζεται για να απαντηθούν τα ερωτήματα) φαίνεται ξεκάθαρα πόσο αποδοτική είναι η χρήση ευρετηρίων.

Τα ερωτήματα των χρηστών εκτελούνται πολύ πιο γρήγορα όταν υπάρχουν ευρετήρια, και ειδικά στην περίπτωση όπου δημιουργούμε ευρετήριο στο πεδίο color, υπάρχει 5 φορές επιτάχυνση. Αυτό ήταν αναμενόμενο, καθώς, όπως αναφέραμε, στην μια περίπτωση η βάση χρειάστηκε να σκανάρει όλα τα έγγραφα της συλλογής, ενώ στην άλλη συμβουλεύτηκε το ευρετήριο στο color και χρειάστηκε να σκανάρει μόλις 195 έγγραφα, τα 192 από τα οποία ταίριαζαν με τον περιορισμό ‘color μεγαλύτερο από 500’.

Τέλος, παρατηρούμε ότι στην περίπτωση που έχω compound ευρετήριο, μπορεί να μην έχουμε την ίδια επιτάχυνση με το ευρετήριο που περιέχει αποκλειστικά το color,

αλλά σε σχέση με την περίπτωση που δεν υπάρχει κανένα ευρετήριο είναι πολύ πιο γρήγορο. Έτσι, ο μηχανισμός της MongoDB που επιτρέπει τη χρήση του προθέματος ενός σύνθετου ευρετηρίου για την απάντηση σχετικών ερωτημάτων αποδεικνύεται πολύ χρήσιμος και ίσως σωτήριος για περιπτώσεις στις οποίες δεν έχουμε κάποιο ευρετήριο που να περιέχει ακριβώς τα πεδία που περιλαμβάνονται στο ερώτημα του χρήστη.

## 3.4.2 Απόδοση ευρετηρίων για ερώτημα που περιέχει δυο πεδία.

Το ερώτημα που επιλέξαμε να τρέξουμε στη βάση είναι το:

```
db.Savage.files.find( {  "metadata.NewDescriptors": {  $elemMatch:  {  "color": { $gt : 500}, "volume": { $gt:0.2 }  } } } ).explain("executionStats");
```

Περιγραφικά το ερώτημα είναι το εξής:

“Βρες ποια από τα έγγραφα της συλλογής Savage.files περιέχουν τουλάχιστον ένα αντικείμενο, το οποίο έχει χρώμα πάνω από 500 και κατέχει περισσότερο από το 20% της σκηνής.”

Χρειάστηκε να χρησιμοποιήσουμε τον δείκτη $elemMatch, ώστε η MongoDB να ψάχνει για αντικείμενα που ικανοποιούν ταυτόχρονα και τους δύο περιορισμούς. Σε διαφορετική περίπτωση θα μας επέστρεφε ως αποτέλεσμα και έγγραφα, τα οποία έχουν κάποιο αντικείμενο με color>500 και κάποιο άλλο με volume>0.2.

Η μέθοδος explain("executionStats") που έχουμε χρησιμοποιήσει μας δείχνει πρώτον πόσα έγγραφα χρειάστηκε να διασχίσει (“σκανάρει”) η MongoDB, ώστε να απαντήσει στο ερώτημα μας, και δεύτερον πόσα έγγραφα από τη συλλογή μας όντως ταίριαξαν ('matched') με τον περιορισμό του χρήστη και επιστράφηκαν ως αποτέλεσμα. Στη συγκεκριμένη περίπτωση όπου το ερώτημα του χρήστη περιέχει το πεδίο color αλλά και το πεδίο volume, πήραμε μετρήσεις για τέσσερις περιπτώσεις:

1. **Χωρίς να φτιάξουμε κανένα ευρετήριο**. Στη συγκεκριμένη περίπτωση η MongoDB χρειάστηκε να σκανάρει όλα τα έγγραφα της συλλογής (δηλαδή 1326), ώστε να επιστρέψει τελικά 62 έγγραφα που ταίριαξαν με το ερώτημά μας.

2. **Με μονού πεδίου ευρετήριο στο πεδίο color**. Συγκεκριμένα δημιουργήσαμε το ευρετήριο με την εξής εντολή:

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

   Σε αυτή την περίπτωση η MongoDB χρειάστηκε να σκανάρει 195 έγγραφα από τη συλλογή, ώστε να επιστρέψει τελικά τα 62 έγγραφα που ταίριαξαν με το ερώτημά μας.

3. **Με σύνθετο ευρετήριο στα πεδία color και volume**, που δημιουργήσαμε ως εξής:

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

   Η MongoDB χρειάστηκε να σκανάρει 74 έγγραφα της συλλογής, ώστε να επιστρέψει τελικά τα 62 έγγραφα που ταίριαξαν με το ερώτημά μας.

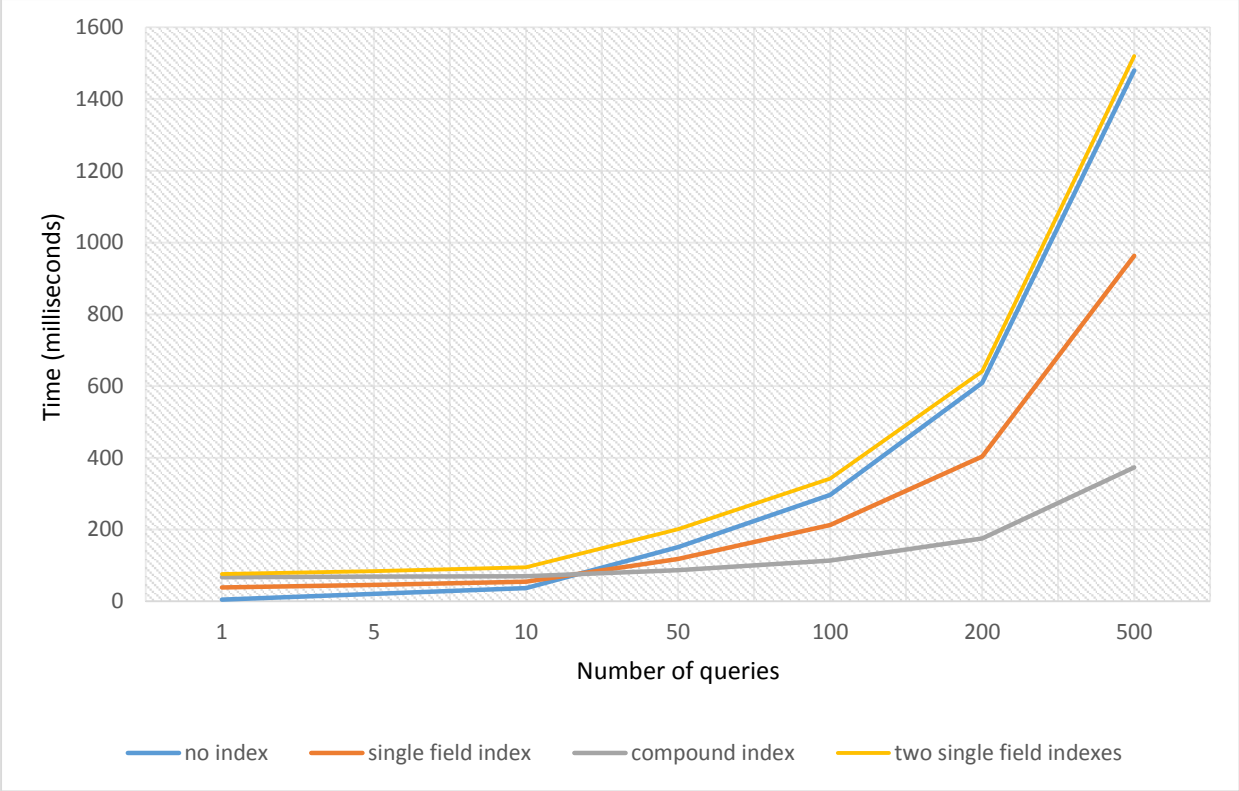
4. **Με δύο ευρετήρια μονού πεδίου, ένα στο πεδίο color και ένα στο πεδίο volume**. Δημιουργήσαμε τα ευρετήρια με τις παρακάτω δύο εντολές:

