ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

«ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΚΑΙ ΥΠΗΡΕΣΙΕΣ»

ΕΙΔΙΚΕΥΣΗ: «ΜΕΓΑΛΑ ΔΕΔΟΜΕΝΑ & ΑΝΑΛΥΤΙΚΗ»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΕΠΕΞΕΡΓΑΣΙΑ ΧΩΡΟΧΡΟΝΙΚΩΝ ΕΡΩΤΗΜΑΤΩΝ ΣΤΗΝ MongoDB»

ΦΟΙΤΗΤΗΣ: ΧΡΗΣΤΟΣ ΓΙΑΝΝΟΓΛΟΥ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: ΜΕ1707

ΕΠΙΒΛΕΠΩΝ: ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ ΧΡΗΣΤΟΣ ΔΟΥΛΚΕΡΙΔΗΣ

ΠΕΙΡΑΙΑΣ, ΣΕΠΤΕΜΒΡΙΟΣ 2019

ΠΕΡΙΛΗΨΗ

Σε αυτή την εργασία η έρευνα επικεντρώθηκε στην επεξεργασία χωροχρονικών ερωτημάτων στην μη σχεσιακή βάση δεδομένων MongoDB. Ως χωροχρονικά δεδομένα ορίζονται τα δεδομένα που αποτελούνται από τις συντεταγμένες και μια χρονική στιγμή, που δείχνει πότε η οντότητα που εξετάζεται βρέθηκε στο συγκεκριμένο σημείο. Σε αυτά τα δεδομένα προστίθεται και ένα id που είναι μοναδικό για κάθε οντότητα. Αυτά τα δεδομένα αποθηκεύτηκαν στη βάση δεδομένων MongoDB. Κάθε έγγραφο (document) της MongoDB περιλαμβάνει το id της οντότητας, τις συντεταγμένες και την αντίστοιχη χρονική στιγμή. Τα ερωτήματα που απαντήθηκαν είναι τα Circle Range και Box Range, που αφορούν την εύρεση δεδομένων μέσα σε ένα κύκλο ή ορθογώνιο αντίστοιχα, για ένα συγκεκριμένο χρονικό διάστημα [t_min, t_max], καθώς επίσης και το ερώτημα της εύρεσης των k κοντινότερων γειτόνων (και k κοντινότερων γειτόνων με διαφορετικό id), που αφορά την εύρεση k δεδομένων (και k δεδομένων με διαφορετικό id) που βρέθηκαν πιο κοντά σε ένα σημείο (x, y) μέσα σε ένα χρονικό διάστημα [t_min, t_max]. Στην πειραματική μελέτη μετρήθηκε ο χρόνος εκτέλεσης του ερωτήματος (query), ο συνολικός χρόνος εκτέλεσης του αλγόριθμου και ο αριθμός των εγγράφων που εξετάστηκαν, προκειμένου να απαντηθεί το ερώτημα. Όσο αφορά τη χρήση ευρετηρίων, όλα τα ερωτήματα απαντήθηκαν χωρίς κανένα ευρετήριο, με ευρετήριο στις συντεταγμένες, με ευρετήριο στο χρόνο και με συνδυαστικό ευρετήριο στις συντεταγμένες και στο χρόνο.

ABSTRACT

The main subject of this thesis was query processing of spatiotemporal data in the NoSQL database MongoDB. The definition for spatiotemporal data is as follows: data objects that combine coordinates (latitude, longitude) with a timestamp, which represent when the entity under examination was present at the given time in the given geographical point. In addition to this data, there exists a unique identification (id) for each entity. This data was stored in the MongoDB database, with each document containing the id, latitude, longitude and the related timestamp. The queries answered were the Circle Range query and the Box Range query, regarding finding data in a circle or a rectangle for a specific period of time [t_min, t_max] respectively. Also, the k-Nearest Neighbours query and k-distinct-id-Nearest Neighbours query, regarding finding the closest coordinates, and the closest coordinates that belong to distinct entities near to a specific point (x,y) in a given period [t_min, t_max] respectively. In the experimental part of this thesis, different metrics were used: the time needed to execute the query, the time of algorithm execution in total and the number of the documents accessed for the query to be answered. Regarding the use of indexes, all the queries were processed: without any index, using index in the field coordinates, using index in the field time, and using compound index which combines the fields coordinates and time.

ΕΥΧΑΡΙΣΤΙΕΣ

Με την περάτωση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή μου κ. Χρήστο Δουλκερίδη για την βοήθεια και την καθοδήγησή του σε όλη τη διάρκεια αυτής της προσπάθειας.

Επίσης, ευχαριστώ όλους τους καθηγητές του τμήματος για όλα όσα μου διδάξανε από την αρχή των σπουδών μου μέχρι και σήμερα, καθώς επίσης και τον συμφοιτητή μου Νικόλαο Κουτρουμάνη για την πολύτιμη βοήθειά του.

Περιεχόμενα

1.ΕΙΣΑΓΩΓΗ

1.1 Πρόλογος

Η σημερινή εποχή χαρακτηρίζεται ως η εποχή των Μεγάλων Δεδομένων, καθώς ολοένα και μεγαλύτερος όγκος δεδομένων συλλέγεται από τις εταιρείες και τους οργανισμούς. Για το τι είναι τα Μεγάλα Δεδομένα έχουν δοθεί πολλοί διαφορετικοί ορισμοί. Όλοι οι ορισμοί συμφωνούν στο γεγονός ότι τα Μεγάλα Δεδομένα έχουν 3 βασικά χαρακτηριστικά. Πρώτο και βασικό χαρακτηριστικό είναι ο όγκος (Volume), δηλαδή τα Μεγάλα Δεδομένα αναφέρονται σε ένα σύνολο δεδομένων τόσο μεγάλο, που τα παραδοσιακά εργαλεία επιχειρηματικής ευφυίας (business intelligence tools) είναι σχεδόν αδύνατον να διαχειριστούν και να επεξεργαστούν. Στη συνέχεια, το επόμενο χαρακτηριστικό είναι γνωστό ως Variety. Αυτά τα δεδομένα είναι πιθανό να βρίσκονται α) δομημένη μορφή, δηλαδή μπορούν να αποθηκευτούν να προσπελαστούν και να επεξεργαστούν με βάση κάποιου συγκεκριμένους κανόνες. Χαρακτηριστικό παράδειγμα δομημένων δεδομένων είναι ένας πίνακας μιας sql βάσης δεδομένων. β) αδόμητη μορφή, δηλαδή δεδομένα που δεν ακολουθούν συγκεκριμένους κανόνες και μπορεί να είναι κείμενο σε αρχεία κειμένου, εικόνες, βίντεο κλπ γ) ημιδομημένη μορφή, που ουσιαστικά είναι ένας συνδυασμός των δύο προηγούμενων κατηγοριών, καθώς μπορούν να εξεταστούν σαν δομημένα δεδομένα, αλλά δεν ακολουθούν κάποιους κανόνες. Ένα παράδειγμα ημιδομημένων δεδομένων είναι τα XML αρχεία. Τέλος, το τρίτο χαρακτηριστικό είναι το Velocity και ουσιαστικά είναι ένα μέτρο του πόσο γρήγορα αυξάνονται αυτά τα δεδομένα [6][7]. Χαρακτηριστικό είναι το παράδειγμα του Facebook, όπου οι χρήστες ανεβάζουν περισσότερες από 900.000.000 φωτογραφίες σε μια μόνο μέρα.

Όπως εύκολα γίνεται κατανοητό, τα Μεγάλα Δεδομένα περιέχουν πάρα πολλές πληροφορίες, που η ανάλυση και η αξιοποίηση τους μπορεί να οδηγήσει σε χρήσιμα συμπεράσματα σε ένα πάρα πολύ μεγάλο πλήθος πεδίων, τόσο επιστημονικών όσο και επαγγελματικών. Μερικά από τα επιτεύγματα που έχουν προέλθει από τη χρήση των Μεγάλων Δεδομένων είναι τα εξής:

- Παροχή καλύτερων υπηρεσιών: Με την ανάλυση της καταναλωτικής συμπεριφοράς των πελατών, είναι δυνατή η προώθηση συγκεκριμένων προϊόντων σε κάθε πελάτη που είναι πολύ πιθανό να τον ενδιαφέρουν. Με αυτό το τρόπο και οι πελάτες μπορούν να κάνουν γρήγορα τις αγορές τους.
- Ανάπτυξη νέων προϊόντων/υπηρεσιών: Γνωρίζοντας τις ανάγκες των πελατών και τις τάσεις τις αγοράς γίνεται ευκολότερη η δημιουργία των αντίστοιχων προϊόντων/υπηρεσιών που θα καλύπτουν τις ανάγκες τους.
- Ανίχνευση απάτης: Συστήματα ανάλυσης Μεγάλων Δεδομένων που χρησιμοποιούνται κυρίως από τράπεζες δίνουν τη δυνατότητα ανίχνευσης συναλλαγών με πιστωτικές κάρτες που μπορούν να χαρακτηριστούν ύποπτες.
- Διάγνωση ασθενειών: Ένας ακόμα τομέας που χρησιμοποιεί και αναλύει Μεγάλα Δεδομένα είναι η ιατρική, καθώς με ανάλυση των συμπτωμάτων ή σε ορισμένες περιπτώσεις ακόμα και με ανάλυση εικόνας μπορεί να γίνει διάγνωση μιας ασθένειας.

Συνεπώς, με εφαρμογή των Μεγάλων Δεδομένων σε πολλούς διαφορετικούς τομείς, είναι απολύτως αναμενόμενη η επεξεργασία όλων αυτών των δεδομένων και η αξιοποίηση τους προκειμένου να ληφθούν οι σωστές αποφάσεις και να πραγματοποιηθεί η εξόρυξη της γνώσης.

Μια κατηγορία Μεγάλων Δεδομένων που η ανάλυση τους εξελίσσεται ραγδαία τα τελευταία χρόνια είναι τα χωροχρονικά δεδομένα. Ως χωροχρονικά δεδομένα ορίζονται τα δεδομένα που αποτελούνται από 3 βασικές μεταβλητές (longitude, latitude, timestamp), δηλαδή τις συντεταγμένες και τη χρονική στιγμή που η οντότητα βρέθηκε στη συγκεκριμένη θέση. Τα συγκεκριμένα δεδομένα έγιναν πάρα πολύ δημοφιλή με την ανάπτυξη της τεχνολογίας και τη δημιουργία συσκευών που είναι εξοπλισμένες με συστήματα GPS. Συνεπώς, πολλές εφαρμογές είτε πρώτα με συλλογή και στη συνέχεια με ανάλυση είτε με ανάλυση σε πραγματικό χρόνο, μπορούν να προσφέρουν στους χρήστες πολύ σημαντικές υπηρεσίες. Η ανάλυση των συγκεκριμένων δεδομένων μπορεί να βρει εφαρμογή σε πολλούς τομείς, μερικοί από τους οποίους παρουσιάζονται παρακάτω:

- Ναυτιλία: Συστήματα πλοήγησης χρησιμοποιούνται από τα πλοία έτσι ώστε να αφενός να είναι πάντα γνωστή η θέση τους, αφετέρου με ανάλυση της τροχιάς υπάρχει η δυνατότητα εξαγωγής χρήσιμων συμπερασμάτων.
- Δημιουργία έξυπνων πόλεων: Ολοένα και περισσότερες πόλεις παρέχουν ανοιχτά δεδομένα, τα οποία πολλές φορές είναι και σε πραγματικό χρόνο (real time), τα οποία μπορούν να χρησιμοποιηθούν για τη δημιουργία εφαρμογών που θα διευκολύνουν τη καθημερινότητα των πολιτών, όπως εφαρμογές εύρεσης θέσεων παρκαρίσματος ή εύρεσης ταξί ή ανάλυσης κίνησης σε κεντρικές λεωφόρους.
- Υπηρεσίες υγείας: Χωροχρονικά δεδομένα χρησιμοποιούνται και από οργανισμούς που ασχολούνται με την Υγεία προκειμένου να πραγματοποιηθεί ανάλυση διάφορων θεμάτων, όπως της εξάπλωσης μιας ασθένειας.
- Οικονομικές συναλλαγές: Οικονομικοί οργανισμοί και τράπεζες χρησιμοποιούν χωροχρονικά δεδομένα προκειμένου να λάβουν σημαντικές αποφάσεις σε διάφορα θέματα όπως οι επενδύσεις.

Προκειμένου να επιτευχθεί ο παραπάνω στόχος είναι απαραίτητη η χρήση των κατάλληλων τεχνολογιών επεξεργασίας των δεδομένων, όπως τεχνολογίες παράλληλης επεξεργασίας δεδομένων και φυσικά πρέπει να υπάρχουν οι κατάλληλες βάσεις δεδομένων που θα μπορέσουν να αποθηκεύσουν όλα αυτά τα δεδομένα. Το πρόβλημα αποθήκευσης των Μεγάλων Δεδομένων λύθηκε χρησιμοποιώντας μη σχεσιακές βάσεις δεδομένων. Μια πάρα πολύ σημαντική μη σχεσιακή βάση δεδομένων που χρησιμοποιήθηκε για τις ανάγκες τις εργασίας είναι η MongoDB.

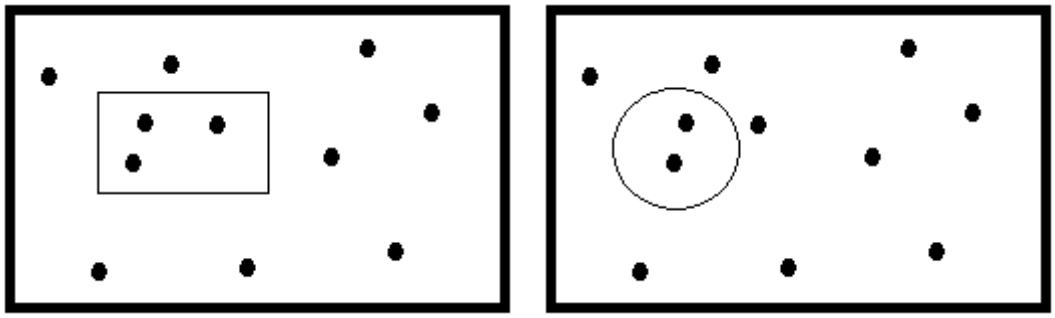
1.2 Σκοπός της εργασίας

Σε αυτή την εργασία, η έρευνα επικεντρώθηκε κυρίως στο κομμάτι των μη σχεσιακών βάσεων δεδομένων και συγκεκριμένα στη noSQL βάση MongoDB. Ειδικότερα, εξετάστηκαν ορισμένοι από τους τρόπους που επιτυγχάνεται η επεξεργασία των ερωτημάτων που αφορούν χωροχρονικά δεδομένα και μέσα από την εφαρμογή αλγορίθμων, που θα αναφερθούν αναλυτικότερα στη συνέχεια, έγινε μια προσπάθεια βελτίωσης της απόδοσής τους.

Ως χωροχρονικά δεδομένα, ορίζονται τα δεδομένα που αποτελούνται κυρίως από 3 παραμέτρους (longitude, latitude, t), δηλαδή τις συντεταγμένες (longitude, latitude) και τη χρονική στιγμή t, που η οντότητα βρέθηκε στην θέση που ορίζουν οι συντεταγμένες. Στις παραπάνω παραμέτρους προστίθεται και το id, που είναι μοναδικό για κάθε οντότητα.

Τα ερωτήματα που απαντήθηκαν αφορούν την εύρεση των εγγραφών, οι οποίες αποτελούνται από τις 4 παραπάνω μεταβλητές (id, lon, lat, t). Τα ερωτήματα είναι τα εξής:

- εύρεση των εγγραφών μέσα σε ένα ορθογώνιο που σχηματίζεται από τα ζεύγη των συντεταγμένων (min_lon, min_lat) και (max_lon, max_lat) κάποιο συγκεκριμένο χρονικό διάστημα [t_min, t_max] (Εικόνα 1)
- εύρεση των εγγραφών μέσα σε ένα κύκλο με κέντρο ένα οποιοδήποτε (longitude, latitude) και ακτίνα r κάποιο συγκεκριμένο χρονικό διάστημα [t_min, t_max] (Εικόνα 1)
- εύρεση των k κοντινότερων εγγραφών που βρέθηκαν πιο κοντά σε ένα συγκεκριμένο σημείο μέσα σε ένα συγκεκριμένο χρονικό διάστημα
- k κοντινότερων εγγραφών με διαφορετικά id, που βρέθηκαν πιο κοντά σε ένα συγκεκριμένο σημείο μέσα σε ένα συγκεκριμένο χρονικό διάστημα



Εικόνα 1 Εύρεση Εγγραφών μέσα σε ένα ορθογώνιο ή έναν κύκλο

Η μέτρηση της αποδοτικότητας των αλγόριθμων έγινε με βάση 2 κριτήρια:

1) τον χρόνο εκτέλεσης του ερωτήματος στη βάση δεδομένων
2) τον αριθμό των εγγράφων της MongoDB, που εξετάστηκαν, προκειμένου να βρεθούν τα έγγραφα που ικανοποιούν όλους τους περιορισμούς

Για την επίτευξη των παραπάνω, χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python, καθώς το MongoDB Atlas που είναι μια cloud βάση δεδομένων στην οποία αποθηκεύτηκαν τα δεδομένα.

1.3 Δομή της εργασίας

Στη συνέχεια, παρουσιάζεται ο τρόπος με τον οποίο έχει δομηθεί η εργασία και τα περιεχόμενα των επόμενων ενοτήτων.

Κεφάλαιο 2 Ανασκόπηση Βιβλιογραφίας: Στην επόμενη ενότητα πραγματοποιείται μια βιβλιογραφική ανασκόπηση, στην οποία παρουσιάζονται οι τύποι των μη σχεσιακών βάσεων δεδομένων, κάποια σχετικά θεωρήματα και η σύνδεση τους με τα χωροχρονικά δεδομένα.

Κεφάλαιο 3 Περιγραφή Αλγόριθμων: Σε αυτή την ενότητα περιγράφονται όλοι οι αλγόριθμοι που χρησιμοποιήθηκαν στην εργασία και παρουσιάζεται ο κώδικας που υλοποιεί τους αλγόριθμους.

Κεφάλαιο 4 Αρχιτεκτονική Συστήματος: Σε αυτή την ενότητα παρουσιάζεται το πως συνδυάζονται οι παραπάνω αλγόριθμοι και η αρχιτεκτονική του τελικού συστήματος.

Κεφάλαιο 5 Πειραματική Μελέτη: Μετά την παρουσίαση του συστήματος εκτελούνται τα πειράματα και παρουσιάζονται τα αποτελέσματα.

Κεφάλαιο 6 Συμπεράσματα: Τέλος, παρουσιάζονται τα συμπεράσματα της εργασίας.

2.ΑΝΑΣΚΟΠΗΣΗ ΒΙΒΛΙΟΓΡΑΦΙΑΣ

2.1 ACID Ιδιότητες [1][3]

Το όνομα των ACID ιδιοτήτων προέκυψε από τα αρχικά των ιδιοτήτων που είναι Atomicity, Consistency, Isolation, Durability. Οι συγκεκριμένες ιδιότητες διαβεβαιώνουν ότι το αποτέλεσμα μιας συναλλαγής (transaction) σε μια SQL βάση δεδομένων θα είναι αξιόπιστο. Στη συνέχεια, παρουσιάζεται η επεξήγηση της κάθε ιδιότητας:

- **Atomicity:** Μια συναλλαγή ονομάζεται atomic όταν εκτελεστούν όλες ή καμία από τις προγραμματισμένες ενέργειες. Το atomicity ως ιδιότητα διασφαλίζει ότι αν ένα transaction διακοπεί τότε οι ενέργειες που είχαν εκτελεστεί ως εκείνη τη στιγμή ακυρώνονται και το transaction είναι σαν να μην εκτελέστηκε ποτέ.
- **Consistency:** Η εκτέλεση ενός transaction στη βάση δεδομένων επιστρέφει πάντα την πιο πρόσφατη τιμή.
- **Isolation:** Οι αλλαγές ου πραγματοποιούνται από την εκτέλεση ενός transaction δεν είναι ορατές στα υπόλοιπα transactions μέχρι αυτές να ολοκληρωθούν. Η συγκεκριμένη ιδιότητα διασφαλίζει ότι τα transactions θα εκτελεστούν σειριακά. Για παράδειγμα, σε ένα τραπεζικό σύστημα που μεταφέρει ένα χρηματικό ποσό από έναν λογαριασμό σε έναν άλλον, το isolation διασφαλίζει ότι τα υπόλοιπα transactions θα δουν αυτό το ποσό, είτε στον έναν λογαριασμό, είτε στον άλλον, αλλά ποτέ και στους δύο λογαριασμούς ή σε κανέναν από τους δύο.
- **Durability:** Το durability διασφαλίζει ότι αν ένα transaction ολοκληρωθεί, οι όποιες αλλαγές πραγματοποιήθηκαν στη βάση δεδομένων θα αποθηκευτούν ακόμα και αν στη συνέχεια υπάρξει κάποιο system failure.

Οι παραπάνω απαιτήσεις κατάφεραν για πάρα πολλά χρόνια να βρουν πεδίο εφαρμογής στα RDBMS (Relational DataBase Management Systems), όπου οι βάσεις δεδομένων ήταν μικρότερες, οριζόντια κλιμακωτές (horizontally scalable) και υπό τη μορφή σχήματος (schema driven). Πλέον, με την εξέλιξη της τεχνολογίας και την συνεχή και ολοένα αυξανόμενη ανάγκη για αποθήκευση δεδομένων στις βάσεις, το μοντέλο των σχεσιακών βάσεων δεδομένων, καθώς και οι παραπάνω ACID ιδιότητες δεν είναι αρκετές για να καλύψουν την αποθήκευση των Μεγάλων Δεδομένων.

Η σημερινή εποχή, η εποχή των Μεγάλων Δεδομένων, χαρακτηρίζεται από συνεχή αύξηση του όγκου των δεδομένων, καθώς και από αποθήκευση δεδομένων που βρίσκονται σε αδόμητη (unstructured) ή ημιδομημένη (semi-structured) μορφή, τα οποία φυσικά δεν ακολουθούν κάποια δομή σχεσιακών δεδομένων (relational data structure) και είναι αποθηκευμένα σε κατανεμημένα υπολογιστικά συστήματα (distributed computing systems).

Με βάση τα παραπάνω δεδομένα, λοιπόν, δημιουργήθηκαν καινούργιες βάσεις δεδομένων που ανταποκρίνονται στις νέες απαιτήσεις.

