ΕΠΕΞΕΡΓΑΣΙΑ ΧΩΡΟΚΕΙΜΕΝΙΚΩΝ ΕΡΩΤΗΜΑΤΩΝ ΣΕ REDIS - REDISEARCH

ΔΗΜΟΠΟΥΛΟΣ ΑΓΓΕΛΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΔΟΥΛΚΕΡΙΔΗΣ ΧΡΗΣΤΟΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΚΑΙ ΥΠΗΡΕΣΙΕΣ

ΚΑΤΕΥΘΥΝΣΗ: ΠΡΟΗΓΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΕΙΡΑΙΑΣ 2020

ΕΥΧΑΡΙΣΤΙΕΣ

Ολοκληρώνοντας την παρούσα εργασία θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου κ. Δουλκερίδη Χρήστο για την υπομονή που έδειξε σε όλο το διάστημα της εκπόνησης της διπλωματικής αυτής εργασίας. Η πολύτιμη βοήθεια που μου πρόσφερε και η επιστημονική καθοδήγηση που μου παρείχε κατά τη διάρκεια της συγγραφής, τόσο με σχόλια και διορθώσεις, όσο και με συμβουλές και συμπληρώσεις, συνέβαλαν στην επιτυχή ολοκλήρωση της διπλωματικής μου.

Επίσης, θα ήθελα να ευχαριστήσω ιδιαιτέρως τον συνάδελφό μου Νίκο Κουτρουμάνη που με προθυμία στήριξε την προσπάθειά μου και με βοήθησε να ξεπεράσω δυσκολίες και εμπόδια που προέκυψαν κατά την εκπόνηση της διπλωματικής αυτής εργασίας.

Τέλος, θα ήταν παράβλεψη να μην ευχαριστήσω την οικογένειά μου που με στήριξε ηθικά και ψυχολογικά, πίστεψε στις ικανότητές μου και αποτέλεσε κινητήρια δύναμη για την ολοκλήρωση της παρούσας εργασίας.

ΠΕΡΙΛΗΨΗ

Ιδιαίτερα την τελευταία δεκαετία η μεγάλη ανάπτυξη των δυνατοτήτων των κινητών τηλεφώνων και ιδιαίτερα η δυνατότητα χρήσης GPS από όλο και περισσότερους χρήστες παράλληλα, έχουν οδηγήσει στην παραγωγή μεγάλου όγκου χωρικών δεδομένων. Σε συνδυασμό με την έξαρση των κοινωνικών δικτύων και ιστοσελίδων κριτικών όπως Twitter και IMDB αντίστοιχα έχει παρουσιαστεί μία ακόμα πολύ ενδιαφέρουσα κατηγορία δεδομένων, αυτή των χωροκειμενικών δεδομένων. Λόγω των παραπάνω, ο ρυθμός παραγωγής αλλά και ο όγκος αυτών των δεδομένων έχουν αυξηθεί σε μεγάλο βαθμό με συνέπεια να προκύπτουν πολλές προκλήσεις με βασικό όμως στόχο την αποτελεσματική αποθήκευση, διαχείριση και προσπέλαση αυτών.

Για την διαχείριση τέτοιου μεγέθους δεδομένων έχουν υιοθετηθεί τα τελευταία χρόνια κατανεμημένα συστήματα βάσεων δεδομένων και πιο συγκεκριμένα αυτά των συστημάτων NoSQL. Παρέχουν πλέον μία σίγουρη επιλογή για την διαχείριση μεγάλου όγκου δεδομένων, προσφέροντας υψηλές επιδόσεις, διαθεσιμότητα και κλιμακωσιμότητα. Έχοντας ως στόχο να παρέχουμε έναν ενιαίο τρόπο προσπέλασης δεδομένων αποθηκευμένα σε βάσεις NoSQL, σε αυτή την διπλωματική εργασία παρουσιάζεται ένα επανασχεδιασμένο Application Programming Interface (API) το οποίο κάνει την διαδικασία προσπέλασης ανεξαρτήτως του συστήματος πολύ εύκολη προς τον χρήστη. Το υπάρχον API προσφέρει πολλά πρότυπα ερωτήματα για την σύνταξη ερωτημάτων και την εκτέλεση λειτουργιών προσπέλασης. Εμπλουτίζεται όμως με την προσθήκη της δυνατότητας εκτέλεσης κειμενικών και χωροκειμενικών ερωτημάτων.

Πολλές σύγχρονες NoSQL βάσεις δεδομένων υποστηρίζουν ευρετηρίαση χωρικών δεδομένων αλλά όχι των χωροκειμενικών που προαναφέρθηκαν. Εκμεταλλευόμενοι αυτό το κενό, παρουσιάζεται η υλοποίηση μίας τεχνικής ευρετηρίασης χωροκειμενικών δεδομένων καθώς επίσης η διενέργεια πειραμάτων με την χρήση της Redis αλλά και του module αυτής RediSearch, με στόχο να μελετήσουμε την επίδοση και την αποδοτικότητα αυτού του συστήματος για τα χωροκειμενικά δεδομένα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΛΙΣΤΑ ΠΙΝΑΚΩΝ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ ΠΡΩΤΟ

1. ΕΙΣΑΓΩΓΗ

Αντικείμενο μελέτης της παρούσας διπλωματικής εργασίας αποτέλεσε το ζήτημα διαχείρισης χωροκειμενικών δεδομένων. Τα δεδομένα αυτά είναι απόρροια της εξάπλωσης φορητών συσκευών με δυνατότητα GPS που παρέχουν τεράστιες δυνατότητες στους χρήστες τους, με αποτέλεσμα να δημιουργείται ένα πολύ μεγάλο πλήθος πληροφοριών προσδιοριζόμενα από τη γεωγραφική τοποθεσία. Πολλά όμως από αυτά τα χωρικά αντικείμενα, σχετίζονται και με κάποια περιγραφή κειμένου όπως για παράδειγμα εικόνες του Instagram, αναρτήσεις του Twitter. Αυτός ο συνδυασμός έχει ως αποτέλεσμα έναν νέο τύπο ερωτημάτων, αυτόν των χωροκειμενικών. Παράλληλα όμως είναι πολλές και οι προκλήσεις που αντιμετωπίζουν τα συστήματα διαχείρισης των δεδομένων αυτών, καθώς αυξάνονται με πρωτοφανή ρυθμό όγκου και ταχύτητας. Η χρήση ενός συστήματος βάσης δεδομένων NoSQL αντί μιας κλασικής σχεσιακής βάσης δεδομένων επιτρέπει τη διαχείριση μεγάλου όγκου δεδομένων, καθώς αυτά παρέχουν επεκτάσιμη κατανεμημένη αποθήκευση και επερώτηση δεδομένων και επίσης είναι πολύ πιο ευέλικτες όσον αφορά τους περιορισμούς που απορρέουν από τις συσχετίσεις πινάκων και τον ορισμό «σχήματος».

1.1 ΟΡΙΣΜΟΣ ΠΡΟΒΛΗΜΑΤΟΣ

Παρόλα τα πολλά προνόμια των NoSQL συστημάτων πολλά από αυτά, ίσως και όλα, δεν υποστηρίζουν άμεσα χωροκειμενική ευρετηρίαση. Υπάρχουν βέβαια πολλές μελέτες που προτείνουν διάφορες τεχνικές και μεθόδους για την υποστήριξη αυτού του τύπου δεδομένων όπως περιγράφεται εδώ [1]. Το παραπάνω καθιστά από μόνο του ένα πρόβλημα και μία πρόκληση για την αποδοτική προσπέλαση τέτοιου τύπου δεδομένων.

Επίσης, η απουσία μίας ενιαίας γλώσσας που θα μπορούσε να λειτουργήσει σε NoSQL αποθήκες δεδομένων για την εκτέλεση ερωτημάτων, αποτελεί κίνητρο για την ύπαρξη ενός API του οποίου στόχος είναι η πρόσβαση δεδομένων σε τέτοιου είδους αποθήκες. Κάθε αποθήκη NoSQL έχει τη δική της γλώσσα ερωτήματος, δυσκολεύοντας τους προγραμματιστές που θέλουν να έχουν πρόσβαση στα αποθηκευμένα δεδομένα, καθώς είναι αναγκαία η εξοικείωση με το λεξιλόγιο της κάθε μία από αυτές. Το API που παρουσιάζεται στην συνέχεια υποστηρίζει πρότυπα ερωτήματα και τελεστές για το σχηματισμό εντολών πρόσβασης. Λαμβάνοντας ως δεδομένο ότι το πρωταρχικό μας ενδιαφέρον της παρούσας εργασίας είναι τα χωροκειμενικά δεδομένα, στόχος είναι να ενσωματώσουμε στο API τη δυνατότητα πρόσβασης σε αυτού του τύπου τα δεδομένα.

1.2 ΣΤΟΧΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ

Σύμφωνα με το [2] παρουσιάζεται μία φιλική προς τον προγραμματιστή διεπαφή για την πρόσβαση δεδομένων σε NoSQL αποθήκες με το όνομα NoDA. Η διεπαφή έχει σχεδιαστεί έχοντας ως βασική σκέψη την παροχή ενός συνόλου εργαλείων και λειτουργιών για την έκφραση εντολών πρόσβασης δεδομένων, με έναν όμως αφηρημένο και ενιαίο τρόπο ανεξαρτήτως του συστήματος NoSQL που υλοποιεί.

Ο στόχος λοιπόν αυτής της διπλωματικής χωρίζεται σε 3 σκέλη.

- Η διεπαφή NoDA στην πρώτη της έκδοση διαχειρίζεται χωρικά και χωρο-χρονικά δεδομένα. Επεκτείνουμε λοιπόν τις δυνατότητες της με την προσθήκη διαχειρίσεις χωροκειμενικών δεδομένων και την εξελίσσουμε με τον επανασχεδιασμό της αρχιτεκτονικής της.

- Υλοποίηση του NoDA API μέσω της RediSearch η οποία υποστηρίζει εγγενώς ευρετηρίαση και ερωτήματα χωρικών και κειμενικών δεδομένων.

- Ανάπτυξη πειραματικού ευρετηρίου και ερωτημάτων χωροκειμενικών δεδομένων μέσω της Redis.

Επικεντρωνόμαστε στις περιπτώσεις χωροκειμενικών ερωτημάτων σε δεδομένα και επομένως μελετάμε διεξοδικά την απόδοση της Redis και κατά συνέπεια του Redisearch module, όσον αφορά την αποδοτικότητα αυτών σε τέτοια δεδομένα. Αλλά επίσης προσπαθούμε να αποκτήσουμε μια βαθύτερη κατανόηση της απόδοσης αυτής της βάσης δεδομένων, καθώς μπορεί να υποστηρίξει την αναπαράσταση γεωχωρικών δεδομένων αλλά και κατάλληλα ευρετήρια για την εκτέλεση χωρικών ερωτημάτων.

1.3 ΔΟΜΗ ΕΡΓΑΣΙΑΣ

Παρακάτω περιγράφεται το πώς είναι δομημένη η παρούσα εργασία στα επόμενα κεφάλαια.

- Το Κεφάλαιο 2 περιγράφει το NoSQL σύστημα που θα μελετήσουμε, δηλαδή την Redis και του module αυτής Redisearch. Καθώς επίσης γίνεται μία αναλυτική αναφορά των δυνατοτήτων που παρέχουν αλλά και των εγγενών λύσεων για χωρική και χωροκειμενική ευρετηρίαση.
- Το Κεφάλαιο 3 περιγράφει την νέα υλοποίηση του NoDA API και των νέων υποστηριζόμενων προτύπων και τελεστών που αναπτύχθηκαν. Επίσης παρουσιάζεται η αρχιτεκτονική του API, προκειμένου να δοθεί μια εικόνα της δομής του αλλά και των δυνατοτήτων επέκτασής του. Τέλος, παρατίθεται μια τεκμηρίωση χρήσης αυτού και παρουσιάζονται περιπτώσεις χρήσης χωρικών και χωροκειμενικών ερωτημάτων.
- Το Κεφάλαιο 4 παρουσιάζει μία προσέγγιση ευρετηρίασης και επερώτησης χωρικειμενικών δεδομένων με την χρήση εγγενών τύπων δεδομένων της Redis, χωρίς την χρήση του Redisearch module.
- Το Κεφάλαιο 5 παραθέτει μία λεπτομερή πειραματική μελέτη πάνω σε ερωτήματα χωρικειμενικών δεδομένων καθώς επίσης συγκρίνει τις επιδόσεις της Redis αλλά και Redisearch. Συγκεκριμένα, εκτελούνται και παρουσιάζονται 2 σκέλη πειραμάτων, με στόχο την κατανόηση και την καταγραφή των επιδόσεων των χωροκειμενικών ευρετηρίων σε αυτό το σύστημα.

- Το Κεφάλαιο 6 παρουσιάζει μία σύνοψη των αποτελεσμάτων της πειραματικής μελέτης και προτείνει ενδιαφέρουσες ιδέες για πιθανή μελλοντική εργασία.

ΚΕΦΑΛΑΙΟ ΔΕΥΤΕΡΟ

2. ΕΠΙΣΚΟΠΗΣΗ REDIS - REDISEARCH ΓΙΑ ΧΩΡΙΚΑ ΚΑΙ ΧΩΡΟΚΕΙΜΕΝΙΚΑ ΔΕΔΟΜΕΝΑ

2.1 REDIS

Η Redis είναι ένα σύστημα αποθήκευσης δεδομένων πάνω στη μνήμη RAM, το οποίο μπορεί να χρησιμοποιηθεί ως βάση δεδομένων, cache ή κατανεμητή μηνυμάτων. Ανήκει στη κατηγορία των key-value stores το οποίο σημαίνει πώς ο τρόπος αποθήκευσης των δεδομένων είναι associative arrays. Υποστηρίζει δομές δεδομένων όπως συμβολοσειρές, hashes, λίστες, σύνολα, ταξινομημένα σύνολα, bitmaps και υπερκαταλόγους. Επίσης υποστηρίζει γεωχωρικά ευρετήρια με ερωτήματα ακτίνας και ροές δεδομένων (streams). Προκειμένου να επιτύχει την εξαιρετική της απόδοση, η Redis φορτώνει το σύνολο δεδομένων στη μνήμη. Ανάλογα με τις ανάγκες μας, μπορούμε να αποθηκεύσουμε μόνιμα το σύνολο δεδομένων είτε στο δίσκο ανά x χρονικό διάστημα, είτε προσθέτοντας κάθε εντολή σε ένα αρχείο καταγραφής (log file).

Η Redis δίνει την δυνατότητα αποθήκευσης των δεδομένων στο δίσκο και παρέχει υψηλή διαθεσιμότητα μέσω του Redis Sentinel και αυτόματο διαμοιρασμό των δεδομένων με το Redis Cluster (Κεφ. 3) με την τεχνική του sharding, επίσης υποστηρίζει ασύγχρονο replication των δεδομένων ακολουθώντας την master-slave λογική, αυτόματη επανασύνδεση και ανακατεύθυνση των κόμβων σε περιπτώσεις σφάλματος (failover) [3].

2.1.1 ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Όπως προαναφέρθηκε η Redis υποστηρίζει αρκετές διαφορετικές δομές δεδομένων, παρακάτω θα δούμε τις πιο σημαντικές και συχνά χρησιμοποιούμενες σύμφωνα με το [15, para. 1.2], [15, Ch. 3, pp. 40-53].

- **Συμβολοσειρές (Strings):** Η πιο απλή δομή δεδομένων είναι αυτή όπου μπορεί να ανατεθεί σε ένα κλειδί μία ακολουθία χαρακτήρων (Strings). Στην ουσία με την δομή αυτή έχουμε την αντιστοίχηση ενός ονόματος (key) με μια τιμή (value). Η δομή αυτή διαθέτει εντολές ακαριαίας εκτέλεσης, εξασφαλίζοντας πως δεν θα υπάρξει ποτέ συνθήκη ανταγωνισμού από δύο πελάτες για το ίδιο κλειδί.
- **Διασυνδεδεμένη Λίστα (Lists):** Αυτή η δομή έχει την μορφή ουράς η οποία είναι ταξινομημένη βάση της σειράς εισαγωγής των δεδομένων, κάθε στοιχείο της λίστας έχει έναν δείκτη για το επόμενο στοιχείο που ακολουθεί. Το κυρίως πλεονέκτημα αυτής της δομής είναι ο σταθερός χρόνο εκτέλεσης της προσθήκης στοιχείων στην πρώτη ή την τελευταία θέση της λίστας, ανεξάρτητα από το μέγεθος της.
- **Hashes:** Η δομή αυτή αποτελείται από ζεύγη πεδίων και τιμών με αποτέλεσμα σε ένα hash να μπορούν να αποθηκευτούν παραπάνω από ένα πεδία με την αντίστοιχη τιμή τους. Αυτό κάνει την δομή hash ιδανική για την αναπαράσταση αντικειμένων, κάθε hash μπορεί να αποθηκεύσει $2^{32} – 1$ ζεύγη πεδίου - τιμής.
- **Σύνολα (Sets):** Οι τιμές που αποθηκεύονται σε ένα σύνολο είναι αταξινόμητες μη επαναλαμβανόμενες ακολουθίες χαρακτήρων, η προσθήκη, αφαίρεση και ο έλεγχος ύπαρξης ενός στοιχείου έχουν σταθερό χρόνο εκτέλεσης ανεξαρτήτως του μεγέθους του συνόλου. Το γεγονός ότι κάθε τιμή είναι μοναδική σε συνδυασμό με τις διαθέσιμες εντολές για την αλληλεπίδραση μεταξύ των συνόλων όπως η ένωση ή η εύρεση της τομής τους, τα καθιστά κατάλληλα για την έκφραση σχέσεων μεταξύ αντικειμένων.

- **Ταξινομημένα σύνολα (Sorted Sets):** Η δομή αυτή μοιάζει με αυτή των συνόλων καθώς αποτελείται από μοναδικές μη επαναλαμβανόμενες τιμές με την διαφορά όμως πώς κάθε μία από αυτές συνοδεύεται από μία βαθμολογία. Πρόκειται για έναν αριθμό βάσει του οποίου γίνεται η ταξινόμηση του συνόλου κατά αύξουσα σειρά. Επιπλέον η ταξινόμηση δεν πραγματοποιείται κατά την αναζήτηση των στοιχείων του ταξινομημένου συνόλου αλλά κατά την εισαγωγή ενός νέου στοιχείου σε αυτό. Στην περίπτωση που δύο στοιχεία έχουν ακριβώς την ίδια βαθμολογία η ταξινόμηση τους γίνεται βάσει του λεξικογραφικού μήκους της τιμής του κάθε στοιχείου.

Τέλος, μία ακόμα όχι ξεχωριστή δομή δεδομένων αλλά σημαντική δυνατότητα η οποία και θα μας απασχολήσει στην συνέχεια, είναι αυτή των γεωχωρικών ευρετηρίων τα οποία περιγράφονται λεπτομερώς στο Κεφ. 2.3.