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

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

   Σε αυτήν την περίπτωση όπου το ερώτημά μας αφορά δύο πεδία για το κάθε ένα από τα οποία υπάρχει ένα ευρετήριο μονού πεδίου θα δούμε την αποδοτικότητα που έχει η διασταύρωση ευρετηρίων (index intersection). Τώρα, η MongoDB χρειάστηκε να σκανάρει 731 έγγραφα της συλλογής, ώστε να επιστρέψει τελικά τα 62 έγγραφα που ταίριαξαν με το ερώτημά μας.

Στο παρακάτω γράφημα, φαίνονται οι χρόνοι σε millisecond που έκανε η εντολή για να εκτελεστεί στην κάθε μία περίπτωση. Συγκεκριμένα φτιάξαμε ένα javascript script το οποίο για την κάθε μία από τις τέσσερις περιπτώσεις τρέχει το παραπάνω ερώτημα 1, 5, 10, 50, 100, 200 και 500 φορές. Στους χρόνους αυτούς συμπεριλαμβάνεται και η δημιουργία του αντίστοιχου index.




Εικόνα 28: Απόδοση ευρετηρίων όταν το ερώτημα περιέχει δυο πεδία.

Όπως και στο προηγούμενο ερώτημα, έτσι και εδώ παρατηρούμε ότι όταν ο αριθμός των ερωτημάτων είναι μικρός η MongoDB επιστρέφει πιο γρήγορα τα αποτελέσματα στην περίπτωση που δεν έχουμε φτιάξει κάποιο ευρετήριο. Ειδικά, στην περίπτωση που δημιουργούμε τα δύο ευρετήρια υπάρχει αρκετά μεγάλη καθυστέρηση σε σχέση με το χρόνο που απαιτείται για να εκτελεστούν περίπου 50 ερωτήματα. Τα ερωτήματα των χρηστών εκτελούνται πολύ πιο γρήγορα όταν υπάρχουν ευρετήρια, και ειδικά στην περίπτωση όπου δημιουργήσαμε το σύνθετο ευρετήριο που είναι ιδανικό γι' αυτό το ερώτημα, υπάρχει σχεδόν 4 φορές επιτάχυνση. Αυτό ήταν αναμενόμενο καθώς όπως αναφέραμε, στη μία περίπτωση η βάση χρειάστηκε να σκανάρει όλα τα έγγραφα της συλλογής, ενώ στην άλλη συμβουλεύτηκε το σύνθετο ευρετήριο και χρειάστηκε να σκανάρει μόλις 74 έγγραφα, τα 64 από τα οποία επέστρεψε ως αποτέλεσμα.

Από εκεί και πέρα, όπως αναφέραμε, τα ερωτήματα των χρηστών προς τη βάση παρουσιάζουν μεγάλη ποικιλία και πολλές φορές είναι δύσκολο να υπάρχει κάποιο ευρετήριο που θα ταιριάζει ακριβώς στα πεδία του ερωτήματος. Όπως παρατηρούμε, το ευρετήριο στο πεδίο color, είναι πολύ χρήσιμο και σε αυτήν την περίπτωση καθώς μειώνει το χρόνο εκτέλεσης των ερωτημάτων στο μισό.

Αντίθετα, όταν δημιουργήσαμε ευρετήριο και στο πεδίο volume, τότε η αυτοματοποιημένη διαδικασία της διασταύρωσης ευρετηρίων φαίνεται να μπέρδεψε κάπως τα πράγματα. Αυτό συμβαίνει διότι τα πεδία color και volume δεν είναι ανεξάρτητα μεταξύ τους, αλλά τα παίρνουμε ως ζεύγη τιμών που

αντιπροσωπεύουν ένα συγκεκριμένο αντικείμενο. Έτσι, όταν η MongoDB βρει ποια έγγραφα τηρούν τον κάθε περιορισμό δεν θα τα επιστρέψει κατευθείαν (όπως θα έκανε αν δεν βάζαμε στο ερώτημά μας τον δείκτη &elemMatch), αλλά θα χρειαστεί να διατρέξει ξανά τα ίδια έγγραφα άλλη μία φορά, ώστε να διαπιστώσει ποια από αυτά τηρούν ταυτόχρονα τους δύο περιορισμούς μας. Αναλυτικότερα, η MongoDB, αρχικά, συμβουλεύεται το ευρετήριο στο color και σκανάρει 195 έγγραφα από τα οποία τα 192 ικανοποιούν τον περιορισμό 'color μεγαλύτερο από 500'. Έπειτα συμβουλεύεται το ευρετήριο στο πεδίο volume και αφού σκανάρει επιπλέον 689 έγγραφα, βρίσκει ότι 529 από αυτά ταιριάζουν στον περιορισμό 'volume μεγαλύτερο από 0,2'.

Έπειτα θα χρειαστεί, όπως είπαμε, να 'διασταυρώσει΄ τα 192 έγγραφα που ικανοποιούν τον πρώτο περιορισμό με τα 529 έγγραφα που ικανοποιούν τον δεύτερο περιορισμό, ώστε να καταλήξει στα τελικά 62 έγγραφα που ικανοποιούν το ερώτημά μας. Επομένως, η MongoDB θα διατρέξει 195 + 689 = 884 έγγραφα και έπειτα για να γίνει η διασταύρωση θα πρέπει να διατρέξει ξανά 192 + 529= 721 έγγραφα από αυτά, δηλαδή συνολικά 1605 έγγραφα. Η μέθοδος explain("executionStats") αναφέρει ότι στο ερώτημα αυτό η MongoDB σκανάρει 884 έγγραφα, εννοώντας ότι 884 είναι τα διαφορετικά έγγραφα τα οποία σκάναρε. Όταν δεν έχουμε καθόλου ευρετήριο η MongoDB σκανάρει όλα τα έγγραφα της συλλογής δηλαδή 1326, ενώ στην περίπτωση με τα δύο ευρετήρια είναι σαν να σκανάρει 1605 έγγραφα. Έτσι εξηγείται και το γεγονός πως αυτή η περίπτωση 'φαίνεται' λίγο πιο αργή από όταν δεν έχουμε καθόλου ευρετήριο. Παρ' όλα αυτά, ο query optimizer επιλέγει να χρησιμοποιήσει τη διασταύρωση ευρετηρίων από το να μην χρησιμοποιήσει καθόλου ευρετήριο. Αυτό το κάνει διότι περιμένει ότι θα υπάρξουν πολλές χιλιάδες ή ακόμη και εκατομμύρια ερωτήματα, όπου σε αυτή την περίπτωση έστω και με μικρή διαφορά η διασταύρωση ευρετηρίων θα έχει καλύτερη απόδοση

Τέλος, όπως αναφέραμε και στη θεωρία, παρ' όλο που η διασταύρωση ευρετηρίων μπορεί να χρησιμοποιηθεί για να απαντήσει μεγαλύτερο αριθμό ερωτημάτων, τα σύνθετα ευρετήρια θα είναι τις περισσότερες φορές πιο αποδοτικά. Στην περίπτωσή μας αυτό οφείλεται στο γεγονός ότι ταξινομεί τα αντικείμενά του κάθε εγγράφου πρώτα ως προς το color και ύστερα ως προς το volume, που είναι ακριβώς αυτό που χρειαζόμαστε στο συγκεκριμένο ερώτημα.