2.2 ΘΕΩΡΗΜΑ CAP

Σύμφωνα με το θεώρημα CAP (Consistency, Availability, Partition tolerance), γνωστό και ως θεώρημα του Brewer, καθώς διατυπώθηκε για πρώτη φορά από τον Eric Brewer το 2000, είναι αδύνατο για ένα σύστημα αποθήκευσης κατανεμημένων δεδομένων να ικανοποιεί ταυτόχρονα και τις τρεις ιδιότητες που παρουσιάζονται παρακάτω:

- **Consistency**: Η εκτέλεση ενός read στη βάση δεδομένων επιστρέφει πάντα την πιο πρόσφατη τιμή των δεδομένων, ανεξαρτήτως του πότε πραγματοποιήθηκε το τελευταίο write.
- **Availability**: Κάθε αίτημα (request) τερματίζει με μια απάντηση χωρίς σφάλματα (errors).
- **Partition Tolerance**: Ακόμα και αν η σύνδεση μεταξύ κάποιων κόμβων έχει διακοπεί, οι λειτουργίες στη βάση δεδομένων συνεχίζουν να πραγματοποιούνται.

Επομένως, όπως γίνεται κατανοητό όταν δημιουργείται ένα κατανεμημένο σύστημα, πρέπει να αποφασιστεί ποια από τις τρεις ιδιότητες δεν θα ικανοποιείται. Στη συνέχεια, παρουσιάζονται όλα τα δυνατά ζεύγη ιδιοτήτων που μπορούν να ικανοποιούνται και σε κάθε περίπτωση αποδεικνύεται γιατί πάντα μια ιδιότητα δεν θα μπορεί να συμπεριληφθεί στο σύστημα [1][3][8].

2.2.1 1^{Η} Περίπτωση: Consistency και Availability

Όταν ένα σύστημα είναι consistent και available, πρακτικά σημαίνει ότι για κάθε αίτημα (request) υπάρχει απάντηση, η οποία είναι και συνεπής (consistent). Από τη στιγμή που η απάντηση είναι συνεπής, υπάρχει συνεχείς επικοινωνία μεταξύ των servers, προκειμένου να υπάρχει εγγύηση για το αποτέλεσμα. Όταν ικανοποιείται και η διαθεσιμότητα (availability), η επικοινωνία δεν μπορεί να διακοπεί σε καμιά περίπτωση. Επομένως, στο σύστημα δεν υπάρχει partition tolerance γιατί αν υπήρχε θα έπρεπε το σύστημα είτε να επιστρέφει σφάλμα, δηλαδή να μην είναι διαθέσιμο (unavailable), είτε η απάντηση να μην είναι ενημερωμένη, δηλαδή να μην είναι συνεπής που με βάση τον ορισμό κάτι τέτοιο δεν μπορεί να ισχύει.

2.2.2 2^{Η} Περίπτωση: Consistency και Partition Tolerance

Όπως και στη πρώτη περίπτωση, εξαιτίας της συνέπειας (consistency) οι servers επικοινωνούν και οι βάσεις είναι συγχρονισμένες. Αλλά το σύστημα είναι και partition tolerant, που σημαίνει ότι οι servers μπορούν να σταματήσουν να επικοινωνούν. Όταν όμως αυτή η επικοινωνία διακοπεί, τα δεδομένα δεν θα είναι συγχρονισμένα και κατά συνέπεια δεν θα υπάρχει απάντηση εφόσον δεν θα υπάρχει consistency. Όταν όμως το σύστημα δεν μπορεί να απαντήσει δεν είναι διαθέσιμο (unavailable).

2.2.3 3Η Περίπτωση: Availability και Partition Tolerance

Όταν το σύστημα είναι partition tolerant υπάρχει δυνατότητα οι servers να μην επικοινωνούν και οι βάσεις να μην είναι συγχρονισμένες. Αλλά λόγω του availability η απάντηση δεν μπορεί να είναι κάποιο σφάλμα (error). Κατά συνέπεια, σε ένα request ο κάθε server θα παρουσιάζει την δική του εκδοχή και φυσικά δεν θα υπάρχει συνέπεια.

2.3 ΤΥΠΟΙ NOSQL ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Υπάρχουν 4 διαφορετικοί τύποι NOSQL βάσεων δεδομένων, οι οποίοι είναι οι εξής: key value databases, document stores databases, columnar databases, graph databases [3][8][9].

2.3.1 Key Value Databases

Οι key Value Databases όπως υποδηλώνει και το όνομά τους συνδυάζουν δύο χαρακτηριστικά. Ένα μοναδικό identifier που είναι το key και μια δομή δεδομένων που είναι το value που χαρακτηρίζεται από το key.

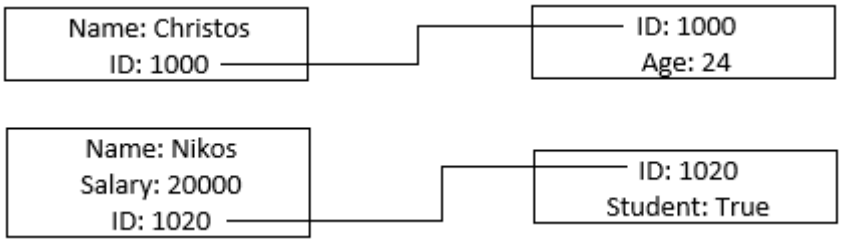
Αυτού του είδους οι βάσεις δεδομένων μοιάζουν με hash tables, όπου ο μοναδικός τρόπος να πραγματοποιηθεί ένα query στα δεδομένων είναι με τη βοήθεια του μοναδικού key. Όλα τα objects (key-value ζευγάρια) είναι replicated για μεγαλύτερο availability.

Key	Value
1	ID: 1000, Name: Tom, Age: 24
2	ID: 1004, Name: John, Age: 27
3	ID: 1012, Name: Mike, Age: 29

Μερικές από τις πιο δημοφιλείς Key Value Stores Databases είναι οι Redis, Amazon DynamoDB, Oracle NoSQL DB.

2.3.2 Document Databases

Σε αυτού του είδους τις NOSQL βάσεις δεδομένων τα στοιχεία που αποθηκεύονται είναι σε μορφή εγγράφων. Σε αυτές τις βάσεις αποθηκεύονται είτε unstructured είτε semi-structured έγγραφα. Με τον όρο «έγγραφα» υποδηλώνεται ένα σύνολο key-values, τα οποί σε μεγάλο βαθμό μοιάζουν με τις key-value databases που παρουσιάστηκαν παραπάνω. Κάθε βάση δεδομένων χρησιμοποιεί pointers για τα πεδία, όπως ακριβώς συμβαίνει με την τεχνική του κατακερματισμού (hashing)[2]. Επίσης, είναι πολύ σημαντικό το γεγονός ότι οι document databases δεν περιέχουν σχήμα. Στην Εικόνα 2.1 παρουσιάζεται ένα παράδειγμα document database:



Εικόνα 2 Παράδειγμα Document Database

Στη παραπάνω παράδειγμα παρουσιάζεται τόσο η δομή των εγγράφων και το πως συνδέονται μεταξύ τους, όσο και το γεγονός ότι είναι schema free.

Σε αυτή τη κατηγορία η MongoDB και η CouchDB είναι οι πιο δημοφιλείς βάσεις δεδομένων.

2.3.3 Column Oriented Databases

Σε αντίθεση με τους πίνακες των SQL βάσεων δεδομένων, οι οποίοι αποθηκεύουν κατά γραμμή (row-based storage)

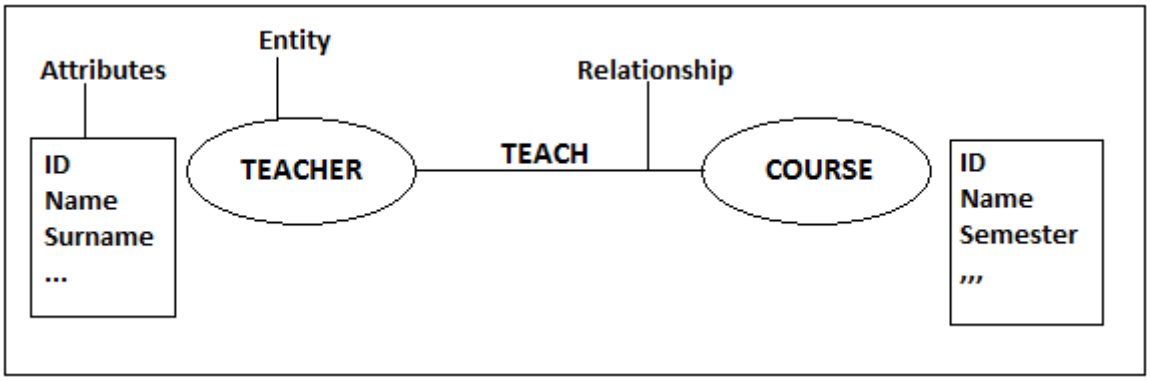
οι column oriented databases κάνουν σειριοποίηση (serialize) όλες τις τιμές ενός column και στη συνέχεια του επόμενου, όπως παρουσιάζεται στον παρακάτω πίνακα:

Name	01: Tom	02: John	03: Mike
Age	01: 24	02: 27	03: 29

Οι πιο δημοφιλείς Column Oriented Stores Databases είναι η HBase και η Cassandra.

2.3.4 Graph Databases

Οι Graph Databases βασίζονται στη θεωρία των γράφων, όπου ένας γράφος αποτελείται από κόμβους (nodes), ιδιότητες (properties) και ακμές (edges). Στις graph databases τα nodes, properties και edges αντιστοιχούν σε entities, attributes και relationships. Στο παρακάτω σχήμα παρουσιάζεται η μορφή ενός graph database:



Εικόνα 3 Παράδειγμα Graph Database

Οι πιο δημοφιλείς Graph Databases είναι η Neo4j και η AllegroGraph.

2.4 MongoDB

Η MongoDB είναι μια από τις πιο δημοφιλείς noSQL βάσεις δεδομένων. Ανήκει στην οικογένεια των Document Databases, που αυτό πρακτικά σημαίνει ότι δεν χρησιμοποιούνται πίνακες, όπως στις υπόλοιπες βάσεις, αλλά documents που η δομή τους μοιάζει περισσότερο με τη δομή των JSON αρχείων. Τα documents είναι αποθηκευμένα σε collections. Τα collections είναι containers για documents, που έχουν παρόμοια δομή χωρίς να είναι απαραίτητο να περιέχουν ακριβώς τα ίδια πεδία. Τέλος, τα collections είναι αποθηκευμένα σε databases [4].

Η MongoDB ως βάση δεδομένων δεν περιέχει ένα προκαθορισμένο schema κι αυτό παρουσιάζει πλεονεκτήματα ως προς το γεγονός ότι είναι δυνατή η διαχείριση και η αποθήκευση δεδομένων που το σχήμα τους αλλάζει πολύ συχνά. Επίσης, η έλλειψη σχήματος και η υποστήριξη αποθήκευσης εμφολευμένων εγγράφων (nested documents) οδηγεί σε πιο αποδοτικά ερωτήματα (queries), καθώς δεν απαιτείται η χρήση join από τη στιγμή που τα documents περιέχουν όλη την απαραίτητη πληροφορία.

Η διαχείριση της βάσης δεδομένων μπορεί να πραγματοποιηθεί είτε μέσα από το command shell, είτε με τη χρήση driver που προσφέρεται σε πολλές γλώσσες προγραμματισμού όπως Python, Java, Scala κλπ

Γενικότερα, οι noSQL βάσεις δεδομένων έχουν ως στόχο την αποθήκευση των Μεγάλων Δεδομένων. Για να επιτευχθεί αυτό κρίνεται αναγκαία η υποστήριξη της οριζόντιας κλιμάκωσης (horizontal scaling), που για τις σχεσιακές βάσεις δεδομένων είναι πολύ δύσκολο. Η MongoDB επιτυγχάνει την οριζόντια κλιμάκωση με χρήση του sharding που είναι μια μέθοδος κατανομής των δεδομένων σε πολλαπλά machines.

Τα query της MongoDB υποστηρίζουν τις CRUD (Create, Read, Update, Delete) λειτουργίες. Συνοπτικά, οι CRUD λειτουργίες είναι οι εξής:

Create: Με το Create προστίθεται ένα νέο document σε ένα collection. Αν το collection δεν υπάρχει δημιουργείται αυτόματα κατά την εισαγωγή του πρώτου document.

Read: Η λειτουργία Read χρησιμοποιείται για την ανάγνωση documents ή πεδίων των documents από ένα collection. Με τη χρήση διαφόρων φίλτρων μπορούν να διαβαστούν οι εγγραφές που ικανοποιούν διάφορες προϋποθέσεις.

Update: Η συγκεκριμένη λειτουργία χρησιμοποιείται για την τροποποίηση ενός document που ήδη υπάρχει σε ένα collection. Όπως και στη Read, με τη χρήση διαφόρων φίλτρων υπάρχει η δυνατότητα επιλογής συγκεκριμένων documents.

Delete: Τέλος, με την delete ο χρήστης μπορεί να διαγράψει ένα document από το collection.

2.5 MongoDB DATA INDEXING

Όπως ήδη έχει αναφερθεί, για τις ανάγκες τις εργασίας χρησιμοποιήθηκε η noSQL βάση δεδομένων MongoDB, καθώς επίσης και το γεγονός ότι για όλα τα queries μετρήθηκε ο χρόνος εκτέλεσης και τα documents που εξετάζουν προκειμένου να επιστρέψουν τα αποτελέσματα. Στην προσπάθεια που πραγματοποιήθηκε για να βελτιωθούν αυτές οι δύο μεταβλητές χρησιμοποιήθηκαν κάποια indexes. Ως ευρετήριο (index) ορίζεται μια δομή που μπορεί να επιταχύνει το χρόνο εκτέλεσης ενός query. Αυτό επιτυγχάνεται με τη διάταξη των δεδομένων σε συγκεκριμένη σειρά, έτσι ώστε κατά τη διαδικασία της αναζήτησης να μην είναι απαραίτητος ο έλεγχος όλων των δεδομένων. Όπως εύκολα γίνεται κατανοητό, το μειονέκτημα των indexes εμφανίζεται κατά τη διαδικασία της εισαγωγής νέων δεδομένων στη βάση, καθώς είναι αναγκαίο η πραγματοποίηση ενός update στο index, με συνέπεια να αυξάνεται ο χρόνος εκτέλεσης του query [4].

Η MongoDB δημιουργεί ένα document id by default που είναι μοναδικό για κάθε νέο document που δημιουργείται. Σε αυτό το id by default υπάρχει ένα index που επιταχύνει την αναζήτηση με βάση το id. Στη συνέχεια παρουσιάζονται όλα τα indexes που υποστηρίζει η MongoDB.

Single Key Index: Ομοίως, με το index στο document id, είναι δυνατή η δημιουργία ενός index σε οποιοδήποτε πεδίο ενός collection και οι εγγραφές του πεδίου να τοποθετηθούν είτε σε αύξουσα, είτε σε φθίνουσα σειρά.

Compound Index: Το compound index λειτουργεί όπως το Single Key Index, με τη διαφορά ότι χρησιμοποιείται για δύο ή περισσότερα πεδία ενός collection. Το Compound Index μπορεί να χρησιμοποιηθεί για μέχρι και 32 πεδία.

Text Index: Το συγκεκριμένο index χρησιμοποιείται για γρηγορότερη αναζήτηση με βάση ένα πεδίο που περιέχει ένα string ή ένα array από string.

MultiKey Index: Το MultiKey Index χρησιμοποιείται για πεδία, που το value είναι array. Συνεπώς, με το συγκεκριμένο index γίνεται μια αντιστοίχιση σε κάθε στοιχείο του πίνακα ξεχωριστά. Τέλος, η MongoDB αυτόματα αποφασίζει αν το value του πεδίου είναι array.

Geospatial Index: Στη συγκεκριμένη κατηγορία ανήκουν δύο indexes. Αξίζει να σημειωθεί ότι η MongoDB στηρίζεται στο GeoHash για τα Geospatial ευρετήρια:

1. **2d Index:** Το 2d Index χρησιμοποιείται για υπολογισμό της ευκλείδειας απόστασης σε έναν πίνακα 2 στοιχείων, που ουσιαστικά είναι το latitude και το longitude.
2. **2dsphere Index:** Ομοίως, με το 2d Index με τη διαφορά ότι δεν υπολογίζεται η ευκλείδεια απόσταση, δηλαδή η απόσταση σε έναν πίνακα δύο διαστάσεων, αλλά την απόσταση σε συνάρτηση με την καμπυλότητα της γης.

Hashed Index: Το συγκεκριμένο index χρησιμοποιείται στον κατατεμαχισμό (hash) μιας τιμής ενός πεδίου. Το συγκεκριμένο index υποστηρίζει μόνο match queries και όχι range-based queries.

Τα indexes της MongoDB παρουσιάζουν κάποιες ιδιότητες. Όλα τα indexes μπορούν να παρουσιάζουν μία ή περισσότερες από τις ιδιότητες που παρουσιάζονται παρακάτω:

Unique Indexes: Με τη unique ιδιότητα, το πεδίο στο οποίο εφαρμόζεται το index δεν μπορεί να έχει duplicate values. Όπως ήδη έχει αναφερθεί το primary key που δημιουργείται by default είναι unique.

Partial Indexes: Τα partial indexes εφαρμόζονται μόνο σε documents που ικανοποιούν κάποια φίλτρα.

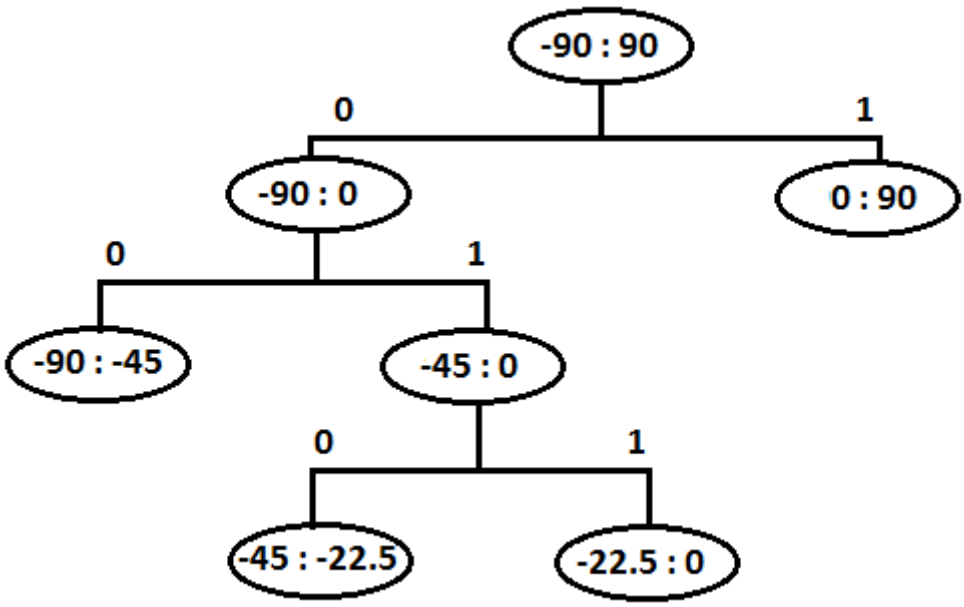
Sparse Indexes: Η sparse ιδιότητα βεβαιώνει ότι το index περιέχει entries μόνο για documents που περιλαμβάνουν το πεδίο που έχει εφαρμοστεί το index.

TTL Indexes: Τα TTL indexes χρησιμοποιούνται από τη MongoDB έτσι ώστε ένα document να διαγράφεται αυτόματα από ένα collection μετά από ένα συγκεκριμένο χρονικό διάστημα.

2.6 ST-HASH INDEX

Το ST-Hash Index είναι ένα Index που εφαρμόζεται σε χωροχρονικά δεδομένα. Αποτελεί μια επέκταση του GeoHash, που αποτελεί ένα αποτελεσματικό Index όσο αφορά 2d χωρικά δεδομένα. Στο ST-Hash Index έχει προστεθεί και η διάσταση του χρόνου. Το ST-Hash ως αλγόριθμος έχει ως σκοπό να μετατρέψει αυτά τα χωροχρονικά δεδομένα σε 1d string [5].

Το πρώτο βήμα του αλγόριθμου είναι η μετατροπή των συντεταγμένων και του χρόνου σε binary κωδικοποίηση. Όσο αφορά τις συντεταγμένες, αυτό επιτυγχάνεται με τη δημιουργία ενός δέντρου απόφασης που πάντα χωρίζει το διάστημα, μέσα στο οποίο βρίσκεται το latitude ή το longitude, στη μέση και ανάλογα με το που βρίσκονται οι συντεταγμένες κάθε φορά παίρνουν την τιμή 0 ή 1. Βασική προϋπόθεση είναι να έχει αποφασιστεί το ύψος (h) του δέντρου. Στο παρακάτω σχήμα παρουσιάζεται το δέντρο απόφασης με βάση το οποίο πραγματοποιείται η binary κωδικοποίηση του lon = -10 σε ένα δέντρο απόφασης με h = 3.



Εικόνα 4 Δέντρο απόφασης με h = 3

Όπως είναι εμφανές η τιμή του lon = -10 θα είναι ίση με 011. Με τον ίδιο τρόπο κωδικοποιείται και το latitude. Όσο αφορά την κωδικοποίηση του time, ένας χρόνος αποτελείται από 365x24x60 = 525600 min, εκτός και αν είναι δίσεκτος που τότε θα αποτελείται από 366x24x60 = 527040 min. Επομένως, χωρίς να ληφθεί υπόψη το έτος βρίσκονται τα min που αντιστοιχούν στη συγκεκριμένη ημερομηνία. Για παράδειγμα στις ημερομηνίες «2015-01-01 00:50:00» και «2016-01-01 00:50:00» τα minutes από την αρχή του έτους είναι 50. Το έτος δεν λαμβάνεται υπόψη, καθώς θα περιλαμβάνεται στο τελικό string. Στην συνέχεια έχοντας τα minutes, δηλαδή έναν αριθμό στο δεκαδικό σύστημα είναι πολύ εύκολο να μετατραπεί σε binary κωδικοποίηση.

Έχοντας δημιουργήσει 3 ακολουθίες από bits, που αντιστοιχούν στις 3 διαστάσεις, είναι δυνατό να προκύψει μια λίστα από bits με την ένωση των τριών ακολουθιών. Το τελευταίο βήμα είναι η μετατροπή της λίστας με τα bits σε ένα Base64 string. Φυσικά, μένει να απαντηθεί το ερώτημα από πόσους χαρακτήρες θα αποτελείται το τελικό string. Η απάντηση βρίσκεται στην ισότητα h = 2l όπου h είναι το ύψος του binary δέντρου απόφασης και l το πλήθος των χαρακτήρων. Συνεπώς, αν για παράδειγμα το ύψος του δέντρου είναι ίσο με 10, το τελικό string θα αποτελείται από 5 χαρακτήρες. Η κωδικοποίηση ολοκληρώνεται με την προσθήκη του χρόνου πριν από το string. Αν για παράδειγμα, το έτος είναι το 2019 και το string το «Re+BP», η τελική κωδικοποίηση θα είναι 2019-Re+BP.