2.1.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΗΣ REDIS

Βασικό χαρακτηριστικό της Redis είναι πως τα δεδομένα αποθηκεύονται προσωρινά στην μνήμη. Εντούτοις, δεν χάνονται κατά την επανεκκίνηση του συστήματος καθώς υπάρχει δυνατότητα μόνιμης αποθήκευσης (persistence) τους στον σκληρό δίσκο με δύο διαφορετικούς τρόπους. Ο πρώτος τρόπος ονομάζεται Redis Database Backup (RDB) κατά τον οποίο αποθηκεύει snapshots των δεδομένων της στον δίσκο χωρίς την παύση της λειτουργίας της. Ως διαδικασία βασίζεται στον ορισμό τακτικών χρονικών διαστημάτων κατά τα οποία η Redis δημιουργεί ένα αντίγραφο του εαυτού της (διαδικασία fork) με σκοπό την εγγραφή των δεδομένων της στον σκληρό δίσκο σε ένα αρχείο που ονομάζεται dump.rdb. Ο δεύτερος τρόπος ονομάζεται Append Only File (AOF) και αφορά την καταγραφή κάθε εντολής εγγραφής που στέλνεται στην Redis στο αρχείο appendonly.aof. Η εκτέλεση αυτής της διαδικασίας μπορεί να πραγματοποιηθεί είτε με την καταγραφή του συνόλου των εντολών που εκτελέστηκαν ανά δευτερόλεπτο ή κατά την λήψη της κάθε εντολής που εκτελείται, χωρίς να επηρεαστεί η απόδοση της Redis καθώς πραγματοποιείται στο παρασκήνιο με την χρήση νήματος (thread). Επιπλέον, όταν το συγκεκριμένο αρχείο αποκτήσει πολύ

μεγάλο μέγεθος, τότε εκτελείται αυτόματα η διαδικασία fork για την επανεγγραφή του αρχείου με το ελάχιστο δυνατό σύνολο των ενεργειών που απαιτούνται, ώστε να δημιουργηθεί το τρέχον σύνολο δεδομένων. Κατά την επανεκκίνηση της Redis χρησιμοποιείται το αρχείο AOF για την ανακατασκευή της βάσης καθώς μας εξασφαλίζει ότι είναι το πιο πλήρες. Προκειμένου να ελαχιστοποιηθεί η πιθανότητα απώλειας δεδομένων, μπορούν να συνδυαστούν οι δύο μέθοδοι στην ίδια βάση [15. para. 4.1].

Η Redis υποστηρίζει την αντιγραφή των δεδομένων σε διάφορους διακομιστές και την μεταξύ τους συνεργασία (replication). Σύμφωνα με αυτή τη διαδικασία σε έναν κύριο κόμβο (master), μπορούν να συνδεθούν βοηθητικοί κόμβοι (slaves) όπου κάθε ένας από αυτούς θα έχει ένα ενημερωμένο αντίγραφο του συνόλου δεδομένων και θα είναι σε θέση να ικανοποιήσει το αίτημα ανάγνωσης δεδομένων από την εφαρμογή. Κατά συνέπεια επιτυγχάνεται η αύξηση της ταχύτητας ανάγνωσης, καθώς τα αιτήματα για ανάγνωση εξυπηρετούνται από τους βοηθητικούς κόμβους και ο κύριος κόμβος ασχολείται μόνο για την εγγραφή πληροφοριών. Επίσης η αποθήκευση των δεδομένων στον σκληρό δίσκο μπορεί να οριστεί να γίνεται σε έναν ή περισσότερους από τους συνδεδεμένους βοηθητικούς κόμβους παρέχοντας έτσι μείωση των εργασιών του κύριου κόμβου. Ταυτόχρονα, λόγο της ύπαρξης πολλών αντιγράφων, αυτή η μέθοδος παρέχει τη δυνατότητα κλιμάκωσης της εφαρμογής και ασφάλειας από την πιθανότητα απώλειας δεδομένων. Η παραπάνω δυνατότητα ονομάζεται Redis Sentinel (στην Redis Enterprise το API αυτού μπορεί να χρησιμοποιηθεί ως service discovery) [4], [5].

Μία ακόμα δυνατότητα που παρέχετε από τη Redis είναι αυτή του διαμοιρασμού των δεδομένων (sharding) σε πολλαπλούς κόμβους. Σε αντίθεση με την προηγούμενη περίπτωση όπου κάθε βοηθητικός κόμβος εξαρτάται από την κύρια βάση για τον συγχρονισμό των δεδομένων και διαθέτει ένα πλήρες αντίγραφο των αυτών, τώρα κάθε κόμβος είναι ανεξάρτητος και διαθέτει ένα τμήμα του συνόλου των δεδομένων. Αυτή η δυνατότητα έχει έναν βασικό περιορισμό μη επιτρέποντας την χρήση εντολών όπου συμμετέχουν πολλαπλά κλειδιά, παρέχει όμως σημαντικές βελτιώσεις όσον αφορά την απόδοση του συστήματος. Συγκεκριμένα αξιοποιώντας την συνολική μνήμη των υπολογιστών που έχουν οριστεί να συμμετέχουν στον διαμοιρασμό,

επιτρέπει την ύπαρξη μεγαλύτερων βάσεων. Αξιοποιώντας ακόμα τους πολλαπλούς πυρήνες των υπολογιστών πετυχαίνει την κλιμάκωση της υπολογιστικής ισχύος του συστήματος. Υλοποίηση αυτής της μεθόδου αποτελεί η Redis Cluster η οποία επιτρέπει τον αυτόματο διαμερισμό των δεδομένων καθώς και την δυνατότητα συνέχισης απάντησης αιτημάτων ακόμα και όταν κάποιοι κόμβοι είναι εκτός λειτουργίας [6].

2.1.3 REDIS PIPELINING

Η Redis είναι ένας διακομιστής TCP που χρησιμοποιεί το μοντέλο διακομιστή-πελάτη (εξυπηρετητή) και το ονομαζόμενο πρωτόκολλο επικοινωνίας αιτήματος / απόκρισης (Request/Response).

Αυτό σημαίνει ότι συνήθως ένα αίτημα ολοκληρώνεται με τα ακόλουθα βήματα:

- Ο πελάτης στέλνει ένα αίτημα στον διακομιστή για το οποίο δεσμεύει ένα socket, συνήθως με τρόπο αποκλεισμού, ώστε να δεχτεί στην συνέχεια την απάντηση του διακομιστή.
- Ο διακομιστής επεξεργάζεται την εντολή και στέλνει την απάντηση πίσω στον πελάτη.

Οι πελάτες και οι διακομιστές συνδέονται μέσω ενός συνδέσμου δικτύωσης. Ένας τέτοιος σύνδεσμος μπορεί να είναι πολύ γρήγορος ή πολύ αργός όταν για παράδειγμα υπάρχουν πολλά hops μεταξύ των δύο κεντρικών υπολογιστών. Όποια και αν είναι η καθυστέρηση του δικτύου, χρειάζεται χρόνος ώστε τα πακέτα να ταξιδέψουν από τον πελάτη στον διακομιστή και πάλι πίσω από τον διακομιστή στον πελάτη κατά την μεταφορά της απάντησης. Αυτός ο χρόνος ονομάζεται RTT (Round Trip Time).

Γίνεται λοιπόν εύκολα αντιληπτό πώς αυτό μπορεί να επηρεάσει τις επιδόσεις όταν ένας πελάτης πρέπει να εκτελέσει πολλά αιτήματα στη σειρά για παράδειγμα, την προσθήκη πολλών στοιχείων στην ίδια λίστα). Για παράδειγμα, εάν ο χρόνος RTT είναι 250 χιλιοστά του δευτερολέπτου, ακόμη και αν ο διακομιστής μπορεί να

επεξεργαστεί 100 χιλιάδες αιτήματα ανά δευτερόλεπτο, θα είμαστε σε θέση να επεξεργαστούμε το πολύ τέσσερα αιτήματα ανά δευτερόλεπτο για αυτόν τον πελάτη.

Ένας διακομιστής αιτήσεων / απόκρισης μπορεί να υλοποιηθεί με τέτοιο τρόπο ώστε να είναι σε θέση να επεξεργαστεί νέα αιτήματα ακόμη και αν ο πελάτης δεν έχει λάβει τις προηγούμενες απαντήσεις. Με αυτόν τον τρόπο είναι δυνατή η αποστολή πολλαπλών εντολών στο διακομιστή χωρίς να χρειάζεται να περιμένουμε μία προς μία τις απαντήσεις αλλά να τις διαβάσουμε σε ένα και μόνο τελικό βήμα.

Αυτή η δυνατότητα ονομάζεται pipelining, και είναι μια τεχνική που χρησιμοποιείται ευρέως εδώ και πολλές δεκαετίες, για παράδειγμα σε πολλές εφαρμογές του πρωτοκόλλου POP3. Η Redis υποστηρίζει αυτήν τη τεχνική, επιταχύνοντας δραματικά τη εκτέλεση πολλαπλών εντολών από το διακομιστή [7].

2.2 REDISEARCH

Η Redisearch είναι μια μηχανή αναζήτησης η οποία έχει υλοποιηθεί και λειτουργεί πάνω στη Redis υποδομή αλλά σε αντίθεση με άλλες λύσεις αναζήτησης κειμένου με χρήση της Redis, δεν χρησιμοποιεί εσωτερικές δομές δεδομένων αυτής όπως ταξινομημένα σύνολα (sorted sets). Αυτό επιτρέπει πιο προηγμένες λειτουργίες, όπως ακριβή αντιστοίχιση φράσεων και αριθμητικό - γεωγραφικό φιλτράρισμα για ερωτήματα κειμένου, τα οποία δεν είναι δυνατά ή αποτελεσματικά με τις παραδοσιακές προσεγγίσεις αναζήτησης κειμένου με την χρήση της Redis. Η RediSearch χρησιμοποιεί τόσο τους δικούς της τύπους δεδομένων όσο και τους ενσωματωμένους τύπους δεδομένων της Redis.

Όσον αφορά την ευρετηρίαση (indexing), η RediSearch υποστηρίζει κειμενικά (text), αριθμητικά (numeric) και γεωγραφικά (geo) ευρετήρια. Συγκεκριμένα για τα γεωγραφικά πεδία χρησιμοποιεί τα γεωγραφικά σύνολα (Geo sets) της Redis (Κεφ. 2.3.1.2). Επίσης, στην λογική της RediSearch, δημιουργούνται επιμέρους ευρετήρια (indexes) για τα field types που έχουμε ορίσει και το πραγματικό περιεχόμενο των

εγγράφων (documents) αποθηκεύεται σε Redis hashes εφόσον το επιθυμούμε. Επιπλέον δίνεται η δυνατότητα να μην προστεθεί κάποιο από τα πεδία σε ευρετήριο [9].

2.2.1 ΟΡΙΣΜΟΣ ΕΥΡΕΤΗΡΙΟΥ ΚΑΙ ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ

Για να πραγματοποιηθεί μία αποτελεσματική αναζήτηση, η RediSearch πρέπει να γνωρίζει πώς να ευρετηριάσει τα έγγραφα. Ένα έγγραφο μπορεί να έχει πολλά πεδία διαφορετικού τύπου.

Συνεπώς, το πρώτο βήμα είναι να δημιουργηθεί ο ορισμός (schema) του ευρετηρίου (Εντολή FT.CREATE), ο οποίος ενημερώνει τη RediSearch πώς να χειριστεί τα έγγραφα που θα προστεθούν.

Στην συνέχεια μπορούν να προστεθούν τα έγγραφα σύμφωνα με το ορισμό που έχουμε δημιουργήσει (Εντολή FT.ADD) και η RediSearch αναλαμβάνει εσωτερικά να αποθηκεύσει και να ευρετηριάσει το κάθε πεδίο με την κατάλληλη δομή δεδομένων καθώς επίσης να τα κατανέμει στο Redis Cluster.

Στα πλαίσια του NoDA για τα παραπάνω βήματα χρησιμοποιείται ο JRediSearch java client.

Εσωτερικά μέσα στο ευρετήριο, η RediSearch χρησιμοποιεί delta encoded λίστες αποτελούμενες από αριθμητικά, αυξανόμενα, αναγνωριστικών εγγράφων των 32-bit. Αυτό σημαίνει πως τα δοθέντα κλειδιά ή αναγνωριστικά που έχουν δοθεί από τον χρήστη, πρέπει να αντικατασταθούν με τα εσωτερικά αναγνωριστικά κατά την ευρετηρίαση και να επιστρέψουν στην αρχική τους μορφή κατά την αναζήτηση [8], [9].

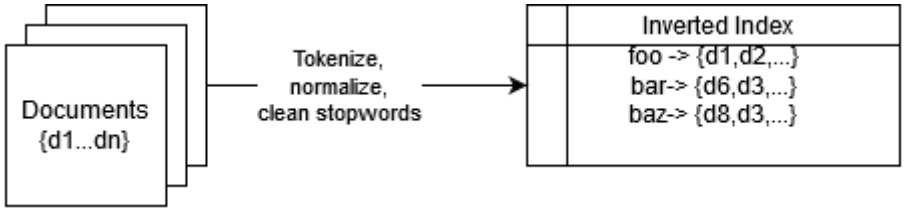
2.2.2 ΧΩΡΙΚΑ ΦΙΛΤΡΑ

Η RediSearch χρησιμοποιεί το Geo API της Redis (Κεφ. 2.3) και συγκεκριμένα την εντολή GEORADIUS για το φιλτράρισμα γεωγραφικών (Geo) δεδομένων και γίνεται ορίζοντας τα πεδία στο ευρετήριο ως GEO. Η Redis έχει πολλές εντολές που σχετίζονται με τη γεωγραφική χωροθέτηση, αλλά σε αντίθεση με άλλες εντολές αυτές οι εντολές δεν διαθέτουν τον δικό τους τύπο δεδομένων. Αυτές οι εντολές στην πραγματικότητα χρησιμοποιούν τα ταξινομημένα σύνολα (Sorted sets) ως τύπο δεδομένων (Κεφ. 2.3.1.1). Κατά την εκτέλεση του ερωτήματος, το γεωγραφικό τμήμα του ερωτήματος (ένα φίλτρο ακτίνας) αποστέλλεται στη Redis, επιστρέφοντας μόνο τα αναγνωριστικά εγγράφων που βρίσκονται εντός αυτής της ακτίνας [9].

2.2.3 ΕΥΡΕΤΗΡΙΑΣΗ ΚΕΙΜΕΝΟΥ

Το θεμελιώδες κομμάτι κάθε μηχανής αναζήτησης είναι αυτό που ονομάζεται Ανεστραμμένο Ευρετήριο (Inverted Index). Με πολύ απλούς όρους, ένα τέτοιο ευρετήριο αναζήτησης είναι ένα σύνολο συσχετίσεων μεταξύ λέξεων ή όρων προς τα αντίστοιχα έγγραφα στα οποία εμφανίζονται.

Εικόνα 1:Δημιουργία Ανεστραμμένου Ευρετηρίου



Αν λοιπόν για παράδειγμα έχουμε δύο έγγραφα, ένα με κείμενο «Hello World» και το άλλο με κείμενο «Hello Friend», μια μηχανή αναζήτησης θα δημιουργήσει ένα ανεστραμμένο ευρετήριο της λέξης «Hello» που περιέχει δύο εγγραφές, μία για καθένα από τα προαναφερθέντα έγγραφα.

Για την αναπαράσταση και μοντελοποίηση των ανεστραμμένων ευρετηρίων σε Redis χρησιμοποιούνται τύποι δεδομένων της Redis, συνήθως ταξινομημένα σύνολα, όπου το αναγνωριστικό (ID) ενός εγγράφου αποθηκεύεται ως "στοιχείο" και η συνάφεια ως "σκορ". Αυτή η προσέγγιση λειτουργεί πολύ καλά γιατί επιτρέπει στους προγραμματιστές να χρησιμοποιήσουν τη τομή και ένωση της Redis με στόχο την πραγματοποίηση αναζήτησης πολλαπλών λέξεων. Από την άλλη πλευρά έχει τους περιορισμούς της καθώς δεν υποστηρίζει την επακριβή αναζήτηση φράσεων, χρησιμοποιεί μεγάλα ποσοστά μνήμης και μπορεί να γίνει πολύ αργή σε μεγάλες τομές εγγραφών. Λόγω αυτών των περιορισμών, η RediSearch εκμεταλλεύεται μία άλλη δυνατότητα για την αποθήκευση των ευρετηρίων, τα Redis DMA (Direct Memory Access) Strings. Στην ουσία εκμεταλλεύεται τα Redis κλειδιά συμβολοσειράς παρέχοντας όμως άμεση πρόσβαση στον δείκτη της μνήμης που έχει δεσμευτεί για τα δεδομένα του κάθε κλειδιού, χωρίς να προηγείται αντιγραφή ή σειριοποίηση αυτών.

Προτού δημιουργηθεί και αποθηκευτεί ένα τέτοιο ευρετήριο με την χρήση της παραπάνω δυνατότητας, κάθε εγγραφή κωδικοποιείται με μία τεχνική η οποία συνδυάζει τα Delta Encoding[1] και Variant Encoding[2].

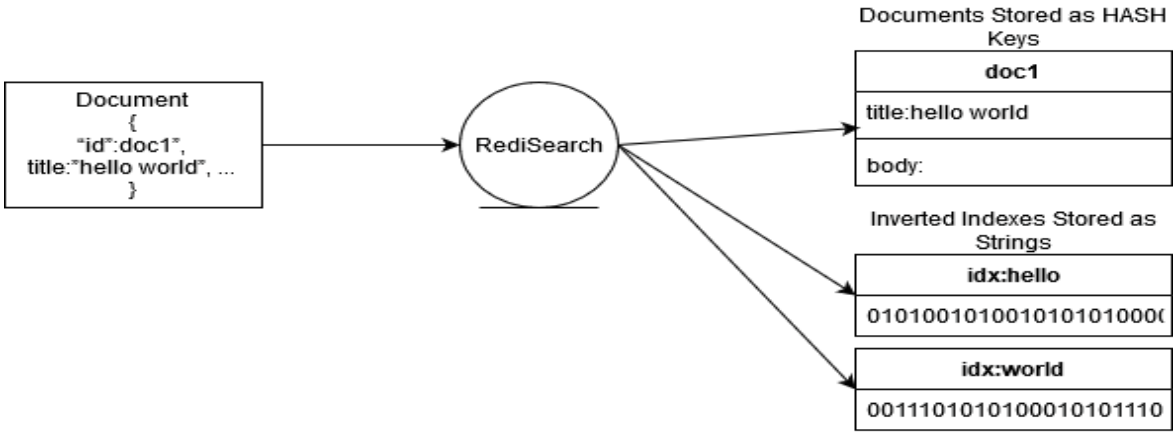
Για κάθε εγγραφή κωδικοποιείται για παράδειγμα:

- Το αναγνωριστικό εγγράφου (ως δέλτα από το προηγούμενο έγγραφο).
- Ο όρος συχνότητα, βάσει της κατάταξης του εγγράφου.
- Αναγνωριστικά που μπορούν να χρησιμοποιηθούν για φιλτράρισμα συγκεκριμένων πεδίων.

Καθώς και αρκετά άλλα μεταδεδομένα τα οποία δεν μας χρειάζονται για την παρούσα μελέτη. Στην Εικόνα 2 παρουσιάζεται με απλό τρόπο πώς πραγματοποιείται η ευρετηρίαση και αποθήκευση των εγγράφων στην RediSearch [10. pp. 4-5].

[1] https://en.wikipedia.org/wiki/Delta_encoding
[2] https://developers.google.com/protocol-buffers/docs/encoding

Εικόνα 2: Αποθήκευση περιεχομένου εγγράφου και ευρετηρίαση δεδομένων



2.3 REDIS GEO API

Η Redis διαθέτει ένα σύνολο εντολών που σχετίζονται με την διαχείριση χωρικών δεδομένων, τις λεγόμενες GEO εντολές. Οι εντολές αυτές παρέχουν δυνατότητες ευρετηρίασης των δεδομένων καθώς και επερώτησης αυτών των ευρετηρίων με εύκολο τρόπο. Οι πιο συνηθισμένες και οι οποίες θα μας απασχολήσουν στην συνέχεια είναι οι ακόλουθες [14].

- **GEOADD key longitude latitude member**: Προσθέτει ένα χωρικό αντικείμενο στο συγκεκριμένο κλειδί (**key**) το οποίο προσδιορίζεται από το γεωγραφικό μήκος (**longitude**), το γεωγραφικό πλάτος (**latitude**) και ένα όνομα (**member**).
- **GEORADIUS key longitude latitude radius m|km|ft|mi**: Επιστρέφει όλα τα αντικείμενα που βρίσκονται μέσα στα όρια του χώρου που έχει καθοριστεί από ένα σημείο ως κέντρο και την ακτίνα (**radius**) από το κέντρο αυτό.