Καταληκτικά, σε πραγματικές εφαρμογές με χιλιάδες χρήστες, όπου η βάση δεδομένων θα πρέπει να απαντάει σε πολλές χιλιάδες και ίσως εκατομμύρια ερωτημάτων, αποτελεί επιτακτική ανάγκη η δημιουργία των κατάλληλων ευρετηρίων, ώστε να υπάρχει γρήγορη απόκριση για το χρήστη, αλλά και σωστή εκμετάλλευση των υπολογιστικών πόρων του συστήματος.

## 3.5 Sharding

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, το σπάσιμο των δεδομένων (sharding) ή horizontal scaling χωρίζει το σύνολο των δεδομένων σε μικρότερα επιμέρους κομμάτια δεδομένων και τα διανέμει σε πολλά διαφορετικά μηχανήματα. Το κάθε ένα από αυτά τα μηχανήματα που ονομάζεται shard, είναι μια ανεξάρτητη βάση δεδομένων. Όμως οι shards στο σύνολό τους, αποτελούν θεωρητικά μία και μοναδική ενιαία βάση δεδομένων.

Το Sharding εφαρμόζεται στο επίπεδο της συλλογής. Στη δική μας περίπτωση, όπου έχουμε αποθηκεύσει τα δεδομένα μας στο GridFS, υπάρχουν δύο συλλογές, η file collection και η chunk collection. Η συλλογή file που αποθηκεύει τα metadata μας έχει πολύ μικρό μέγεθος και όπως προτείνεται στο documentation της MongoDB, δεν ενδείκνυται για να την κάνουμε sharding. Συγκεκριμένα στην περίπτωση μας έχει μέγεθος 120Mb και αν την κάνουμε sharding θα σπάσει σε τέσσερα επιμέρους chunks, από τα οποία θα αποθηκεύσει δύο ο ένας shard, και από ένα οι άλλοι δυο. Όπως φαίνεται δεν είναι καθόλου αποδοτικό ένας ολόκληρος shard να κρατά μόνο ένα ή δύο chunk δεδομένων, καθώς το κόστος της επικοινωνίας μεταξύ των shard αφαιρεί όλα τα πλεονεκτήματα που έχει το sharding όταν μιλάμε για τόσο μικρό όγκο δεδομένων.

Αντίθετα, η συλλογή chunk που έχει μέγεθος 8846Mb, είναι ιδανική για να φανεί η αποδοτικότητα του sharding στα Big Data. Παρακάτω, θα αναλύσουμε τον τρόπο με τον οποίο υλοποιήσαμε το cluster μας αλλά και πως κάναμε τον διαχωρισμό δεδομένων (data partition) στη συγκεκριμένη συλλογή.

### 3.5.1 Δημιουργία cluster και διαχωρισμός δεδομένων

Όπως αναφέραμε και προηγουμένως, για την παρούσα διπλωματική έχουν δοθεί τρεις servers. Συγκεκριμένα, οι ip διευθύνσεις τους είναι 147.102.19.131, 147.102.19.132 και 147.102.19.133. Το shared cluster που υλοποιήσαμε, αποτελείται από τρεις configuration servers, τρεις shards και έναν query router. Ιδανικά, κάθε ένα από αυτά τα components πρέπει να δημιουργηθεί σε διαφορετικό server, αλλά σε αυτήν την περίπτωση θα χρειαζόμασταν 7 διαφορετικούς servers. Αυτό γίνεται, ώστε στην περίπτωση που, για παράδειγμα, κάποιος από τους configuration servers βγει εκτός λειτουργίας, το cluster να συνεχίζει να είναι λειτουργικό. Στη δική μας περίπτωση, όπου δεν θα έχουμε cluster παραγωγής, αλλά θέλουμε μόνο να δούμε

την αποδοτικότητα του sharding, αρκεί οι τρεις shards να βρίσκονται σε διαφορετικά μηχανήματα. Τους τρεις configuration servers και τον query server, θα τους σηκώσουμε “εικονικά” στο ίδιο μηχάνημα. Συγκεκριμένα, στον server με διεύθυνση ip 147.102.19.133, θα δημιουργήσουμε και τους τρεις configuration servers, τον query router, και τον έναν από τους shards. Στους υπόλοιπους δύο servers θα δημιουργήσουμε τους άλλους δύο shards. Όλα αυτά τα components (εκτός από τον query server) θα υλοποιηθούν ουσιαστικά με μία mongod διεργασία (που όπως αναφέραμε στο κεφάλαιο 3.3 είναι ο primary mongodb daemon), η οποία κάθε φορά θα έχει ενεργοποιημένα διαφορετικά flags, ώστε να επιτελέσει το ρόλο του server για τον οποίο προορίζεται. Ο Query router υλοποιείται με τη διεργασία mongos, η οποία είναι υπεύθυνη για τη δρομολόγηση των ερωτημάτων του πελάτη στο sharded cluster.

Παρακάτω θα περιγράψουμε αναλυτικά τις εντολές που τρέξαμε για να δημιουργήσουμε το cluster. Όσον αφορά τον server με ip 147.102.19.133:

1. Ανοίγουμε ένα terminal και πληκτρολογούμε τις εξής εντολές:

   ```
   sudo mkdir -p /data/db

   sudo mkdir /data/configdb1

   sudo chown ntua /data/configdb1

   mongod --configsvr --dbpath /data/configdb1 --port 26050
   ```

   Συγκεκριμένα για την τελευταία εντολή, δημιουργήσαμε **τον πρώτο configuration server**, ο οποίος θα αποθηκεύει τα metadata στην τοποθεσία /data/configdb1, και τρέχει στην port 26050.

2. Έπειτα ανοίγουμε ένα άλλο terminal, και ακολουθούμε την ίδια σχεδόν διαδικασία για **τον δεύτερο configuration server**:

   ```
   sudo mkdir /data/configdb2

   sudo chown ntua /data/configdb2

   mongod --configsvr --dbpath /data/configdb2 --port 26051
   ```

   Εδώ, πρέπει να τονίσουμε ότι ο δεύτερος configuration server θα τρέχει σε διαφορετική port και συγκεκριμένα την 26051. Αν δημιουργούσαμε configuration servers σε διαφορετικούς servers, τότε δεν θα χρειαζόταν να διευκρινίσουμε την port στην οποία θα τρέχουν, μιας και θα είχαν διαφορετική ip διεύθυνση. Εδώ, που και οι τρεις configuration servers τρέχουν κάτω από την ίδια ip διεύθυνση είναι απαραίτητο να βρίσκονται σε διαφορετική port.

3. Τέλος, για τον **τρίτο configuration server**, ανοίγουμε ένα άλλο terminal και γράφουμε:

   sudo mkdir /data/configdb3

   sudo chown ntua /data/configdb3

   mongod --configsvr --dbpath /data/configdb3 --port 26052

   Τώρα, οι τρεις configuration servers είναι έτοιμοι και αναμένουν για επικοινωνία ο κάθε ένας στη δική του port.