3.ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΩΝ

Σε αυτό το κεφάλαιο περιγράφονται οι αλγόριθμοι που χρησιμοποιήθηκαν στην εργασία. Τα ερωτήματα που απαντήθηκαν αφορούν χωροχρονικά δεδομένα. Τα βασικά χαρακτηριστικά που έχουν τα δεδομένα που μελετήθηκαν είναι οι συντεταγμένες καθώς και η χρονική στιγμή που συνδέεται με το αντίστοιχο σημείο (συντεταγμένες). Κάθε έγγραφο στη βάση δεδομένων περιέχει ένα id, τις συντεταγμένες και την αντίστοιχη χρονική στιγμή. Ο σκοπός των παρακάτω αλγόριθμων είναι να απαντήσουν κάποια ερωτήματα σχετικά με χωροχρονικά δεδομένα και είτε με τη χρήση index είτε με εφαρμογή κάποιων άλλων αλγόριθμων να βελτιωθεί ο χρόνος εκτέλεσης του query, καθώς και ο αριθμός των εγγράφων που εξετάζονται.

Η βάση δεδομένων που χρησιμοποιήθηκε είναι η MongoDB. Ένα από τα πλεονεκτήματα της MongoDB, όσο αφορά τα χωροχρονικά δεδομένα, είναι ότι με χρήση της συνάρτησης geoWithin(), υποστηρίζονται τα ερωτήματα Box Range Query και Circle Range Query. Συνεπώς, μπορεί να βρεθεί το τελικό αποτέλεσμα με μόνο ένα query. Το μειονέκτημα είναι ότι το ερώτημα για την εύρεση των κοντινότερων γειτόνων δεν υποστηρίζεται και ως εκ τούτου είναι απαραίτητος ο σχεδιασμός μιας πιο πολύπλοκης λύσης. Για αυτό το λόγο εφαρμόστηκαν δύο λύσεις όπως παρουσιάζεται στη συνέχεια.

Τα ερωτήματα που απαντήθηκαν είναι τα εξής:

- **k-nearest neighbour:** Στο συγκεκριμένο ερώτημα αναζητούνται τα k κοντινότερα σημεία σε ένα σημείο (x, y) μέσα σε ένα χρονικό διάστημα [t_min, t_max]. Τα ερωτήματα απαντήθηκαν με δύο τρόπους:
    1. Ο πρώτος τρόπος περιλαμβάνει έλεγχο μόνο με βάση τον χρόνο και στη συνέχεια εύρεση των k κοντινότερων γειτόνων από τα σημεία που πέρασαν με επιτυχία τον έλεγχο. Πρόκειται για έναν σχετικά μη αποδοτικό τρόπο καθώς δεν χρησιμοποιείται κανένας χωρικός περιορισμός, σε πολλές περιπτώσεις επιστρέφεται μεγάλος αριθμός δεδομένων (υπάρχει περίπτωση όλα τα δεδομένα να ικανοποιούν τον χρονικό περιορισμό) και στη συνέχεια στη μνήμη για όλα αυτά τα δεδομένα πρέπει να πραγματοποιηθεί μέτρηση της απόστασης και σύγκριση μεταξύ τους.
    2. Ο δεύτερος τρόπος περιλαμβάνει χρήση στατιστικών στοιχείων. Κατά την εισαγωγή των δεδομένων στη βάση, ο χώρος που βρίσκονται τα δεδομένα χωρίζεται σε κελιά και δημιουργείται ένα αρχείο που δείχνει πόσα δεδομένα βρίσκονται σε κάθε κελί. Αυτό έχει ως αποτέλεσμα όταν το σημείο για το οποίο γίνεται η αναζήτηση των κοντινότερων γειτόνων να βρίσκεται σε ένα κελί με πολλά σημεία να ελέγχονται μόνο τα σημεία που βρίσκονται σε αυτό και στα γειτονικά κελιά και όχι όλα τα σημεία της βάσης δεδομένων. Θεωρητικά, αυτός ο τρόπος είναι περισσότερο αποδοτικός, καθώς δεν εξετάζονται όλα τα δεδομένα.
- **k-nearest neighbour με unique id:** Πρόκειται για το ίδιο ερώτημα με το k-nearest neighbour με τη διαφορά ότι τα k σημεία που θα είναι οι κοντινότεροι γείτονες πρέπει να προέρχονται από διαφορετικές οντότητες, δηλαδή να έχουν διαφορετικό id. Όπως και το παραπάνω ερώτημα, έτσι κι αυτό απαντήθηκε με τους δύο τρόπους που περιγράφονται παραπάνω. Φυσικά, σε σύγκριση με τα αντίστοιχα ερωτήματα του k-nearest neighbour απαιτείται περισσότερος χρόνος, καθώς και ο έλεγχος περισσότερων δεδομένων, καθώς είναι πολύ πιθανό οι k κοντινότεροι γείτονες να περιέχουν το ίδιο id.

- **Box Range Query:** Σε αυτό το ερώτημα δίνεται ένα ορθογώνιο και ένα χρονικό διάστημα και αναζητούνται όλα τα σημεία που βρέθηκαν μέσα στο ορθογώνιο το συγκεκριμένο χρονικό διάστημα.
- **Circle Range Query:** Όπως στο Box Range Query, αναζητούνται τα σημεία που βρέθηκαν μέσα σε έναν κύκλο ένα συγκεκριμένο χρονικό διάστημα.
- **Aggregate Range Query:** Η διαδικασία εύρεσης των δεδομένων είναι ίδια όπως τα box και circle range query. Η διαφορά είναι ότι στο τέλος εκτελείται κάποια ενέργεια στα δεδομένα που βρέθηκαν όπως για παράδειγμα να επιστρέφεται στον χρήστη το άθροισμα όλων των δεδομένων.

Όλα τα ερωτήματα απαντήθηκαν με 4 διαφορετικούς τρόπους όσο αφορά τη χρήση index και φυσικά πραγματοποιήθηκε σύγκριση των αποτελεσμάτων που παράγει η κάθε λύση. Οι λύσεις που εφαρμόστηκαν είναι οι εξής:

- Χωρίς κανένα index
- Με index στο χρόνο
- Με index στις συντεταγμένες
- Με index στις συντεταγμένες και στο χρόνο

Επίσης, μια παραλλαγή των ερωτημάτων είναι να μην επιστρέφονται τα id, αλλά για κάθε id να επιστρέφεται όλο το trajectory, δηλαδή όλες τα ζεύγη [συντεταγμένες, χρονική στιγμή]. Αυτό μπορεί να απαντηθεί με δύο τρόπους:

1) για κάθε id να πραγματοποιείται μια αναζήτηση στο collection και να επιστρέφονται όλα τα documents που έχουν το ίδιο id. Για παράδειγμα, αν το collection μοιάζει με το collection που ακολουθεί
   {{id : 1, lon : 50, lat : 50, t : 1034324567},
   {id : 2, lon : 42, lat : 45, t : 1036434781},
   {id : 1, lon : 51, lat : 52, t : 1034324659}
   {…}}
   τότε για κάθε id που αναζητείται πρέπει να ελέγχονται τα id όλων των εγγράφων.
2) να δημιουργηθεί ένα δεύτερο collection, στο οποίο κάθε document θα περιέχει όλο το trajectory. Με αυτό το τρόπο γίνεται αναζήτηση στο δεύτερο collection ενός document που θα περιέχει το συγκεκριμένο id. Αυτό το δεύτερο collection μπορεί να παρομοιαστεί με τον εξής πίνακα:
   {1: [{50, 50, 1034324567}, {51, 52, 1034324659}, …], 2: [{42, 45, 1036434781}, …]}

Από τους δύο παραπάνω τρόπους, τα καλύτερα αποτελέσματα παρουσιάζονται με τον δεύτερο τρόπο που έχει συγκεντρωμένα όλα τα στοιχεία ενός id σε μια μόνο εγγραφή. Το μειονέκτημα του πρώτου τρόπου είναι ότι πρέπει να ψάξει όλη τη βάση για να βρει τις εγγραφές που έχουν το ίδιο id.

Τέλος, αξίζει να σημειωθεί ότι όλα τα πειράματα πραγματοποιήθηκαν με πολλές διαφορετικές παραμέτρους.

3.1 Σύνδεση στη βάση δεδομένων

```
def connect_to_mongodb(MONGO_HOST, MONGO_PORT, MONGO_DB,
MONGO_USER, MONGO_PASSWORD, collection_name):
      client = MongoClient(MONGO_HOST, MONGO_PORT)
      db = client[MONGO_DB]
      db.authenticate(MONGO_USER, MONGO_PASSWORD)
      print("Connection Succeed")
      return db[collection_name]
```

Η παραπάνω συνάρτηση πραγματοποιεί την σύνδεση στη βάση δεδομένων. Τα ορίσματα της συνάρτησης είναι τα host, port, database name, username, password, collection name. Η συνάρτηση επιστρέφει το collection της βάσης, έχοντας πραγματοποιήσει τη σύνδεση σε αυτό, έτοιμο για να χρησιμοποιηθεί από τις υπόλοιπες συναρτήσεις, οι οποίες θα παρουσιαστούν παρακάτω. Το # μπροστά από το db.authenticate υποδηλώνει ότι η συγκεκριμένη γραμμή έχει μπει σε σχόλιο, καθώς στα παραδείγματα που θα παρουσιαστούν η σύνδεση στη βάση δεν απαιτεί username και password.

Ένα παράδειγμα χρήσης της παραπάνω συνάρτησης είναι το εξής:

```
mycol = connect_to_mongodb('localhost', 27017, 'trajectories', '', '','trajectory')
```

Στο παραπάνω παράδειγμα το host ισούται με localhost, το port είναι το 27017, το όνομα της βάσης είναι trajectories, η σύνδεση στην βάση δεν απαιτεί username και password και από τη βάση θα χρησιμοποιηθεί το collection objects.

Η παραπάνω γραμμή μπορεί να χρησιμοποιηθεί είτε μέσα σε κάποια άλλη συνάρτηση, είτε το mycol να χρησιμοποιηθεί σαν όρισμα σε κάποια άλλη συνάρτηση.

3.2 Circle Range Query

Η Cirlcle Range συνάρτηση υπολογίζει πόσες εγγραφές υπήρξαν μέσα σε έναν κύκλο μια συγκεκριμένη χρονική στιγμή.

Τα ορίσματα της συνάρτησης είναι (x, y) που αντιστοιχούν στο κέντρο του κύκλου, r που αντιστοιχεί στην ακτίνα του κύκλου και (t_min, t_max) που είναι το χρονικό διάστημα από t_min ως t_max.

Η mongo περιλαμβάνει τον geospatial query operator $geowithin, ο οποίος μπορεί να ακολουθείται από κάποιους άλλους τελεστές όπως $center για κύκλο ή $box για ορθογώνιο και χρησιμοποιείται για να βρίσκονται συντεταγμένες που περιέχονται μέσα σε μια γεωγραφική περιοχή.

Επομένως, το query ζητάει όλες τις εγγραφές όπου το lon και το lat μέσα στον κύκλο με κέντρο (x, y) και ακτίνα r και παράλληλα ικανοποιείται και ο χρονικός περιορισμός.

```
def range_circle(mycol, x, y, r, t_min, t_max):

    query='{"$and":[{"coordinates":{"$geoWithin":{"$center":
[['+str(x)+','+str(y)+'],'+str(r)+']}}}
,{"t":{"$gt":'+str(t_min)+'}},{"t":{"$lt":'+str(t_max)+'}}]}'
```

```
    k1 = json.loads(query)
```

3.3 Box Range Query

Η Box Range συνάρτηση υπολογίζει πόσες εγγραφές υπήρξαν μέσα σε ένα ορθογώνιο ένα συγκεκριμένο χρονικό διάστημα.

Τα ορίσματα της συνάρτησης είναι (x_min, y_min) που αντιστοιχούν στην κάτω αριστερά γωνία του ορθογωνίου, (x_max, y_max) που αντιστοιχούν στην πάνω δεξιά γωνία του ορθογωνίου και (t_min, t_max) που είναι το χρονικό διάστημα από t_min ως t_max.

Ομοίως με το circle range query, εκτελείται πρώτα το query που βρίσκει τις εγγραφές που βρίσκονται μέσα στον ορθογώνιο και ικανοποιούν τον χρονικό περιορισμό.

```
def range_box(mycol, x_min, x_max, y_min, y_max, t_min, t_max):

   query = '{"$and":[{"coordinates":{"$geoWithin":
{"$box":[['+str(x_min)+','+str(y_min)+'] ,
['+str(x_max)+','+str(y_max)+']]}}},{"t":{"$gt":'+str(t_min)+'}},{"t
":{"$lt":'+str(t_max)+'}}]}'

  k1 = json.loads(query)
```

3.4 Box Range query aggregate

Ο παραπάνω αλγόριθμος μπορεί να εκτελεστεί και ως εξής: Εκτελούνται όλα τα βήματα μέχρι να βρεθούν όλες οι εγγραφές που ικανοποιούν τους περιορισμούς, με τη διαφορά ότι η συνάρτηση περιέχει μια ακόμα παράμετρο. Στο τέλος, η συνάρτηση δεν επιστρέφει όλες τις εγγραφές, αλλά ότι ορίζει η παράμετρος agg. Παρακάτω παρουσιάζεται η περίπτωση όπου το agg είναι ίσο με count και κατά συνέπεια η συνάρτηση επιστρέφει το πλήθος των εγγραφών.

```
def range_box(mycol, x_min, x_max, y_min, y_max, t_min, t_max):

  query = '{"$and":[{"coordinates":{"$geoWithin":
{"$box":[['+str(x_min)+','+str(y_min)+'] ,  ['+str(x_max)+','
+str(y_max)+']]}}},{"t":{"$gt":'+str(t_min)+'}},{"t":{"$lt":'+str(t_
max)+'}}]}'

  k1 = json.loads(query)

  cursor = mycol.find(k1)

  documents = []

  counter = 0

  for document in cursor:

       documents.insert(counter, document)
```

```
        counter += 1
  if agg == "count":
      return len(documents)
```

3.5 Εύρεση k κοντινότερων γειτόνων

Η k_neighbors συνάρτηση υπολογίζει τους k κοντινότερους γείτονες ενός σημείου (x,y) που ικανοποιούν ένα χρονικό περιορισμό, δηλαδή το t βρίσκεται ανάμεσα σε t_min και t_max.

Αρχικά η συνάρτηση λαμβάνει ως ορίσματα τα (mycol, x, y, k, t_min, t_max) που αντιστοιχούν στα εξής: το mycol αντιστοιχεί στη σύνδεση σε κάποιο collection της βάσης, όπως παρουσιάστηκε παραπάνω, x και y είναι οι συντεταγμένες για τις οποίες πρέπει να βρεθούν οι κοντινότεροι γείτονες. k είναι ο αριθμός των κοντινότερων γειτόνων και t_min, t_max το χρονικό διάστημα μέσα στο οποίο θα πραγματοποιηθεί η αναζήτηση των κοντινότερων γειτόνων.

Στη συνέχεια καλείται η παραπάνω συνάρτηση, η οποία πραγματοποιεί τη σύνδεση με τη βάση δεδομένων, όπως παρουσιάζεται παρακάτω:

```
def k_neghbors(mycol,x,y,k,t_min,t_max):
```

Η λογική που ακολουθείται είναι η εξής: Πρώτα βρίσκονται όλες οι εγγραφές που ικανοποιούν τον χρονικό περιορισμό δηλαδή βρίσκονται μεταξύ t_min και t_max και στη συνέχεια υπολογίζονται οι k κοντινότερες εγγραφές στο x και y. Ο υπολογισμός των εγγραφών που βρίσκονται μεταξύ t_min και t_max γίνεται με τις εξής εντολές:

```
query='{"$and":[{"t":{"$gt":'+str(t_min)+'}},
{"t":{"$lt":'+str(t_max)+'}}]}'
    k1 = json.loads(query)
    cursor = mycol.find(k1)
```

Στη συνέχεια τα αποτελέσματα αποθηκεύονται στη λίστα documents

```
 documents = []
 counter = 0
 for document in cursor:
        documents.insert(counter, document)
        counter += 1
```

Το επόμενο βήμα είναι για κάθε εγγραφή του documents να υπολογιστεί η Ευκλείδια απόσταση από τις παραμέτρους (x, y). Για να πραγματοποιηθεί η παραπάνω διαδικασία, οι παράμετροι (x, y) αποθηκεύονται σε έναν πίνακα b1 και τα στοιχεία (lon, lat) των εγγραφών του document σε έναν πίνακα b2. Με χρήση της συνάρτησης linalg.norm(b1-b2) του πακέτου numpy της Python

υπολογίζεται η ευκλείδια απόσταση και αποθηκεύεται σε μια λίστα distances μαζί με τον αύξων αριθμό της εγγραφής του document.

```
b1 = np.array((x,y))

distances = []

counter = 0

for doc in documents:

    b2 = np.array((doc['coordinates'][0],doc['coordinates'][1]))

    dist = np.linalg.norm(b1-b2)

    distances.append((counter,dist))

    counter += 1
```

Τέλος, κάνοντας μια ταξινόμηση στη λίστα distances με τις αποστάσεις σε αύξουσα σειρά, μπορεί να γίνει η επιλογή των k πρώτων εγγραφών και με χρήση του αύξων αριθμού πραγματοποιείται εύρεση των εγγραφών στη λίστα documents.

Στην Python, αν το k είναι μεγαλύτερο από το πλήθος των εγγραφών στο distances, θα υπάρξει σφάλμα. Επομένως, γίνεται και ο έλεγχος των δύο τιμών και αν το k είναι μεγαλύτερο, τότε επιστρέφονται όλες οι εγγραφές:

```
distances = sorted(distances, key=lambda distance: distance[1])

document = []

if len(distances) > k:

    for j in range(k):

        document.append(documents[distances[j][0]])

else:

    for j in range(len(distances)):

        document.append(documents[distances[j][0]])

return document
```

3.6 Εύρεση k κοντινότερων γειτόνων με μοναδικό id

Μια παραλλαγή της συνάρτησης που παρουσιάστηκε παραπάνω είναι να βρεθούν οι k κοντινότεροι γείτονες που θα έχουν διαφορετικό id. Πρακτικά αυτό σημαίνει ότι αν για ένα σημείο (x, y) βρεθούν τα 3 κοντινότερα σημεία, αυτά τα σημεία θα πρέπει να μην έχουν τα ίδια id ή διαφορετικά να μην είναι σημεία του ίδιου trajectory.

Ο αλγόριθμος είναι ίδιος με την παραπάνω συνάρτηση μέχρι το σημείο που γίνεται η ταξινόμηση στη λίστα distances με τη διαφορά ότι στη λίστα distances, αποθηκεύεται και το id της κάθε εγγραφής, εκτός από την απόσταση από το (x, y) και τον αύξων αριθμό.

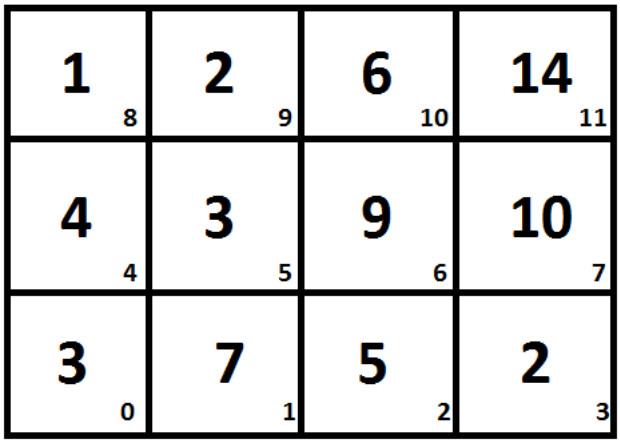
Στη συνέχεια, έχοντας ταξινομημένη τη λίστα, η λογική που ακολουθείται είναι να βρεθούν τα πρώτα k στοιχεία με διαφορετικό id.

Επομένως για κάθε στοιχείο της λίστας distances ελέγχεται η τελική λίστα document. Αν η λίστα δεν είναι άδεια, τότε συγκρίνονται τα δύο id, αν είναι διαφορετικά τότε το στοιχείο που αντιστοιχεί στη συγκεκριμένη εγγραφή του distances εισάγεται στην τελική λίστα. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να γίνει εισαγωγή των k στοιχείων.

```
for j in range(len(distances)):
        if len(document) < k:
            if document != []:
                for doc in document:
                    traj_object = documents[distances[j][0]]
                    if traj_object['id'] != doc['id']:
                        document.append(documents[distances[j][0]])
            else:
                document.append(documents[distances[j][0]])
        else:
            break
    return document
```

3.7 Εισαγωγή Δεδομένων στη Βάση και Αποθήκευση Στατιστικών Στοιχείων

Η λογική που ακολουθήθηκε στον συγκεκριμένο αλγόριθμο είναι η εξής: χωρίζοντας τον χώρο που βρίσκονται τα δεδομένα σε κελιά και αποθηκεύοντας τον αριθμό των εγγραφών που βρίσκονται σε κάθε κελί, κάποια ερωτήματα μπορούν να απαντηθούν με προσπέλαση λιγότερων εγγραφών από τη βάση δεδομένων (των εγγραφών που βρίσκονται στο συγκεκριμένο κελί ή και των γειτονικών κελιών). Όπως φαίνεται στο παρακάτω σχήμα, κάθε κελί περιέχει έναν συγκεκριμένο αριθμό εγγραφών. Αν αναζητηθούν οι 2 κοντινότεροι γείτονες ενός σημείου που βρίσκεται μέσα στο κελί 8, τότε δεν θα εξεταστούν όλα τα δεδομένα, αλλά μόνο τα δεδομένα που βρίσκονται μέσα στο κελί 8 και σε όλα τα γειτονικά κελιά, δηλαδή τα κελιά 4, 5, 9. Αυτά τα τέσσερα κελιά περιέχουν συνολικά 10 εγγραφές. Το μειονέκτημα αυτού του αλγόριθμου είναι ότι σε περίπτωση που αυτά τα δεδομένα δεν ικανοποιούν τον χρονικό περιορισμό, πρέπει να εξεταστούν τα επόμενα γειτονικά κελιά, δηλαδή τα κελιά 0, 1, 2, 6, 10.



Εικόνα 5 Δημιουργία κελιών στο χώρο με πληροφορία για τον αριθμό των εγγράφων

Αρχικά ο χρήστης μπορεί να εισάγει των αριθμό των κελιών στον άξονα x και στον άξονα y με τις εντολές:

```
xcells = int(input("number of xcells: "))
```

```
ycells = int(input("number of ycells: "))
```

και να διαβάσει το csv αρχείο με την εντολή read_csv του πακέτου pandas.

```
df = pd.read_csv(.csv file path)
```

Στη συνέχεια γίνεται εύρεση των μικρότερων και μεγαλύτερων lon και lat και αποθήκευση στις κατάλληλες μεταβλητές.

```
min_lon = min(df['lon'])
```

```
max_lon = max(df['lon'])
```

```
min_lat = min(df['lat'])
```

```
max_lat = max(df['lat'])
```

Πλέον, γνωρίζοντας το ορθογώνιο μέσα στο οποίο βρίσκονται όλες οι συντεταγμένες, καθώς και τον αριθμό των κελιών x και y, μπορεί να γίνει ο έλεγχος για κάθε εγγραφή έτσι ώστε να βρεθεί το κελί μέσα στο οποίο βρίσκεται.