2.3.1 ΔΟΜΕΣ ΤΗΣ REDIS ΓΙΑ ΧΩΡΙΚΑ ΔΕΔΟΜΕΝΑ

Παρακάτω περιγράφονται οι δομές δεδομένων τις Redis που χρησιμοποιούνται για την αναπαράσταση των χωρικών δεδομένων.

2.3.1.1 ΤΑΞΙΝΟΜΗΜΕΝΑ ΣΥΝΟΛΑ

Τα ταξινομημένα σύνολα της Redis είναι μια δομή δεδομένων που αποθηκεύει μια βαθμολογία για κάθε μέλος της, επιτρέποντας την ταξινόμηση και την αναζήτηση των μελών βάσει της κατάταξης και της βαθμολογίας τους. Τα μέλη του συνόλου είναι Redis Strings και οι βαθμολογίες είναι αριθμοί κινητής υποδιαστολής των 64-bit.

Η δομή δεδομένων του Ταξινομημένου Συνόλου εμφανίζει ασυμπτωτική πολυπλοκότητα χώρου O(N), πράγμα που σημαίνει ότι η κατανάλωση μνήμης είναι γραμμικά ανάλογη με τον αριθμό (και τα μεγέθη) των μελών σε αυτό. Η υπολογιστική πολυπλοκότητα των περισσότερων λειτουργιών του Ταξινομημένου Συνόλου είναι λογαριθμική. Πιο συγκεκριμένα,

- Η προσθήκη παρουσιάζει πολυπλοκότητα O(log(N)) όπου N είναι το σύνολο τον αντικειμένων του ταξινομημένου συνόλου.
- Η αφαίρεση παρουσιάζει πολυπλοκότητα O(M*log(N)) όπου N είναι το σύνολο τον αντικειμένων του ταξινομημένου συνόλου και M ο αριθμός των αντικειμένων που θα αφαιρεθούν.
- Η αναζήτηση παρουσιάζει πολυπλοκότητα O(log(N)+M) όπου N είναι το σύνολο τον αντικειμένων του ταξινομημένου συνόλου και M το πλήθος των αντικειμένων που θα επιστραφούν.

Η λογαριθμική πολυπλοκότητα σημαίνει ότι το κόστος επεξεργασίας των λειτουργιών διατηρείται σχετικά χαμηλό, ακόμη και για εξαιρετικά μεγάλα σύνολα [12, p. 6].

Πρόκειται για έναν εγγενή (native) τύπο δεδομένων της Redis ο οποίος μπορεί να θεωρηθεί ένα σύνολο μοναδικών μελών (οι επαναλήψεις δεν αποθηκεύονται) με κάθε μέλος να συνδέεται με έναν αριθμό (ονομαζόμενο σκορ) που λειτουργεί ως ένας

φυσικός μηχανισμός ταξινόμησης. Ενώ τα μέλη δεν μπορούν να επαναληφθούν, οποιοσδήποτε αριθμός μελών μπορεί να μοιραστεί την ίδια βαθμολογία καθώς επίσης αναφέρθηκε παραπάνω η προσθήκη, αφαίρεση και ανάκτηση περιοχών (κατά βαθμό ή βαθμολογία) πραγματοποιείται με σχετικά χαμηλή χρονική πολυπλοκότητα. Όλα τα παραπάνω οδηγούν στο γεγονός πώς το ταξινομημένο σύνολο είναι ένα ευρετήριο από την φύση του [11].

2.3.1.2 ΓΕΩΓΡΑΦΙΚΑ ΣΥΝΟΛΑ

Τα γεωγραφικά σύνολα είναι απλά ταξινομημένα σύνολα αλλά με διαφορετικό όνομα. Η ουσιαστική διαφορά ωστόσο, εκτείνεται στον τρόπο με τον οποίο χρησιμοποιείται η βαθμολογία. Ενώ με τα κανονικά ταξινομημένα σύνολα η βαθμολογία μπορεί να τεθεί σε οποιαδήποτε αριθμητική πτυχή των δεδομένων, στα γεωγραφικά σύνολα χρησιμοποιείται για την αποθήκευση της τοποθεσίας.

Οι τοποθεσίες όπως γνωρίζουμε περιγράφονται από ένα ζεύγος συντεταγμένων, το γεωγραφικό μήκος και το γεωγραφικό πλάτος, ενώ η βαθμολογία είναι ένας ενιαίος αριθμός. Συνεπώς, η κύρια λειτουργικότητα που παρέχεται από τα γεωγραφικά σύνολα (Geo Sets) σε σχέση με τα κανονικά ταξινομημένα σύνολα (Sorted Sets) είναι η δυναμική εκτέλεση κωδικοποίησης και αποκωδικοποίησης σε ζεύγη συντεταγμένων και αριθμητικών βαθμολογιών. Η Redis εφαρμόζει το σύστημα Geohash για τη μετάφραση/εναλλαγή μεταξύ αυτών των δύο αναπαραστάσεων [12, p. 7].

2.3.2 GEO ΕΝΤΟΛΕΣ & ΠΩΣ ΛΕΙΤΟΥΡΓΟΥΝ

Η τεχνική που χρησιμοποιείται ώστε να συμπληρωθεί το ταξινομημένο σύνολο ονομάζεται Geohash. Το Geohash είναι ένα σύστημα κωδικοποίησης του γεωγραφικού πλάτους και μήκους που εφευρέθηκε από τον Gustavo Niemeyer,

πρόκειται μια δυαδική συμβολοσειρά στην οποία κάθε χαρακτήρας υποδεικνύει τμηματικές διαιρέσεις γεωγραφικού μήκους και πλάτους του παγκόσμιου ορθογωνίου [-180,180] × [−90,90]. Η πρώτη διαίρεση χωρίζει το ορθογώνιο σε δύο τετράγωνα ([−180,0] × [−90,90] και [0,180] × [−90,90]). Τα σημεία που βρίσκονται στα αριστερά της κάθετης διαίρεσης ξεκινούν με το «0» και αυτά στο δεξί μισό ξεκινούν με το «1». Η επόμενη διαίρεση που πραγματοποιείται είναι οριζόντια, όπου τα σημεία κάτω από την οριζόντια γραμμή λαμβάνουν το «0» και αυτά που βρίσκονται από πάνω το «1». Αυτός ο διαχωρισμός συνεχίζεται έως ότου επιτευχθεί η επιθυμητή ακρίβεια [16]. Ενώ το αρχικό σύστημα Geohash χρησιμοποιεί base-32 αναπαράσταση της κωδικοποιημένης μορφής, η Redis χρησιμοποιεί μια αριθμητική αναπαράσταση αυτής ως βαθμολογία ενός μέλους στο ταξινομημένο σύνολο. Η αριθμητική υλοποίηση GeoHash της Redis προέρχεται από την Ardb[3] αλλά έχει υλοποιηθεί εξαρχής από τον Matt Stancliff[4], η οποία βασίζεται στην αναπαράσταση ενός 52 bit ακέραιου αριθμού. Η βαθμολογία ενός ταξινομημένου συνόλου μπορεί να αναπαραστήσει έναν ακέραιο 52 bit χωρίς απώλεια ακρίβειας.

Δοθέντος του κέντρο και της ακτίνας του κύκλου, η Redis πραγματοποιεί μια Geohash αναζήτηση του κελιού που περιέχει το κέντρο και των 8 κελιών που το περιβάλλουν και καλύπτουν την ακτίνα (σύνολο 9 κελιά). Πρακτικά πραγματοποιείται ερώτημα εύρους για τα όρια του κάθε κελιού. Τέλος, ελέγχεται η απόσταση από το κέντρο προς κάθε μέλος και αν είναι μικρότερη ή ίση με την ακτίνα, τότε περιλαμβάνεται στην απάντηση [12, p. 8], [13].

[3] https://github.com/yinqiwen/ardb
[4] https://matt.sh/redis-geo#_origin-story

2.4 ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ ΣΕ ΣΥΣΤΑΔΑ - REDIS CLUSTER

Σύμφωνα με το [6] στο παρόν Κεφάλαιο γίνεται αναλυτική περιγραφή της Redis Cluster. Η ανοιχτού κώδικα Redis είναι μια single-threaded διεργασία με σκοπό να παρέχει ταχύτητα και απλότητα. Μια μεμονωμένη διαδικασία Redis δεσμεύεται από τους πυρήνες του επεξεργαστή στον οποίο εκτελείται και τη διαθέσιμη μνήμη του διακομιστή.

Η ανοιχτού κώδικα Redis αλλά και το Redis Enterprise Software (RS) υποστηρίζει συστάδα βάσεων δεδομένων (database clustering) για να επιτρέπει στους πελάτες να μοιράζουν το φόρτο εργασίας μιας Redis διεργασίας σε πολλούς πυρήνες και να χρησιμοποιεί τη μνήμη RAM πολλαπλών διακομιστών. Μία συστάδα βάσεων δεδομένων (database cluster) είναι ένα σύνολο Redis διεργασιών όπου κάθε διεργασία διαχειρίζεται ένα υποσύνολο του συνολικού χώρου των κλειδιών της βάσης δεδομένων.

Σε μία συστάδα (cluster) RS, το σύνολο των κλειδιών χωρίζεται σε τμήματα της βάσης δεδομένων (database shards). Ανά πάσα στιγμή, ένα τμήμα βρίσκεται σε έναν μόνο κόμβο και διαχειρίζεται από αυτόν τον κόμβο. Κάθε κόμβος σε μία συστάδα βάσης δεδομένων Redis μπορεί να διαχειριστεί πολλαπλά τμήματα. Ο χώρος των κλειδιών στα τμήματα χωρίζεται σε θυρίδες κατακερματισμού (hash slots). Η θυρίδα ενός κλειδιού καθορίζεται από τον ετικέτα κατακερματισμού (hash) του ονόματος του κλειδιού ή μέρος αυτού.

Η RS συστάδα βάσεων δεδομένων είναι διαφανής για τον Redis πελάτη (client) που συνδέεται με τη βάση δεδομένων. Ο πελάτης Redis αποκτά πρόσβαση στη βάση δεδομένων μέσω ενός μοναδικού τελικού κόμβου (endpoint) που δρομολογεί αυτόματα όλες τις λειτουργίες στα σχετικά τμήματα. Μπορούμε να συνδέσουμε δηλαδή μια εφαρμογή σε μια διαδικασία Redis μοναδικού κόμβου ή σε μια συστάδα βάσης δεδομένων χωρίς καμία διαφορά στη λογική της εφαρμογής.

Από την άλλη πλευρά, στην ανοιχτού κώδικα έκδοση της Redis (OSS), ο πελάτης πρέπει γνωρίζει το σύνολο των κόμβων που απαρτίζουν την συστάδα και να είναι

αρκετά έξυπνος ώστε να διεκπεραιώνει την σωστή δρομολόγηση των εντολών προς του κόμβους και τα τμήματα αυτού.

2.4.1 ΠΟΤΕ ΥΠΑΡΧΕΙ ΟΦΕΛΟΣ ΑΠΟ ΤΗΝ ΤΜΗΜΑΤΟΠΟΙΗΣΗ

Η τμηματοποίηση του συνόλου κλειδιών είναι ένας αποτελεσματικός τρόπος κλιμάκωσης της Redis που πρέπει να χρησιμοποιείται όταν:

- Το σύνολο δεδομένων είναι αρκετά μεγάλο ώστε να επωφεληθεί από τη χρήση πόρων RAM περισσότερων του ενός διακομιστή. Η τμηματοποίηση του συνόλου δεδομένων προτείνεται μόλις φτάσει το μέγεθος των 25 GB.
- Οι λειτουργίες που εκτελούνται στην βάση δεδομένων είναι απαιτητικές για τον επεξεργαστή (CPU) με αποτέλεσμα την υποβάθμιση της απόδοσης. Έχοντας πολλούς επεξεργαστικούς πυρήνες να διαχειριστούν τα τμήματα της βάσης δεδομένων, το φορτίο των λειτουργιών κατανέμεται μεταξύ τους.

2.4.2 ΥΠΟΣΤΗΡΙΖΟΜΕΝΕΣ ΠΟΛΙΤΙΚΕΣ ΤΜΗΜΑΤΟΠΟΙΗΣΗΣ

Ο τρόπος με τον οποίο θα πραγματοποιηθεί η τμηματοποίηση των κλειδιών σε μια Redis βάση δεδομένων, εξαρτάται από την επιλογή της μεθόδου κατακερματισμού. Η Redis παρέχει τις παρακάτω επιλογές.

- **Κλειδιά με ετικέτα κατακερματισμού**: Η ετικέτα κατακερματισμού ενός κλειδιού είναι οποιαδήποτε συμβολοσειρά μεταξύ του «{» και του «}» στο όνομα του κλειδιού. Αυτό σημαίνει ότι όταν το όνομα ενός κλειδιού περιλαμβάνει το μοτίβο «{…}», η ετικέτα κατακερματισμού χρησιμοποιείται ως είσοδος για τη συνάρτηση κατακερματισμού. Για παράδειγμα, τα ακόλουθα ονόματα κλειδιών έχουν την ίδια ετικέτα κατακερματισμού και επομένως, αντιστοιχίζονται στην ίδια θυρίδα: foo{bar}, {bar}baz, foo{bar}baz.

- **Κλειδιά χωρίς ετικέτα κατακερματισμού**: όταν ένα κλειδί δεν περιέχει το μοτίβο «{…}», ολόκληρο το όνομα του κλειδιού χρησιμοποιείται για κατακερματισμό.

Μπορούμε να χρησιμοποιήσουμε το μοτίβο "{...}" για να κατευθύνουμε σχετικά κλειδιά στην ίδια θυρίδα κατακερματισμού, ώστε να εκτελούνται λειτουργίες πολλαπλών κλειδιών σε αυτά τα κλειδιά. Από την άλλη πλευρά, η μη χρήση ετικέτας κατακερματισμού στο όνομα του κλειδιού οδηγεί σε (στατιστικά) ομοιόμορφη κατανομή των κλειδιών στα τμήματα, εφόσον και αυτά προέρχονται από κανονική κατανομή. Εάν η εφαρμογή μας δεν εκτελεί λειτουργίες πολλαπλών κλειδιών, δεν χρειάζεται να δημιουργήσουμε ονόματα κλειδιών με ετικέτες κατακερματισμού. Επιπλέον, στην Redis Enterprise υποστηρίζονται προσαρμοσμένες πολιτικές τμηματοποίησης τις οποίες μπορεί να ορίσει ο χρήστης.

2.4.3 ΛΕΙΤΟΥΡΓΙΕΣ ΠΟΛΛΑΠΛΩΝ ΚΛΕΙΔΙΩΝ

Στις λειτουργίες σε πολλαπλά κλειδιά σε μια συσταδοποιημένη βάση δεδομένων συναντάμε τους ακόλουθους περιορισμούς:

- **Εντολές πολλαπλών κλειδιών**: Η Redis προσφέρει πολλές εντολές που δέχονται πολλαπλά κλειδιά ως ορίσματα. Σε μια βάση δεδομένων σε συστάδα, οι περισσότερες εντολές πολλαπλών κλειδιών δεν επιτρέπουν την προσπέλαση σε όλες τις θυρίδες των κλειδιών. Οι εντολές πολλαπλών κλειδιών που επιτρέπονται σε πολλαπλές θυρίδες και μόνο στην Redis Enterprise είναι: DEL, MSET, MGET, EXISTS, UNLINK, TOUCH
- **Γεωγραφικές εντολές**: Στις εντολές GEORADIUS / GEOREADIUSBYMEMBER, η επιλογή STORE και STOREDIST μπορεί να χρησιμοποιηθεί μόνο όταν όλα τα κλειδιά που συμμετέχουν βρίσκονται στην ίδια θυρίδα.

- **Συναλλαγές (Transactions)**: Όλες οι ομαδοποιημένες λειτουργίες εντός WATCH / MULTI / EXEC πρέπει να εκτελούνται για κλειδιά που έχουν αντιστοιχιστεί στην ίδια θυρίδα.
- **Lua Scripts**: Όλα τα κλειδιά που χρησιμοποιούνται από ένα σενάριο Lua πρέπει να βρίσκονται στην ίδια θυρίδα και πρέπει να παρέχονται ως ορίσματα στις εντολές EVAL / EVALSHA. Η χρήση κλειδιών σε σενάριο Lua που δεν παρασχέθηκαν ως ορίσματα ενδέχεται να παραβιάσει την έννοια τμηματοποίησης.

ΚΕΦΑΛΑΙΟ ΤΡΙΤΟ

3. API ΠΡΟΣΒΑΣΗΣ NoSQL ΔΕΔΟΜΕΝΩΝ

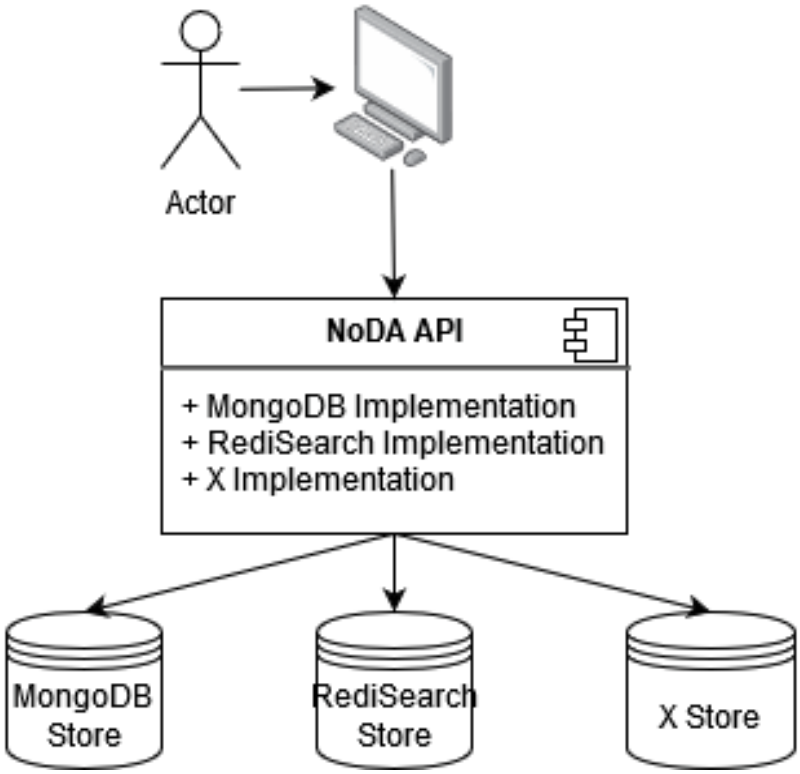
3.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ API

Όπως περιγράφεται και στο [2], η υλοποίηση του API βασίστηκε στην ιδέα της δημιουργίας μιας εργαλειοθήκης πρότυπων ερωτημάτων για NoSQL συστήματα βάσεων δεδομένων. Κύριος στόχος αυτής της ιδέας ήταν να διευκολύνει τη διαδικασία πρόσβασης στα δεδομένα που είναι αποθηκευμένα σε μια NoSQL βάση δεδομένων. Αυτό επιτυγχάνεται παρέχοντας προς το χρήστη μια φιλική διεπαφή για την έκφραση και την εκτέλεση ερωτημάτων με απλό και ενιαίο τρόπο. Επίσης, παρέχεται ένα σύνολο τελεστών με σκοπό να καταφέρουμε να ορίσουμε την λειτουργική συμπεριφορά ορισμένων πρότυπων εντολών. Όπως απεικονίζεται και στην Εικόνα 3, το API ορίζει τα υποστηριζόμενα πρότυπα ερωτήματα και τους τελεστές με τέτοιο τρόπο ώστε η υλοποίηση ανά υποστηριζόμενη βάση δεδομένων να είναι διαφανής από τους χρήστες.