4. Κατόπιν, θα δημιουργήσω τον **query router**, στην port 27020, οπότε αυτή είναι η port στην οποία θα πρέπει να συνδέονται οι πελάτες για να θέτουν τα ερωτήματά τους στο sharded cluster. Ανοίγουμε ένα terminal και γράφουμε:

   mongos --configdb 147.102.19.133:26050,147.102.19.133:26051,147.102.19.133:26052 --port 27020

   Ουσιαστικά με αυτή την εντολή, δημιουργούμε τον query router, και τον ενημερώνουμε για την "τοποθεσία" των config servers.

5. Τέλος, σε ξεχωριστό terminal δημιουργούμε τον **πρώτο shard server** με τις εντολές:

   sudo chown -R ntua /var/lib/mongodb

   mongod -shardsvr -dbpath /var/lib/mongodb

   Με την τελευταία εντολή, δημιουργούμε τον shard server και τον ενημερώνουμε, ώστε να χρησιμοποιήσει την τοποθεσία /var/lib/mongodb για την αποθήκευση των δεδομένων του. Σε αυτή την τοποθεσία, όμως, βρίσκεται ήδη αποθηκευμένη βάση δεδομένων, η οποία προς το παρόν παραμένει αυτούσια σε αυτόν τον shard. Εδώ δεν διευκρινίζουμε σε ποια port να τρεξει ο shard server, οπότε τρέχει στην default port που είναι η 27018.

Έπειτα, συνδεόμαστε στον server με ip 147.102.19.132, και δημιουργούμε **τον δεύτερο shard server**. Ανοίγουμε ένα terminal και πληκτρολογούμε:

sudo mkdir -p /data/db

sudo chown ntua /data/db

mongod -shardsvr

Και σε αυτήν την περίπτωση, ο shard server δημιουργείται στην default port που είναι η 27018, εδώ όμως δεν αντιμετωπίζουμε το πρόβλημα που υπήρχε προηγουμένως με τους config servers, μιας και οι shard servers, βρίσκονται σε διαφορετικά μηχανήματα.

Τέλος, συνδεόμαστε στον server με ip 147.102.19.131, και δημιουργούμε **τον τρίτο shard server**, ακριβώς με τον ίδιο τρόπο.

Πλέον, όλα τα components του cluster μας είναι έτοιμα και το μόνο που απομένει είναι να ενημερώσω τον query server ποια είναι η τοποθεσία των τριών shard servers που θα αποτελέσουν το cluster. Επομένως, ανοίγουμε ένα terminal σε οποιονδήποτε από τους τρεις φυσικούς servers επιθυμούμε και συνδεόμαστε ως administrator στον query router με την παρακάτω εντολή:

```
mongo 147.102.19.133:27020/admin
```

Έπειτα εκτελούμε τις παρακάτω εντολές και το cluster μας είναι έτοιμο για λειτουργία:

```
sh.addShard("147.102.19.133:27018")

sh.addShard("147.102.19.131:27018")

sh.addShard("147.102.19.132:27018")
```

Προς το παρόν, ολόκληρη η βάση δεδομένων βρίσκεται στον server με ip 147.102.19.133. Χρειάζεται να ενεργοποιήσουμε τη δυνατότητα του sharding για τη συγκεκριμένη βάση δεδομένων και ύστερα να ορίσουμε shard key και τον τρόπο με τον οποίο θα γίνει ο διαχωρισμός των δεδομένων.

Αναλυτικότερα, ορίζουμε το shard key {files_id : 1 }. Το files_id (τύπου ObjectId) είναι το μοναδικό αναγνωριστικό κάθε σκηνής. Όπως αναφέραμε και στο προηγούμενο κεφάλαιο, όταν επιλέξουμε για shard key κάποιο πεδίο με τύπο ObjectId που οι τιμές του αυξάνονται με σταθερό και προβλέψιμο ρυθμό, δεν θα υπάρχει καλό write scaling, καθώς όλα τα νέα έγγραφα θα τοποθετούνται στο ίδιο chunk και άρα στον ίδιο shard. Για να επιλύσουμε αυτό το πρόβλημα επιλέγουμε τη μέθοδο Hashed base partitioning, όπου περνάει την τιμή τύπου ObjectId από την hash function και κατανέμει τα έγγραφα ομοιόμορφα εκεί που προστάζει το αποτέλεσμα της.

Επομένως, στο ίδιο terminal που συνδεθήκαμε προηγουμένως ως administrator γράφουμε τις παρακάτω εντολές:

```
db.runCommand({enablesharding: "Savage1"})

use Savage1

db.Savage.chunks.createIndex({files_id : 1 })

sh.shardCollection( "Savage1.Savage.chunks" , {files_id : 1 })
```

Σε αυτό το σημείο, η MongoDB πρώτα σπάει τη συλλογή files σε chunks, και συγκεκριμένα σε 274 chunks, τα οποία βρίσκονται όλα στο ίδιο shard. Έπειτα ενεργοποιείται ο balancer και αρχίζει να διαμοιράζει τα chunks στους άλλους δύο shards. Μετά από περίπου 20 λεπτά θα έχει επέλθει ισορροπία στο cluster μας, και

κάθε shard θα περιέχει 91 chunks (εκτός από έναν που θα περιέχει 92, αφού το 274 δεν διαιρείται ακριβώς με τον αριθμό των shard μας αλλά έχει υπόλοιπο 1).

Τώρα μπορεί οποιοσδήποτε χρήστης να συνδεθεί στο mongo shell του query server που βρίσκεται στη διεύθυνση 147.102.19.133:27020 μέσω της εντολής

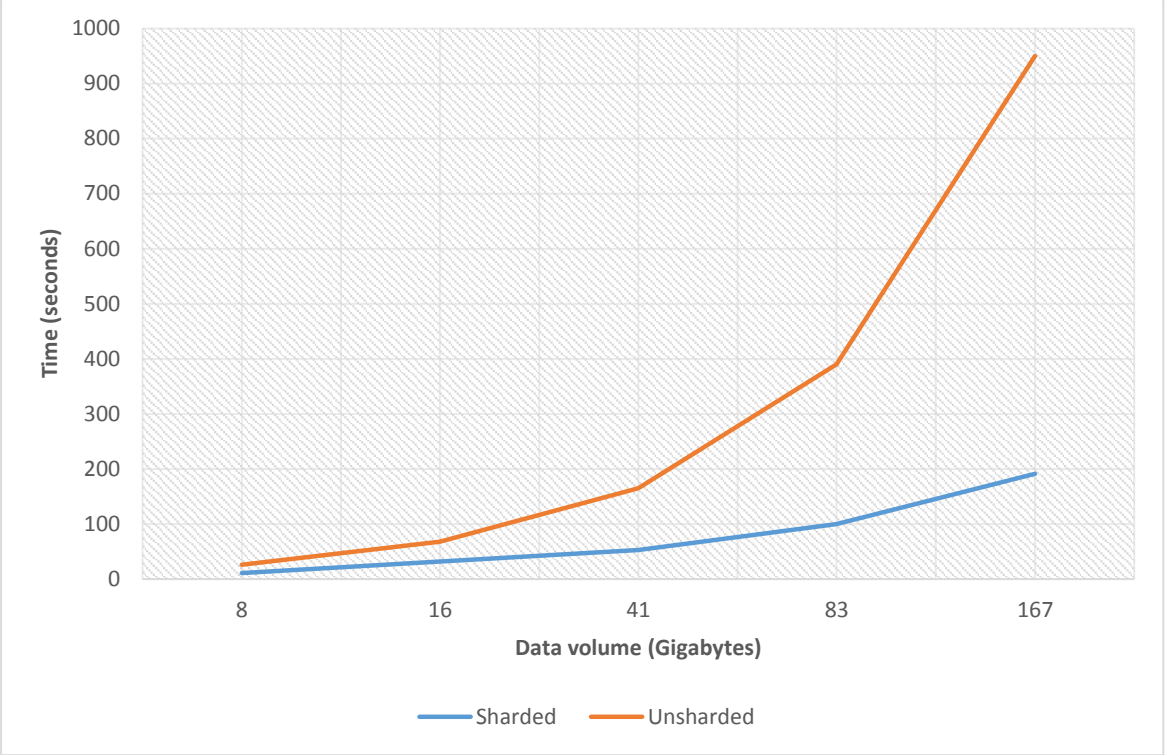
mongo 147.102.19.133:27020

και να τρέξει τα ερωτήματα που επιθυμεί στο sharded cluster.