Ο έλεγχος που πραγματοποιείται για κάθε εγγραφή είναι αν το lon είναι μεγαλύτερο από το `min_lon+x*(max_lon-min_lon)/xcells` που ουσιαστικά είναι η αριστερή πλευρά του ορθογωνίου με την αρίθμηση του x να ξεκινάει από το 0 και μικρότερο από το `min_lon+(x+1)*(max_lon-min_lon)/xcells` που είναι η δεξιά πλευρά. Ομοίως για το lat κάθε εγγραφής πρέπει να είναι μεγαλύτερο από το `min_lat+y*(max_lat-min_lat)/ycells` και μικρότερο από το `min_lat+(y+1)*(max_lat-min_lat)/ycells`.

Εφόσον βρεθεί το ορθογώνιο στο οποίο ανήκει η εγγραφή, εισάγεται στη βάση δεδομένων με τις εντολές:

```
mydict = { "id": str(df['sourcemmsi'][i]), "t": int(df['ts'][i]), "coordinates":[df['lon'][i], df['lat'][i]] }

mycol.insert_one(mydict)
```

Τα min και max lon και lat του ορθογωνίου, τα x και y που αναφέρονται στο κελί καθώς και ένας μοναδικός αύξων αριθμός για κάθε κελί αποθηκεύονται σε μια λίστα.

```
coords_list.append([y*xcells+x, lon_start, lon_end, lat_start, lat_end, x, y])
```

Όταν ολοκληρωθεί η διαδικασία και έχει πραγματοποιηθεί η εισαγωγή όλων των εγγραφών στη βάση δεδομένων, για κάθε κελί βρίσκεται το άθροισμα όλων των μοναδικών αριθμών που αναφέρονται σε αυτό και έχουν εισαχθεί στη λίστα.

Τέλος, σε ένα αρχείο statistics.txt για κάθε κελί εισάγονται τα εξής στοιχεία: άθροισμα των εγγραφών, min longitude, max longitude, min latitude, max latitude, x, y.

3.8 Εισαγωγή Δεδομένων σε Δεύτερο Collection

Όπως θα παρουσιαστεί παρακάτω, σε ορισμένες περιπτώσεις υπάρχει η ανάγκη να επιστραφεί ολόκληρο το trajectory και όχι μόνο μια εγγραφή αυτού. Συνεπώς, είναι περισσότερο αποδοτικό να δημιουργηθεί ένα δεύτερο collection, στο οποίο κάθε document θα περιέχει ολόκληρο το trajectory δηλαδή τα δεδομένα θα είναι της μορφής id και ένας πίνακας με όλες τις τριάδες (lon, lat, t). Αυτό το collection θα λειτουργήσει σαν index. Για την εισαγωγή κάθε δεδομένου στο δεύτερο collection καλείται η συνάρτηση:

```
def insert_into_col(col_idx, traj_id, t, lon, lat):
```

η οποία λαμβάνει ως ορίσματα το collection στο οποίο θα γίνει η σύνδεση καθώς και τα δεδομένα της εγγραφής (id, timestamp και συντεταγμένες).

Στη συνέχεια, πραγματοποιείται αναζήτηση σε αυτό το collection για τυχόν ύπαρξη του του traj_id και αν υπάρχει, η συγκεκριμένη εγγραφή αποθηκεύεται σε μια λίστα:

```
query = '{"id":"'+traj_id+'"}'
    k1 = json.loads(query)
    cursor = col_idx.find(k1)
    documents = []
    counter = 0
    for document in cursor:
        documents.insert(counter, document)
```

Αν η λίστα είναι άδεια, δηλαδή δεν υπάρχει το συγκεκριμένο id στο collection, τότε τα ορίσματα της συνάρτησης εισάγονται σαν ένα document στο collection.

```
if len(documents) == 0:
    col_dict = { "id": traj_id, "trajectories":[[t, lon, lat]] }
    col_idx.insert(col_dict)
```

Διαφορετικά το document που περιέχει το traj_id γίνεται update και στον πίνακα trajectories προστίθεται η εγγραφή [t, lon, lat]

```
else:
    trajectories = [t, lon, lat]
    old_query = {"id":traj_id}
    push = {"$push":{"trajectories":trajectories}}
    col_idx.update_one(old_query, push)
```

3.9 Εύρεση k κοντινότερων γειτόνων με χρήση στατιστικών στοιχείων

Για την εύρεση των k κοντινότερων γειτόνων θα χρησιμοποιηθεί το .txt αρχείο, στο οποίο έχει αποθηκευτεί ο αριθμός των εγγραφών που βρίσκονται σε κάθε κελί. Με αυτό τον τρόπο η αναζήτηση επικεντρώνεται σε ένα κελί ή και σε κάποια γειτονικά κελιά και ο αριθμός των εγγραφών που θα εξεταστούν είναι μικρότερος από το σύνολο όλων των εγγραφών.

Οι παράμετροι της συνάρτησης είναι το mycol με το οποίο έχει πραγματοποιηθεί η σύνδεση στη βάση δεδομένων, τα x και y που είναι οι συντεταγμένες στις οποίες θα αναζητηθούν οι κοντινότεροι γείτονες, το k που είναι ο αριθμός των κοντινότερων γειτόνων και τα tmin και tmax που είναι τα χρονικό διάστημα μέσα στα οποία θα πρέπει να βρίσκονται οι εγγραφές.

Αρχικά, διαβάζεται το αρχείο στο οποίο βρίσκονται τα δεδομένα και αποθηκεύονται σε ένα dataframe. Αυτό επιτυγχάνεται με την συνάρτηση read_csv του πακέτου pandas.

```
df = pd.read_csv('statistics.txt', header = None)
```

Τα δεδομένα που αποθηκεύτηκαν στο df έχουν την εξής μορφή:

	0	1	2	3	4	5	6
0	43	-8.926487	-6.936496	45.179054	46.587353	0	0
1	2	-4.946505	-2.956514	45.179054	46.587353	2	0
2	29	-2.956514	-0.966523	45.179054	46.587353	3	0
3	307	-6.936496	-4.946505	46.587353	47.995652	1	1
4	1682	-4.946505	-2.956514	46.587353	47.995652	2	1
5	425	-2.956514	-0.966523	46.587353	47.995652	3	1
6	287	-6.936496	-4.946505	47.995652	49.403951	1	2
7	1764	-4.946505	-2.956514	47.995652	49.403951	2	2
8	5	-2.956514	-0.966523	47.995652	49.403951	3	2
9	10	-6.936496	-4.946505	49.403951	50.812250	1	3
10	10	-4.946505	-2.956514	49.403951	50.812250	2	3
11	1	-2.956514	-0.966523	49.403951	50.812250	3	3

Εικόνα 6 Παράδειγμα στατιστικών δεδομένων

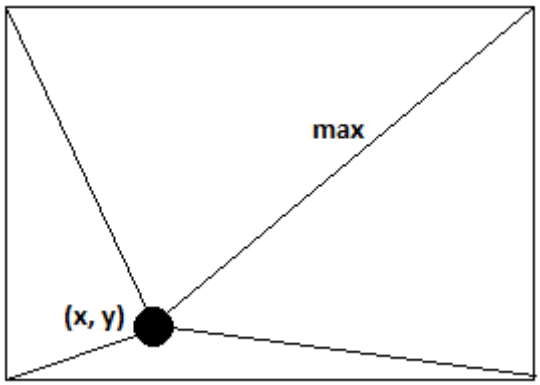
Η στήλη 0 παρουσιάζει τον αριθμό των εγγραφών που βρίσκονται σε κάθε κελί, οι στήλες 1 και 2 αναφέρονται στα min και max longitude, οι στήλες 3 και 4 στα min και max latitude και οι στήλες 5 και 6 στον αύξων αριθμό του κελιού στον οριζόντιο και κάθετο άξονα αντίστοιχα.

Στη συνέχεια από το df επιλέγεται η εγγραφή για την οποία η παράμετρος x της συνάρτησης είναι μεγαλύτερη από την τιμή της στήλης 1 και μικρότερη από την τιμή της στήλης 2 και αντίστοιχα η παράμετρος y που είναι μεγαλύτερη από την τιμή της στήλης 3 και μικρότερη από την τιμή της στήλης 4. Οι τιμές των στηλών 5 και 6 της εγγραφής που επιλέχθηκε αποθηκεύονται στις μεταβλητές xcell και ycell.

```
df1 = df.loc[(df[1]<=x) & (df[2]>=x) & (df[3]<=y) & (df[4]>=y)]

has_docs = True

xcell = int(df1[5])

ycell = int(df1[6])
```

Για τη συγκεκριμένη εγγραφή αν η τιμή της στήλης 0 είναι μεγαλύτερη από το k, δηλαδή στο συγκεκριμένο κελί βρίσκονται περισσότερες από k εγγραφές, τότε υπολογίζεται ποια γωνία του ορθογωνίου έχει τη μεγαλύτερη απόσταση από το (x, y).

```
if sum(df1[0]) >= k:
        dist = np.array((x,y))
        dist13= np.array((df1[1][df1.index[0]],df1[3][df1.index[0]]))
        dist24= np.array((df1[2][df1.index[0]],df1[4][df1.index[0]]))
        dist14= np.array((df1[1][df1.index[0]],df1[4][df1.index[0]]))
        dist23= np.array((df1[2][df1.index[0]],df1[3][df1.index[0]]))
         max_distance = max(np.linalg.norm(dist-dist13),\
                            np.linalg.norm(dist-dist24),\
                            np.linalg.norm(dist-dist14),\
                            np.linalg.norm(dist-dist23))
```



Εικόνα 7 Υπολογισμός της πιο μακρινής γωνίας ενός κελιού από το σημείο (x, y)

Μετά την εύρεση της γωνίας με τη μεγαλύτερη απόσταση εκτελείται η range_circle που παρουσιάστηκε παραπάνω. Η συγκεκριμένη συνάρτηση επιστρέφει όλες τις εγγραφές που βρίσκονται μέσα σε έναν κύκλο με κέντρο το (x, y) και ακτίνα ίση με την μεγαλύτερη απόσταση της γωνίας του ορθογωνίου.

```
documents = range_circle(mycol, x, y, max_distance, tmin, tmax)
```

Τέλος, για όλες τις εγγραφές που επέστρεψε η range_circle υπολογίζεται η απόσταση από το (x, y),

τα αποτελέσματα ταξινομούνται από το μικρότερο στο μεγαλύτερο

```
b1 = np.array((x,y))
distances = []
counter = 0
for doc in documents:
        b2 = np.array((doc['coordinates'][0],doc['coordinates'][1]))
        dist = np.linalg.norm(b1-b2)
        distances.append((counter,dist))
        counter += 1
distances = sorted(distances, key=lambda distance: distance[1])
```

και αν τα αποτελέσματα είναι περισσότερα ή ίσα με k επιλέγονται τα πρώτα k αποτελέσματα.

```
document = []
if len(distances) >= k:
    for j in range(k):
        document.append(documents[distances[j][0]])
    has_docs = False
```

```
    return document
```

Στην περίπτωση που ο αριθμός των εγγραφών που βρίσκονται στο συγκεκριμένο κελί είναι μικρότερος από k ή το query που εκτελέστηκε για το συγκεκριμένο κελί επέστρεψε λιγότερα από k αποτελέσματα, επιλέγονται όλα τα γειτονικά κελιά και η διαδικασία επαναλαμβάνεται για το ορθογώνιο που θα σχηματιστεί από όλα τα κελιά. Η διαδικασία επαναλαμβάνεται μέχρι ο αριθμός των εγγραφών που βρίσκονται στο ορθογώνιο που σχηματίζεται να είναι μεγαλύτερος από k. Στο παρακάτω παράδειγμα το i παίρνει τιμές από 1 ως 4, καθώς όταν έγινε ο διαχωρισμός του χώρου που βρίσκονται οι συντεταγμένες σε κελιά, θεωρήσαμε ότι ο μεγαλύτερος αριθμός κελιών στον x και στον y άξονα είναι 4.

```
if sum(df1[0])<k or has_docs==True:
        for i in range(1, 4):
           df2 = df.loc[(df[5]>=xcell-i) & (df[5]<=xcell+i) &
           (df[6]>=ycell-i) & (df[6]<=ycell+i)]
           if sum(df2[0]) >= k:
           .
           .
           .
           break
```

3.10 Εύρεση k κοντινότερων γειτόνων με διαφορετικά id με χρήση στατιστικών στοιχείων

Στη περίπτωση που αναζητούνται k κοντινότεροι γείτονες με διαφορετικά id, η λογική που ακολουθείται είναι ίδια με το παραπάνω ερώτημα, η διαφορά παρουσιάζεται μετά την ταξινόμηση των στοιχείων που επέστρεψε το query.

Όπως παρουσιάζεται στο παρακάτω τμήμα κώδικα δημιουργούνται δύο λίστες, η document στην οποία θα γίνει εισαγωγή των k κοντινότερων εγγραφών και η docs_id στην οποία θα γίνει εισαγωγή μόνο των ids των k κοντινότερων εγγραφών. Για κάθε στοιχείο του distances, αν η λίστα documents περιέχει λιγότερα από k στοιχεία τότε εξετάζονται δύο περιπτώσεις. Αν η documents περιέχει στοιχεία, συγκρίνονται τα αντίστοιχα ids του docs_id με το id του distances και αν δεν υπάρχει στο docs_id τότε πραγματοποιείται η εισαγωγή στο document. Διαφορετικά αν η document είναι άδεια, τότε εισάγεται το στοιχείο toy distances και το αντίστοιχο id στο docs_id.

```
distances = sorted(distances, key=lambda distance: distance[1])
document = []
docs_id = []
for j in range(len(distances)):
    if len(document) < k:
```

```
            if document != []:
                traj_object = documents[distances[j][0]]
                if traj_object['id'] not in docs_id:
                    docs_id.append(traj_object['id'])
                    document.append(documents[distances[j][0]])
            else:
                traj_object = documents[distances[j][0]]
                docs_id.append(traj_object['id'])
                document.append(traj_object)
        else:
            break
    if len(document) == k:
        return document
```

3.11 Εξαγωγή όλου του trajectory

Μια παραλλαγή των ερωτημάτων που παρουσιάστηκαν στην προηγούμενη ενότητα είναι στο τέλος για κάθε id που βρίσκεται, επιστρέφεται ολόκληρο το trajectory, δηλαδή όλες οι εγγραφές που ανήκουν στο συγκεκριμένο id. Προκειμένου να επιστραφεί όλο το trajectory τα παραπάνω ερωτήματα που επιστρέφουν μέρος αυτών δεν αλλάζουν καθόλου. Η διαφορά είναι στο γεγονός ότι υπάρχει ένα επιπλέον βήμα στο τέλος της διαδικασίας και αυτό είναι η χρήση των unique ids για την εύρεση ολόκληρου του trajectory.

Η εύρεση ολόκληρου του trajectory μπορεί να πραγματοποιηθεί με δύο τρόπους. Ο πρώτος τρόπος είναι για κάθε id που επέστρεψε κάποιο από τα παραπάνω ερωτήματα να αναζητηθούν όλα τα ίδια id στο collection. Πρόκειται για έναν σχετικά μη αποδοτικό τρόπο, καθώς απαιτεί τον έλεγχο όλων των δεδομένων. Φυσικά, η διαδικασία θα μπορούσε να επιταχυνθεί με τη χρήση ενός index στο id. Ο δεύτερος τρόπος απαιτεί τη δημιουργία ενός δεύτερου collection στο οποίο κάθε document περιλαμβάνει τα εξής δεδομένα id, [(lon, lat, t), (lon, lat, t)…], δηλαδή ένα id και στη συνέχεια έναν πίνακα με όλες τις τριάδες (lon, lat, t). Με αυτό το τρόπο τα documents που θα εξεταστούν είναι λιγότερα.

Και οι δύο τρόποι βασίζονται στον ίδιο κώδικα, η διαφορά τους είναι στο όρισμα που θα εισαχθεί στην τελευταία συνάρτηση, δηλαδή σε ποιο από τα δύο collections θα γίνει η σύνδεση.

Όπως παρουσιάζεται στο παρακάτω τμήμα κώδικα, τα δεδομένα που επιστρέφονται από τη συνάρτηση range_box() αποθηκεύονται στο results. Στη συνέχεια, όλα τα id αποθηκεύονται στη λίστα ids. Το ids μετατρέπεται σε dataframe και πλέον στο ids αποθηκεύονται όλα τα unique ids, δηλαδή κάθε id εμφανίζεται μόνο μια φορά. Για κάθε id της λίστας ids αποθηκεύονται οι τιμές “id”, η τιμή

του id και ":" στις λίστες key_list, value_list, operator_list αντίστοιχα. Αυτές οι τρεις λίστες εισάγονται ως ορίσματα στη συνάρτηση filter_or().

```
results = range_box(mycol, -
3.4032917,46.61752,0,48.9,1443687610,1943689790)

ids = []

for r in results:

    ids.append(r['id'])

ids = pd.DataFrame(ids)

ids = pd.unique(ids[0])

key_list = []

value_list = []

operator_list = []

for i in ids:

    key_list.append("id")

    value_list.append(i)

    operator_list.append(":")

results = filter_or(mycol1, key_list, value_list, operator_list)
```

Η συνάρτηση filter_or() αναζητά στη βάση όλα τα keys του key_list που συνδέονται με τις τιμές του value_list μέσω της αντίστοιχης τιμής του operator_list. Για παράδειγμα αν οι τιμές των key_list, value_list, operator_list είναι ["name", "age"], ["John", 27], [":", ">"] αντίστοιχα, τότε στο collection αναζητούνται όλα ta document που είτε το πεδίο name ισούται John είτε η τιμή του πεδίου age είναι μεγαλύτερη του 27. Επομένως, στη περίπτωση με τα trajectories που όλες οι τιμές του key_list είναι "id" και όλες οι τιμές του operator_list είναι όλες ":", αναζητούνται όλα τα id με κάποια από τις τιμές του value_list. Στη συνέχεια παρουσιάζεται ο κώδικας της συνάρτησης filter_or:

```
def filter_or(col,a,b,c):

    query = ''

    for i in range(len(a)):

        a[i] = '"'+a[i]+'"'

        if type(b[i]) == str:

            b[i] = '"'+b[i]+'"'

        if i == 0:

            if c[i] == ':':

                query += '{'+a[i]+c[i]+str(b[i])+'}'
```

```
        else:
            if c[i] == ':':
                query += ',{'+a[i]+c[i]+str(b[i])+'}'
```

Για κάθε εγγραφή του a, εφόσον γίνουν κάποιες αλλαγές στις τιμές του key και του value αν είναι απαραίτητες στη μεταβλητή query προστίθενται οι αντίστοιχες τιμές που περιέχουν οι τρεις λίστες. Όταν το i δεν ισούται με 0 τότε προστίθεται ένα κόμμα (,) πριν το query προκειμένου η δομή του να είναι σωστή. Φυσικά, στο filter_or υπάρχουν και οι περιπτώσεις που c[i] ισούται με (>=, >, <=, <) αλλά δεν έχουν συμπεριληφθεί στην παραπάνω περιγραφή καθώς απαιτείται μόνο η περίπτωση της ισότητας.

```
query = '{"$or":['+query+"]}"
    k1 = json.loads(query)
    cursor = col.find(k1)
    documents = []
    counter = 0
    for document in cursor:
        documents.insert(counter, document)
        counter += 1
    return documents
```

Τέλος, στο query προστίθεται το or, πραγματοποιείται η αναζήτηση και τα αποτελέσματα αποθηκεύονται σε μια λίστα.

Αν η σύνδεση έχει πραγματοποιηθεί στο αρχικό collection, τότε στο τέλος εκτελείται και το παρακάτω τμήμα κώδικα, που ουσιαστικά κάνει μια ομαδοποίηση των δεδομένων ανάλογα με το id στο οποίο ανήκουν.

```
results = pd.DataFrame(results)
results = results.groupby('id')
for traj_id, data in results:
    print(traj_id)
    print(data)
```

4.ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

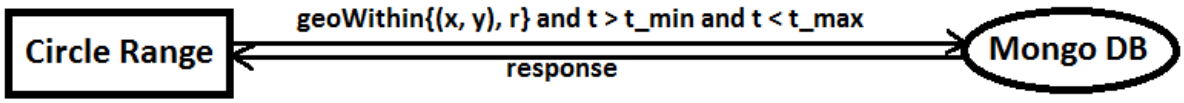
Σε αυτό το κεφάλαιο παρουσιάζεται συνοπτικά η αρχιτεκτονική του συστήματος. Συγκεκριμένα, παρουσιάζονται οι 4 βασικές συναρτήσεις, που είναι οι Circle Range, Box Range, k κοντινότεροι γείτονες, k κοντινότεροι γείτονες με χρήση στατιστικών δεδομένων. Οι υπόλοιποι αλγόριθμοι του κεφαλαίου 3, αποτελούν μέρος των βασικών συναρτήσεων.

4.1 Circle Range

Η συνάρτηση Circle Range δέχεται ως ορίσματα τις τιμές (x, y, r, t_min, t_max), δηλαδή αναζητούνται οι εγγραφές που βρίσκονται μέσα στον κύκλο με κέντρο (x,y) και ακτίνα r και στο χρονικό διάστημα [t_min, t_max]. Για το query χρησιμοποιήθηκε η συνάρτηση geoWithin της Mongo. Η συνάρτηση geoWithin χρησιμοποιείται για την εύρεση documents που βρίσκονται μέσα σε συγκεκριμένα σχήματα. Για αυτή την εργασία χρησιμοποιήθηκε ο κύκλος και το ορθογώνιο. Για τον κύκλο η σύναρτηση geowithin δέχεται μια παράμετρο την center η οποία αποτελείται από ένα πίνακα δύο στοιχείων που αντιστοιχούν στις συντεταγμένες και από έναν ακόμη αριθμό που αντιστοιχεί στην ακτίνα. Ένα παράδειγμα της geowithin για την εύρεση documents μέσα σε κύκλο με κέντρο [50, 50] και ακτίνα r = 10 είναι το εξής:

db.places.find({ loc: { $geoWithin: { $center: [[50, 50], 10] } } })

Συνεπώς, η geoWithin σε συνδυασμό με τα δύο φίλτρα για το χρονικό διάστημα επιστρέφει το τελικό αποτέλεσμα χωρίς να απαιτείται κάποια άλλη ενέργεια.