Η πρώτη έκδοση του API πραγματοποιήθηκε στα πλαίσια της διπλωματικής [17]. Το API σε αυτή του την έκδοση παρέχει ένα σύνολο από ευρέως χρησιμοποιούμενα πρότυπα ερωτήματα καθώς επίσης και ένα σύνολο τελεστών για την έκφραση

σύνθετων ερωτημάτων. Επιπλέον έχουν υποστηριχτεί τα χωρικά και χωρο-χρονικά ερωτήματα. Εκτός από τους κοινούς τελεστές, όπως οι δυαδικοί (boolean) και οι συγκριτικοί που έχουν ήδη περιγραφεί, το API για τις ανάγκες της παρούσας διπλωματικής εμπλουτίζεται με τελεστές προσανατολισμένους σε κειμενικά και χωροκειμενικά δεδομένα.

Εικόνα 3: Απεικόνιση επιπέδου αφαίρεσης του NoDA



3.2 ΤΕΚΜΗΡΙΩΣΗ ΤΟΥ API

Η τεκμηρίωση του API έχει περιγραφεί πλήρως στο [17, para. 3.2] όσον αφορά το σύνολο των τελεστών που υποστηρίζει και της αρχιτεκτονικής του σχεδίασης. Στα κεφάλαια που ακολουθούν περιγράφονται οι νέοι τελεστές που προστέθηκαν στα πλαίσια της παρούσας διπλωματικής καθώς και αλλαγές που προέκυψαν στην σχεδίαση μετά την αναγνώριση αδυναμιών.

3.2.1 ΠΡΟΤΥΠΑ ΕΡΩΤΗΜΑΤΑ ΚΑΙ ΤΕΛΕΣΤΕΣ

Στον Πίνακα 1 παρουσιάζονται τα πρότυπα ερωτήματα που προστέθηκαν. Στον Πίνακα 2 ξαναπαρουσιάζονται οι Γεωγραφικοί τελεστές. Στον Πίνακα 3 και 4 παρουσιάζονται οι νέοι Κειμενικοί και Χωροκειμενικοί τελεστές αντίστοιχα. Στον Πίνακα 5 παρουσιάζονται οι νέοι τελεστές Συνάθροισης (aggregate) που υποστηρίζονται.

Πίνακας 1: Υποστηριζόμενα Πρότυπα Ερωτήματα

Πρότυπα Ερωτήματα	**Ορίσματα**	**Φάση**
groupBy	(String fieldName, String... fieldNames)	Ορισμού
printScreen	()	Εκτέλεσης
count	()	Εκτέλεσης
aggregate	(AggregateOperator aop, AggregateOperator... aops)	Ορισμού

- groupBy: Εκτελεί μια λειτουργία ομαδοποίησης σε ένα πεδίο, το όνομα του οποίου δίνεται ως το πρώτο όρισμα. Συγκεκριμένα, ομαδοποιεί τις ίδιες τιμές ενός συγκεκριμένου πεδίου σε ομάδες.
- printScreen: Εκτυπώνει τα αποτελέσματα του συνόλου των λειτουργιών στην οθόνη του χρήστη.
- count: Βρίσκει το σύνολο των αποτελεσμάτων του συνόλου των λειτουργιών.
- aggregate: Εκτελεί την λειτουργία συνάθροισης δοθέντος ορισμένων αντικειμένων τύπου AggregateOperator ως ορίσματα. Μπορεί να συνδυαστεί με το βασικό ερώτημα groupBy.

Όπως έχει οριστεί και στο [17], τα πρότυπα ερωτήματα που συσχετίζονται με τη φάση ορισμού μπορούν να χρησιμοποιηθούν σε συνδυασμό μεταξύ τους όσες φορές χρειαστεί. Είναι σχεδιαστικά νωθρά (lazy) ως προς την εκτέλεση τους που σημαίνει ότι ορίζουν μια ακολουθία λειτουργιών χωρίς να εκτελούνται στην πραγματικότητα. Αυτές οι λειτουργίες εκτελούνται όταν καλείται ένα πρωτότυπο που ανήκει στη φάση εκτέλεσης.

Πίνακας 2: Υποστηριζόμενοι Γεωγραφικοί Τελεστές

Γεωγραφικοί Τελεστές	**Ορίσματα**
inGeoCircleKm	(String fieldName, Coordinates point, double radius)
inGeoCircleMeters	(String fieldName, Coordinates point, double radius)
inGeoCircleMiles	(String fieldName, Coordinates point, double radius)
inGeoPolygon	(String fieldName, Coordinates c1, Coordinates c2, Coordinates c3, Coordinates... cs)
inGeoRectangle	(String fieldName, Coordinates lowerBoundPoint, Coordinates upperBoundPoint)
geoNearestNeighbors	(String fieldName, Coordinates point, int neighbors)

Οι γεωγραφικοί τελεστές δεν έχουν αλλάξει ως προς την λογική που εξυπηρετούν και τον τρόπο λειτουργίας τους σε σχέση με την πρώτη έκδοση. Παρακάτω περιγράφεται ξανά η λειτουργικότητα τους με στόχο την καλύτερη κατανόηση των χωροκειμενικών τελεστών.

- inGeoPolygon: Επιλέγει τις εγγραφές των οποίων η χωρική έκταση που αντιπροσωπεύεται από ένα συγκεκριμένο πεδίο (το όνομά της δίδεται ως το πρώτο όρισμα), είναι εντελώς μέσα σε ένα πολύγωνο. Το πολύγωνο ορίζεται από τα γωνιακά του σημεία (οι συντεταγμένες του περνάνε ως ορίσματα - απαιτούνται τουλάχιστον τρία).
- inGeoBox: Επιλέγει τις εγγραφές των οποίων η χωρική έκταση που αντιπροσωπεύεται από ένα συγκεκριμένο πεδίο (το όνομά της δίδεται ως το πρώτο όρισμα), είναι εξ ολοκλήρου μέσα σε ένα πλαίσιο (τετράγωνο). Το πλαίσιο καθορίζεται από τα κατώτερα και ανώτερα σημεία οριοθέτησης του (οι συντεταγμένες των οποίων περνούν ως το δεύτερο και το τρίτο όρισμα αντίστοιχα).

- inGeoCircleKm: Επιλέγει τις εγγραφές των οποίων η χωρική έκταση που αντιπροσωπεύεται από ένα συγκεκριμένο πεδίο (το όνομά του δίδεται ως το πρώτο όρισμα), είναι εντελώς εντός ενός κύκλου. Ο κύκλος ορίζεται από το σημείο του κέντρου του (οι συντεταγμένες του οποίου περνούν ως το δεύτερο όρισμα) και την ακτίνα του στη μονάδα του χιλιομέτρου (δίδεται ως το τρίτο όρισμα).
- inGeoCircleMeters: Επιλέγει τις εγγραφές των οποίων η χωρική έκταση που αντιπροσωπεύεται από ένα συγκεκριμένο πεδίο (το όνομά του δίδεται ως το πρώτο όρισμα), είναι εντελώς εντός ενός κύκλου. Ο κύκλος ορίζεται από το σημείο του κέντρου του (οι συντεταγμένες του οποίου περνούν ως το δεύτερο όρισμα) και την ακτίνα του στη μονάδα του μέτρου (δίδεται ως το τρίτο όρισμα).
- inGeoCircleMiles: Επιλέγει τις εγγραφές των οποίων η χωρική έκταση που αντιπροσωπεύεται από ένα συγκεκριμένο πεδίο (το όνομά του δίδεται ως το πρώτο όρισμα), είναι εντελώς εντός ενός κύκλου. Ο κύκλος ορίζεται από το σημείο του κέντρου του (οι συντεταγμένες του οποίου περνούν ως το δεύτερο όρισμα) και την ακτίνα του στη μονάδα των μιλίων (δίδεται ως το τρίτο όρισμα).
- nearestNeighbors: Επιλέγει έναν καθορισμένο αριθμό εγγραφών (δίδεται ως το τρίτο όρισμα) των οποίων η χωρική έκταση που αντιπροσωπεύεται από ένα συγκεκριμένο πεδίο (το όνομά του δίδεται ως το πρώτο όρισμα), είναι η πλησιέστερη σε ένα συγκεκριμένο σημείο (οι συντεταγμένες των οποίων περνούν ως δεύτερο όρισμα).

Πίνακας 3: Υποστηριζόμενοι Κειμενικοί Τελεστές

Κειμενικοί Τελεστές	**Ορίσματα**
anyKeywords	(String fieldName, String keyword, String… keywords)
allKeywords	(String fieldName, String keyword1, String keyword2, String… keywords)

Οι κειμενικοί τελεστές μπορούν να χρησιμοποιηθούν ως ορίσματα του πρότυπου ερωτήματος φίλτρου, δεδομένου ότι είναι ένας υποτύπος του τύπου FilterOperator. Η λειτουργικότητά τους περιγράφεται ως εξής.

- anyKeywords: Επιλέγει τις εγγραφές που ικανοποιούν τουλάχιστον μία από τις λέξεις κλειδιά που έχουν δοθεί ως ορίσματα (απαιτείται τουλάχιστον ένα).
- allKeywords: Εκτελεί τη λογική λειτουργία σύζευξης και επιλέγει τις εγγραφές που ικανοποιούν όλες τις λέξεις κλειδιά που έχουν δοθεί ως ορίσματα (απαιτούνται τουλάχιστον δύο).

Πίνακας 4: Υποστηριζόμενοι Χωρο- Κειμενικοί Τελεστές

Χωρο - Κειμενικοί Τελεστές	**Ορίσματα**
inGeoTextualCircleKm	(String fieldName, Coordinates point, double radius, ConditionalTextualOperator cto)
inGeoTextualCircleMeters	(String fieldName, Coordinates point, double radius, ConditionalTextualOperator cto)
inGeoTextualCircleMiles	(String fieldName, Coordinates point, double radius, ConditionalTextualOperator cto)
inGeoTextualPolygon	(String fieldName, ConditionalTextualOperator cto, Coordinates

	c1, Coordinates c2, Coordinates c3, Coordinates... cs)
inGeoTextualRectangle	(String fieldName, Coordinates lowerBoundPoint, Coordinates upperBoundPoint, ConditionalTextualOperator cto)
topRankInGeoTextualCircleKm	(String fieldName, Coordinates point, double radius, Collection<String> keywords, int topK)
topRankInGeoTextualCircleMeters	(String fieldName, Coordinates point, double radius, Collection<String> keywords, int topK)
topRankInGeoTextualCircleMiles	(String fieldName, Coordinates point, double radius, Collection<String> keywords, int topK)
topRankInGeoTextualPolygon	(String fieldName, Collection<String> keywords, int topK, Coordinates c1, Coordinates c2, Coordinates c3, Coordinates... cs)
topRankInGeoTextualRectangle	(String fieldName, Coordinates lowerBoundPoint, Coordinates upperBoundPoint, Collection<String> keywords, int topK)

Οι χωροκειμενικοί τελεστές μπορούν να χρησιμοποιηθούν ως ορίσματα του πρότυπου ερωτήματος φίλτρου, δεδομένου ότι είναι ένας υποτύπος του τύπου FilterOperator. Η λειτουργικότητά τους περιγράφεται ως εξής.

- inGeoTextualCircleKm: Εκτελεί την λογική του τελεστή inGeoCircleKm καθώς επίσης και ενός ConditionalTextualOperator (δίδεται ως τέταρτο όρισμα) και επιστρέφει το σύνολο των δεδομένων που επαληθεύουν και τους δύο τελεστές.
- inGeoTextualCircleMeters: Εκτελεί την λογική του τελεστή inGeoCircleMeters καθώς επίσης και ενός ConditionalTextualOperator (δίδεται ως τέταρτο όρισμα) και επιστρέφει το σύνολο των δεδομένων που επαληθεύουν και τους δύο τελεστές.
- inGeoTextualCircleMiles: Εκτελεί την λογική του τελεστή inGeoCircleMiles καθώς επίσης και ενός ConditionalTextualOperator (δίδεται ως τέταρτο όρισμα) και επιστρέφει το σύνολο των δεδομένων που επαληθεύουν και τους δύο τελεστές.
- inGeoTextualPolygon: Εκτελεί την λογική του τελεστή inGeoPolygon καθώς επίσης και ενός ConditionalTextualOperator (δίδεται ως δεύτερο όρισμα) και επιστρέφει το σύνολο των δεδομένων που επαληθεύουν και τους δύο τελεστές.
- inGeoTextualRectangle: Εκτελεί την λογική του τελεστή inGeoRectangle καθώς επίσης και ενός ConditionalTextualOperator (δίδεται ως τέταρτο όρισμα) και επιστρέφει το σύνολο των δεδομένων που επαληθεύουν και τους δύο τελεστές.
- topRankInGeoTextualCircleKm: Εκτελεί την λογική του τελεστή inGeoCircleKm, επίσης επιλέγει τις εγγραφές που ικανοποιούν όλες τις λέξεις κλειδιά που έχουν δοθεί ως όρισμα.
- topRankInGeoTextualCircleMeters: Εκτελεί την λογική του τελεστή inGeoCircleMeters, επίσης επιλέγει τις εγγραφές που ικανοποιούν όλες τις λέξεις κλειδιά που έχουν δοθεί ως όρισμα.
- topRankInGeoTextualCircleMiles: Εκτελεί την λογική του τελεστή inGeoCircleMiles, επίσης επιλέγει τις εγγραφές που ικανοποιούν όλες τις λέξεις κλειδιά που έχουν δοθεί ως όρισμα.

- topRankInGeoTextualPolygon: Εκτελεί την λογική του τελεστή inGeoPolygon, επίσης επιλέγει τις εγγραφές που ικανοποιούν όλες τις λέξεις κλειδιά που έχουν δοθεί ως όρισμα.
- topRankInGeoTextualRectangle: Εκτελεί την λογική του τελεστή inGeoRectangle, επίσης επιλέγει τις εγγραφές που ικανοποιούν όλες τις λέξεις κλειδιά που έχουν δοθεί ως όρισμα.

Όλοι οι topRank* τελεστές επιστρέφουν τις topk (δίδεται ως παράμετρος) συναφείς εγγραφές όπου η συνάφεια καθορίζεται με βάση κάποια βαθμολογία (score).

Πίνακας 5: Υποστηριζόμενοι Τελεστές Συνάθροισης (Aggregate)

Τελεστές Συνάθροισης	**Ορίσματα**
countDistinct	(String fieldName)
countNonNull	(String fieldName)

Οι τελεστές συνάθροισης μπορούν να χρησιμοποιηθούν ως ορίσματα του πρότυπου ερωτήματος συνάθροισης (aggregate), εκτελώντας μια λειτουργία για κάθε προκύπτουσα ομάδα. Η λειτουργικότητά τους περιγράφεται ως εξής.

- countDistinct: Υπολογίζει των αριθμό των μοναδικών τιμών μίας μεταβλητής της οποία το όνομα έχει δοθεί ως όρισμα.
- countNonNull: Υπολογίζει των αριθμό των μη κενών (non null) τιμών μίας μεταβλητής της οποία το όνομα έχει δοθεί ως όρισμα.

3.2.3 ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ ΧΩΡΙΚΩΝ ΚΑΙ ΧΩΡΟΚΕΙΜΕΝΙΚΩΝ ΕΡΩΤΗΜΑΤΩΝ

Οι χωρικοί τελεστές του Πίνακα 2 και οι χωροκειμενικοί του Πίνακα 4 μας παρέχουν πρόσβαση σε χωρικά και χωροκειμενικά δεδομένα καθώς μπορούν να χρησιμοποιηθούν για την έκφραση είτε χωρικών είτε χωροκειμενικών ερωτημάτων.

Τα παρακάτω παραδείγματα αφορούν περιπτώσεις χρήσης του API για την εκτέλεση ερωτημάτων χωρικού εύρους σε ακτίνα κύκλου, αλλά και ερωτημάτων χωροκειμενικού εύρους σε ακτίνα κύκλου.

Κώδικας 1: Λειτουργία ερωτήματος χωρικού κύκλου - Μετράει τα σημεία που απέχουν το πολύ 10 μίλια από ένα σημείο δοθέντων των συντεταγμένων του.

```
1 import gr.ds.unipi.noda.api.client.NoSqlDbSystem;
2 import gr.ds.unipi.noda.api.core.nosqldb.NoSqlDbOperators;
3 import gr.ds.unipi.noda.api.core.operators.FilterOperators;
4 import
gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators.Coordinates;
5
6 public class NoSqlApi {
7   public static void main(String args[]) {
8     public NoSqlDbSystem localhost =
NoSqlDbSystem.RediSearch().Builder().host("localhost").port(12000).build();
9     public NoSqlDbOperators tweets =localhost.operateOn("tweets");
10     int count =tweets.filter(FilterOperators.inGeoCircleMiles("location",
Coordinates.newCoordinates(42.0970023,-79.2353259),10)).count();
11     localhost.closeConnection();
12  }
13}
```

Κώδικας 2: Λειτουργία ερωτήματος χωροκειμενικού κύκλου - Μετράει τα σημεία που απέχουν το πολύ 50 χιλιόμετρα από ένα σημείο δοθέντων των συντεταγμένων του και οποιασδήποτε από τις δοθείσες λέξεις-κλειδιά που ικανοποιούνται.

```
1 import gr.ds.unipi.noda.api.client.NoSqlDbSystem;
2 import gr.ds.unipi.noda.api.core.nosqldb.NoSqlDbOperators;
3 import gr.ds.unipi.noda.api.core.operators.FilterOperators;
4 import
gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators.Coordinates;
5
6 public class NoSqlApi {
7   public static void main(String args[]) {
8     public NoSqlDbSystem localhost =
NoSqlDbSystem.RediSearch().Builder().host("localhost").port(12000).build();
9     public NoSqlDbOperators tweets =localhost.operateOn("tweets");
10     int count =tweets.filter(FilterOperators.inGeoTextualCircleKm("location",
Coordinates.newCoordinates(42.0970023,-79.2353259),50,
FilterOperators.anyKeywords("text","latest","the","apollo","great","fish"))).count();
11     localhost.closeConnection();
12  }
13}
```

Κώδικας 3: Λειτουργία ερωτήματος χωροκειμενικού κύκλου - Μετράει τα σημεία που απέχουν το πολύ 500 μέτρα από ένα σημείο δοθέντων των συντεταγμένων του και μόνο εφόσον ικανοποιούνται όλες οι λέξεις-κλειδιά που έχουν δοθεί.

```
1 import gr.ds.unipi.noda.api.client.NoSqlDbSystem;
2 import gr.ds.unipi.noda.api.core.nosqldb.NoSqlDbOperators;
3 import gr.ds.unipi.noda.api.core.operators.FilterOperators;
4 import
gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators.Coordinates;
5
6 public class NoSqlApi {
7   public static void main(String args[]) {
8     public NoSqlDbSystem localhost =
NoSqlDbSystem.RediSearch().Builder().host("localhost").port(12000).build();
9     public NoSqlDbOperators tweets =localhost.operateOn("tweets");
10     int count =tweets.filter(FilterOperators.inGeoTextualCircleMeters("location",
Coordinates.newCoordinates(42.0970023,-79.2353259),500,
FilterOperators.allKeywords("text","latest","the","apollo","great","fish"))).count();
11     localhost.closeConnection();
12  }
13}
```

3.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ API

3.3.1 MODULES

Το API στην νέα του σχεδίαση έχει υιοθετήσει την λογική των Modules[5]. Τα 2 βασικά modules που το αποτελούν είναι τα **nosql-operators-client** και **nosql-operators-core**. Το κάθε NoSQL σύστημα αποθήκευσης που υποστηρίζεται προστίθεται ως ένα νέο ξεχωριστό module, παράδειγμα **nosql-operators-XDataBase**.