## 3.5.2 Απόδοση του sharded cluster

Για να μετρήσουμε την απόδοση του sharded cluster, γράψαμε σε java το πρόγραμμα **shard_performance.java**, το οποίο βρίσκεται στο παράρτημα. Αρχικά, συνδεόμαστε ως πελάτης στον query router που βρίσκεται στην port 27020. Ύστερα, θέτουμε στη συλλογή files το ερώτημα: “Ποια έγγραφα περιέχουν κάποιο αντικείμενο που έχει στο πεδίο color τιμή μεγαλύτερη από 0;”. Ουσιαστικά, με αυτό το ερώτημα θέλω να μου επιστρέψει όλα τα έγγραφα που περιέχουν κάποια σκηνή, αφού αν το έγγραφο δεν έχει το πεδίο χρώμα δεν θα επιστρέψει ως αποτέλεσμα. Αφού βρει αυτά τα έγγραφα, μας επιστρέφει ένα κέρσορα, ο οποίος δείχνει στο πρώτο από αυτά. Έπειτα, εμείς διατρέχουμε αυτόν τον κέρσορα, και για κάθε έγγραφο-σκηνή βρίσκουμε τα αντίστοιχα chunks από τα οποία αποτελείται και ενημερώνουμε σε κάθε ένα από αυτά το πεδίο last_visit, βάζοντας για νέα τιμή την ώρα και την ημερομηνία κατά την οποία τα επισκεφτήκαμε. Με αυτόν τον τρόπο, εκτελούμε πολλές λειτουργίες update στη συλλογή chunk collection, ώστε να δούμε την απόδοση που έχει το sharded cluster.

Τέλος, γράψαμε ένα script που τρέχει αυτό το πρόγραμμα επαναληπτικά 1, 2, 5, 10 και 20 φορές. Επομένως, με αυτό το script, καταφέραμε να πάρουμε μετρήσεις για τις περιπτώσεις που ο όγκος δεδομένων μας είναι 8, 16, 41, 83 και 167 Gigabytes αντίστοιχα (αφού όπως είπαμε η συλλογή chunk collection έχει μέγεθος περίπου 8GB). Τρέξαμε αυτό το script μια φορά για την περίπτωση που η βάση είναι ολόκληρη σε ένα server, χωρίς να έχουμε χρησιμοποιήσει την τεχνική του sharding και άλλη μία φορά για την περίπτωση που η βάση είναι διαμοιρασμένη στους τρεις servers του sharded cluster. Όλοι οι χρόνοι που φαίνονται στο παρακάτω διάγραμμα είναι σε δευτερόλεπτα.




Εικόνα 29: Απόδοση του sharded cluster.

Αρχικά, παρατηρούμε ότι όταν τα δεδομένα στα οποία εκτελούμε τη λειτουργία είναι της τάξης των λίγων (5-10) Gigabytes η απόδοση του sharded cluster είναι σχεδόν ίδια με όταν η βάση είναι unsharded. Αυτό οφείλεται στο ότι το κόστος επικοινωνίας μεταξύ των shards του cluster είναι μεγάλο σε σχέση με τους υπολογιστικούς πόρους που απαιτούνται για να επιτελέσουμε τη λειτουργία update σε αυτό το μικρό όγκο δεδομένων. Αντίθετα, όταν εκτελούμε τη λειτουργία update 10 ή 20 φορές στη συλλογή που έχουμε, δηλαδή ο όγκος των δεδομένων είναι της τάξης των εκατοντάδων Gigabytes, τότε παρατηρούμε τις πραγματικές δυνατότητες του sharding. Αναλυτικότερα, όπως βλέπουμε, ο χρόνος εκτέλεσης των updates στο sharded cluster αυξάνεται σχεδόν γραμμικά σε αντίθεση με το χρόνο εκτέλεσης στην unsharded συλλογή, ο οποίος αυξάνεται με εκθετικό ρυθμό.

Συμπερασματικά, στα cluster παραγωγής, όπου η MongoDB απαιτείται να αποθηκεύσει εκατοντάδες Terabytes ή ακόμη και Petabytes, αποτελεί επιτακτική ανάγκη η χρήση της μεθόδου sharding, ώστε να υπάρχει γρήγορη απόκριση για το χρήστη, αλλά και σωστή εκμετάλλευση των υπολογιστικών πόρων του συστήματος.

# 3.6 Replication

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, η αντικατάσταση (replication) στην ουσία είναι η διαδικασία συγχρονισμού των δεδομένων μέσω πολλαπλών εξυπηρετητών. Η φύλαξη πολλαπλών αντιγράφων των δεδομένων σε μια σειρά από διαφορετικούς εξυπηρετητές, εγγυάται την αδιάκοπη λειτουργία της βάσης δεδομένων ακόμα και μετά την κατάρρευση ενός εξυπηρετητή λόγω αστοχίας υλικού του υπολογιστή, ενώ ταυτόχρονα, διευκολύνει την ανάκτηση τυχόν χαμένων δεδομένων και τη διατήρηση αντιγράφων.

Σε αυτή τη διπλωματική δημιουργήσαμε, ένα εικονικό replica set, στο όποιο εισάγαμε τη βάση δεδομένων μας και έπειτα ελέγξαμε τη συμπεριφορά του σε περίπτωση partition, δηλαδή στην περίπτωση όπου ένας η δύο servers του replica set τεθούν εκτός λειτουργίας.

Αναλυτικότερα, για να στήσουμε το replica set που ονομάσαμε “ena” , ανοίγουμε το terminal και πληκτρολογούμε:

```
sudo mkdir -p /data

sudo mkdir /data/replicaset/a

sudo chown ntua /data/replicaset/a

sudo mongod --replSet ena --dbpath /data/replicaset/a --port 27000
```

Η βάση δεδομένων μας βρίσκεται στην τοποθεσία /data/replicaset/a. Με αυτόν τον τρόπο δημιουργούμε το πρώτο μέλος του replica set, το οποίο τρέχει στην port 27000. Έπειτα, σε άλλο terminal πληκτρολογούμε:

```
sudo mkdir /data/replicaset/b

sudo chown ntua /data/replicaset/b

sudo mongod --replSet ena --dbpath /data/replicaset/b --port 27001
```

Έτσι, δημιουργήσαμε το δεύτερο μέλος του replica set στην port 27001. Τέλος, με τις παρακάτω εντολές δημιουργούμε το τρίτο μέλος του replica set στην port 27002:

```
sudo mkdir /data/replicaset/c

sudo chown ntua /data/replicaset/c

sudo mongod --replSet ena --dbpath /data/replicaset/c --port 27002
```

Προς το παρόν, το replica set δεν είναι λειτουργικό, αφού δεν έχουν έρθει σε επικοινωνία τα τρία μέλη του. Θα ορίσουμε ως Primary, το replica set member που τρέχει στην port 27000, διότι σε αυτό υπάρχει αποθηκευμένη η βάση δεδομένων μας.