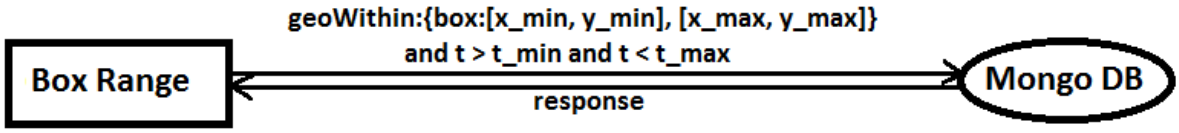
Εικόνα 8 Συνάρτηση Circle Range

4.2 Box Range

Ομοίως, η συνάρτηση Box Range δέχεται ως ορίσματα τις τιμές (x_min, y_min, x_max, y_max, t_min, t_max). Οι εγγραφές που επιστρέφονται πρέπει να βρίσκονται μέσα στο ορθογώνιο με μικρότερες συντεταγμένες (x_min, y_min) και μεγαλύτερες τις (x_max, y_max) και χρονικό διάστημα [t_min, t_max]. Όπως την Circle Range, έτσι και για την Box Range χρησιμοποιήθηκε η geoWithin συνάρτηση. Σε αυτή τη περίπτωση, η geowithin δέχεται μια παράμετρο την box, η οποία αποτελείται από δύο πίνακες με συντεταγμένες για τα [x_min, y_min] και [x_max, y_max]. Ένα παράδειγμα της geoWithin για την εύρεση των εγγραφών που βρίσκονται μέσα στο ορθογώνιο με γωνίες [0, 0] και [50, 50] είναι το εξής:

db.places.find({ loc: { $geoWithin: { $box: [[0, 0], [50, 50]] } } })

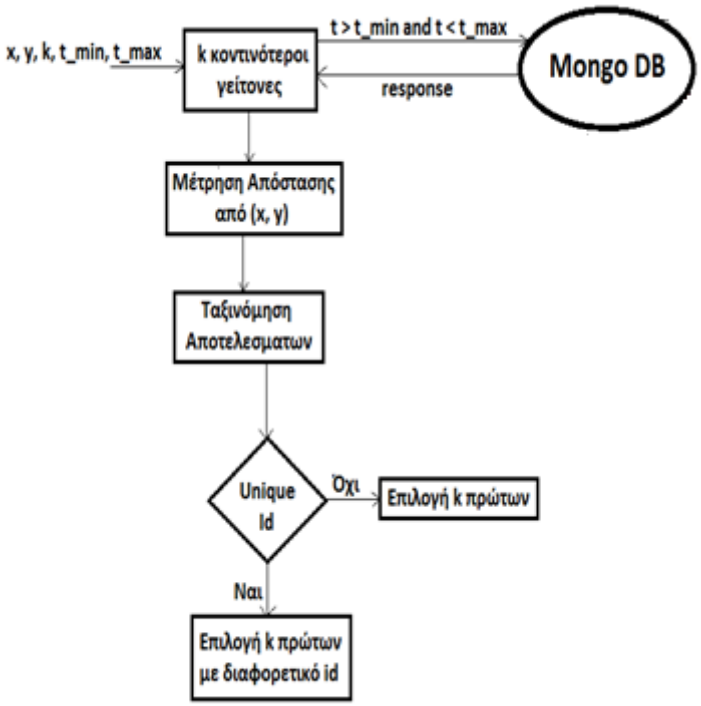
Όπως και στην geoWithin στην περίπτωση του κύκλου, έτσι και στην εύρεση στοιχείων μέσα σε ορθογώνιο, όταν χρησιμοποιηθούν και τα φίλτρα για το χρονικό διάστημα, το query επιστρέφει το τελικό αποτέλεσμα χωρίς να απαιτείται κάποια άλλη ενέργεια.



Εικόνα 9 Συνάρτηση Box Range

4.3 k Κοντινότεροι Γείτονες

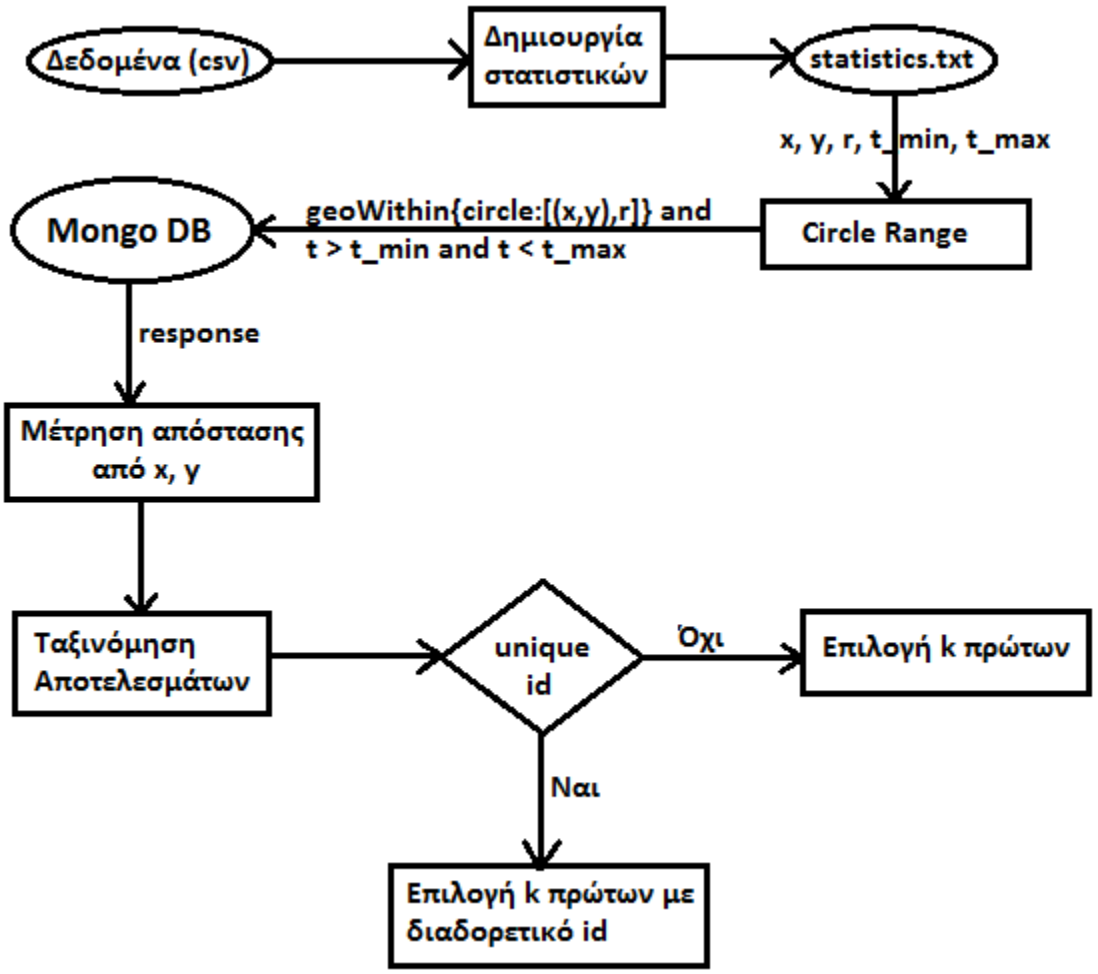
Αρχικά, η συνάρτηση των k κοντινότερων γειτόνων δέχεται ως ορίσματα τις τιμές (x, y, k, t_min, t_max) που αντιστοιχούν στις συντεταγμένες (x, y) για τις οποίες θα αναζητηθούν οι κοντινότεροι γείτονες, στον αριθμό των κοντινότερων γειτόνων (k) και στο χρονικό διάστημα (t_min, t_max) μέσα στο οποίο πρέπει να βρίσκονται οι συντεταγμένες. Αρχικά εκτελείται το query στη βάση δεδομένων, το οποίο περιλαμβάνει μόνο τα t_min και t_max, δηλαδή αναζητούνται οι εγγραφές που ικανοποιούν μόνο τον χρονικό περιορισμό. Στη συνέχεια για όλα τα αποτελέσματα (response) που θα επιστρέψει το query, πραγματοποιείται η μέτρηση της απόστασης από το (x, y) και η ταξινόμηση από το πιο κοντινό στο μακρινό. Τέλος, αν δεν υπάρχει ενδιαφέρον για το id των εγγραφών επιλέγονται οι k πρώτες. Διαφορετικά, αν οι εγγραφές πρέπει να είναι διαφορετικών οντοτήτων, επιλέγονται οι k πρώτες με διαφορετικό id.



Εικόνα 10 Συνάρτηση Εύρεσης k κοντινότερων γειτόνων χωρίς τη χρήση στατιστικών δεδομένων

4.4 k Κοντινότεροι Γείτονες Με Χρήση Στατιστικών Δεδομένων

Τέλος, η συνάρτηση για την εύρεση των κοντινότερων γειτόνων με τη χρήση των στατιστικών δεδομένων, διαβάζει το αρχείο με τα στατιστικά δεδομένα. Το συγκεκριμένο αρχείο μεταξύ άλλων πληροφοριών για τα δεδομένα περιέχει και έναν μοναδικό αύξων αριθμό για κάθε κελί, ο οποίος και εξάγεται για το κελί στο οποίο βρίσκονται οι συντεταγμένες (x, y). Έχοντας, τα δεδομένα για το κελί, εξάγεται η απόσταση από την πιο μακρινή γωνία, η οποία θα χρησιμοποιηθεί ως ακτίνα r για την κλήση της συνάρτησης Circle Range. Η πιο μακρινή γωνία του ορθογωνίου θα χρησιμοποιηθεί, εφόσον το οθογώνιο περιέχει περισσότερα από k στοιχεία. Σε διαφορετική περίπτωση, μέτρηση αφορά όλα τα γειτονικά κελιά. Σε κάθε περίπτωση, το επόμενο βήμα είναι να κληθεί η Circle Range. Στα αποτελέσματα που θα επιστρέψει η Circle Range, εφαρμόζονται τα ίδια βήματα, όπως στη συνάρτηση για την εύρεση των κοντινότερων γειτόνων χωρίς τη χρήση των στατιστικών δεδομένων, δηλαδή μετριέται η απόσταση για κάθε εγγραφή, τα αποτελέσματα ταξινομούνται και ανάλογα με το αν υπάρχει ενδιαφέρον για το id των εγγραφών, επιστρέφονται οι αντίστοιχες k εγγραφές.



Εικόνα 11 Συνάρτηση Εύρεσης k κοντινότερων γειτόνων με χρήση στατιστικών δεδομένων

5.ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ

Στην πειραματική μελέτη εξετάστηκαν οι αλγόριθμοι που περιγράφονται παραπάνω, δηλαδή οι αλγόριθμοι Box Range, Circle Range και k κοντινότεροι γείτονες. Στα πειράματα που πραγματοποιήθηκαν, η μελέτη επικεντρώθηκε στον αριθμό των εγγραφών της βάσης δεδομένων, που πρέπει να προσπελάσει ο κάθε αλγόριθμος προκειμένου να επιστρέψει το τελικό αποτέλεσμα, καθώς και στον χρόνο εκτέλεσης των queries.

Στα πειράματα που περιγράφονται σε αυτό το κεφάλαιο, όλα τα ερωτήματα εκτελέστηκαν χωρίς κανένα index, καθώς και με index στο χρόνο, στις συντεταγμένες και compound index, που, ουσιαστικά, είναι ο συνδυασμός χρόνου με συντεταγμένες. Επίσης, για το ερώτημα της εύρεσης των κοντινότερων γειτόνων, που στον χώρο δημιουργούνται κελιά, εξετάστηκαν περιπτώσεις όπου ο αριθμός των κελιών αλλάζει.

Προκειμένου να βρεθεί ο αριθμός των εγγραφών που εξετάζει ο κάθε αλγόριθμος χρησιμοποιήθηκε η συνάρτηση explain. Μετά την εκτέλεση της συνάρτησης find, εκτελέστηκε η εντολή print(cursor.explain()['executionStats']). Η συνάρτηση explain μπορεί να πάρει μία από τις 3 παρακάτω παραμέτρους:

- **queryPlanner:** Η MongoDB εκτελεί ένα query optimizer προκειμένου να αποφασίσει ποιο πλάνο είναι το καλύτερο και επιστρέφει πληροφορίες σχετικά με αυτό.
- **executionStats:** Το καλύτερο πλάνο εκτελείται και επιστρέφονται στατιστικά δεδομένα που περιγράφουν την εκτέλεση του.
- **allPlansExecution:** Το καλύτερο πλάνο εκτελείται και επιστρέφονται στατιστικά δεδομένα για αυτό, όπως επίσης στατιστικά δεδομένα επιστρέφονται και για όλα τα υποψήφια πλάνα.

Όπως είναι προφανές η παράμετρος για την οποία υπάρχει ενδιαφέρον είναι η «executionStats» και συγκεκριμένα τα πεδία που ονομάζεται «totalDocsExamined» και «executionTimeMillis». Η τιμή του πεδίου «totalDocsExamined» δείχνει πόσες εγγραφές προσπέλασε ο αλγόριθμος προκειμένου να βρεθεί το σωστό αποτέλεσμα. Το «executionTimeMillis» μετράει το χρόνο εκτέλεσης του query σε milliseconds.

Στα πειράματα πραγματοποιήθηκαν χωρίς τη χρήση index, καθώς επίσης και με τη χρήση index είτε στο πεδίο «t», δηλαδή στο χρόνο είτε στις συντεταγμένες ή και με τη χρήση και των δύο index. Οι εντολές για τη χρήση των index είναι οι εξής:

```
> db.trajectory.createIndex({t:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Εικόνα 12 Δημιουργία Index στο πεδίο t

Η τιμή «1» στο t ότι οι τιμές του πεδίου θα ταξινομηθούν σε αύξουσα σειρά, αν η τιμή ήταν «-1», τότε η σειρά θα ήταν φθίνουσα.

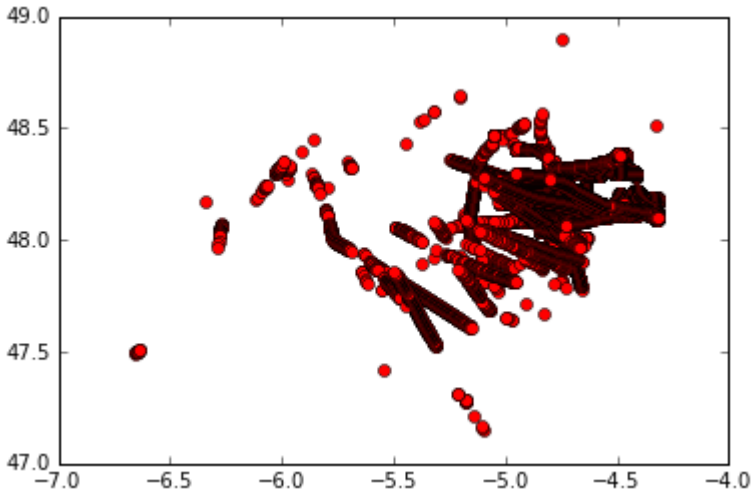
```
> db.trajectory.createIndex({coordinates:"2d"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Εικόνα 13 Δημιουργία Index στο πεδίο coordinates

Η τιμή «2d» δείχνει ότι το συγκεκριμένο πεδίο περιέχει ζεύγη τιμών όπως είναι οι συντεταγμένες. Φυσικά, επειδή το συγκεκριμένο index έχει σχεδιαστεί για συντεταγμένες, η πρώτη τιμή, το longitude, πρέπει να βρίσκεται στο διάστημα [-180,180] και το latitude στο [-90, 90]. Σε διαφορετική περίπτωση θα υπάρξει error.

5.1 Datasets

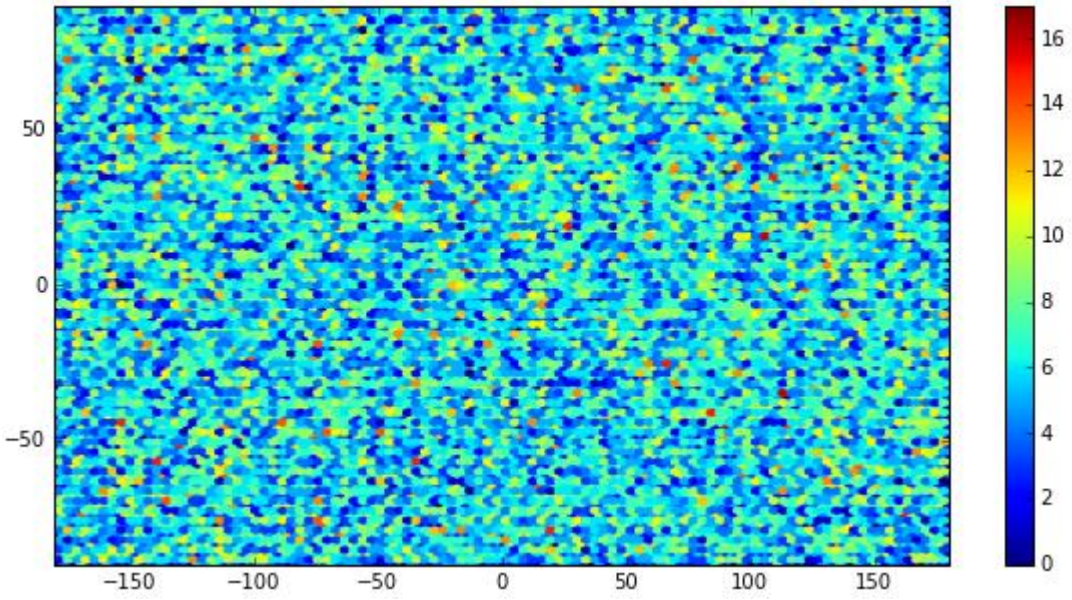
Το πρώτο dataset προέρχεται από ένα μέρος ενός dataset του zenodo repository. Ουσιαστικά, πρόκειται για 67.484 εγγραφές που ανήκουν σε 70 οντότητες, η συγκέντρωση των οποίων βρίσκεται μέσα στο ορθογώνιο που προκύπτει από τις συντεταγμένες (-7, 47) και (-4, 49) όπως παρουσιάζεται και στο παρακάτω σχήμα και το χρονικό διάστημα μέσα στο οποίο βρίσκονται οι εγγραφές είναι το (1443650402, 1443722123). Κάθε εγγραφή αποτελείται από τα εξής στοιχεία: sourcemmsi, altitude, speedoverground, courseoverground, lon, lat, ts. Τα στοιχεία που αποθηκεύτηκαν στη βάση δεδομένων είναι το sourcemmsi ως id, τα lon και lat που αφορούν τις συντεταγμένες και το ts που είναι το timestamp για κάθε εγγραφή.



Εικόνα 14 Συντεταγμένες εγγραφών πρώτου dataset

Το δεύτερο dataset δημιουργήθηκε με τυχαίο τρόπο χρησιμοποιώντας την random.uniform() συνάρτηση της Python. Η συγκεκριμένη συνάρτηση χρησιμοποιήθηκε με παραμέτρους (-180,180), (-90, 90) και (1400000000, 16000000000), οι οποίες αντιστοιχούν στο εύρος των τιμών των μεταβλητών lon, lat και time. Στη βάση δεδομένων αποθηκεύτηκαν 20 οντότητες, δηλαδή 20

διαφορετικά id με 5000 εγγραφές το καθένα, δηλαδή σύνολο 70000 εγγραφές. Στο παρακάτω γράφημα παρουσιάζεται η κατανομή των δεδομένων.



Εικόνα 15 Heatmap κατανομής δεδομένων

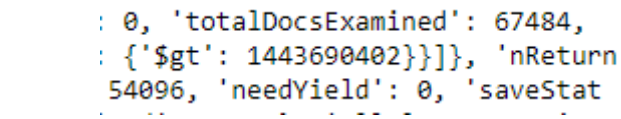
5.2 Πειραματική Μελέτη με χρήση του πρώτου dataset

5.2.1 Εύρεση k κοντινότερων γειτόνων

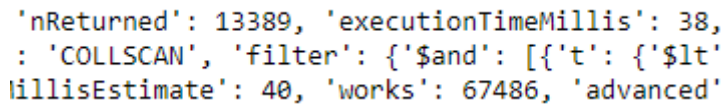
Για την εύρεση των k κοντινότερων γειτόνων εξετάστηκαν οι περιπτώσεις όπου ο χώρος δεν χωρίζεται σε κελιά και όταν χωρίζεται σε 4x4 κελιά, 10x10 κελιά, 1000x1000 κελιά και 2000x2000 κελιά. Σε όλες αυτές τις περιπτώσεις εκτελέστηκαν 4 πειράματα όσο αφορά το index (χωρίς καθόλου index, με index στις συντεταγμένες, με index στο χρόνο, με index στο χρόνο και στις συντεταγμένες).

Χωρίς τη χρήση στατιστικών δεδομένων

Ο πρώτος αλγόριθμος που μελετήθηκε ήταν ο k_neighbors χωρίς την χρήση στατιστικών δεδομένων καθώς επίσης και χωρίς την χρήση indexes. Όπως ήταν αναμενόμενο, ο συγκεκριμένος αλγόριθμος προκειμένου να επιστρέψει το τελικό αποτέλεσμα θα έπρεπε να προσπελάσει όλες τις εγγραφές. Οι παράμετροι του αλγόριθμου ήταν οι εξής: (-4.75, 48.25, 2, 1443690402, 1443700123), δηλαδή αναζητήθηκαν οι 2 κοντινότεροι γείτονες στο σημείο (-4.75, 48.25) που βρίσκονται στο χρονικό διάστημα [-4.75, 48.25]. Το σημείο δεν επιλέχθηκε τυχαία, καθώς όπως παρατηρείται στο παραπάνω γράφημα, υπάρχουν πάρα πολλά σημεία που βρίσκονται κοντά σε αυτό. Βέβαια, θεωρητικά θα υπάρξει βελτίωση όταν χρησιμοποιηθούν στατιστικά δεδομένα σε συνδυασμό με κάποιο index στο χώρο, καθώς όπως παρουσιάστηκε παραπάνω, ο συγκεκριμένος αλγόριθμος επιστρέφει μόνο τα δεδομένα που ικανοποιούν τον χρονικό περιορισμό. Παρακάτω παρουσιάζεται ένα μέρος του αποτελέσματος όπως αυτό παρουσιάστηκε από την συνάρτηση explain.



Ο αλγόριθμος εξέτασε 67484 εγγραφές και επέστρεψε μόλις 13389 σε 38 ms και η διάρκεια όλης της διαδικασίας ήταν 6.44 sec.

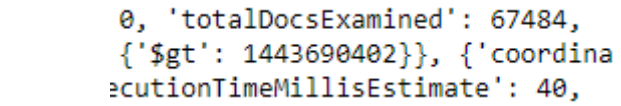


	Documents Examined	Execution Time (ms)	Documents Returned	Total Time (sec)
Without index	67484	38	13389	6.44
Time index	13389	15	13389	6.14
2d index	67484	38	13389	6.42
Compounding index	67484	34	13389	5.37

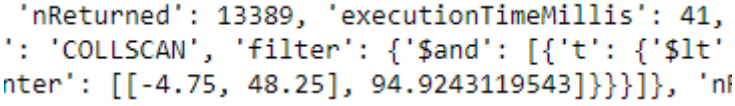
Και στα δύο παραπάνω διαγράμματα, όπως ήταν αναμενόμενο τα καλύτερα αποτελέσματα παρουσιάζονται με τη χρήση του index στο χρόνο, καθώς το query που εκτελείται εξετάζει μόνο το χρόνο και στη συνέχεια στα δεδομένα που επιστρέφονται, εξετάζονται οι παράμετροι στο χώρο. Χωρίς την χρήση του time index εξετάζονται όλα τα έγγραφα και ο χρόνος εκτέλεσης είναι σχετικά υψηλός. Συνεπώς, όταν δεν χρησιμοποιούνται στατιστικά δεδομένα τα καλύτερα αποτελέσματα παρουσιάζονται με τη χρήση του time index.