- **nosql-operators-client:** Παρέχει και υλοποιεί όλες τις απαραίτητες μεθόδους για την αρχικοποίηση μίας σύνδεσης με τη εκάστοτε NoSQL βάση δεδομένων.
- **nosql-operators-core:** Αποτελεί την καρδία του API καθώς παρέχει όλες τις απαραίτητες διεπαφές, κλάσεις και σταθερές για την επέκταση του και υποστήριξη των επιμέρους NoSQL βάσεων δεδομένων, διατηρώντας την ομοιογένεια αυτού.

3.3.2 ΠΑΚΕΤΑ ΚΑΙ ΚΛΑΣΕΙΣ JAVA

Παρακάτω παρουσιάζεται πώς οργανώνεται η νέα δομή σε πακέτα και στις αντίστοιχες κλάσεις που τα αποτελούν.

- gr.ds.unipi.noda.api.client
    - **NoSqlDbSystem** abstract class

[5] https://en.wikipedia.org/wiki/Modular_programming

Αυτή η κλάση έχει υλοποιηθεί με το builder design pattern, καθώς χρησιμοποιείται απευθείας από τους χρήστες του API με σκοπό τον καθορισμό των στοιχείων της NoSQL βάσης δεδομένων, όπως την IP, την θύρα, καθώς επίσης και του ονόματος του πίνακα - δομής στην οποία θα πραγματοποιηθεί η εκτέλεση λειτουργιών ερωτημάτων. Επεκτείνεται και υλοποιείται από τις XSystem κλάσεις όπου X είναι το όνομα της NoSQL βάσης δεδομένων που υποστηρίζεται.

- gr.ds.unipi.noda.api.core.constants
  - **AggregationKeywords** enum

    Ο συγκεκριμένος enumerator ορίζει τα προκαθορισμένα ψευδώνυμα (alias) που χρησιμοποιούνται από τους υλοποιημένους τελεστές συνάθροισης των modules του κάθε NoSQL συστήματος που υποστηρίζεται.

  - **Commons** class

    Αυτή κλάση αρχικοποιεί και προσφέρει στατικές μεταβλητές που ισχύουν για όλα τα επιμέρους modules του κάθε NoSQL συστήματος που υποστηρίζεται.

  - **StringPool** class

    Αυτή η κλάση αρχικοποιεί και προσφέρει ένα σύνολο από συχνά χρησιμοποιούμενες συμβολοσειρές (Strings) για την καθολική τους χρήση από τα επιμέρους modules του κάθε NoSQL συστήματος που υποστηρίζεται.

- gr.ds.unipi.noda.api.core.nosqldb
  - **NoSqlConnectionFactory** abstract class

    Αυτή η αφηρημένη κλάση αναπαριστά τον γεννήτορα των πρότυπων ερωτημάτων και αρχικοποίησης των τελεστών. Επίσης ορίζει τις μεθόδους για τον τερματισμό τον συνδέσεων προς την βάση δεδομένων. Επεκτείνεται και υλοποιείται από τις XConnectionFactory

κλάσεις όπου Χ είναι το όνομα της NoSQL βάσης δεδομένων που υποστηρίζεται.

- **NoSqlConnectionManager** abstract class

  Αυτή η κλάση αναπαριστά τον διαχειριστή συνδέσεων μιας NoSQL βάσης δεδομένων, ορίζοντας αφηρημένες μεθόδους που πρέπει να υλοποιηθούν από έναν διαχειριστή συνδέσεων ενός συγκεκριμένου NoSQL συστήματος. Δέχεται ως Java Generic παράμετρο τον τύπο αντικειμένου που αντιπροσωπεύει τη σύνδεση με μια συγκεκριμένη βάση δεδομένων NoSQL με σκοπό την αποθήκευσή της σε ένα HashMap. Υπάρχουν ορισμένες μέθοδοι που έχουν ήδη υλοποιηθεί σε αυτήν την κλάση όπως η getConnection καθώς η λειτουργικότητά τους είναι κοινή για όλα τα NoSQL συστήματα. Οι ConnectionManager κλάσεις που επεκτείνουν την αφηρημένη κλάση ονομάζονται XConnectionManager όπου Χ είναι το όνομα της NoSQL βάσης δεδομένων.

- **NoSqlDbConnector** interface

  Η συγκεκριμένη διεπαφή αναπαριστά τον connector της NoSQL βάσης δεδομένων, ορίζει τις απαραίτητες μεθόδους που πρέπει να υλοποιηθούν από τον connector ενός συγκεκριμένου NoSQL συστήματος βάσης δεδομένων. Η υλοποίηση αυτής έχει όλες τις πληροφορίες και τις λογικές που χρειάζονται για να πραγματοποιηθεί μία σύνδεση στην βάση δεδομένων. Οι connector κλάσεις που υλοποιούν τη διεπαφή ονομάζονται XConnector όπου Χ είναι το όνομα της NoSQL βάσης δεδομένων.

- **NoSqlDbOperators** abstract class

  Αυτή η κλάση ορίζει όλα τα πρότυπα ερωτήματα που προσφέρονται από το API. Πρέπει να υλοποιείται από κλάσεις XOperators όπου το Χ είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τα

πρότυπα ερωτήματα για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators
    - **Operator** interface

      Η συγκεκριμένη διεπαφή παρέχει μια αφηρημένη μέθοδο για κάθε NoSQL σύστημα που υποστηρίζεται από το API. Συγκεκριμένα, επιστρέφει έναν τύπο αντικειμένου ο οποίος μπορεί να αρχικοποιηθεί από την κάθε μια συγκεκριμένη NoSQL βάση δεδομένων, εκφράζοντας έναν τελεστή (operator). Η διεπαφή επεκτείνεται από τη διεπαφή FilterOperator και υλοποιείται από τις αφηρημένες κλάσεις AggregateOperator και SortOperator.

    - **AggregateOperators** class

      Αυτή η κλάση περιέχει στατικές μεθόδους, οι οποίες προσφέρουν στους χρήστες του API όλες τις αρχικοποιήσεις των τελεστών που είναι υποτύποι του AggregateOperator τύπου.

    - **FilterOperators** class

      Αυτή η κλάση περιέχει στατικές μεθόδους, οι οποίες προσφέρουν στους χρήστες του API όλες τις αρχικοποιήσεις των τελεστών που είναι υποτύποι του FilterOperator τύπου.

    - **SortOperators** class

      Αυτή η κλάση περιέχει στατικές μεθόδους, οι οποίες προσφέρουν στους χρήστες του API όλες τις αρχικοποιήσεις των τελεστών που είναι υποτύποι του SortOperator τύπου.

- gr.ds.unipi.noda.api.core.operators.aggregateOperators
    - **AggregateOperator** abstract class

      Αυτή η κλάση ορίζει τον τύπο AggregateOperator για τους τελεστές Aggregate (συνάθροισης). Επεκτείνεται από όλες τις αφηρημένες

AggregateOperator κλάσεις των υποστηριζόμενων NoSQL modules. Περιλαμβάνει δύο μεταβλητές των οποίων οι τύποι είναι String, στις οποίες αποθηκεύονται το όνομα του πεδίου και ένα ψευδώνυμο (alias). Όλες οι κλάσεις που επεκτείνουν την συγκριμένη, εφαρμόζουν ένα συγκεκριμένο τελεστή συνάθροισης στο όνομα του πεδίου, προβάλλοντας το αποτέλεσμα αυτού με το δοθέν ψευδώνυμο.

- **BaseAggregateOperatorFactory** abstract class

  Αυτή η κλάση ορίζει όλους τους τελεστές συνάθροισης που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XAggregateOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των τελεστών συνάθροισης για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.sortOperators
  - **SortOperator** abstract class

    Αυτή η κλάση ορίζει τον τύπο SortOperator για τους τελεστές Ταξινόμησης. Επεκτείνεται από όλες τις αφηρημένες SortOperator κλάσεις των υποστηριζόμενων NoSQL modules. Περιέχει μία μεταβλητή τύπου String, αποθηκεύοντας το όνομα του πεδίου στο οποίο θα γίνει η ταξινόμηση.

  - **BaseSortOperatorFactory**abstract class

    Αυτή η κλάση ορίζει όλους τους τελεστές ταξινόμησης που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XSortOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των τελεστών ταξινόμησης για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.filterOperators
    - **FilterOperator** interface

      Αυτή η διεπαφή καθορίζει τον τύπο FilterOperator για τους τελεστές φίλτρου. Υλοποιείται από τις αφηρημένες κλάσεις ComparisonOperator, GeographicalOperator, GeoTextualOperator, TextualOperator και LogicalOperator

- gr.ds.unipi.noda.api.core.operators.filterOperators.comparisonOperators
    - **ComparisonOperator** abstract class

      Αυτή η κλάση ορίζει τον τύπο ComparisonOperator για τους τελεστές σύγκρισης. Επεκτείνεται από όλες τις αφηρημένες ComparisonOperator κλάσεις των υποστηριζόμενων NoSQL modules. Περιέχει δύο μεταβλητές τύπου String και generic (ο generic τύπος δίνεται ως παράμετρος), αποθηκεύοντας το όνομα του πεδίου και την τιμή αυτού αντίστοιχα. Αυτές οι μεταβλητές αρχικοποιούνται από τις τελικές κλάσεις που το υλοποιούν προκειμένου να εφαρμοστεί ένας συγκεκριμένος τύπος τελεστή σύγκρισης.

    - **BaseComparisonOperatorFactory** abstract class

      Αυτή η κλάση ορίζει όλους τους τελεστές σύγκρισης που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XComparisonOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των τελεστών σύγκρισης για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators
    - **Coordinates**

Αυτή η κλάση αντιπροσωπεύει τις συντεταγμένες ενός σημείου. Περιέχει δύο μεταβλητές τύπου double, στις οποίες αποθηκεύονται οι τιμές γεωγραφικού μήκους και πλάτους.

- gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators.geometries
  - **Geometry** abstract class

    Αυτή η αφηρημένη κλάση ορίζει τον γενικό τύπο μιας γεωμετρίας. Περιέχει μία μεταβλητή, τύπου πίνακα από Coordinates, όπου αποθηκεύονται οι συντεταγμένες της κάθε γεωμετρίας που την επεκτείνει. Επεκτείνεται από τις κλάσεις Point, Circle, Polygon και Rectangle.

  - **Point** class

    Αυτή η κλάση αναπαριστά την γεωμετρία ενός σημείου αποθηκεύοντας τις συντεταγμένες του σημείου αυτού.

  - **Circle** class

    Αυτή η κλάση αναπαριστά την γεωμετρία ενός κύκλου αποθηκεύοντας τις συντεταγμένες του κέντρου αυτού. Περιέχει μία μεταβλητή τύπου double στην οποία αποθηκεύεται η ακτίνα του κύκλου.

  - **Polygon** class

    Αυτή η κλάση αναπαριστά την γεωμετρία ενός πολυγώνου αποθηκεύοντας τις συντεταγμένες των κορυφών αυτού (τουλάχιστον 3).

  - **Rectangle** class

    Αυτή η κλάση αναπαριστά την γεωμετρία ενός παραλληλογράμμου αποθηκεύοντας τις συντεταγμένες του ελάχιστου και μέγιστου ορίου που το αντιπροσωπεύουν.

- gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators.geographicalOperators
    - **GeographicalOperator** abstract class

      Αυτή η κλάση ορίζει τον τύπο GeographicalOperator για τους γεωγραφικούς τελεστές. Επεκτείνεται από όλες τις αφηρημένες GeographicalOperator κλάσεις των υποστηριζόμενων NoSQL modules. Περιέχει δύο μεταβλητές τύπου String και generic η οποία επεκτείνει τον τύπο Geometry (ο generic τύπος δίνεται ως παράμετρος), αποθηκεύοντας το όνομα του πεδίου και την γεωμετρία που αφορά τον εκάστοτε τελεστή. Αυτές οι μεταβλητές αρχικοποιούνται από τις τελικές κλάσεις που το υλοποιούν προκειμένου να εφαρμοστεί ένας συγκεκριμένος τύπος γεωγραφικού τελεστή.

    - **BaseGeographicalOperatorFactory** abstract class

      Αυτή η κλάση ορίζει όλους τους γεωγραφικούς τελεστές που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XGeographicalOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των γεωγραφικών τελεστών για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.filterOperators.geoperators.geoTextualOperators
    - **GeoTextualOperator** abstract class

      Αυτή η κλάση ορίζει τον τύπο GeoTextualOperator για τους χωροκειμενικούς τελεστές. Επεκτείνεται από τις αφηρημένες κλάσεις GeoTextualConstraintlOperator. Περιέχει μια μεταβλητή τύπου GeographicalOperator, στην οποία αποθηκεύεται ο γεωγραφικός τελεστής για τον οποίο υλοποιείται.

- **GeoTextualConstraintOperator** abstract class

  Αυτή η κλάση ορίζει τον τύπο GeoTextualConstraintOperator για τους χωροκειμενικούς τελεστές. Επεκτείνεται από όλες τις αφηρημένες GeoTextualConstraintOperator κλάσεις των υποστηριζόμενων NoSql modules. Περιέχει μια μεταβλητή τύπου ConditionalTextualOperator, στην οποία αποθηκεύεται ο κειμενικός τελεστής για τον οποίο υλοποιείται και μία τύπου GeographicalOperator όπου αποθηκεύεται ο χωρικός τελεστής. Αυτές οι μεταβλητές αρχικοποιούνται από τις τελικές κλάσεις που το υλοποιούν προκειμένου να εφαρμοστεί ένας συγκεκριμένος τύπος χωροκειμενικού τελεστή.

- **BaseGeoTextualOperatorFactory** abstract class

  Αυτή η κλάση ορίζει όλους τους χωροκειμενικούς τελεστές που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XGeoTextualOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των χωροκειμενικών τελεστών για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.filterOperators.logicalOperators
  - **LogicalOperator** abstract class

    Αυτή η κλάση ορίζει τον τύπο LogicalOperator για τους λογικούς τελεστές. Επεκτείνεται από όλες τις αφηρημένες LogicalOperator κλάσεις των υποστηριζόμενων NoSQL modules. Περιέχει μια μεταβλητή της οποίας ο τύπος είναι πίνακας FilterOperator, στην οποία αποθηκεύονται αντικείμενα τύπου FilterOperator. Αυτό επιτρέπει τις τελικές κλάσεις που το υλοποιούν να εφαρμόζουν έναν συγκεκριμένο λογικό τύπο πάνω σε πολλούς τελεστές φίλτρου.

  - **BaseLogicalOperatorFactory** abstract class

Αυτή η κλάση ορίζει όλους τους λογικούς τελεστές που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XLogicalOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των λογικών τελεστών για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.filterOperators.textualOperators
  - **TextualOperator** abstract class

    Αυτή η κλάση ορίζει τον τύπο TextualOperator για τους κειμενικούς τελεστές. Επεκτείνεται από τις αφηρημένες κλάσεις ConditionalTextualOperator. Περιέχει δύο μεταβλητές τύπου String και πίνακα από String, αποθηκεύοντας το όνομα του πεδίου και τις λέξεις κλειδιά. Αυτές οι μεταβλητές αρχικοποιούνται από τις τελικές κλάσεις που το υλοποιούν προκειμένου να εφαρμοστεί ένας συγκεκριμένος τύπος κειμενικού τελεστή.

  - **BaseTextualOperatorFactory** abstract class

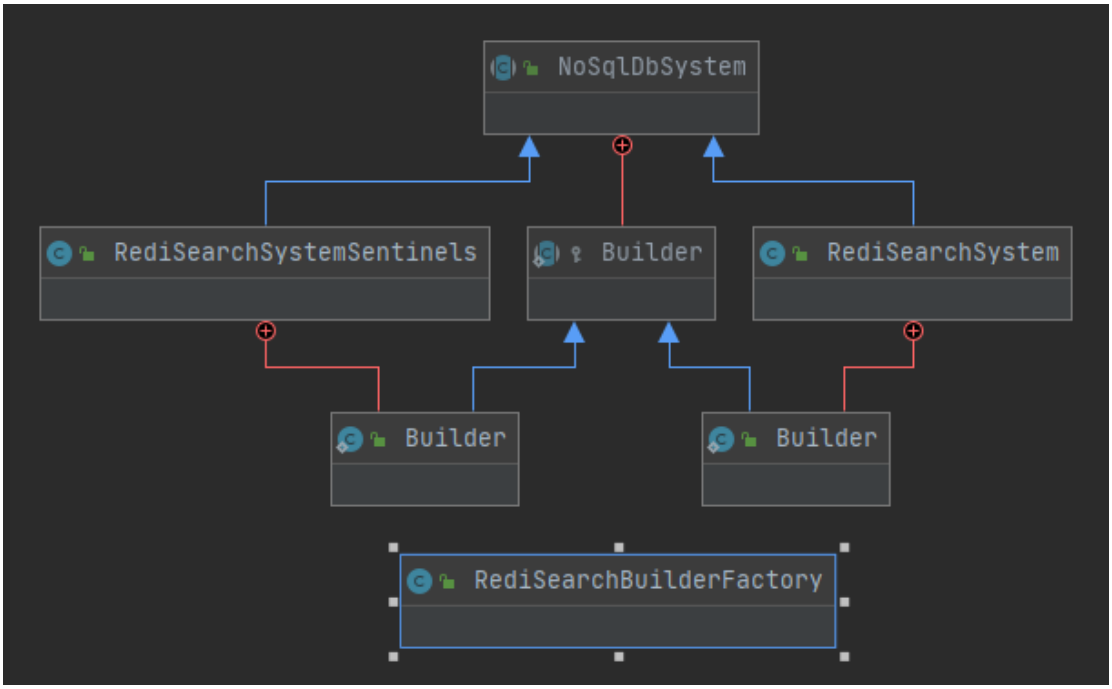
    Αυτή η κλάση ορίζει όλους τους κειμενικούς τελεστές που προσφέρονται από το API. Πρέπει να επεκτείνεται και να υλοποιείται από τις κλάσεις XTextualOperatorFactory όπου το X είναι το όνομα μιας NoSQL βάσης δεδομένων όπως της Redis - RediSearch. Σκοπός αυτών των κλάσεων είναι να υλοποιήσουν τους γεννήτορες των κειμενικών τελεστών για το κάθε συγκεκριμένο NoSQL σύστημα βάσης δεδομένων.