Έπειτα, ενημερώνουμε τον primary για τις port, στις οποίες βρίσκονται οι δύο άλλοι servers, οι οποίοι θα αποτελέσουν τους Secontaries. Οπότε, ανοίγουμε ένα ξεχωριστό terminal και πληκτρολογούμε:

```
mongo --port 27000

rs.initiate()

rs.conf()

rs.add("OpenStack1:27001")

rs.add("OpenStack1:27002")
```

Σε αυτό το σημείο, ο Primary server είναι έτοιμος και λειτουργικός, οπότε αρχίζει να στέλνει τη βάση δεδομένων στους δύο άλλους servers, οι οποίοι είναι σε κατάσταση αναμονής. Όταν ολοκληρωθεί αυτή η διαδικασία, μετά από 10 λεπτά περίπου, το status των δύο άλλων server αλλάζει και γίνεται Secontary.

Κατόπιν, θέτουμε εκτός λειτουργίας τον ένα από τους δύο secondary servers και παρατηρούμε, όπως ήταν αναμενόμενο, ότι τα άλλα δύο μέλη του replica set είναι απολύτως λειτουργικά, οπότε έχουμε πρόσβαση για να εκτελέσουμε οποιαδήποτε λειτουργία θέλουμε στη βάση δεδομένων. Γι' αυτό το λόγο εκτελούμε μία λειτουργία insert, η οποία εκτελείται κανονικά. Έπειτα ξαναθέτουμε σε λειτουργία τον secondary server που είχαμε βγάλει εκτός λειτουργίας και εκτελούμε ένα ερώτημα find σε αυτόν για να βρούμε το έγγραφο που είχαμε τοποθετήσει προηγουμένως στον primary server. Το αποτέλεσμα είναι ο secondary να μας επιστρέψει το έγγραφο που αναζητάμε, αφού με το που τέθηκε ξανά σε λειτουργία συγχρόνισε τα δεδομένα του με αυτά του primary server.

Καταληκτικά, οι μεγάλες σύγχρονες βάσεις δεδομένων, όπου αποτελούνται από servers σε πολλά μέρη μιας χώρας ή ακόμη και σε πολλές διαφορετικές χώρες, αντιμετωπίζουν το πρόβλημα του partition. Δηλαδή, πέφτει το δίκτυο σε κάποιο σημείο είτε από φυσικά αίτια (φυσική καταστροφή), είτε για άλλους λόγους, όπως για παράδειγμα, η αντικατάσταση κατεστραμμένων μηχανημάτων, με αποτέλεσμα οι servers που αποθηκεύουν τα δεδομένα της βάσης μας να μην μπορούν να επικοινωνήσουν μεταξύ τους. Όμως, οι περισσότερες σύγχρονες εφαρμογές απαιτούν να υπάρχει πάντα διαθεσιμότητα (availability) της βάσης δεδομένων, και ένας τρόπος για να επιτευχθεί αυτό είναι η αντικατάσταση (replication).

## 3.7 Map-Reduce

Για να μετρήσουμε την απόδοση του map-reduce framework χρειαζόμαστε ένα ερώτημα προς εκτέλεση στη βάση, το οποίο να έχει μεγάλες απαιτήσεις σε υπολογιστικούς πόρους, αλλά παράλληλα να έχει και αρκετά μεγάλη σημασία για ένα multimedia dataset. Ένα τέτοιο ερώτημα είναι η εύρεση του ιστογράμματος χρώματος των σκηνών.

Για να υπολογίσουμε το ιστόγραμμα χρώματος χρειάζεται, αφενός να συγκεντρώσουμε τα επί μέρους χρώματα που υπάρχουν σε όλα τα αντικείμενα μιας σκηνής, αφετέρου να υπολογίσουμε τον όγκο του κάθε αντικειμένου αλλά και αθροιστικά, ούτως ώστε να σταθμίσουμε τις τιμές των χρωμάτων.

Αναλυτικότερα, από κάθε αντικείμενο μιας σκηνής υπολογίζουμε τον όγκο της, και το χρωματικό κελί (που όπως αναφέραμε προηγουμένως κυμαίνεται από 1 έως 512) στο οποίο αντιστοιχεί και τον προσθέτουμε σε αυτό το κελί. Όταν διατρέξουμε όλα τα αντικείμενα μιας σκηνής, αθροίζουμε τις τιμές όλων των κελιών για να βρούμε το άθροισμα όλων των όγκων της σκηνής και διαιρούμε την τιμή του κάθε κελιού με το άθροισμα αυτό. Οπότε, τελικά τα κελιά έχουν άθροισμα 1 και η τιμή κάθε κελιού δείχνει το ποσοστό του "όγκου" της σκηνής που περιέχει το αντίστοιχο χρώμα. Με αυτόν τον τρόπο, υπολογίζουμε το ιστόγραμμα χρώματος όλων των σκηνών και εξάγουμε το μέσο ποσοστό για το κάθε κελί, για τις σκηνές στις οποίες συναντάται. Παίρνουμε, συνεπώς, ένα μέτρο για το πόσο κυρίαρχο ρόλο παίζει το κάθε χρωματικό κελί όταν βρίσκεται σε μια σκηνή.

Σε προηγούμενη εργασία που πραγματοποιήθηκε στο πλαίσιο του προγράμματος i-promotion, είχε γραφτεί ένα πρόγραμμα, το οποίο με τη χρήση της μεθόδου map-reduce, υπολόγιζε το ιστόγραμμα χρώματος των σκηνών. Το πρόγραμμα αυτό, περιείχε δύο ζεύγη συναρτήσεων map-reduce για αυτόν τον σκοπό. Το πρώτο ζεύγος map-reduce ομαδοποιούσε τα αντικείμενα ανά σκηνή και χρησίμευε στο να σταθμιστούν οι τιμές των όγκων των αντικειμένων της κάθε σκηνής, ως προς τον όγκο της ίδιας της σκηνής. Το δεύτερο ζεύγος map reduce συναρτήσεων, έπαιρνε τους σταθμισμένους όγκους των αντικειμένων της κάθε σκηνής που υπολογίστηκαν προηγουμένως, και ομαδοποιούσε τα αντικείμενα ανά χρώμα, ώστε να υπολογίσει τελικά τον συνολικό (σταθμισμένο ανά σκηνή) όγκο που καταλαμβάνει το κάθε ένα χρώμα.

Στην παρούσα διπλωματική, ύστερα από ανάλυση του τρόπου με τον οποίο δημιουργείται το ιστόγραμμα χρώματος, αλλά και τη λεπτομερή παρατήρηση των δεδομένων μας, καταφέραμε να φτιάξουμε το ιστόγραμμα χρώματος χρησιμοποιώντας μόνο ένα ζεύγος map-reduce συναρτήσεων. Συγκεκριμένα, παρατηρήσαμε ότι κάθε έγγραφο της βάσης μας περιέχει από μια σκηνή. Η συνάρτηση map παίρνει ως είσοδο κάθε έγγραφο ξεχωριστά, και εφαρμόζει πάνω σε αυτό τις λειτουργίες της.

Οπότε, τα αντικείμενα που υπάρχουν στα δεδομένα μας, ήδη έρχονται ομαδοποιημένα ανά σκηνή στη συνάρτηση map. Επομένως, το πρώτο ζεύγος map-reduce, στο οποίο αναφερθήκαμε πριν, μπορεί να συγχωνευτεί και οι υπολογισμοί που επιτελούσε να εκτελεστούν στη δική μας συνάρτηση map.