Με διαχωρισμό σε 4x4 κελιά

Χωρίς την χρήση indexes και ο δεύτερος αλγόριθμος k_neighbors_with_statistics, που χρησιμοποιεί στατιστικά δεδομένα επέστρεψε το ίδιο αποτέλεσμα, δηλαδή χρειάστηκε να ερευνήσει όλα τα έγγραφα προκειμένου να βρει το σωστό αποτέλεσμα. Για τον k_neighbors_with_statistics, ο χώρος χωρίστηκε σε 4x4 ίσα κελιά



Όπως και ο πρώτος αλγόριθμος, έτσι και ο k_neighbors_with_statistics επιστρέφει 13389 αποτελέσματα σε σχεδόν ίδιο χρόνο, 41 ms και η συνολική διάρκεια ήταν 6.21 sec.



Χωρίς τη χρήση κάποιου index στο χώρο ή στο χρόνο, θα εξετάζονται πάντα όλα τα έγγραφα από τον k_neighbors_with_statistics.

Στην περίπτωση που δημιουργηθεί ένα index στο πεδίο t, δηλαδή στο χρόνο, ο πρώτος αλγόριθμος εξετάζει 13389 εγγραφές, αυτές που επιστρέφει και ο χρόνος εκτέλεσης του query είναι τα 15ms και η διάρκεια ήταν 6.14 sec.

Ομοίως και ο k_neighbors_with_statistics εξετάζει 13389 εγγραφές και τις επιστρέφει όλες, αλλά αυτό γίνεται σε 21ms. Η διαφορά είναι πολύ πιθανό να οφείλεται στο γεγονός ότι ο δεύτερος αλγόριθμος εξετάζει και τους χωρικούς περιορισμούς, καθώς όπως παρουσιάστηκε παραπάνω, δημιουργείται ένας κύκλος και ελέγχεται ποια δεδομένα βρίσκονται μέσα σε αυτόν. Παρόλα αυτά, ο αλγόριθμος επιστρέφει το τελικό αποτέλεσμα σε 5.89 sec.

Στην περίπτωση που στον k_neighbors_with_statistics εισαχθεί μόνο το “2d” index, εξετάζονται και πάλι 67484 εγγραφές σε 142ms. Όταν χρησιμοποιηθεί ένα compounding index εξετάζονται 13389 εγγραφές, δηλαδή οι εγγραφές που επιστρέφονται σε 82ms. Ο χρόνος ολοκλήρωσης του αλγόριθμου είναι 6.29 sec και 6.01 sec αντίστοιχα.

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (4x4) Without index	67484	41	13389	6.21
Statistics (4x4) Time index	13389	21	13389	5.89
Statistics (4x4) 2d index	67484	142	13389	6.29
Statistics (4x4) Compounding index	13389	82	13389	6.01

Όταν ο χώρος χωρίζεται σε 4x4 κελιά και πάλι ο χρόνος ελαχιστοποιείται όταν χρησιμοποιηθεί κάποιο index στο χρόνο. Αυτό που θεωρητικά δεν θα έπρεπε να συμβαίνει είναι να αυξάνεται ο χρόνος εκτέλεσης του query όταν χρησιμοποιηθεί το 2d index.

Όσο αφορά τα έγγραφα που εξετάζονται, τα καλύτερα αποτελέσματα παρουσιάζονται όταν χρησιμοποιηθεί το time index ή ο συνδυασμός των δύο index. Το 2d index ακόμα δεν βελτιώνει τα αποτελέσματα καθώς όλα τα έγγραφα είναι συγκεντρωμένα σε ένα κελί και εξετάζονται όλα για να επιστραφεί το τελικό αποτέλεσμα.

Με διαχωρισμό σε 10x10 κελιά

Όταν ο χώρος χωριστεί σε 10x10 χωρίς τη χρήση κάποιου index δεν αλλάζει κάτι σε σύγκριση με τα 4x4 κελιά. Εξετάζονται όλες οι εγγραφές σε 41 ms. Όταν χρησιμοποιηθεί ένα index στο χώρο, εξετάζονται και πάλι όλες οι εγγραφές σε 145 ms. Όταν χρησιμοποιηθεί μόνο index στο χρόνο, εξετάζονται 13389 εγγραφές σε 20 ms. Όταν χρησιμοποιηθεί compounding index στο χώρο και στο χρόνο εξετάζονται και πάλι 13389 εγγραφές, αλλά αυτό συμβαίνει σε 84 ms. Οι χρόνοι ολοκλήρωσης του αλγόριθμου είναι 6.46, 6.01, 6.26, 6.17 sec αντίστοιχα

Παρατηρείται, λοιπόν, ότι για το συγκεκριμένο dataset, που όλα τα σημεία βρίσκονται συγκεντρωμένα μέσα σε ένα κελί με διαστάσεις 2x3, για να υπάρξει βελτίωση θα πρέπει ο χώρος να χωριστεί σε πάρα πολλά κελιά.

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (10x10) Without index	67484	41	13389	6.46
Statistics (10x10) Time index	13389	20	13389	6.01
Statistics (10x10) 2d index	67484	145	13389	6.26
Statistics (10x10) Compounding index	13389	84	13389	6.17

Όταν ο χώρος χωριστεί σε 10x10 κελιά, ουσιαστικά δεν υπάρχει καμιά βελτίωση σε σύγκριση με την παραπάνω περίπτωση που ο χώρος χωρίζεται σε 4x4 κελιά. Αυτό συμβαίνει, καθώς όταν τα έγγραφα είναι συγκεντρωμένα μέσα σε ένα μικρό ορθογώνιο (3x2) πρέπει είτε αυτό το ορθογώνιο να χωριστεί σε περισσότερα κελιά, είτε όλος ο χώρος να χωριστεί σε πάρα πολλά κελιά και αυτό είναι που συμβαίνει στα επόμενα πειράματα που ο χώρος χωρίζεται σε 1000x1000 και 2000x2000 κελιά.

Με διαχωρισμό σε 1000x1000 κελιά

Όταν ο χώρος χωριστεί σε 1000x1000 κελιά, χωρίς κανένα index εξετάζονται όλες οι εγγραφές και τα αποτελέσματα επιστρέφονται σε 6 sec.

```
: True, 'nReturned': 11154, 'executionTimeMillis': 41,
'totalDocsExamined': 67484, 'executionStages': {'stage'
{'$and': [{'t': {'$lt': 1443700123}}, {'t': {'$gt': 144
```

Αν όμως χρησιμοποιηθεί ένα index στο χώρο, τα αποτελέσματα είναι τα εξής:

```
ue, 'nReturned': 11154, 'executionTimeMillis': 128,
 'totalDocsExamined': 57680, 'executionStages': {'s
{'$and': [{'coordinates': {'$geoWithin': {'$center'
```

Η συνολική διάρκεια είναι 5.83 sec. Σίγουρα δεν είναι το καλύτερο δυνατό αποτέλεσμα, αλλά σε σύγκριση με τα προηγούμενα πειράματα, που επέστρεφαν όλο το dataset, υπάρχει μια μικρή βελτίωση, καθώς εξετάζονται λιγότερα έγγραφα και επιστρέφονται ακόμα λιγότερα. Όταν χρησιμοποιηθεί ένα index στο χρόνο, τα αποτελέσματα είναι περισσότερο θετικά με τη διάρκεια όλου του αλγόριθμου να είναι 5.43 sec.

```
ue, 'nReturned': 11154, 'executionTimeMillis': 21,
 'totalDocsExamined': 13389, 'executionStages': {'s
ordinates': {'$geoWithin': {'$center': [[-4.75, 48
```

Αλλά τα καλύτερα αποτελέσματα παρουσιάζονται όταν χρησιμοποιηθεί ένα compounding index, με τη συνολική διάρκεια να είναι στα 5.34 sec.

```
ue, 'nReturned': 11154, 'executionTimeMillis': 64,
 'totalDocsExamined': 11415, 'executionStages': {'s
ordinates': {'$geoWithin': {'$center': [[-4.75, 48
```

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (1000x1000) Without index	67484	41	11154	6.0
Statistics (1000x1000) Time index	13389	21	11154	5.43
Statistics (1000x1000) 2d index	57680	128	11154	5.83
Statistics (1000x1000) Compounding index	11415	64	11154	5.34

Όταν ο χώρος χωριστεί σε 1000x1000 κελιά, ο χρόνος εκτέλεσης για το 2d index εξακολουθεί να είναι υψηλός, αν και σε σύγκριση με τις προηγούμενες μετρήσεις είναι χαμηλότερος. Ο χρόνος εκτέλεσης ελαχιστοποιείται όταν χρησιμοποιηθεί το time index. Γενικότερα, παρατηρείται ότι και τα 3 index παρουσιάζουν μια μικρή βελτίωση.

Στο συγκεκριμένο διάγραμμα παρατηρείται ότι με τη χρήση του 2d index έχει ξεκινήσει να υπάρχει μια βελτίωση, καθώς εξετάζονται λιγότερα έγγραφα. Όσο αφορά την ελαχιστοποίηση των documents που εξετάζονται, time index και compounding index παρουσιάζουν τα καλύτερα αποτελέσματα, με το compounding index να εξετάζει λιγότερα έγγραφα.

Με διαχωρισμό σε 2000x2000 κελιά

Όταν ο χώρος χωριστεί σε 2000x2000 κελιά τότε χωρίς κανένα index εξετάζονται όλα τα έγγραφα και επιστρέφονται 13389 σε 40 ms και τα τελικά αποτελέσματα σε 5.75 sec. Όταν χρησιμοποιηθεί ένα time index, τότε όπως και στα παραπάνω πειράματα εξετάζονται 13389 έγγραφα σε 20 ms και η συνολική διάρκεια είναι 3.61 sec. Με 2d index ελέγχονται 13836 έγγραφα σε 27 ms και τέλος με τη χρήση του compounding index ελέγχονται 3439 έγγραφα σε 18 ms. Η συνολική διάρκεια είναι 3.79 και 3.59 sec αντίστοιχα. Αξίζει να σημειωθεί ότι με 2000x2000 κελιά, ο αλγόριθμος επιστρέφει 1710 έγγραφα.

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (2000x2000) Without index	67484	40	1710	3.99
Statistics (2000x2000) Time index	13389	20	1710	3.61
Statistics (2000x2000) 2d index	13836	27	1710	3.79
Statistics (2000x2000) Compounding index	3439	18	1710	3.59

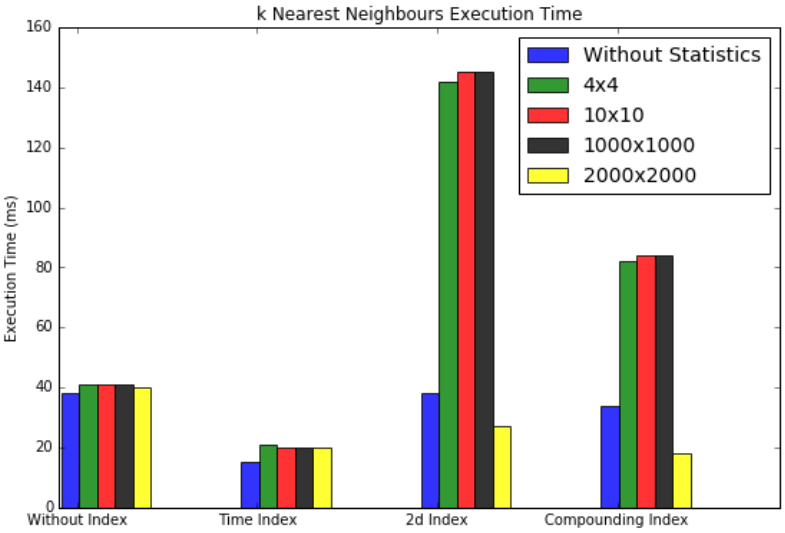
Παρατηρείται ότι όταν ο χώρος χωριστεί σε 2000x2000 κελιά και χρησιμοποιηθεί ένα compounding index, τα έγγραφα που εξετάζονται είναι λιγότερα. Σε διαφορετική περίπτωση όταν χρησιμοποιηθούν λίγα κελιά δεν υπάρχει σχεδόν καμία βελτίωση ή η βελτίωση είναι πάρα πολύ μικρή. Επίσης, ο μικρότερος χρόνος ολοκλήρωσης εμφανίζεται όταν χρησιμοποιηθούν στατιστικά δεδομένα, ο χώρος χωριστεί σε 2000x2000 κελιά και χρησιμοποιηθεί κάποιο index.

Όταν ο χώρος χωρίζεται σε 2000x2000 κελιά, παρατηρείται σημαντική βελτίωση στον χρόνο εκτέλεσης με τη χρήση του 2d index, καθώς ο χρόνος μειώθηκε από 128ms σε 27ms. Επίσης, σε

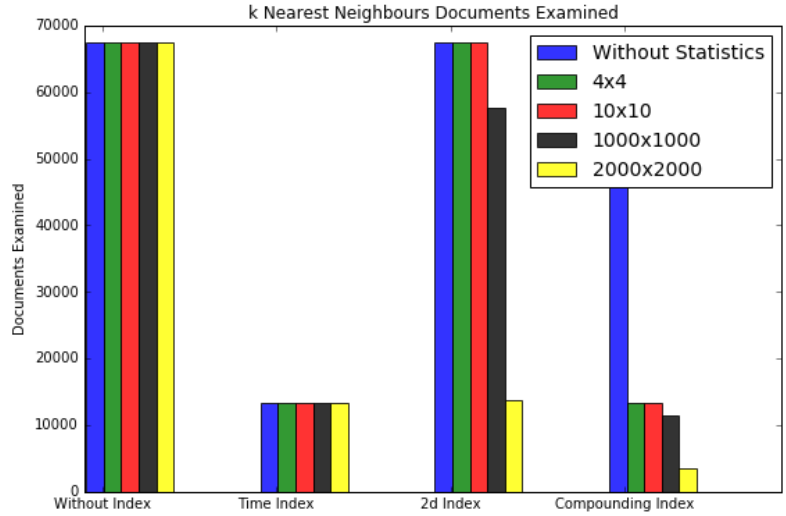
αντίθεση με τις προηγούμενες μετρήσεις, με τη χρήση του compounding index το query εκτελείται σε λιγότερο χρόνο από το time index.

Όσο αφορά τα documents που εξετάζονται, το compounding index πολύ καλύτερα αποτελέσματα σε σύγκριση με τα άλλα δύο index.

Στη συνέχεια παρουσιάζονται συνοπτικά τα αποτελέσματα των παραπάνω πειραμάτων.



Εικόνα 16 Χρόνος Εκτέλεσης k Nearest Neighbours για το πρώτο σύνολο δεδομένων



Εικόνα 17 Αριθμός Εγγράφων που εξετάστηκαν για το πρώτο σύνολο δεδομένων του k Nearest Neighbours

Με τη χρήση των παραπάνω γραφημάτων είναι περισσότερο εμφανές ότι ο καλύτερος συνδυασμός χρόνου εκτέλεσης του query και εγγράφων που εξετάζονται είναι στην περίπτωση που ο χώρος χωρίζεται σε 2000x2000 κελιά και χρησιμοποιείται ένα compounding index. Γενικότερα, όταν η διασπορά των δεδομένων είναι πολύ μικρή, καλύτερη λύση είναι να χωριστεί ο χώρος σε πάρα πολλά κελιά και να χρησιμοποιηθεί ένα index στο χώρο και στο χρόνο.

5.2.2 Εύρεση k κοντινότερων γειτόνων με διαφορετικό id

Όπως παρουσιάστηκε στην προηγούμενη ενότητα μια παραλλαγή του παραπάνω αλγόριθμου, που επιστρέφει τους k κοντινότερους γείτονες, είναι η εύρεση των k κοντινότερων γειτόνων που θα προέρχονται από διαφορετικές οντότητες, δηλαδή μεταξύ τους θα έχουν διαφορετικό id.

Θεωρητικά, σε σύγκριση με τον παραπάνω αλγόριθμο ο χρόνος εκτέλεσης του αλγόριθμου (Total Time) θα είναι μεγαλύτερος, όπως και τα έγγραφα που εξετάζονται μετά το query θα είναι περισσότερα. Αλλά, όπως και στον προηγούμενο αλγόριθμο, τα καλύτερα αποτελέσματα αναμένονται όταν χρησιμοποιηθεί ένα compounding index και ο χώρος χωριστεί σε 2000x2000 κελιά.

Στους παρακάτω πίνακες παρουσιάζονται τα αποτελέσματα όλων των πειραμάτων.

Χωρίς τη χρήση στατιστικών δεδομένων

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Without index	67484	34	13389	10.59
Time index	13389	17	13389	6.51
2d index	67484	34	13389	6.69
Compounding index	67484	34	13389	5.72

Με διαχωρισμό σε 4x4 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (4x4) Without index	67484	52	13389	7.23
Statistics (4x4) Time index	13389	23	13389	6.13
Statistics (4x4) 2d index	67484	144	13389	6.35
Statistics (4x4) Compounding index	67484	34	13389	6.08

Με διαχωρισμό σε 10x10 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (10x10) Without index	67484	43	13389	6.92

Statistics (10x10) Time index	13389	20	13389	6.45
Statistics (10x10) 2d index	67484	146	13389	6.57
Statistics (10x10) Compounding index	67484	83	13389	6.23

Με διαχωρισμό σε 1000x1000 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (1000x1000) Without index	67484	41	11154	6.19
Statistics (1000x1000) Time index	13389	21	11154	5.72
Statistics (1000x1000) 2d index	57680	129	11154	5.91
Statistics (1000x1000) Compounding index	11415	65	11154	5.41

Με διαχωρισμό σε 2000x2000 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (2000x2000) Without index	67484	42	1710	4.01
Statistics (2000x2000) Time index	13389	21	1710	3.82
Statistics (2000x2000) 2d index	13836	27	1710	3.91
Statistics (2000x2000)	3439	20	1710	3.70

Compounding index				

Όπως ήταν αναμενόμενο ο συνολικός χρόνος εκτέλεσης, σε σύγκριση με τον προηγούμενο αλγόριθμο που δεν ελέγχει το id των εγγραφών που επιστρέφονται, είναι λίγο μεγαλύτερος. Οι υπόλοιπες μετρήσεις είναι σχεδόν ίδιες γιατί ο αλγόριθμος που επιστρέφει τις εγγραφές δεν αλλάζει.

5.2.3 Circle Range

Η Circle Range συνάρτηση όπως είναι αναμενόμενο χωρίς κανένα index, θα πρέπει να ελέγξει όλες τις εγγραφές προκειμένου να επιστρέψει κάποιο αποτέλεσμα. Σημαντική βελτίωση υπάρχει με την εισαγωγή index στο χώρο ή στο χρόνο. Οι παράμετροι της συνάρτησης είναι (mycol, -4.75, 48.25, 0.08, 1443690402, 1443700123), δηλαδή το ερώτημα που κλήθηκε να απαντήσει είναι να βρεθούν όλες οι εγγραφές που βρίσκονται μέσα στον κύκλο με κέντρο (-4.75, 48.25) και ακτίνα 0.08, καθώς επίσης και στο χρονικό διάστημα [1443690402, 1443700123].

Αρχικά, όπως αναφέρθηκε χωρίς κανένα index η συνάρτηση εξετάζει όλες τις εγγραφές.

```
'totalDocsExamined': 67484,
gt': 1443690402}}, {'coordina
isEstimate': 30, 'works': 67
```

Ο χρόνος εκτέλεσης ήταν 40 ms και το query επέστρεψε 221 έγγραφα.

```
'nReturned': 221, 'executionTimeMillis': 40,
'COLLSCAN', 'filter': {'$and': [{'t': {'$lt':
er': [[-4.75, 48.25], 0.08]}}}]}, 'nReturned'
```

Αν πραγματοποιηθεί εισαγωγή ενός index στο χώρο, τότε η συνάρτηση εξετάζει 3457 εγγραφές σε 7 ms, δηλαδή η βελτίωση είναι πολύ μεγάλη.

```
'totalKeysExamined': 3459,
{'$geoWithin': {'$center'
221, 'executionTimeMillisE
```

```
'nReturned': 221, 'executionTimeMillis': 7,
'FETCH', 'filter': {'$and': [{'coordinates'
{'t': {'$gt': 1443690402}}]}, 'nReturned':
```

Αν πραγματοποιηθεί και εισαγωγή index μόνο στο χρόνο, τότε οι εγγραφές που εξετάζονται είναι 13389 σε 19 ms.

```
'totalDocsExamined': 13389,
48.25], 0.08]}}}, 'nReturne
ldYield': 0, 'saveState': 10
```

```
'nReturned': 221, 'executionTimeMillis': 19,
': 'FETCH', 'filter': {'coordinates': {'$geoWi
lisEstimate': 20, 'works': 13390, 'advanced':
```

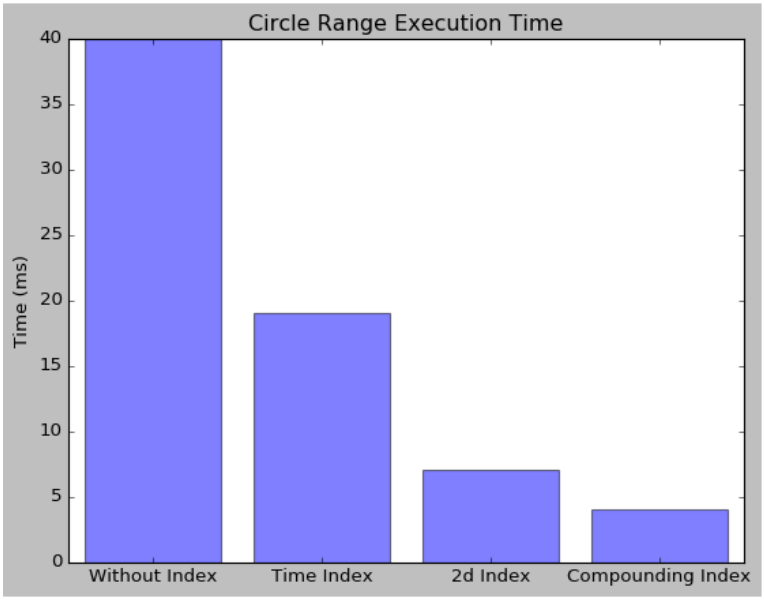
Αν χρησιμοποιούνταν ένα compounding index, τότε τα αποτελέσματα είναι τα καλύτερα σε σύγκριση με τις προηγούμενες μετρήσεις, καθώς εξετάζονται μόνο 766 εγγραφές σε 4 ms.