- gr.ds.unipi.noda.api.core.operators.filterOperators.textualOperators.conditionalTextualOperators
  - **ConditionalTextualOperator** abstract class

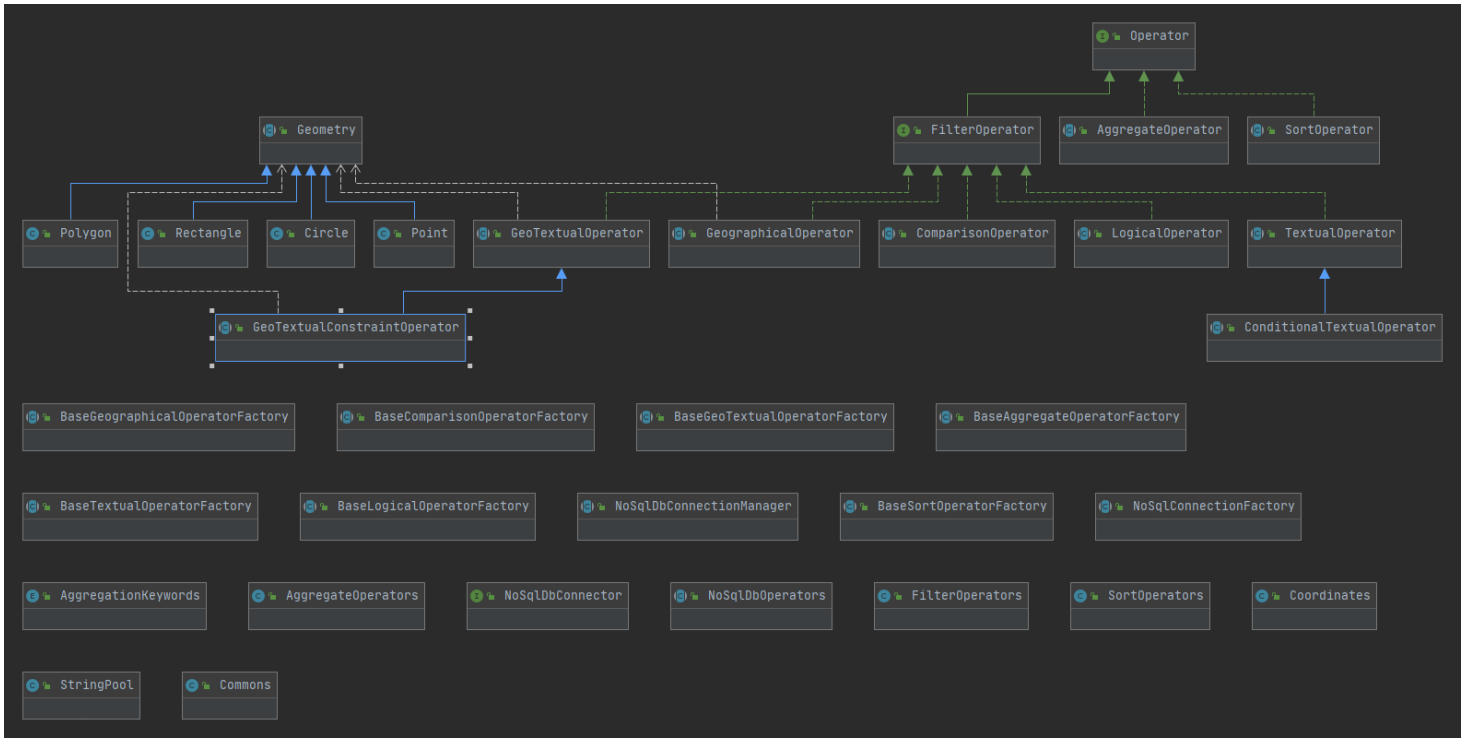
Αυτή η κλάση ορίζει τον τύπο ConditionalTextualOperator για τους κειμενικούς τελεστές. Επεκτείνει την αφηρημένη κλάση TextualOperator και

επεκτείνεται από όλες τις αφηρημένες ConditionalTextualOperator κλάσεις των υποστηριζόμενων NoSql modules. Σκοπός αυτής είναι η ανάπτυξη κειμενικών τελεστών που υποστηρίζουν ακριβής αντιστοίχιση φράσης-λέξης.

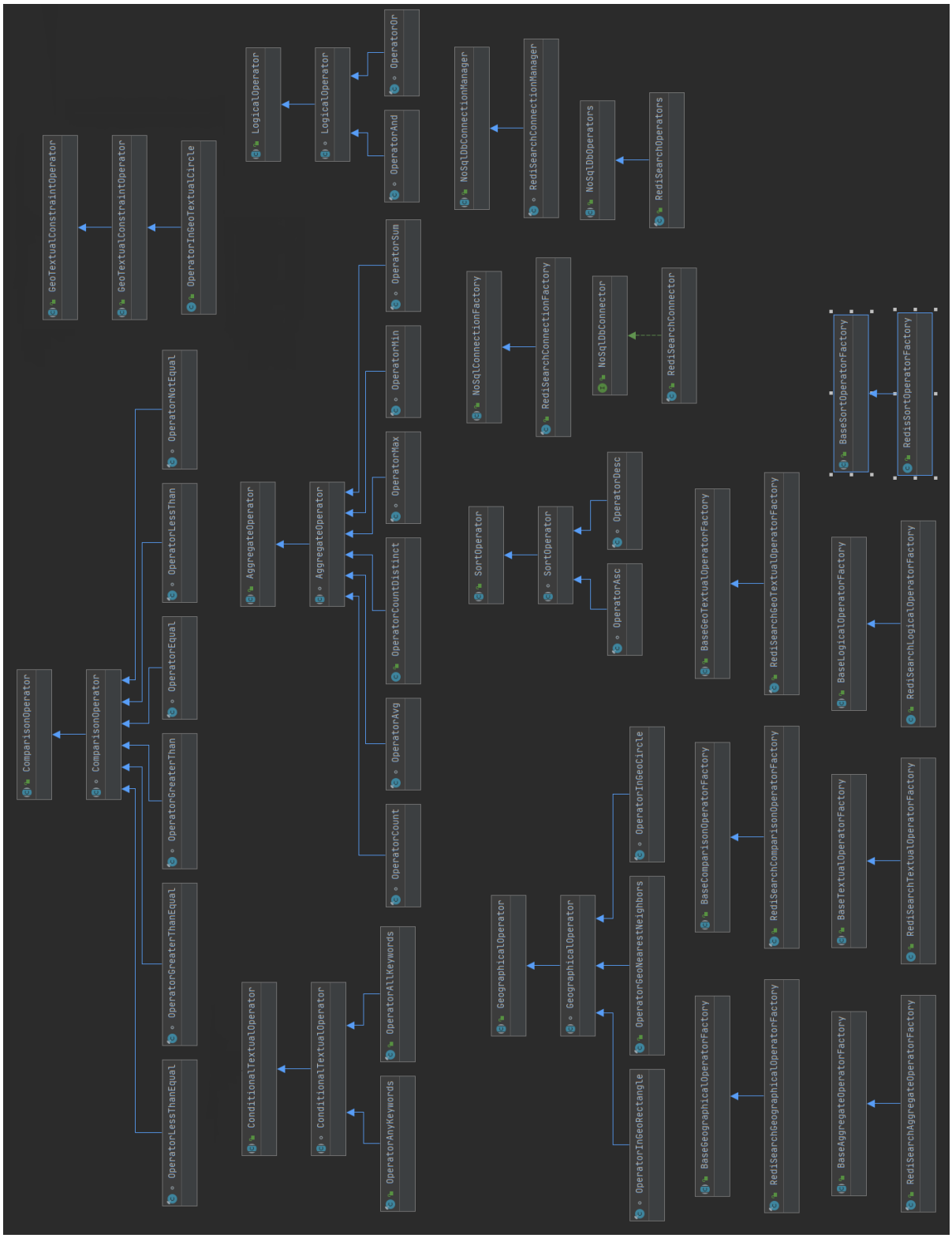
Εικόνα 4: Διάγραμμα κλάσεων του module nosql-operators-client.



Εικόνα 5: Διάγραμμα κλάσεων του module nosql-operators-core.



Εικόνα 6: Διάγραμμα κλάσεων του module nosql-operators-redisearch.



3.4 ΥΠΟΣΤΗΡΙΞΗ NOSQL ΣΥΣΤΗΜΑΤΟΣ ΚΑΙ ΕΠΕΚΤΑΣΗ ΤΟΥ API

Για την υποστήριξη ενός νέου συστήματος βάσης δεδομένων NoSQL έστω με όνομα X με στόχο την επέκταση του API, θα πρέπει αρχικά να δημιουργηθεί ένα νέο module με όνομα της μορφής nosql-operators-X το οποίο θα έχει ως υποχρεωτικό dependency το module nosql-operators-core. Για λόγους ομοιομορφίας το νέο module θα πρέπει να έχει ένα πακέτο gr.ds.unipi.noda.api.X του οποίου τα υποπακέτα θα πρέπει να ακολουθήσουν την ιεραρχία του gr.ds.unipi.noda.api.core.operators πακέτου. Στην συνέχεια θα πρέπει να δημιουργηθούν οι παρακάτω κλάσεις στα πακέτα τους, σύμφωνα με την δομή του nosql-operators-core αντίστοιχα.

- XOperators
- XConnector
- XConnectionManager
- XConnectionFactory

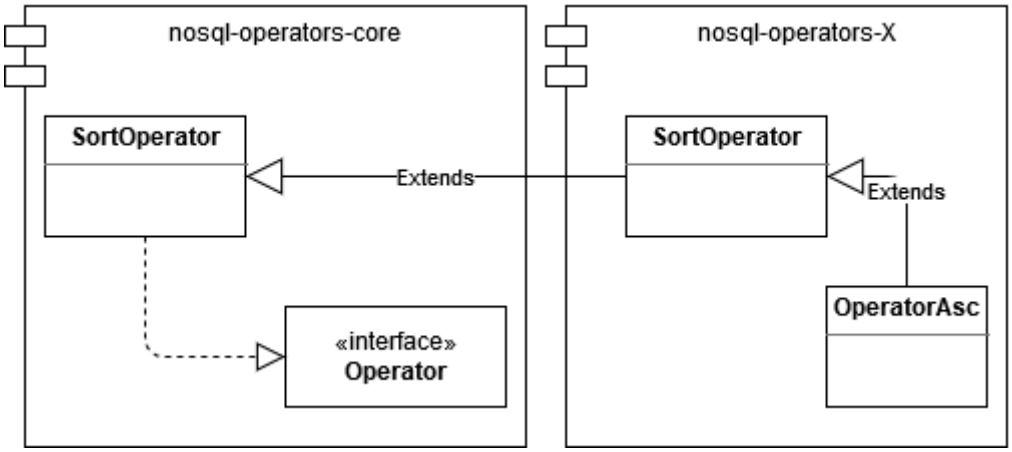
Αυτές οι κλάσεις θα πρέπει να υλοποιούν τις αφηρημένες μεθόδους των κλάσεων γονέων που επεκτείνουν. Τα ορίσματα γενικού τύπου (generic) των κλάσεων XConnector και XConnectionManager πρέπει να είναι ένας τύπος αντικειμένου μέσω του οποίου μπορεί να πραγματοποιηθεί η διαχείριση της σύνδεσης με τη βάση δεδομένων. Αυτός ο τύπος θα πρέπει να προσφέρεται από το εγγενές API της βιβλιοθήκης πελάτη της βάσης δεδομένων που υλοποιείται.

Η κλάση XConnector πρέπει να υλοποιεί τις μεθόδους createConnection, hashCode και equals που χρησιμοποιούνται στο NoSqlConnectionManager για τη δημιουργία και τη αποθήκευση της σύνδεσης σε ένα HashMap. Αυτή η κλάση πρέπει να περιέχει μεταβλητές οι οποίες χρησιμοποιούνται μόνο για τη δημιουργία σύνδεσης μέσω του εγγενούς API της βάσης δεδομένων. Η λειτουργικότητα της μεθόδου equals πρέπει να βασίζεται στις μεταβλητές που έχουν οριστεί. Η κλάση XConnectionManager θα πρέπει να υλοποιεί τις μεθόδους closeConnection και closeConnections όπου χρησιμοποιούνται για το κλείσιμο ανοιχτών συνδέσεων μέσω του αντικειμένου του

οποίου ο τύπος ορίζεται από το γενικό όρισμα. Οι κλάσεις XConnectionManager και XConnector χρησιμοποιούνται στην κλάση XOperators (το αντικείμενο τύπου XConnector περνά ως όρισμα μέσω του κατασκευαστή). Η κλάση XOperators υλοποιεί όλες τις πρότυπες μεθόδους ερωτήματος που ορίζονται στη διεπαφή NoSqlDbOperators [17].

Για να αποκτήσουμε την έκφραση ενός τελεστή σε έναν τύπο αντικειμένου Y το οποίο υποθέτουμε ότι μπορεί να χρησιμοποιηθεί από τη βάση δεδομένων NoSQL για την εκτέλεση λειτουργιών, θα πρέπει να ακολουθηθεί η παρακάτω διαδικασία. Για κάθε σύστημα X θα πρέπει να υλοποιούνται οι XOperator αφηρημένες κλάσεις η οποίες επεκτείνουν τις αντίστοιχες του nosql-operators-core module οι οποίες πρέπει να υλοποιούν την getOperatorExpression() αλλά επίσης διευκολύνουν σαν ενδιάμεσο βήμα για την δήλωση πρόσθετων μεταβλητών οι μεθόδων που μπορεί να είναι χρήσιμες για το συγκεκριμένο σύστημα (Κώδικας 4). Στην συνέχεια αυτές οι αφηρημένες κλάσεις επεκτείνονται από την τελική κλάση του κάθε τελεστή που θέλουμε να αναπαραστήσουμε (Κώδικας 5). Στην παρακάτω εικόνα φαίνεται η δομή αυτής της λογικής.

Εικόνα 7: Λογική ανάπτυξης και επέκτασης Τελεστή του API.



Επιπλέον, θα πρέπει να γίνουν και κάποιες ενέργειες στο nosql-operators-client module οι οποίες περιγράφονται παρακάτω.

- Δημιουργία πακέτου gr.ds.unipi.noda.api.client.X
    - Δημιουργία της XSystem κλάσης σκοπός της οποίας είναι να οριστούν σύμφωνα με τον builder σχεδιασμό, συγκεκριμένες μεταβλητές και ορίσματα που μπορεί να χρειάζονται από την X βάση δεδομένων για την υποστήριξη διαφόρων δυνατοτήτων σύνδεσης με αυτή. Αυτή η κλάση θα πρέπει να περιέχει έναν XConnector τον οποίο θα πρέπει και να αρχικοποιεί.
    - Δημιουργία της κλάσης XBuilderFactory (μπορεί να είναι πολλές αναλόγως το εγγενές API). Αυτή η κλάση πρέπει να περιέχει στατικές μεθόδους, οι οποίες προσφέρουν στους χρήστες του API όλες τις αρχικοποιήσεις του Builder που έχει οριστεί στο XSystem.

Στην Εικόνα 6 παρουσιάζεται το διάγραμμα κλάσεων της Redisearch υλοποίησης σύμφωνα με τις παραπάνω οδηγίες.

Κώδικας 4: Παράδειγμα υλοποίησης Redisearch της αφηρημένης κλάσης SortOperator.

```
1 import io.redisearch.aggregation.SortedField;
2
3 abstract class SortOperator extends
gr.ds.unipi.noda.api.core.operators.sortOperators.SortOperator<SortedField> {
4    SortOperator(String fieldName) {
5        super(fieldName);
6    }
7    protected abstract SortedField getOperatorField();
8    @Override
9    public SortedField getOperatorExpression() {
10        return getOperatorField();
11    }
12 }
```

Κώδικας 5: Παράδειγμα υλοποίησης Redisearch της τελικής κλάσης OperatorAsc.

```
1 import gr.ds.unipi.noda.api.core.constants.StringPool;
2 import io.redisearch.aggregation.SortedField;
3
4 final class OperatorAsc extends SortOperator {
5   private OperatorAsc(String fieldName) {
6     super(fieldName);
7   }
8   @Override
9   protected SortedField getOperatorField() {
10     return SortedField.asc(StringPool.AT.concat(getFieldName()));
11   }
12   static OperatorAsc newOperatorAsc(String fieldName) {
13     return new OperatorAsc(fieldName);
14   }
15 }
```

ΚΕΦΑΛΑΙΟ ΤΕΤΑΡΤΟ

4. ΕΥΡΕΤΗΡΙΑΣΗ ΧΩΡΟ - ΚΕΙΜΕΝΙΚΩΝ ΔΕΔΟΜΕΝΩΝ ΣΕ REDIS

Έχοντας ως πρωταρχικό μας στόχο την όσο το δυνατόν γρηγορότερη εκτέλεση των ερωτημάτων άλλα και την κλιμάκωση των δεδομένων στις περιπτώσεις μεγάλου όγκου αυτών, διατηρώντας τους χρόνους σε ικανοποιητικά επίπεδα, λαμβάνοντας επίσης υπόψιν μας όλες τις παραπάνω δυνατότητες και περιορισμούς της Redis, καταλήξαμε στην λύση που θα περιγραφεί στην συνέχεια η οποία αφορά μία textual first προσέγγιση. Αυτό σημαίνει πώς προσπαθούμε να σχεδιάσουμε μια κατάλληλη δομή ευρετηρίου στην Redis όπου κατά την εκτέλεση ενός ερωτήματος έχουμε ως προτεραιότητα την επαλήθευση των λέξεων κλειδιών που χαρακτηρίζουν ένα αντικείμενο και στην συνέχεια την εφαρμογή του χωρικού φίλτρου.

Για την υλοποίηση χωροκειμενικού ευρετηρίου στην Redis χρησιμοποιήθηκε το GEO API (Κεφ. 2.3) αυτής. Για να πετύχουμε τα παραπάνω, δοθέντος ενός συνόλου δεδομένων θα δημιουργηθούν κλειδιά για κάθε λέξη που υπάρχει στο σύνολο δεδομένων μας. Σε κάθε κλειδί θα προστεθούν τα ids του εγγράφου στο οποίο εμφανίζεται καθώς επίσης και οι συντεταγμένες του όπως ορίζει το GEO API.

Για παράδειγμα, δοθέντος ενός συνόλου δεδομένων **D** => I1[id1, (x1,y1), {k1,k2,k3}], I2[id2, (x2,y2), {k4,k1,k5}] κάθε χωροκειμενικό αντικείμενο αντιπροσωπεύεται από μία τριπλέτα I(*id, loc, doc*), όπου *id* τα αναγνωριστικά των εγγραφών, *loc* οι συντεταγμένες ενός γεωγραφικού σημείου και *doc* οι λέξεις κλειδιά που το χαρακτηρίζουν.
Θα παραχθεί η ακόλουθη δομή δεδομένων. Στην ουσία παράγεται ένα ανεστραμμένο ευρετήριο με την χρήση ενός Redis Geo Set. Κλειδί της δομής είναι το *ki* το οποίο αντιπροσωπεύει την κάθε λέξη-κλειδί, η οποία και αντιστοιχίζεται στις συντεταγμένες και το αναγνωριστικό του κάθε αντικειμένου στο οποίο εμφανίζεται.

Παραγόμενη Δομή Ευρετηρίου		
Κλειδί	**Συντεταγμένες**	**Id**
k1	(x1,y1)	id1
k1	(x2,y2)	id2
k2	(x1,y1)	id1
k3	(x1,y1)	id1
k4	(x2,y2)	id2
k5	(x2,y2)	id2

Τα ερωτήματα τα οποία μελετάμε και θα υποστηριχτούν είναι τα εξής:

1. **BooleanAnd Range Query** Το συγκεκριμένο ερώτημα θα επιστρέφει το σύνολο των αντικειμένων τα οποία επαληθεύουν όλες τις λέξεις κλειδιά και επιπλέον ανήκουν στο γεωγραφικό χώρο που έχουν δοθεί ως ορίσματα.
2. **BooleanOr Range Query** Το συγκεκριμένο ερώτημα θα επιστρέφει το σύνολο των αντικειμένων τα οποία επαληθεύουν έστω μία από τις λέξεις κλειδιά και επιπλέον ανήκουν στο γεωγραφικό χώρο που έχουν δοθεί ως ορίσματα.

Για κάθε ερώτημα στο ευρετήριο, τα ορίσματα τα οποία απαιτούνται είναι:

- Οι συντεταγμένες του κέντρου του κύκλου. Γεωγραφικό μήκος και πλάτος.
- Η ακτίνα του κύκλου για την οποία θα γίνει η αναζήτηση.
- Η γεωγραφική μονάδα μέτρησης της ακτίνας. Μέτρα, χιλιόμετρα, μίλια.
- Το σύνολό των λέξεων κλειδιών που θέλουμε να επαληθεύουν το ερώτημά μας.