Συγκεκριμένα, γράψαμε το πρόγραμμα **new_map _red.java**, το οποίο βρίσκεται στο παράρτημα. Όπως αναφέραμε, στο δικό μας πρόγραμμα γράψαμε μονάχα ένα ζεύγος map reduce συναρτήσεων, ώστε να δημιουργήσουμε το ιστόγραμμα χρωμάτων. Αναλυτικότερα η συνάρτηση map παίρνει ως είσοδο ένα ένα τα έγγραφα της συλλογής file και επιτελεί τους ακόλουθους υπολογισμούς:

1. Διατρέχει όλα τα αντικείμενα που περιέχει το συγκεκριμένο έγγραφο και κρατάει στη μεταβλητή scenevolume, το άθροισμα των όγκων τους. Όπως είπαμε, κάθε έγγραφο είναι και μία 3d σκηνή, οπότε το scenevolume θα είναι τελικά ο συνολικός όγκος της συγκεκριμένης σκηνής.
2. Έπειτα, διατρέχει ξανά όλα τα αντικείμενα της ίδιας σκηνής, ώστε να πάρουμε το χρώμα και τον όγκο που έχουν. Κατόπιν, διαιρούμε τον όγκο κάθε αντικειμένου με τον συνολικό όγκο της σκηνής που βρήκαμε προηγουμένως ώστε να υπολογίσουμε τον σταθμισμένο όγκο του (newvolume), δηλαδή το ποσοστό της σκηνής που καταλαμβάνει.
3. Τέλος, για κάθε ένα αντικείμενο, κάνουμε emit το ζεύγος τιμών {color : newvolume}

Έπειτα, η MongoDB ομαδοποιεί τα ζεύγη αυτά, ως προς την τιμή του κλειδιού, δηλαδή του color. Επομένως, η συνάρτηση reduce λαμβάνει ως είσοδο τους σταθμισμένους όγκους του κάθε χρώματος. Αυτό που κάνει είναι απλά να τους αθροίσει και έπειτα, το ιστόγραμμα χρώματος είναι έτοιμο.

Σε αυτή τη διπλωματική δεν θα επικεντρωθούμε στην παρουσίαση του ιστογράμματος, μιας και αυτό έγινε στην προγενέστερη εργασία, αλλά θα αναλύσουμε την απόδοση των δύο προγραμμάτων που εξήγησα προηγουμένως.

Ειδικότερα, ο χρόνος που απαιτείται για τη δημιουργία του ιστογράμματος χρώματος από το πρόγραμμα με τα δύο ζεύγη map-reduce συναρτήσεων είναι περίπου 1190 milisecond. Στο πρόγραμμα που δημιουργήσαμε εμείς, ο χρόνος που απαιτείται είναι περίπου 910 milisecond. Μπορεί με την πρώτη όψη να μην φαίνεται μεγάλη η διαφορά στην απόδοση των δύο προγραμμάτων, αλλά πετύχαμε μείωση του χρόνου εκτέλεσης κατά 17,6%. Στη συγκεκριμένη περίπτωση, αυτή η επιτάχυνση μεταφράζεται μόνο σε 210 millisecond, διότι η συλλογή files έχει πολύ μικρό μέγεθος (120Megabytes).

Σε πραγματικά συστήματα, όπου ο όγκος των δεδομένων θα είναι της τάξης των χιλιάδων Gigabytes ή ακόμη και πολλών terabytes, η επιτάχυνση του προγράμματος μας θα ήταν υπερπολύτιμη, καθώς θα εξοικονομούσε πολλά λεπτά και ίσως ώρες απο τους χρήστες, ενώ ταυτόχρονα, θα ελευθέρωνε τους υπολογιστικούς πόρους των μηχανημάτων που θα χρησιμοποιούσε, ώστε να επιτελέσουν κάποια άλλη εργασία.

# Βιβλιογραφία


[1] Site from which we took our data: https://savage.nps.edu/Savage/

[2] MongoDB site: https://www.mongodb.org/

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

[4] P. A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems".

[5] Y. Saito and M. Shapiro, "Optimistic Replication".

[6] N. Fatemi, M. Lalmas and T. Rölleke, "How to retrieve multimedia documents described by MPEG-7".

[7] Y. Chu, L.-T. Chia and S. S. Bhowmick, "Looking at Mapping, Indexing & Querying of MPEG-7 Descriptors in RDBMS with SM3".