'totalDocsExamined': 766, 'nReturned': 221, 'executionTimeMillis': 4,
8.25], 0.08]}}}, 'nReturne FETCH', 'filter': {'coordinates': {'$geoWith
.d': 0, 'saveState': 27, 'r stimate': 0, 'works': 3459, 'advanced': 221,

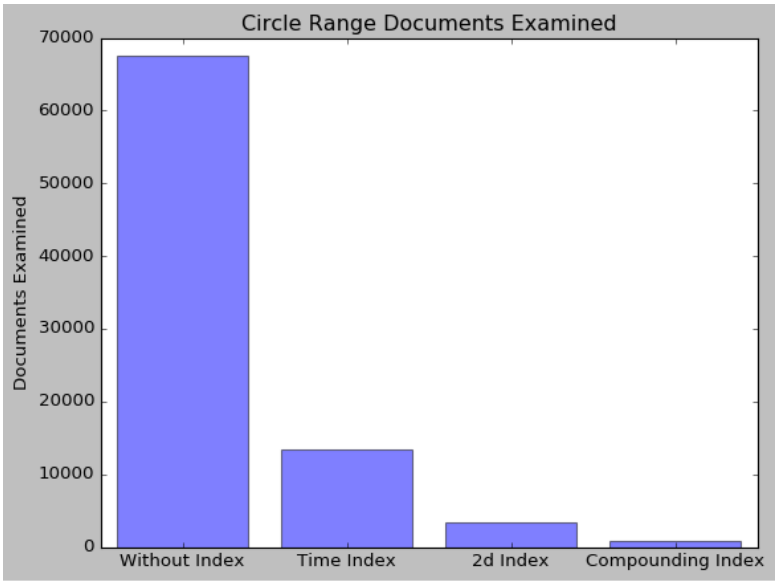
Στον παρακάτω πίνακα παρουσιάζονται συγκεντρωτικά τα αποτελέσματα όλων των μετρήσεων για το Circle Range Query.

	Documents Examined	Execution Time (ms)	Total Time (sec)
Without index	67484	40	2.96
Time index	13389	19	2.58
2d index	3459	7	2.56
Compounding index	766	4	2.53

Τα γραφήματα των αποτελεσμάτων είναι τα εξής:



Εικόνα 18 Circle Range: Χρόνος Εκτέλεσης Query



Εικόνα 19 Circle Range: Αριθμός Εγγράφων που εξετάστηκαν

Τα αποτελέσματα είναι απολύτως αναμενόμενα, καθώς όταν δεν χρησιμοποιείται κανένα index εξετάζονται όλα τα έγγραφα και η χρονική διάρκεια είναι υψηλή συγκριτικά με τις υπόλοιπες μετρήσεις. Το γεγονός ότι το 2d index παρουσιάζει καλύτερα αποτελέσματα από το χρονικό index, όσο αφορά το χρόνο εκτέλεσης του query, οφείλεται στο ότι η ακτίνα του κύκλου είναι πολύ μικρή και το χρονικό διάστημα σχετικά μεγάλο. Σε ένα διαφορετικό query, όπου το χρονικό διάστημα θα ήταν πολύ μικρό και η ακτίνα του κύκλου πολύ μεγάλη, προφανώς το χρονικό index θα παρουσίαζε καλύτερα αποτελέσματα. Παρόλα αυτά σε κάθε περίπτωση, ο συνδυασμός των δύο index, δηλαδή το compounding index είναι αυτό που παράγει τα καλύτερα αποτελέσματα.

5.2.4 Box Range

Η Box Range συνάρτηση χρησιμοποιήθηκε με παραμέτρους (mycol, -4.9, 48.1, -4.7, 48.3, 1443690402, 1443700123), δηλαδή το ορθογώνιο σχηματιζότανε από τις γωνίες (-4.9, 48.1) και (-4.7, 48.3) και το χρονικό διάστημα μέσα στο οποίο πρέπει να βρίσκονται τα δεδομένα είναι [1443690402, 1443700123].

Όπως σε όλα τα παραδείγματα, χωρίς index εξετάζονται όλα τα δεδομένα σε 41 ms. Φυσικά, το query επιστρέφει μόνο 1642 έγγραφα.

```
'totalDocsExamined': 67484,                    'nReturned': 1642, 'executionTimeMillis': 41,
$gt': 1443690402}}, {'coordi                   : 'COLLSCAN', 'filter': {'$and': [{'t': {'$lt'
imeMillisEstimate': 30, 'wor                   ox': [[-4.9, 48.1], [-4.7, 48.3]]}}}]}, 'nRetu
```

Με χρήση index στο χώρο εξετάζονται 8797 εγγραφές σε 20 ms.

```
'totalKeysExamined': 8797,                     'nReturned': 1642, 'executionTimeMillis': 20,
: {'$and': [{'coordinate                       'executionStages': {'stage': 'FETCH', 'filter
: 1443700123}}, {'t': {'$g                      [[-4.9, 48.1], [-4.7, 48.3]]}}}, {'t': {'$lt
```

Αν χρησιμοποιηθεί μόνο το index στο χρόνο εξετάζονται 13389 έγγραφα σε 30 ms.

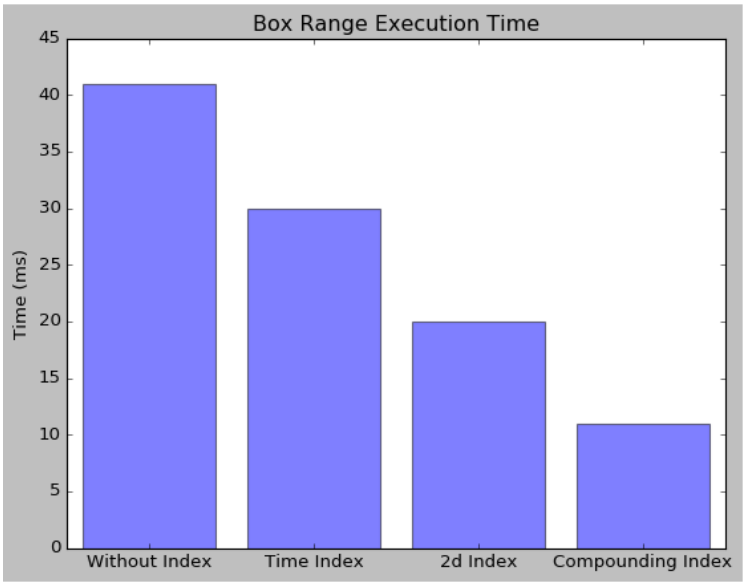
```
ue, 'nReturned': 1642, 'executionTimeMillis': 30,
 'totalDocsExamined': 13389, 'executionStages': {
{'coordinates': {'$geoWithin': {'$box': [[-4.9, 4
```

Συνεπώς, με χρήση index είτε στο χώρο, είτε στο χρόνο η αποδοτικότητα του αλγόριθμου αυξάνεται. Παρόλα αυτά όπως και στον Circle Range αλγόριθμο, η μεγαλύτερη αποδοτικότητα εμφανίζεται όταν χρησιμοποιηθεί ένα compounding index. Όπως παρουσιάζεται παρακάτω, εξετάζονται 2173 έγγραφα σε 11 ms.

```
ue, 'nReturned': 1642, 'executionTimeMillis': 11,
 'totalDocsExamined': 2173, 'executionStages': {'st
{'coordinates': {'$geoWithin': {'$box': [[-4.9, 48
```

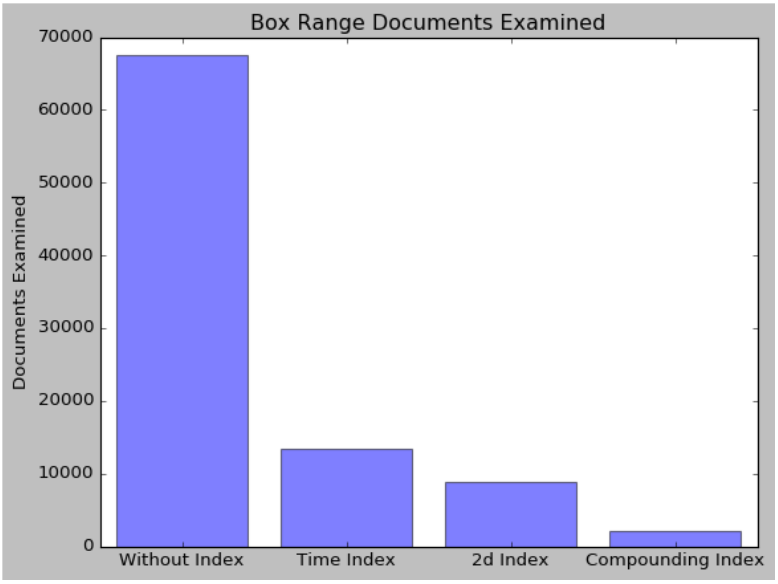
Συγκεντρωτικά, τα αποτελέσματα του αλγόριθμου είναι τα εξής:

	Documents Examined	**Execution Time (ms)**	**Total Time (sec)**
Without index	67484	41	3.61
Time index	13389	30	3.50
2d index	8797	20	3.41
Compounding index	2173	11	3.33



Εικόνα 20 Box Range: Χρόνος εκτέλεσης Query

Στο παραπάνω διάγραμμα όταν δεν χρησιμοποιείται κανένα index, ο χρόνος εκτέλεσης του query είναι σχετικά υψηλός και με τη χρήση του compounding index ο χρόνος εκτέλεσης ελαχιστοποιείται.



Εικόνα 21 Box Range: Αριθμός Εγγράφων που εξετάστηκαν

Τα αποτελέσματα μοιάζουν πάρα πολύ με τον Range Circle αλγόριθμο, καθώς το 2d index παρουσιάζει καλύτερα αποτελέσματα από το time index, καθώς λιγότερα έγγραφα ικανοποιούν τον χωρικό περιορισμό, από αυτά που ικανοποιούν τον χρονικό περιορισμό. Τέλος, όπως ήταν αναμενόμενο το compounding index παρουσιάζει τα καλύτερα αποτελέσματα.

5.2.5 Εξαγωγή όλου του trajectory

Όπως παρουσιάστηκε παραπάνω μια παραλλαγή των παραπάνω ερωτημάτων είναι για κάθε id που επιστρέφει ο αλγόριθμος, να πραγματοποιείται αναζήτηση όλου του trajectory. Οι μετρήσεις αφορούν τον χρόνο εκτέλεσης του query που θα επιστρέψει όλες τις εγγραφές που περιλαμβάνει το συγκεκριμένο id, όταν η αναζήτηση πραγματοποιηθεί στο αρχικό collection και όταν πραγματοποιηθεί στο collection που λειτουργεί ως inverted index. Φυσικά, αναμένεται η αναζήτηση στο collection που λειτουργεί ως inverted index να πραγματοποιηθεί πολύ γρηγορότερα. Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα των μετρήσεων του χρόνου στα δυο collection. Οι παράμετροι των αλγόριθμων είναι ίδιοι με τα παραπάνω ερωτήματα.

	Αρχικό Collection Time (ms)	**Inverted Index Collection Time (ms)**
Box Range	79	10
Circle Range	48	0
k Nearest Neighbours	26	0

5.3 Πειραματική Μελέτη με χρήση του Δεύτερου Dataset

5.3.1 Εύρεση k κοντινότερων γειτόνων

Όπως και στο παραπάνω dataset ο αλγόριθμος μελετήθηκε χωρίς τη χρήση στατιστικών δεδομένων και indexes, καθώς και με τη χρήση μόνο indexes ή μόνο στατιστικών δεδομένων ή συνδυασμό των δύο. Το σημείο στο οποίο θα αναζητηθούν οι κοντινότεροι γείτονες είναι το (1, 1) και το χρονικό διάστημα είναι το (1475000000, 1525000000). Για τη χρήση των στατιστικών δεδομένων, ο χώρος χωρίστηκε σε 10x10, 25x25 και 40x40 κελιά.

Στους παρακάτω πίνακες παρουσιάζονται τα αποτελέσματα όλων των πειραμάτων.

Χωρίς τη χρήση στατιστικών δεδομένων

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Without index	70000	35	17356	6.57
Time index	17356	22	17356	6.22
2d index	70000	34	17356	6.35
Compounding index	70000	34	17356	6.34

Χωρίς τη χρήση στατιστικών δεδομένων, ο αλγόριθμος επιστρέφει μόνο τα έγγραφα που ικανοποιούν τον χρονικό περιορισμό. Επομένως, ουσιαστική μείωση στο χρόνο υπάρχει μόνο όταν χρησιμοποιηθεί κάποιο index στο χρόνο.

Όπως και στο παραπάνω γράφημα, όταν χρησιμοποιηθεί κάποιο index στο χρόνο, μειώνεται πάρα πολύ ο αριθμός των εγγράφων που εξετάζονται. Σε διαφορετική περίπτωση εξετάζονται όλα τα έγγραφα.

Με διαχωρισμό σε 10x10 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (10x10) Without index	70000	43	1222	3.15
Statistics (10x10) Time index	17356	32	1222	3.10
Statistics (10x10) 2d index	8680	20	1222	3.01
Statistics (10x10) Compounding index	2124	12	1222	2.98

Όταν ο χώρος χωριστεί σε 10x10 κελιά, όχι μόνο υπάρχει βελτίωση των αποτελεσμάτων και με τα υπόλοιπα index, αλλά ο χρόνος εκτέλεσης του query με τη χρήση 2d index και compounding index είναι μικρότερος από το χρόνο που απαιτείται όταν χρησιμοποιηθεί time index. Χωρίς τη χρήση κάποιου index ο χρόνος παραμένει υψηλός.

Αντίστοιχα και τα έγγραφα που εξετάζονται είναι λιγότερα, με το compounding index να παρουσιάζει τα καλύτερα αποτελέσματα, καθώς τα 2124 έγγραφα που εξετάζονται σε σύγκριση με το αρχικό 70.000, είναι πολύ μεγάλη βελτίωση.

Με διαχωρισμό σε 25x25 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (25x25) Without index	70000	43	69	3.18
Statistics (25x25) Time index	17356	33	69	2.79
Statistics (25x25) 2d index	477	1	69	2.17
Statistics (25x25) Compounding index	477	1	69	2.79

Όταν ο χώρος χωριστεί σε περισσότερα κελιά (25x25), το 2d index και το compounding index παρουσιάζουν ακόμα καλύτερα αποτελέσματα και ουσιαστικά δεν μπορεί να υπάρξει μεγαλύτερη βελτίωση όσο αφορά τον χρόνο εκτέλεσης του query, καθώς ο χρόνος εκτέλεσης είναι 1 ms.

Η βελτίωση των αποτελεσμάτων αποτυπώνεται και στο παραπάνω γράφημα που παρουσιάζει τα έγγραφα που εξετάζει ο αλγόριθμος. Φυσικά, η όποια βελτίωση αφορά τη χρήση 2d index και compounding index, καθώς ο διαχωρισμός του χώρου σε κελιά δεν έχει καμιά σχέση με το time index και ως εκ τούτου δεν μπορεί να επηρεάσει τα τελικά αποτελέσματα.

Με διαχωρισμό σε 40x40 κελιά

	Documents Examined	**Execution Time (ms)**	**Documents Returned**	**Total Time (sec)**
Statistics (40x40) Without index	70000	42	58	2.53
Statistics (40x40) Time index	17356	31	58	2.39
Statistics (40x40) 2d index	473	1	58	2.16

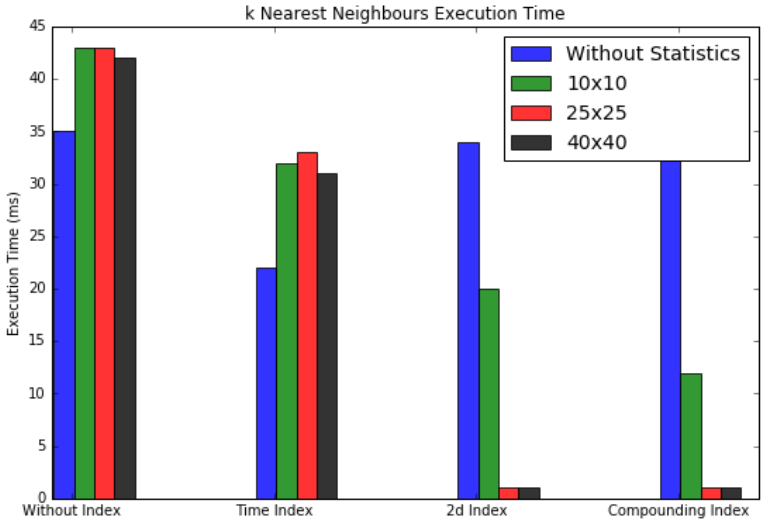
Statistics (40x40) Compounding index	108	1	58	2.09

Παρατηρείται ότι σε όσο περισσότερα κελιά χωρίζεται ο χώρος, τόσο καλύτερα τα αποτελέσματα του αλγόριθμου. Φυσικά, πάντα με τη χρήση κάποιου index είτε στο χώρο, είτε index που συνδυάζει χώρο και χρόνο. Όσο περισσότερα τα κελιά που χωρίζεται ο χώρος, τόσο λιγότερα τα έγγραφα που επιστρέφει ο αλγόριθμος, συνεπώς είναι λιγότερος ο συνολικός χρόνος εκτέλεσης του αλγόριθμου.

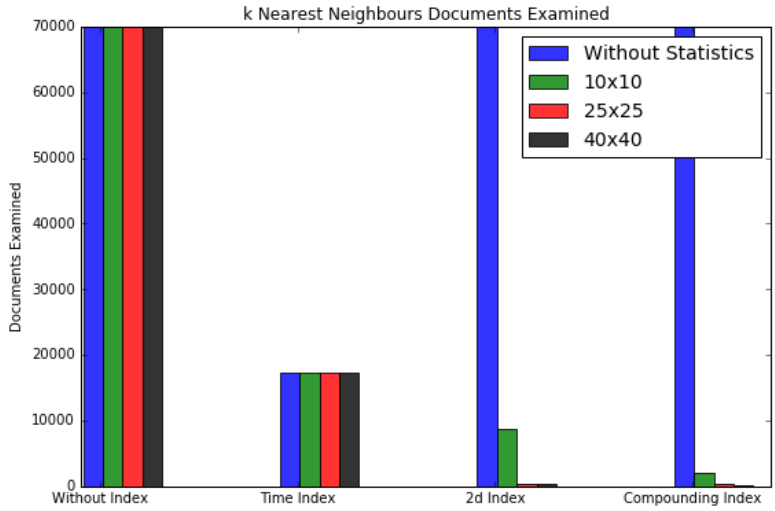
Όπως ειπώθηκε παραπάνω, όταν χώρος χωρίστηκε σε 25x25 κελιά, υπήρξε μια βελτιστοποίηση των αποτελεσμάτων (ιδιαίτερα με τη χρήση 2d και compounding index), συνεπώς με τον διαχωρισμό σε περισσότερα κελιά δεν μπορεί να υπάρξει κάποια βελτίωση των αποτελεσμάτων όσο αφορά των χρόνο εκτέλεσης του query.

Τέλος, αν η μελέτη επικεντρωθεί στα έγγραφα που εξετάζει ο αλγόριθμος, τότε υπάρχει μια μικρή βελτίωση, καθώς ο αλγόριθμος εξετάζει 473 και 108 έγγραφα με τη χρήση 2d και compounding index, που είναι λιγότερα από τα έγγραφα που εξετάζονται όταν ο χώρος χωριστεί σε 25x25 κελιά.

Στη συνέχεια παρουσιάζονται συνοπτικά τα αποτελέσματα των παραπάνω πειραμάτων.



Εικόνα 22 Χρόνος Εκτέλεσης k Nearest Neighbours για το δεύτερο σύνολο δεδομένων



Εικόνα 23 Αριθμός Εγγράφων που εξετάστηκαν για το δεύτερο σύνολο δεδομένων του k Nearest Neeighbours

5.3.2 Εύρεση k κοντινότερων γειτόνων με διαφορετικό id

Γενικότερα, όταν η αναζήτηση αφορά εγγραφές με διαφορετικό id, ο συνολικός χρόνος εκτέλεσης του αλγόριθμου αναμένεται να είναι μεγαλύτερος, καθώς πολλές φορές οι κοντινότερες εγγραφές περιέχουν το ίδιο id, επομένως πρέπει να γίνει επαναϋπολογισμός των κοντινότερων εγγραφών και επίσης πάντα πραγματοποιείται έλεγχος ότι οι εγγραφές δεν περιέχουν το ίδιο id. Στον παρακάτω πίνακα παρουσιάζεται μόνο ο συνολικός χρόνος εκτέλεσης του αλγόριθμου, καθώς τα query δεν αλλάζουν σε σύγκριση με τον παραπάνω αλγόριθμο.

	Without Statistics	**Statistics (10x10)**	**Statistics (25x25)**	**Statistics (40x40)**
Without index	8.12	3.79	3.59	4.72
Time index	7.07	3.44	3.01	3.18
2d index	7.19	3.55	2.82	2.56
Compounding index	7.14	3.60	2.86	2.50

Όπως ήταν αναμενόμενο ο συνολικός χρόνος εκτέλεσης του αλγόριθμου που αναζητά εγγραφές με διαφορετικό id όταν συγκρίνεται με τον παραπάνω αλγόριθμο που δεν ελέγχει το id είναι μεγαλύτερος σε όλες τις περιπτώσεις.

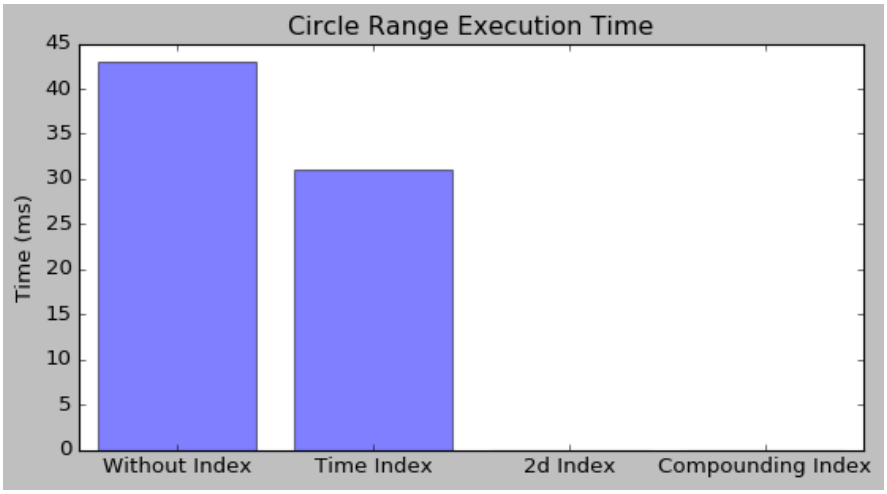
5.3.3 Circle Range

1ο Πείραμα

Αρχικά, ο Circle Range αλγόριθμος εκτελέστηκε με τις εξής παραμέτρους: (1,1, 2, 1475000000, 1525000000), δηλαδή αναζητήθηκε ο κύκλος με κέντρο (1, 1) και ακτίνα 2 και χρονικό διάστημα [1475000000, 1525000000]. Τα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα.