Στο τέλος κάθε ερωτήματος, το παραγόμενο αποτέλεσμα θα είναι το σύνολο των ids και συντεταγμένων τα οποία επαληθεύουν τις λέξεις κλειδιά που έχουν δοθεί ως ορίσματα και περιλαμβάνονται στην γεωγραφική ακτίνα.
Συγκεκριμένα ακολουθήθηκαν τα παρακάτω βήματα:

4.1 ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ

Αρχικά δημιουργούμε τα κλειδιά και εισάγουμε τα δεδομένα με τη χρήση της εντολής GEOADD για την κάθε λέξη ξεχωριστά. Η εντολή αυτή προσθέτει στο κλειδί το μέλος που έχουμε ορίσει, υπολογίζοντας και θέτοντας την παραγόμενη βαθμολογία όπως περιγράφεται στα Κεφ. 2.3.1.1, Κεφ. 2.3.1.2.
Για τον διαχωρισμό των λέξεων από τις προτάσεις καθώς και για την εξάλειψη των stop words χρησιμοποιήθηκε η βιβλιοθήκη Lucene.

Δομή εντολής: GEOADD key longitude latitude member [longitude latitude member ...]

- GEOADD **casino** 13.361389 38.115556 "**docid1**". Δημιουργείται το **key casino** και ως **member** θέτουμε το **id** του document στο οποίο εμφανίζεται το keyword αυτό, ο συνδυασμός **key - member** είναι μοναδικός όπως προαναφέρθηκε σύμφωνα με της προδιαγραφές Redis για τα ταξινομημένα σύνολα.

4.2 ΕΚΤΕΛΕΣΗ ΕΡΩΤΗΜΑΤΟΣ

Κατά την εκτέλεση του ερωτήματος εκτελούνται πολλαπλές GEORADIUS εντολές τόσες όσες είναι και οι λέξεις που δίνονται ως όρισμα στο εκάστοτε ερώτημα. Η εντολή GEORADIUS είναι υπεύθυνη για την εκτέλεση του ερωτήματος ακτίνας όπως περιγράφεται στο Κεφ. 2.3.2.

Δομή εντολής: GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC] [STORE key] [STOREDIST key]

- GEORADIUS casino 15 37 200 km WITHCOORDS
- GEORADIUS play 15 37 200 km WITHCOORDS

Σε αυτό το σημείο, εκμεταλλευόμαστε την δυνατότητα του pipelining. Δηλαδή, το σύνολο των GEORADIUS εντολών αποστέλλεται στον Redis διακομιστή ως μία εντολή. Έτσι δεν χρειάζεται ο client να περιμένει μία μία τις απαντήσεις, αλλά μπορεί να τις διαχειριστεί όλες σε ένα τελικό βήμα. Κερδίζουμε μεγάλο βαθμό στην αποτελεσματικότητα του ερωτήματος καθώς ο χρόνος αυτού πλέον εξαρτάται από την πιο αργή GEORADIUS εκτέλεση και όχι από τον συνολικό χρόνο αυτών.

Τέλος, για το τελικό BooleanOr ή BooleanAnd των αποτελεσμάτων βάση των λεκτικών παραμέτρων που δόθηκαν ως είσοδος στο ερώτημα, επιλέχθηκε η εκτέλεση τους στον πελάτη. Στις μέρες μάς, η πλειοψηφία των συσκευών που θα εκτελέσουν τον ρόλο ενός πελάτη, από κάποιο μεγάλων δυνατοτήτων υπολογιστή μέχρι και κάποια IOT μικροσυσκευή, διαθέτουν αρκετή επεξεργαστική ισχύ και μνήμη RAM για την διεκπεραίωση μιας τέτοιας λειτουργίας.

Αυτή η επιλογή δεν έγινε τυχαία αλλά όπως αναφέρθηκε παραπάνω στους περιορισμούς της Redis Cluster, δεν επιτρέπονται η λειτουργίες πολλαπλών κλειδιών για κλειδιά που βρίσκονται σε διαφορετικές θυρίδες κατακερματισμού. Αυτό σημαίνει πώς η χρήση των εντολών ZINTERSTORE (εντολή της Redis για τον υπολογισμό της τομής πολλαπλών sets) και ZUNIONSTORE (εντολή της Redis για τον υπολογισμό της ένωσης πολλαπλών sets) για την εκτέλεση του BooleanAnd και του BooleanOr αντίστοιχα στον Redis διακομιστή, θα απαιτούσε την ύπαρξη όλων των κλειδιών στον ίδιο Redis κόμβο, γεγονός που οδηγεί στον μη διαμοιρασμό των κλειδιών στο σύνολο των τμημάτων της συστάδας.

4.3 ΨΕΥΔΟΚΩΔΙΚΑΣ

Είσοδος Αλγορίθμου: Τα ορίσματα που δίνονται στην είσοδο του αλγορίθμου περιγράφονται ως εξής.

- lon: Τιμή τύπου double όπου αφορά το γεωγραφικό μήκος του σημείου.
- lat: Τιμή τύπου double όπου αφορά το γεωγραφικό πλάτος του σημείου.
- radius: Τιμή τύπου double όπου αφορά την γεωγραφική ακτίνα.
- unit: Enumerator που περιγράφει την μονάδα μέτρησης της ακτίνας.
- keywords[]: Λίστα τύπου συμβολοσειράς όπου αφορά τις λέξεις κλειδιά.
- QueryType: Enumerator που περιγράφει τον τύπο του ερωτήματος.

Έξοδος Αλγορίθμου: Τα αποτελέσματα που παράγονται ως έξοδος του αλγορίθμου περιγράφονται ως εξής.

- union[] or intersection[]: Λίστες τύπου GeoRadiusResponse η οποίες παράγονται από το BooleanOr ή BooleanAnd των λέξεων κλειδιών. Ο τύπος GeoRadiusResponse περιέχει το id και τις συντεταγμένες ενός σημείου.

1. **Server Calls:**

```
redis.call("PIPELINE")
responses[] = {}
foreach keyword of keywords do
        geoRadiusResponse[] = redis.call("GEORADIUS {keyword} {lon} {lat} {radius} {unit} WITHCOORD")
        responses.add(geoRadiusResponse[])
end
redis.call("SYNC PIPELINE")
```

2. **BooleanOr Client Logic:**

```
union[] = distinctByMember(flatMapList(responses))
```

3. **BooleanAnd Client Logic:**

```
intersection[] = {}
foreach response of responses do
        intersectByMember(response, intersection)
end
```

Υπόμνημα:

- responses[]: Η μεταβλητή αυτή αποθηκεύει το σύνολο τον απαντήσεων της κάθε GEORADIUS κλήσης. Στην πραγματικότητα είναι μία λίστα από GeoRadiusResponse λίστες.
- flatMapList: Μέθοδος η οποία μετατρέπει τις πολλαπλές GeoRadiusResponse λίστες σε μία.
- distinctByMember: Μέθοδος η οποία φιλτράρει και επιστρέφει τα μοναδικά ids για την αποφυγή διπλότυπων.
- intersectByMember: Μέθοδος η οποία φιλτράρει και επιστρέφει τα ids αυτά τα οποία εμφανίζονται σε όλες τις GeoRadiusResponse λίστες.

4.3.1 ΕΝΑΛΛΑΚΤΙΚΕΣ ΠΡΟΣΕΓΓΙΣΕΙΣ

Οι εναλλακτικές προσεγγίσεις που αξιολογήθηκαν στα πλαίσια υλοποίησης χωροκειμενικού ευρετηρίου στην Redis ήταν οι εξής:

1. Η προσέγγιση που περιγράφηκε παραπάνω αλλά με την διαφορά της εκτέλεσης των BooleanOr και BooleanAnd στον διακομιστή με την χρήση των εντολών ZUNIONSTORE και ZINTERSTORE αντίστοιχα. Απορρίφθηκε καθώς όπως αναφέρθηκε δεν είναι αποδοτική στην κλιμάκωση μεγάλου όγκου δεδομένων. Επίσης, μας προσθέτει το κόστος μιας επιπλέον τελικής GEORADIUS εντολής, πάνω στο παραγόμενο κλειδί της ZUNIONSTORE ή ZINTERSTORE εντολής με σκοπό τη ανάκτηση των αποτελεσμάτων που πληρούν τα κριτήρια του ερωτήματος.

Πιθανόν μία τέτοια προσέγγιση να είναι πιο επιθυμητή καθώς αφαιρεί πολυπλοκότητα από τον πελάτη όταν έχουμε μικρό όγκο δεδομένων και ερωτήματα μερικών λέξεων όπου δεν μας ενδιαφέρει ο διαμοιρασμός τον κλειδιών μας σε πολλαπλούς κόμβους.

2. Αντί της δημιουργία κλειδιών για κάθε λέξη των δεδομένων μας, μία spatial first λύση ήταν η δημιουργία ενός μοναδικού κλειδιού της μορφής
    - GEOADD **documents** 13.361389 38.115556 "**docid1:casino**"
    - GEOADD **documents** 13.361389 38.115556 "**docid1:play**"

    στο οποίο θέτουμε ως μέλος (member) το id του εγγράφου σε συνδυασμό με την εκάστοτε λέξη. Με αυτή την προσέγγιση προκύπτει μια λιγότερο πολύπλοκη υλοποίηση του ευρετηρίου και αντίστοιχα κατά το ερώτημα μας μία μοναδική εκτέλεση GEORADIUS εντολής. Και σε αυτή την προσέγγιση το BooleanOr και BooleanAnd των λεκτικών παραμέτρων των αποτελεσμάτων θα πρέπει να εκτελεστούν στον πελάτη.

    Η παραπάνω προσέγγιση απορρίφθηκε για τους λόγους ότι επίσης δεν είναι αποδοτική στην κλιμάκωση μεγάλου όγκου δεδομένων αλλά κυρίως λόγω του ότι έχοντας μόνο ένα κλειδί, ο χρόνος εκτέλεσης ερωτήματος εξαρτάται άμεσα από το μέγεθος του συνόλου των δεδομένων που έχουν εισαχθεί σε αυτό.

ΚΕΦΑΛΑΙΟ ΠΕΜΠΤΟ

5. ΠΕΙΡΑΜΑΤΑ ΚΑΙ ΣΥΓΚΡΙΣΕΙΣ

Σε αυτό το κεφάλαιο περιγράφονται πειράματα το οποία διεξήχθησαν πάνω σε Twitter δεδομένα, τα οποία αποθηκεύσαμε στην Redis βάση δεδομένων με σκοπό να μελετήσουμε την επίδοση χωροκειμενικών ερωτημάτων.

Το σύνολο των δεδομένων αφορούν Tweets, έστω *D*, μεγέθους 16.9GB από τα οποία επιλέχθηκαν τα παρακάτω χαρακτηριστικά.

1. Tweet ID
2. Coordinates - Longitude, Latitude
3. Text

Η μελέτη χωρίστηκε σε 2 σκέλη, διαφορετικού μεγέθους δεδομένων το καθένα. Στο πρώτο σκέλος χρησιμοποιήθηκε υποσύνολο του *D*, έστω *D1*, το οποίο αποτελείται από 1000 εγγραφές. Στο δεύτερο σκέλος χρησιμοποιήθηκε υποσύνολο του *D*, έστω *D2*, το οποίο αποτελείται από 87000 εγγραφές.

Οι προσεγγίσεις που μελετήθηκαν και έγινε σύγκριση των αποτελεσμάτων τους είναι οι εξής.

- Χωροκειμενικό ευρετήριο σε Redis όπως περιγράφεται στο Κεφ. 4, έστω *Redis*.
- Χρήση του RediSearch module όπως περιγράφεται στο Κεφ. 2.2, έστω *RediSearch*.

Για κάθε προσέγγιση εκτελέστηκαν και τα 2 σκέλη πειραμάτων. Επιπλέον, λόγω του ότι κάθε προσέγγιση απαιτεί για την λειτουργία της τον δικό της τύπο και οργάνωση των δεδομένων, το κάθε υποσύνολο (*D1, D2*) εισήχθηκε 2 φορές (Κεφ. 5.1). Όλα τα παραπάνω θα εισαχθούν σε μία Redis συστάδα βάσεων δεδομένων (Redis Cluster) η οποία αποτελείται από 2 Redis Enterprise κόμβους (nodes) των 2 τμημάτων (shards) ο κάθε ένας. Η έκδοση της Redis που χρησιμοποιήθηκε είναι η 5.0.7 και του

RediSearch module η 1.6.10. Η εγκατάσταση αυτών έγινε σε περιβάλλον Docker με την χρήση των Docker containers, στο οποία δόθηκαν οι εξής υπολογιστικοί πόροι:

- 6 πυρήνες CPU
- 16GB μνήμης RAM
- 60GB χώρο στο δίσκο

οι οποίοι είναι αρκετοί για το μέγεθος του cluster και του όγκου των δεδομένων για την μελέτη που θα πραγματοποιήσουμε.

Ο υπολογιστής που χρησιμοποιήθηκε έχει τις παρακάτω προδιαγραφές.

- AMD Ryzen 7 3700X 8-core χρονισμένο στα 3,6GHz (Base Freq.) - 4,4GHz (Max Boost Clock)
- 32GB DDR4 3200MHz RAM
- 250GB NVMe SSD
- Windows 10 Pro Version: 2004 / Docker: Ubuntu 18.04.4 LTS

5.1 ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ

Προαπαιτούμενο βήμα για την εκτέλεση της πειραματικής μελέτης είναι η εισαγωγή των δεδομένων μας στην Redis βάση δεδομένων. Όπως προαναφέρθηκε κάθε προσέγγιση απαιτεί την δίκια της εισαγωγή των δεδομένων. Για την *Redis* όπως περιγράφεται στο Κεφ. 4.1 και για την *RediSearch* όπως περιγράφεται στο Κεφ. 2.2.1. Με στόχο μία σύντομη προεπισκόπηση των κλειδιών που δημιουργούνται για κάθε προσέγγιση, καταγράφηκαν τα παρακάτω στοιχεία όπως εμφανίζονται στους Πίνακες 6, 7 κατά της εισαγωγή του κάθε υποσυνόλου (*D1, D2*) σε κάθε μια από αυτές.

Ο Πίνακας 6 και 7 παρουσιάζουν για κάθε υποσύνολο δεδομένων τον αριθμό των κλειδιών που δημιουργούνται στην Redis βάση δεδομένων, το μέγεθος αυτών που καταλαμβάνουν στην μνήμη κάθε τμήματος (shard) και τον χρόνο που απαιτήθηκε για την ολοκλήρωση της εισαγωγής τους.

Πίνακας 6: Εισαγωγή Δεδομένων στην RediSearch

RediSearch	Σύνολο Δεδομένων	Κλειδιά	Μέγεθος/ τμήμα	Χρόνος εισαγωγής δεδομένων
	D1	8560	0.7MB	476 millis
	D2	342266	28.7 - 29.7MB	679 millis

Πίνακας 7: Εισαγωγή Δεδομένων στην Redis

Redis	Σύνολο Δεδομένων	Κλειδιά	Μέγεθος ανά τμήμα	Χρόνος εισαγωγής δεδομένων
	D1	4509	0.2MB	411 millis
	D2	129357	12.7 - 17.7MB	424 millis

Παρατηρούμε πως ο απαιτούμενος χρόνος εισαγωγής κυμαίνετε σε χαμηλά και περίπου ίδια επίπεδα ανεξαρτήτως της προσέγγισης ή του συνόλου δεδομένων. Υπάρχει όμως έντονη διαφορά στο πλήθος των κλειδιών και κατά συνέπεια στο μέγεθος μνήμης που καταλαμβάνουν αυτά σε κάθε προσέγγιση.

Συγκεκριμένα, όπως φαίνεται στην κολόνα «Κλειδιά» η *Redis* προσέγγιση δημιουργεί αισθητά λιγότερα κλειδιά σε σχέση με την *RediSearch*. Αυτό έχει ως επακόλουθο πως η *Redis* προσέγγιση είναι πιο αποδοτική όσον αφορά την μνήμη αλλά προφανώς υπολείπεται των δυνατοτήτων που παρέχει το RediSearch module όπως document scoring, term frequencies.

5.2 ΕΚΤΕΛΕΣΗ ΕΡΩΤΗΜΑΤΩΝ

Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκε το εργαλείο JMH - Java Microbenchmark Harness το οποίο μας βοηθάει στο να υλοποιήσουμε σωστά microbenchmarks στην Java.

Με την χρήση του παραπάνω εργαλείου και σύμφωνα με τις δυνατότητες που μας παρέχει, η εκτέλεση των πειραμάτων και η συλλογή των δεδομένων έγινε όπως θα περιγραφεί παρακάτω.

Το JMH θα δημιουργήσει 2 Forks δηλαδή το κάθε Benchmark θα εκτελεστεί από 2 φορές. Το κάθε Fork περιέχει 7 κύκλους Warmup (προθέρμανση του JVM των οποίων τα αποτελέσματα αγνοούνται) και 2 κύκλους Measurement. Οι υπολογισμοί και τα αποτελέσματα προκύπτουν από τον συνολικό αριθμό των Forks επί των Measurements (Fork x Measurement), στην περίπτωσή μας δηλαδή 4 επαναλήψεις. Για τα πειράματα μας θα χρησιμοποιήσουμε την λειτουργία *Throughput* του JMH η οποία μας δίνει ως αποτέλεσμα των μετρικών το σύνολο των επιτυχημένων εκτελέσεων ενός ερωτήματος ανά δευτερόλεπτο (ops/s).

Οι Java υλοποιήσεις που θα συγκριθούν είναι οι εξής:

- NoDAredisearch: Υλοποίηση των ερωτημάτων χρησιμοποιώντας το NoDA API και την Redisearch υλοποίηση αυτού. Στο παρασκήνιο εκμεταλλεύεται τον JRedisearch client.
- Redisearch: Υλοποίηση των ερωτημάτων χρησιμοποιώντας τον JRedisearch client.
- RedisMultiKeys: Υλοποίηση των ερωτημάτων χρησιμοποιώντας τον Jedis client και την χρήση του χωροκειμενικού δείκτη που περιγράψαμε.

Οι δυο πρώτες υλοποιήσεις εκμεταλλεύονται το API του RediSearch module και η διεξαγωγή των ερωτημάτων πραγματοποιείται στο σύνολο των κλειδιών της *RediSearch* προσέγγισης, όπως παρουσιάστηκε στην προηγούμενη ενότητα. Η τρίτη υλοποίηση εκμεταλλεύεται το API της Redis και η διεξαγωγή των ερωτημάτων πραγματοποιείται στο σύνολο των κλειδιών της *Redis* προσέγγισης, όπως παρουσιάστηκε στην προηγούμενη ενότητα.

5.2.1 ΣΚΕΛΟΣ 1: ΣΥΝΟΛΟ ΔΕΔΟΜΕΝΩΝ D1

Αυτή η ομάδα πειραμάτων υλοποιήθηκε με σκοπό μία γενικότερη θεώρηση και σύγκριση των υλοποιήσεων που θα περιγράψαμε στο Κεφ. 5.2. Στόχος εδώ είναι να καταλάβουμε για ένα μικρό σύνολο δεδομένων πώς αποδίδουν οι υλοποιήσεις χωρίς να τις φτάσουμε στα άκρα τους.

Για τα ερωτήματα επιλέχθηκαν οι λέξεις-κλειδιά **apollo**, **the** και **latest** καθώς επίσης οι συντεταγμένες **42.0970023**, **-79.2353259** τα οποία επαληθεύουν σίγουρα την εγγραφή με id 806864584581578753 και έστω το ονομάζουμε *S1*.
Σε συνδυασμό με τα μεγέθη ακτίνας έχουμε τα παρακάτω ερωτήματα.

Πίνακας 8: Χωρο- Κειμενικά ερωτήματα προς εκτέλεση

Ερώτημα	**Σταθερές Μεταβλητές Ερωτήματος**	**Μέγεθος Ακτίνας**
Q1	S1	10 χιλιόμετρα
Q2	S1	100 χιλιόμετρα
Q3	S1	1000 μίλια
Q4	S1, Χωρίς τις λέξεις [the, latest]	1000 μίλια
Q5	S2	10 χιλιόμετρα
Q6	S2	100 χιλιόμετρα
Q7	S2	500 χιλιόμετρα
Q8	S1	500 χιλιόμετρα

Οι Πίνακες 9, 10, 11, 12 παρουσιάζουν την εκτέλεση της τομής (BooleanAnd) των λέξεων-κλειδιών των ερωτημάτων ενώ ο Πίνακας 13 την ένωση (BooleanOr) αυτών.