# Παράρτημα

/*

 *

 *



*/

```
package com.mongodb;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.stream.JsonReader;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.bson.BSONObject;
import org.bson.types.ObjectId;
import org.jaxen.function.SumFunction;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
```

```
public class metatroph {



   public static void main(String[] args) throws JSONException {



      try {
         long startTime = System.currentTimeMillis();

         Mongo mongo = new Mongo("localhost", 27017);

         DB Mydb = mongo.getDB("Savage");

         DBCollection collection = Mydb.getCollection("Savage.files");

         DBCursor cursor = collection.find();

         double length, height, depth, width, volume, newvol;
         int cell;
         double[] volumetable;
         int[] celltable;
         int volumecnt;
         double sceneVolumeSum;



         while (cursor.hasNext()) {

            BasicDBObject doc = (BasicDBObject) cursor.next();

            BasicDBObject info = (BasicDBObject) doc.get("metadata");
            ObjectId identity = (ObjectId) doc.get("_id");
```

```
BasicDBList newdescriptors = new BasicDBList();

volumetable =  new double[1000000];
celltable =  new int[1000000];
volumecnt=0;

sceneVolumeSum = 0.0;

JsonParser jsonParser = new JsonParser();
try {

   JsonArray DescriptorCollection = jsonParser.parse(String.valueOf(info))

         .getAsJsonObject().get("Mpeg7")

         .getAsJsonObject().getAsJsonArray("Description").get(1)

         .getAsJsonObject().get("MultimediaContent")

         .getAsJsonObject().get("StructuredCollection")

         .getAsJsonObject().getAsJsonArray("Collection").get(1)

         .getAsJsonObject().getAsJsonArray("DescriptorCollection");



   length = DescriptorCollection.size();

   for (int i = 0; i < length; i++) {

      try {
```

```
JsonElement BoundingBox3DSize =

      DescriptorCollection.getAsJsonArray().get(i).getAsJsonObject()

            .getAsJsonObject().getAsJsonArray("Descriptor").get(0)

            .getAsJsonObject().get("BoundingBox3DSize");


height = BoundingBox3DSize.getAsJsonObject().get("BoxHeight").getAsDouble();

width = BoundingBox3DSize.getAsJsonObject().get("BoxWidth").getAsDouble();

depth = BoundingBox3DSize.getAsJsonObject().get("BoxDepth").getAsDouble();

volume = height * width * depth;

if (Double.isNaN(height)) {

   continue;
}


JsonElement RGB =

      DescriptorCollection.getAsJsonArray().get(i).getAsJsonObject()

            .getAsJsonObject().getAsJsonArray("Descriptor").get(1)

            .getAsJsonObject().get("Geometry3D")

            .getAsJsonObject().get("DominantColor3D")
```

```
                        .getAsJsonObject().get("Value")

                        .getAsJsonObject().get("Index");


//Since we reached here, we have gotten both the volume and the color

//Otherwise an exception would have been thrown

                String[] RGBarr = RGB.getAsString().split(" ");

                cell = (int) (Math.floor((double) Integer.parseInt(RGBarr[0]) / 32) + 8 * Math.floor((double) Integer.parseInt(RGBarr[1]) / 32) + 64 * Math.floor((double) Integer.parseInt(RGBarr[2]) / 32));

                sceneVolumeSum += volume;

                volumetable[volumecnt] = volume;
                celltable[volumecnt] = cell;
                volumecnt++;


              } catch (Exception e1) {


              }

            }


            if (sceneVolumeSum == 0.0) {
              //System.out.println("Something went wrong");
              continue;
            }
```

```
            for (int i = 0; i < volumecnt; i++) {
               BasicDBObject newdescriptor = new BasicDBObject();
               newdescriptor.put("color", celltable[i]);
               newvol = volumetable[i] / sceneVolumeSum;
               newdescriptor.put("volume", newvol);
               newdescriptors.add(newdescriptor);
            }

            BasicDBObject neo = new BasicDBObject();
            neo.append("$set", new BasicDBObject().append("metadata.NewDescriptors",
newdescriptors));
            BasicDBObject searchQuery = new BasicDBObject().append("_id", identity);
            collection.update(searchQuery, neo);


         } catch (Exception e) {

            //e.printStackTrace(); //fields don't exist

         }


      }
      long endTime   = System.currentTimeMillis();
      long totalTime = endTime - startTime;
      System.out.println(totalTime);

   } catch (IOException ex) {

      Logger.getLogger(RetrieveMp7.class.getName()).log(Level.SEVERE,
```

```
                null, ex);

        }


    }
}
```

```
package com.mongodb;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.stream.JsonReader;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.bson.BSONObject;
import org.bson.types.ObjectId;
import org.jaxen.function.SumFunction;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import java.util.Date;
import java.text.SimpleDateFormat;
```

```
import java.util.Calendar;


public class shard_performance {


   public static void main(String[] args) throws JSONException {


      try {
         long startTime = System.currentTimeMillis();


         Mongo mongo = new Mongo("localhost", 27017);


         DB Mydb = mongo.getDB("Savage");


         Date today;
         SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd-hh.mm.ss");
         String folderName;



            BasicDBObject query = new BasicDBObject("metadata.NewDescriptors.color", new
BasicDBObject("$gt", 0));
            DBCollection collection = Mydb.getCollection("Savage.files");
            DBCollection collection1 = Mydb.getCollection("Savage.chunks");
            DBCursor cursor = collection.find(query);


            try {
               while (cursor.hasNext()) {
                  //System.out.println(cursor.next());
                  BasicDBObject doc = (BasicDBObject) cursor.next();
                  ObjectId identity = (ObjectId) doc.get("_id");
```

```
                BasicDBObject neo = new BasicDBObject();
                today = Calendar.getInstance().getTime();
                folderName = formatter.format(today);
                neo.append("$set", new BasicDBObject().append("last_visit", folderName));
                BasicDBObject searchQuery = new BasicDBObject().append("files_id", identity);
                collection1.updateMulti(searchQuery, neo);
              }

            } finally {
              cursor.close();
            }

          long endTime = System.currentTimeMillis();
          long totalTime = endTime - startTime;
          System.out.println(totalTime);
        } catch (IOException ex) {

          Logger.getLogger(RetrieveMp7.class.getName()).log(Level.SEVERE,

              null, ex);

        }


    }

}
```

```
package com.mongodb;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.net.UnknownHostException;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MapReduceCommand;
import com.mongodb.MapReduceOutput;
import com.mongodb.MongoClient;

public class new_map_red {

   private String dbName;
   private DB db;
   private MongoClient mongoClient;

   private static final String map1 = "function(){"
         + "try{"
         + "var length = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection.length;"
         + "var i = 0;"
         + "var toEmit;"
         + "var scenevolume=0;"
         + "var UNDERFLOW  = 0.000000001;"
         + "var OVERFLOW  = 10000000;"
         + "var toEmit2 = [];"
         + "var maxINT = 100000000;"
```

```
+ "for(var j = 0; j < length; j++){"
+ "try {"
+ "var v1 = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxHeight;"
+ "var v2 = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxDepth;"
+ "var v3 = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxWidth;"
+ "var volume = v1*v2*v3;"
+ "if(volume < UNDERFLOW || volume > OVERFLOW)"
+ "{ volume = 0;}"
+ "if( scenevolume + volume > maxINT){"
+ " scenevolume = maxINT;"
+ " }"
+ " else"
+ " scenevolume += volume;"
+ "}"
+ "catch(err3){"
+ "}"
+ "}"
```

```
+ "for(var j = 0; j < length; j++){"
+ "try {"
+ "var v1 = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxHeight;"
+ "var v2 = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxDepth;"
+ "var v3 = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxWidth;"
+ "var volume = v1*v2*v3;"
```

```
        + "if(volume < UNDERFLOW || volume > OVERFLOW)"
        + "             volume = 0;"
        + "volume = volume/scenevolume;"
        + "var RGB = this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Collection[1].DescriptorCollection[j].Descriptor[1].Geometry3D.DominantColor3D.Value.Index;"
        + "var colors = RGB.split(' ');"
        + "var dim1 = Math.floor(colors[0] / 32);"
        + "var dim2 = Math.floor(colors[1] / 32);"
        + "var dim3 = Math.floor(colors[2] / 32);"
        + "var cell = dim3 * 64 + dim2 * 8 + dim1;"
        + "                     if(!isNaN(volume))"
        + "                             emit(cell, volume);"

        + "}catch(err){"
//                              + "                 print(err);"
        + "}"
        + "}"
        + "}"
        + "catch(err2){"
        //                      + "                 print(err2);"
        + "}"
        + "};";
```

```
    private static final String reduce2 = "function(k, values) {"
        + "var sum = 0;"
        + "var count = 0;"
        + "var MAX_INT = 100000000;"
        + "for (var i = 0; i < values.length; i++){"
        + "             if(!isNaN(values[i])){"
```

```
        + "                     if( sum + values[i] > MAX_INT){"
        + "                             sum = max_INT;"
        + "                     }"
        + "                     else{"
        + "                             sum += values[i];"
        + "                     }"
        + "                     count++;"
        + "             }"
        + "}"
        + "return (k, sum / count);"
        + "};";
```

```
    private static final String outputCollection1 = "out1";
```

```
    private void executeQuery(BasicDBObject searchQuery, String collectionName, String outputCollection ,String map, String reduce) throws FileNotFoundException{

        MapReduceCommand mr = new MapReduceCommand(db.getCollection(collectionName), map, reduce, outputCollection, MapReduceCommand.OutputType.REPLACE, searchQuery);
        MapReduceOutput out = db.getCollection(collectionName).mapReduce(mr);
        for (DBObject o : out.results()) {
            System.out.println(o.toString());
        }
    }

    public new_map_red(MongoClient mongoClient, String dbName){
        this.dbName = dbName;
        this.mongoClient = mongoClient;
        this.db = mongoClient.getDB(dbName);
    }
```

```
    public static void main(String[] args) throws Exception{

        MongoClient mongoClient = new MongoClient("localhost", 27017);

        new_map_red mc = new new_map_red(mongoClient, "Savage");
        int i;
        long startTime = System.currentTimeMillis();
        BasicDBObject searchQuery = new BasicDBObject();

        for (i = 0; i < 1; i++) {
            mc.executeQuery(searchQuery, "Savage.files", outputCollection1, map1, reduce2);
        }
        long endTime = System.currentTimeMillis();
        System.out.println(endTime - startTime);
    }
}
```