Πίνακας 9: Τομή λέξεων του Q1

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	986,319	± 19,654	ops/s
redisearch	4	985,677	± 25,420	ops/s
redisMultiKeys	4	1425,047	± 27,945	ops/s

Πίνακας 10: Τομή λέξεων του Q2

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	969,848	± 36,937	ops/s
redisearch	4	970,826	± 36,080	ops/s
redisMultiKeys	4	1281,189	± 331,160	ops/s

Πίνακας 11: Τομή λέξεων του Q3

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	968,673	± 63,778	ops/s
redisearch	4	978,276	± 8,361	ops/s
redisMultiKeys	4	1259,024	± 29,227	ops/s

Πίνακας 12: Τομή λέξεων του Q4

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	971,086	± 85,039	ops/s
redisearch	4	983,449	± 21,602	ops/s
redisMultiKeys	4	1376,750	± 15,041	ops/s

Πίνακας 13: Ένωση λέξεων του Q3

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	905,284	± 128,523	ops/s
redisearch	4	971,647	± 18,638	ops/s
redisMultiKeys	4	1267,084	± 31,913	ops/s

Στους παραπάνω πίνακες βλέπουμε τα αποτελέσματα εκτέλεσης των ερωτημάτων για κάθε μία από τις υλοποιήσεις μας. Συγκρίνοντας την κολόνα **Σκορ** των πινάκων παρατηρούμε τα ακόλουθα. Ανεξαρτήτως του πλήθους των λέξεων κλειδιών αλλά και της ακτίνα του ερωτήματος η απόδοση και των 3 λύσεων δεν παρουσιάζει σημαντικές μεταβολές αλλά παραμένει στα ίδια επίπεδα. Επίσης η NoDAredisearch υλοποίηση πρόσθετη πολύ μικρό έως καθόλου overhead σε σχέση με αυτή της απλής Redisearch, το οποίο μας αποδεικνύει ευχάριστα πώς το NoDA API δεν επιφέρει καθυστερήσεις. Τέλος, βλέπουμε πώς η RedisMultiKeys υλοποίηση έχει ένα μικρό προβάδισμα έναντι αυτών της Redisearch.

5.2.2 ΣΚΕΛΟΣ 2: ΣΥΝΟΛΟ ΔΕΔΟΜΕΝΩΝ D2

Αυτή η ομάδα πειραμάτων υλοποιήθηκε με σκοπό μία ειδικότερη σύγκριση των υλοποιήσεων NoDAredisearch και RedisMultiKeys. Χρησιμοποιώντας ένα αρκετά

μεγαλύτερο σύνολο δεδομένων, στόχος μάς εδώ είναι να κατανοήσουμε εις βάθος την απόδοση της RediSearch και του χωροκειμενικού ευρετηρίου σε Redis που περιγράψαμε σε προηγούμενο κεφάλαιο, όσον αφορά τα χωροκειμενικά ερωτήματα. Εδώ θα χρησιμοποιήσουμε το *S1* της παραγράφου 5.2.1 αλλά επίσης και ένα ερώτημα *S2* = *S1* + Λέξεις Κλειδιά (great, fish)

Πίνακας 14: Τομή λέξεων του Q1

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	878,715	± 167,644	ops/s
redisMultiKeys	4	1223,312	± 30,368	ops/s

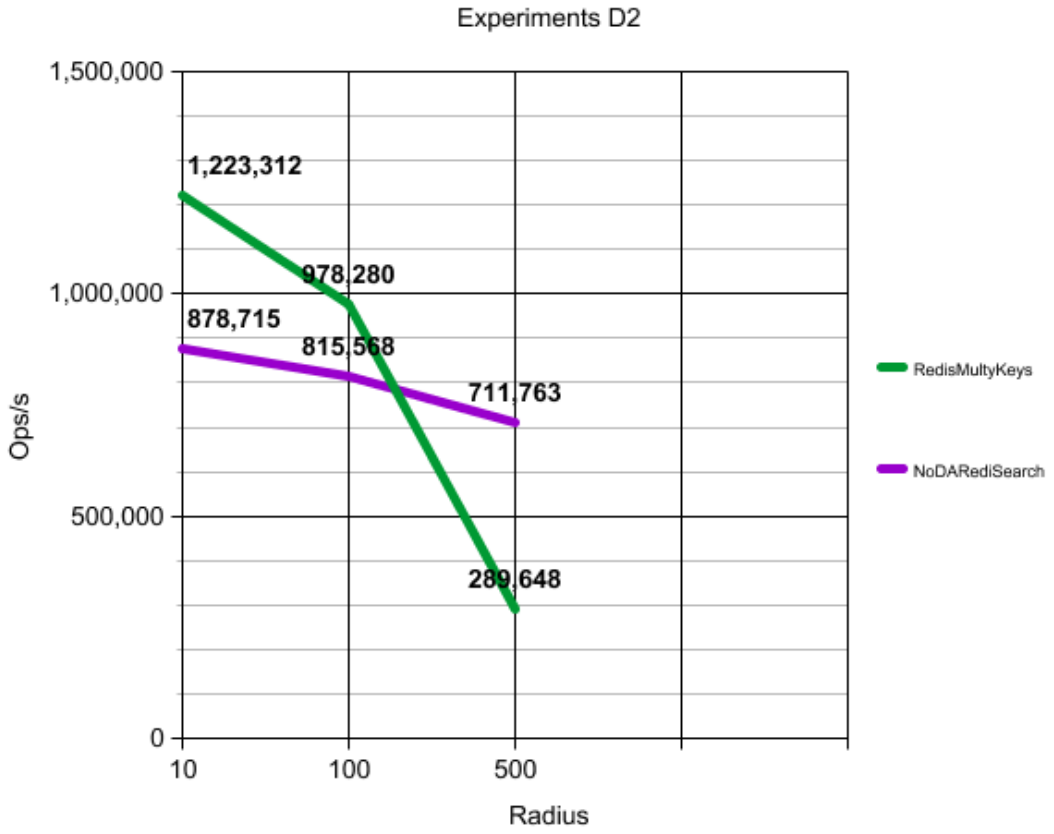
Πίνακας 15: Τομή λέξεων του Q2

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	815,568	± 88,984	ops/s
redisMultiKeys	4	978,280	± 41,205	ops/s

Πίνακας 16: Τομή λέξεων του Q8

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	711,763	± 46,005	ops/s
redisMultiKeys	4	289,648	± 58,863	ops/s

Εικόνα 8: Γράφημα Επιδόσεων



Στην Εικόνα 8 παρουσιάζεται η σύγκριση επίδοσης της RedisMultiKeys και NoDAredisearch υλοποίησης, όπως προκύπτει από τα αποτελέσματα των Πινάκων 14, 15, 16. Παρατηρούμε πώς η απόδοση της RedisMultiKeys υλοποίησης πέφτει αρκετά συγκριτικά με αυτή της NoDAredisearch όσο αυξάνεται η ακτίνα του χωρικού ερωτήματος. Για το σύνολο δεδομένων *D1* δεν παρατηρήθηκε μία τέτοια συμπεριφορά, γιατί όμως συμβαίνει αυτό για το σύνολο δεδομένων *D2*;

Η απάντηση κρύβεται στην λέξη κλειδί των μεταβλητών *S1* "latest". Συγκεκριμένα, το κλειδί latest που δημιουργείται στην Redis, για το *D1* περιέχει μόλις 14 εγγραφές ενώ για το *D2* 689. Επιπλέον η διαφορά με ακτίνα 100 χιλιόμετρα (Πίνακας 15) σε σχέση με τα 500 χιλιόμετρα (Πίνακας 16) φαίνεται καθαρά εκτελώντας τις παρακάτω Redis εντολές.

- GEORADIUS latest 42.0970023 -79.2353259 100 km WITHCOORD -> 31 αποτελέσματα.

- GEORADIUS latest 42.0970023 -79.2353259 500 km WITHCOORD -> 421 αποτελέσματα.

Στην συνέχεια εκτελούμε τα πειράματα για το *S2* στο οποίο έχουμε 2 παραπάνω λέξεις κλειδιά όπου συμμετέχουν στο ερώτημα.

Πίνακας 17: Τομή λέξεων του Q5

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	878,841	± 128,341	ops/s
redisMultiKeys	4	1247,826	± 93,453	ops/s

Πίνακας 18: Τομή λέξεων του Q7

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	731,644	± 72,382	ops/s
redisMultiKeys	4	266,508	± 9,105	ops/s

Πάλι επιβεβαιώνουμε πώς η απόδοση των ερωτημάτων μας δεν επηρεάζεται από το πλήθος των λέξεων κλειδιών αυτού καθώς παραμένουν στα ίδια επίπεδα με αυτά του *S1*.

Στη συνέχεια θα εμβαθύνουμε στην εντολή GEORADIUS και συγκεκριμένα στην επιλογή WITHCOORD. Η συγκεκριμένη επιλογή ενημερώνει την Redis πώς στα αποτελέσματα του ερωτήματος, επιθυμούμε να λάβουμε και τις συντεταγμένες των αποτελεσμάτων εκτός του μελών (members). Αυτό σημαίνει πώς για κάθε αποτέλεσμα η Redis χρειάζεται να κάνει decode το score, το οποίο όπως έχει προαναφερθεί είναι ένα GeoHash, ώστε να παράξει τις συντεταγμένες αυτού.

Υποψιαζόμαστε λοιπόν πως το παραπάνω επιφέρει κόστος στο ερώτημα. Με στόχο να το ανακαλύψουμε εκτελέσαμε πάλι το τελευταίο ερώτημα αλλά χωρίς την επιλογή WITHCOORD στον κώδικα μας.

Πίνακας 19: Τομή λέξεων του Q7

Υλοποίηση	**Επαναλήψεις**	**Σκορ**	**Περιθώριο Σφάλματος**	**Μονάδα Μέτρησης**
NoDAredisearch	4	712,146	± 35,713	ops/s
redisMultiKeys	4	627,487	± 14,241	ops/s

Παρατηρούμε πώς η επίδοση του ερωτήματος αυξήθηκε αρκετά σε σχέση με πριν, με το μειονέκτημα ότι δεν λαμβάνουμε τις συντεταγμένες των αποτελεσμάτων στην απάντηση του ερωτήματος παρά μόνο τα μέλη δηλαδή στην προκειμένη περίπτωση, τα αναγνωριστικά των εγγράφων (id's).

Από τις παραπάνω παρατηρήσεις κατά την εκτέλεση των πειραμάτων γίνεται αντιληπτό πώς η επίδοση εκτέλεσης των ερωτημάτων εξαρτάται κυρίως από το μέγεθος των κλειδιών που συμμετέχουν σε αυτά και το πλήθος των αποτελεσμάτων που επιστρέφονται από το κάθε ένα.

ΚΕΦΑΛΑΙΟ ΕΚΤΟ

6. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Σε αυτή την διπλωματική παρουσιάστηκε ένα API που επιτρέπει την ενοποιημένη πρόσβαση σε δεδομένα, αποθηκευμένα σε NoSQL βάσεις δεδομένων. Το API προσφέρει πρότυπα ερωτήματα αλλά και ένα σύνολο τελεστών για την σύνταξη και έκφραση λειτουργιών πρόσβασης στα δεδομένα αυτών. Η λειτουργικότητά του επεκτάθηκε στα χωροκειμενικά δεδομένα, δεδομένου ότι παρέχονται χωρικοί, κειμενικοί και χωροκειμενικοί τελεστές. Πραγματοποιήθηκε η υλοποίηση αυτού πάνω στο module της Redis, RediSearch, το οποίο παρουσιάστηκε με παραδείγματα και πειράματα.

Επίσης, μελετήθηκε εις βάθος ο τρόπος λειτουργίας της Redis και στο Κεφάλαιο 4 περιγράφηκε μία λύση ευρετηρίασης χωροκειμενικών δεδομένων σε αυτή η οποία εκμεταλλεύεται σε μεγάλο βαθμό τεχνικές όπως του διαμοιρασμού των κλειδιών σε μία βάση δεδομένων Redis σε συστάδα και pipelining με στόχο την κλιμάκωση των ευρετηρίων και της μέγιστης απόδοσης αυτών τον ερωτημάτων, καθώς και παρουσιάστηκαν εναλλακτικοί τρόποι προσέγγισης.

Ακόμα ένας στόχος αυτής της διπλωματικής ήταν η πειραματική αξιολόγηση της απόδοσης της RediSearch μέσο του NoDA API και της προσαρμοσμένης Redis λύσης σε χωροκειμενικά ερωτήματα. Σύμφωνα με το Κεφάλαιο 5, είδαμε πως και οι 2 λύσεις εκτελούν αποτελεσματικά χωροκειμενικά ερωτήματα με την χρήση των εγγενών τύπων δεδομένων και ευρετηρίων που παρέχουν. Παρατηρήθηκε πώς κυρίως η απόδοση τους επηρεάζεται από το μέγεθος των κλειδιών, δηλαδή του συνόλου των τιμών που περιέχουν σε συνδυασμό με την ακτίνα του ερωτήματος και όχι από το πλήθος των λέξεων κλειδιών που λαμβάνουν μέρος σε αυτό. Ειδικά για την προσαρμοσμένη Redis λύση παρατηρήθηκε πώς αποδίδει καλύτερα σε ερωτήματα μικρής ακτίνας ή σε ερωτήματα όπου συμμετέχουν κλειδιά μικρού όγκου ενώ η

απόδοση μειώνεται αρκετά σε ερωτήματα μεγάλης ακτίνας και κλειδιών μεγάλου όγκου. Επίσης άλλη μία ενδιαφέρουσα παρατήρηση είναι αυτή του κόστους που επιφέρει η επιλογή WITHCOORD στην εκτέλεση της εντολής GEORADIUS και πώς η παράλειψη αυτής βελτιώνει αισθητά την απόδοση εκτέλεσης του χωροκειμενικού ερωτήματος. Στην άλλη πλευρά είχαμε την RediSearch για την οποία παρατηρήθηκε πώς διατηρεί τον μέσο όρο αποδοτικότητάς της σε σταθερά επίπεδα χωρίς να επηρεάζεται έντονα από τον όγκο τον δεδομένων, την ακτίνα του ερωτήματος ή του μεγέθους τον κλειδιών που συμμετέχουν.

Πολλές θα μπορούσαν να είναι οι ιδέες και κατευθύνσεις για μελλοντικές εργασίες. Αρχικά το NoDA API θα μπορούσε να επεκταθεί υποστηρίζοντας περισσότερες NoSQL βάσεις δεδομένων καθώς επίσης και να εμπλουτιστεί με περισσότερους τελεστές και δυνατότητες όπως για παράδειγμα την υποστήριξη ασύγχρονων και reactive εντολών, κάτι το οποίο είναι πολύ διαδεδομένο στα σύγχρονα συστήματα. Επίσης, όσον αφορά την λύση ευρετηρίασης χωροκειμενικων δεδομένων, θα μπορούσαν να μελετηθούν άλλες προσεγγίσεις όπου περιγράφηκαν, για παράδειγμα διαφορετικός τρόπος διαμοιρασμού των κλειδιών στην συστάδα και πιθανές τεχνικές για εκμετάλλευση των εγγενών εντολών της Redis για τομή και ένωση των αποτελεσμάτων. Επίσης, θα μπορούσε να πραγματοποιηθεί κάποια υλοποίηση χωρίς την επιλογή WITHCOORDS στην οποία θα αποθηκεύονται τα πραγματικά δεδομένα σε κάποια άλλη δομή της Redis, παρόμοια δηλαδή με την λογική της RediSearch. Ακόμα, μετά των λεπτομερών πειραμάτων αλλά και της καλής πλέον γνώσης της Redis και RediSearch, θα είχε μεγάλο ενδιαφέρον η μελέτη για την δημιουργία ενός νέου module της Redis, αντίστοιχο της RediSearch, το οποίο θα εκμεταλλεύεται τις δυνατότητες που περιγράφηκαν σε αυτή την διπλωματική και θα είναι στοχευόμενο στην ευρετηρίαση και επερώτηση χωροκειμενικών δεδομένων.

Βιβλιογραφία

[1] Y. Ma, Y. Zhang and X. Meng, "ST-HBase: A Scalable Data Management System," pp. 155-166*, WAIM 2013,* Jun. 2013.

[2] N. Koutroumanis, P. Nikitopoulos, A. Vlachou and C. Doulkeridis, "NoDA: Unified NoSQL Data Access Operators for Mobility Data," *SSTD 2019,* pp. 174-177, Aug. 2019.

[3] "Introduction to Redis," *Redis*. [Online]. Available: https://redis.io/topics/introduction. [Accessed: 12-Sep-2020].

[4] "Redis Sentinel Documentation," *Redis*. [Online]. Available: https://redis.io/topics/sentinel. [Accessed: 12-Sep-2020].

[5] "Discovery Service," *Discovery Service | Redis Labs Documentation Center*. [Online]. Available: https://docs.redislabs.com/latest/rs/concepts/data-access/discovery-service/. [Accessed: 12-Sep-2020].

[6] "Database clustering," *Database clustering | Redis Labs Documentation Center*. [Online]. Available: https://docs.redislabs.com/latest/rs/concepts/high-availability/clustering/. [Accessed: 12-Sep-2020].

[7] "Using pipelining to speedup Redis queries," *Redis*. [Online]. Available: https://redis.io/topics/pipelining. [Accessed: 12-Sep-2020].

[8] "RediSearch Full Command Documentation," *Command Reference - RediSearch Documentation*. [Online]. Available: https://oss.redislabs.com/redisearch/Commands.html. [Accessed: 12-Sep-2020].

[9] "RediSearch Technical Overview," *RediSearch Technical Overview - RediSearch Documentation*. [Online]. Available: https://oss.redislabs.com/redisearch/Overview/. [Accessed: 12-Sep-2020].

[10] D. Volk, "RediSearch: A High Performance Search Engine as a Redis Module," *Redis*. [Online]. Available: https://lp.redislabs.com/rs/915-NFD-128/images/WP-RedisLabs-RediSearch-103-Web.pdf. [Accessed: 12-Sep-2020].

[11] "Sorted Sets as Indexes," *Redis*, 23-Jul-2020. [Online]. Available: https://redislabs.com/redis-best-practices/indexing-patterns/sorted-sets-indexes/. [Accessed: 12-Sep-2020].

[12] I. Haber, "Redis for Geospatial Data," *Redis*. [Online]. Available: https://lp.redislabs.com/rs/915-NFD-128/images/WP-RedisLabs-Geospatial-Redis.pdf. [Accessed: 12-Sep-2020].

[13] C. Greco, "Introducing the Geo API in Redis," 7-July-2015. [Online]. Available: https://cristian.regolo.cc/2015/07/07/introducing-the-geo-api-in-redis.html. [Accessed: 12-Sep-2020].

[14] "Geospatial Commands," *Redis*, 19-Aug-2020. [Online]. Available: https://redislabs.com/redis-best-practices/indexing-patterns/geospatial/. [Accessed: 12-Sep-2020].

[15] J. L. Carlson, *Redis in action*. Shelter Island: Manning, 2013.

[16] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon, "Spatio-temporal indexing in non-relational distributed databases," 2013 IEEE International Conference on Big Data, pp. 291–299, Oct. 2013.

[17] N. Koutroumanis, "Design and Implementation of a NoSQLData Access Interface," M.S. thesis, University of Piraeus, Piraeus, 2019. [Online]. Available: http://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/11932/Nikolaos_Koutroumanis.pdf.