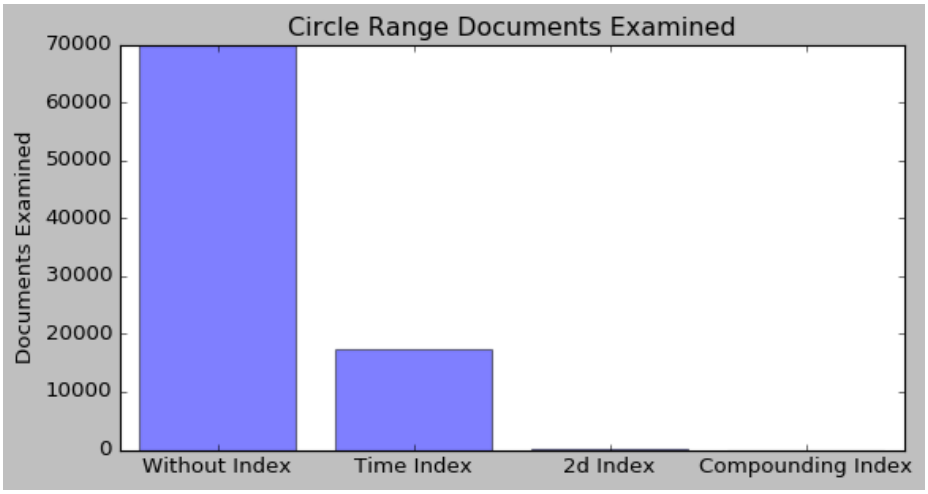
	Documents Examined	Execution Time (ms)	Documents Returned
Without index	70000	43	3
Time index	17356	31	3
2d index	38	0	3
Compounding index	8	0	3

Παρατηρείται ότι με συνδυασμό των δύο Index, το αποτέλεσμα είναι το βέλτιστο, καθώς το query εκτελείται σε 0 ms και εξετάζονται μόνο 8 έγγραφα. Στη συνέχεια, παρουσιάζονται τα γραφήματα που προκύπτουν από τον παραπάνω πίνακα.



Εικόνα 24 Circle Range: Χρόνος Εκτέλεσης Query (2ο Σύνολο Δεδομένων, 1ο Πείραμα)

Όπως παρουσιάζεται στο παραπάνω γράφημα ο χρόνος εκτέλεσης όταν χρησιμοποιείται 2d index ή compounding index τείνει στο 0 ms. Αυτό οφείλεται στο γεγονός ότι ο κύκλος είναι πάρα πολύ μικρός. Σε διαφορετική περίπτωση όταν χρησιμοποιηθεί time index ο χρόνος εκτέλεσης είναι μικρότερος από την περίπτωση που δεν χρησιμοποιείται κανένα index.



Εικόνα 25 Circle Range: Αριθμός Εγγράφων που εξετάστηκαν (2ο Σύνολο Δεδομένων, 1ο Πείραμα)

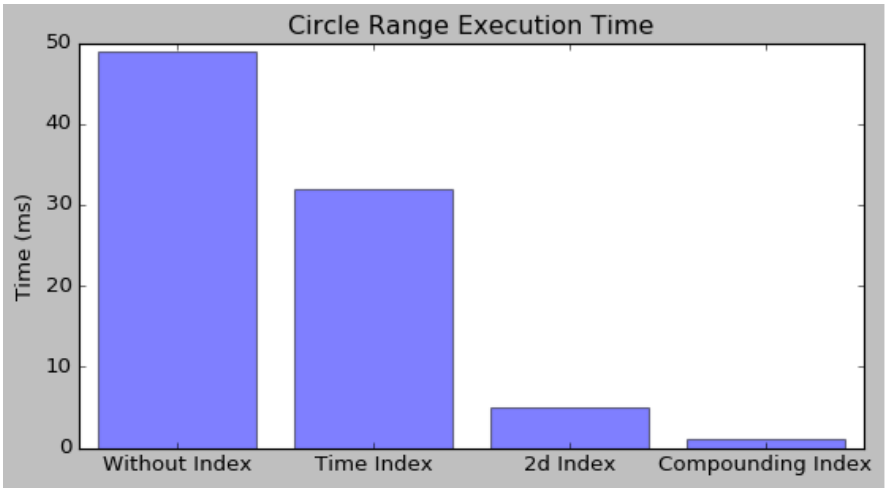
Ομοίως και τα έγγραφα που εξετάζονται είναι 38 και 8 για 2d index και compounding index αντίστοιχα, που είναι και το καλύτερο αποτέλεσμα. Σημαντική είναι και η βελτίωση που υπάρχει όταν χρησιμοποιηθεί το time index.

2ο Πείραμα

Αν αλλάξουν οι παράμετροι και πραγματοποιηθεί αναζήτηση των εγγραφών που βρίσκονται μέσα στο κύκλο με κέντρο (1, 1) και ακτίνα 10 στο ίδιο χρονικό διάστημα τότε τα αποτελέσματα είναι τα εξής:

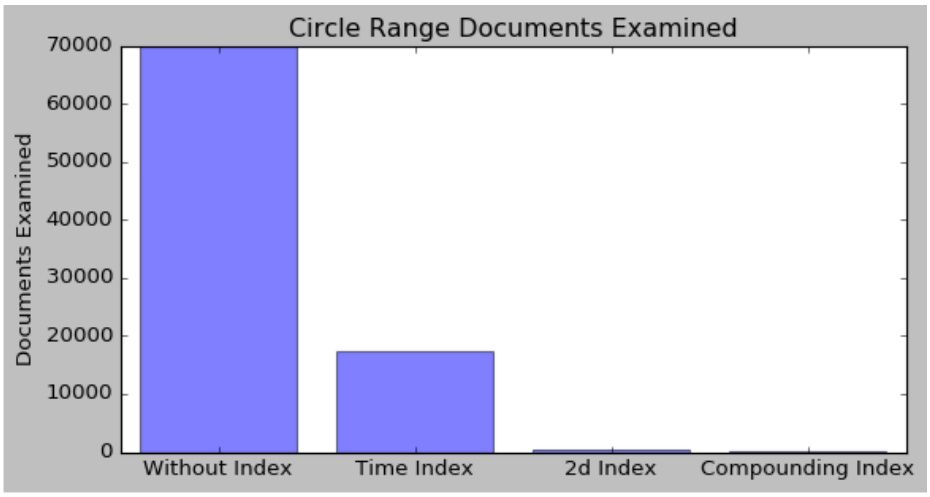
	Documents Examined	**Execution Time (ms)**	**Documents Returned**
Without index	70000	49	73
Time index	17356	32	73
2d index	500	5	73
Compounding index	116	1	73

Με αυτά τα νέα δεδομένα και πάλι όπως ήταν αναμενόμενο με τη χρήση του compounding index επιτυγχάνονται τα καλύτερα αποτελέσματα, αλλά σε σύγκριση με τα παραπάνω αποτελέσματα ο χρόνος εκτέλεσης είναι λίγο μεγαλύτερος κυρίως όταν χρησιμοποιείται 2d ή compounding index. Αυτό οφείλεται στο γεγονός ότι εξετάζονται περισσότερα έγγραφα.



Εικόνα 26 Circle Range: Χρόνος Εκτέλεσης Query (2ο Σύνολο Δεδομένων, 2ο Πείραμα)

Στο παραπάνω γράφημα είναι περισσότερο εμφανές ότι με αυτές τις νέες παραμέτρους ο χρόνος εκτέλεσης του query αυξήθηκε ελάχιστα κυρίως στις περιπτώσεις που χρησιμοποιείται 2d index.



Εικόνα 27 Circle Range: Αριθμός Εγγράφων που εξετάστηκαν (2ο Σύνολο Δεδομένων, 2ο Πείραμα)

Ομοίως αυξήθηκε και ο αριθμός των εγγράφων που εξετάζονται. Όταν χρησιμοποιείται time index δεν παρατηρείται κάποια διαφορά, καθώς το χρονικό διάστημα έχει παραμείνει ίδιο.

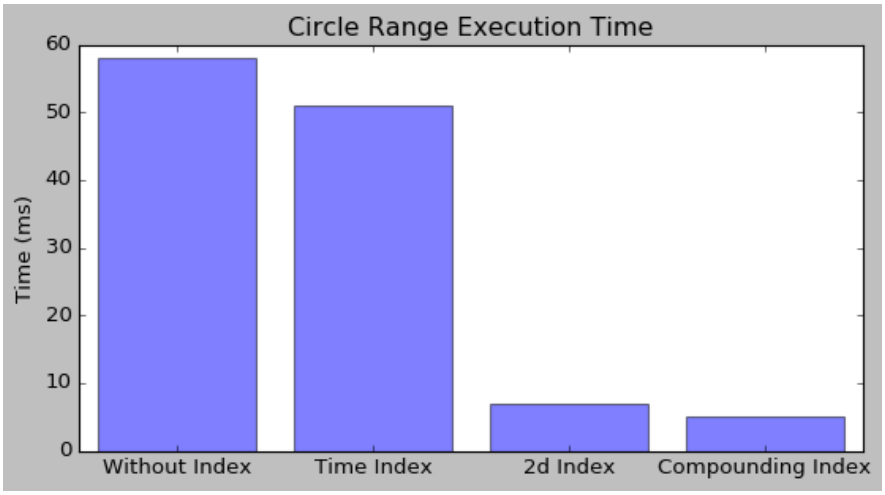
3ο Πείραμα

Τέλος, οι μετρήσεις θα εκτελεστούν με τις εξής παραμέτρους (1, 1, 15, 1450000000, 1550000000), δηλαδή εκτός από την ακτίνα αυξάνεται και το χρονικό διάστημα μέσα στο οποίο θα αναζητηθούν τα δεδομένα.

	Documents Examined	**Execution Time (ms)**	**Documents Returned**
Without index	70000	58	377
Time index	35005	51	377

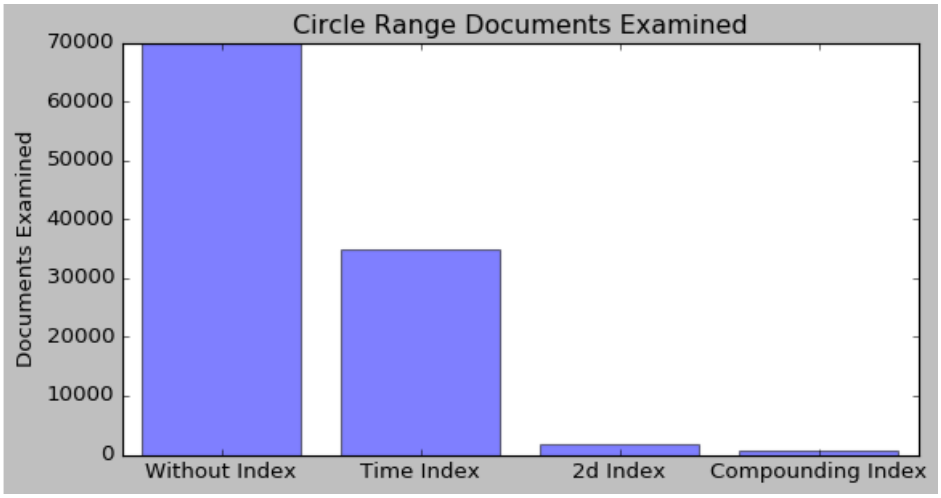
2d index	1738	7	377
Compounding index	823	5	377

Παρατηρείται ο χρόνος εκτέλεσης του query είναι μεγαλύτερος σε σύγκριση με τα παραπάνω πειράματα σε όλες τις περιπτώσεις.



Εικόνα 28 Circle Range: Χρόνος Εκτέλεσης Query (2ο Σύνολο Δεδομένων, 3ο Πείραμα)

Το γεγονός ότι ο χρόνος εκτέλεσης του query αυξήθηκε σε όλες τις περιπτώσεις είναι περισσότερο ξεκάθαρο στο παραπάνω γράφημα.



Εικόνα 29 Circle Range: Αριθμός Εγγράφων που εξετάστηκαν (2ο Σύνολο Δεδομένων, 3ο Πείραμα)

Μαζί με τον χρόνο εκτέλεσης του query, αυξήθηκαν και τα έγγραφα που εξετάζονται σε όλες τις περιπτώσεις, αλλά κυρίως όταν χρησιμοποιείται time index, καθώς το χρονικό διάστημα διπλασιάστηκε.

5.3.4 Box Range

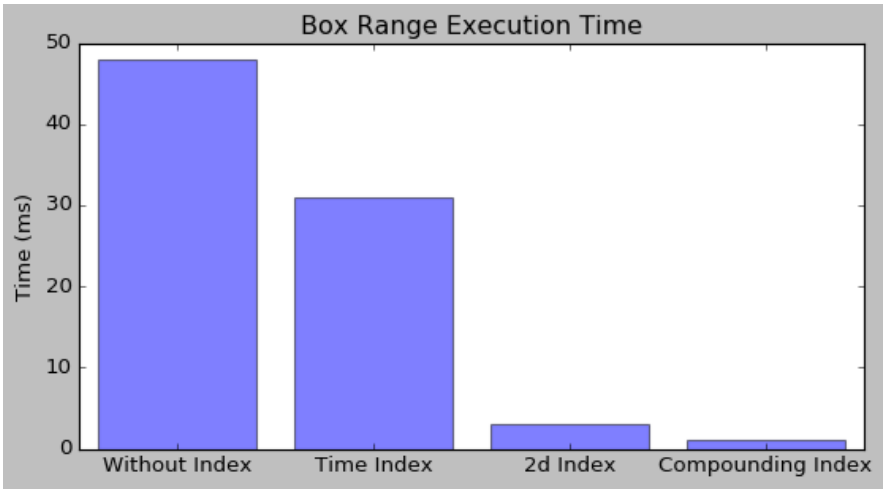
Ο Box Range αλγόριθμος εκτελέστηκε με 3 διαφορετικές παραμέτρους. Όπως και στον Circle Range, αρχικά αυξήθηκε ο χώρος μέσα στον οποίο αναζητούνται τα δεδομένα και στη συνέχεια το χρονικό διάστημα.

1ο Πείραμα

Αρχικά οι παράμετροι ήταν (0, 0, 3, 3, 1475000000, 1525000000).

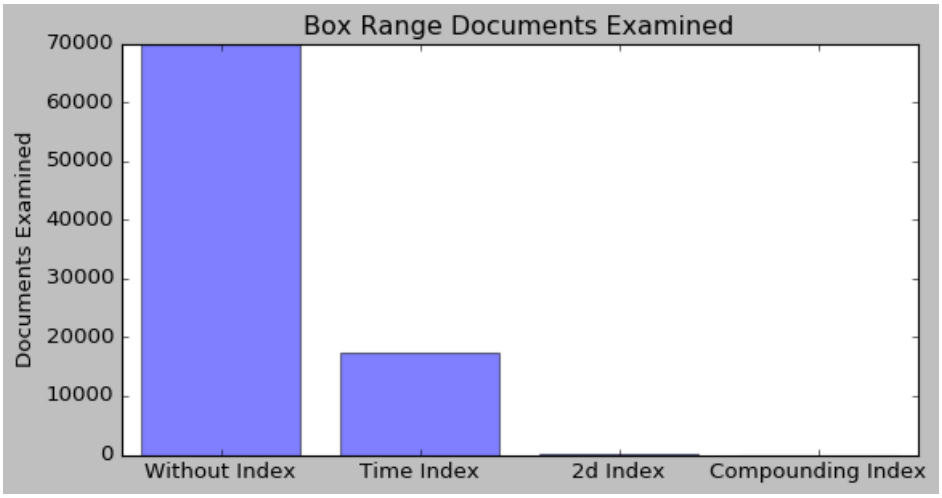
	Documents Examined	**Execution Time (ms)**	**Documents Returned**
Without index	70000	48	4
Time index	17356	31	4
2d index	28	3	4
Compounding index	6	1	4

Όπως και στον circle range αλγόριθμο, ο συνδυασμός των δύο index οδηγεί στην εξέταση λιγότερων εγγράφων και κατά συνέπεια, στην ελαχιστοποίηση του χρόνου εκτέλεσης του query.



Εικόνα 30 Box Range: Χρόνος Εκτέλεσης Query (2ο Σύνολο Δεδομένων, 1ο Πείραμα)

Στο παραπάνω γράφημα το compounding index παρουσιάζει τον μικρότερο χρόνο εκτέλεσης του query, ο οποίος τείνει στο 1 ms και ακολουθεί το 2d index με 3 ms. Στις άλλες 2 περιπτώσεις ο χρόνος είναι αρκετά μεγαλύτερος.



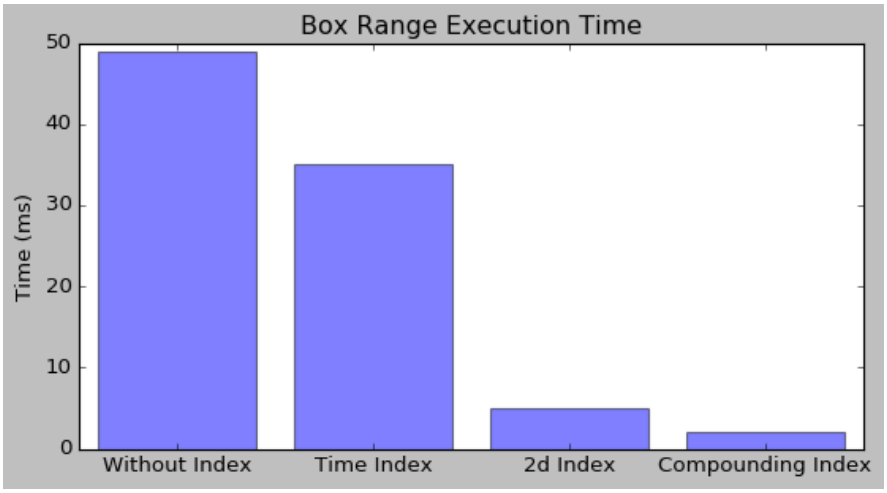
Εικόνα 31 Box Range: Αριθμός Εγγράφων που εξετάστηκαν (2ο Σύνολο Δεδομένων, 1ο Πείραμα)

Όσο αφορά τα έγγραφα που εξετάζονται, χωρίς κανένα index εξετάζονται όλα τα έγγραφα και με time index υπάρχει μια σημαντική μείωση των εγγράφων. Φυσικά τα καλύτερα αποτελέσματα παρουσιάζονται με 2d index και compounding index (28 και 6 αντίστοιχα).

2ο Πείραμα

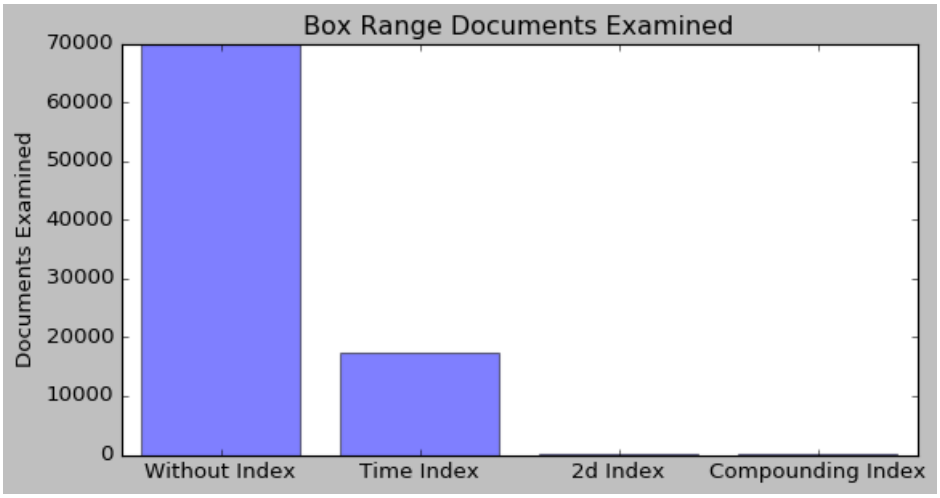
Όταν αυξηθεί μόνο το ορθογώνιο μέσα στο οποίο πραγματοποιείται η αναζήτηση των δεδομένων (0, 0, 10, 10), αυξάνεται ελάχιστα και ο χρόνος εκτέλεσης των query και τα αποτελέσματα είναι τα εξής:

	Documents Examined	**Execution Time (ms)**	**Documents Returned**
Without index	70000	49	28
Time index	17356	35	28
2d index	178	5	28
Compounding index	49	2	28



Εικόνα 32 Box Range: Χρόνος Εκτέλεσης Query (2ο Σύνολο Δεδομένων, 2ο Πείραμα)

Στο παραπάνω γράφημα ο χρόνος εκτέλεσης έχει αυξηθεί ελάχιστα κυρίως στη περίπτωση που χρησιμοποιείται 2d index compounding index. Αυτό οφείλεται στο γεγονός ότι ο αριθμός των εγγράφων που εξετάζονται έχει αυξηθεί για το 2d index και για το compounding index.



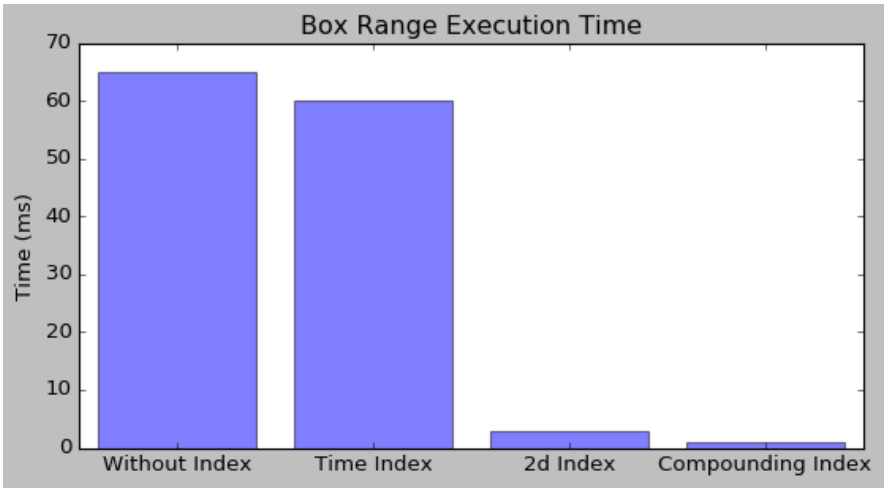
Εικόνα 33 Box Range: Αριθμός Εγγράφων που εξετάστηκαν (2ο Σύνολο Δεδομένων, 2ο Πείραμα)

3ο Πείραμα

Τέλος, όταν αυξηθεί το χρονικό διάστημα και τον ορθογώνιο παραμείνει όπως το πρώτο πείραμα, δηλαδή οι παράμετροι είναι (0, 0, 3, 3, 1430000000, 1570000000) τότε παρουσιάζονται τα παρακάτω αποτελέσματα.

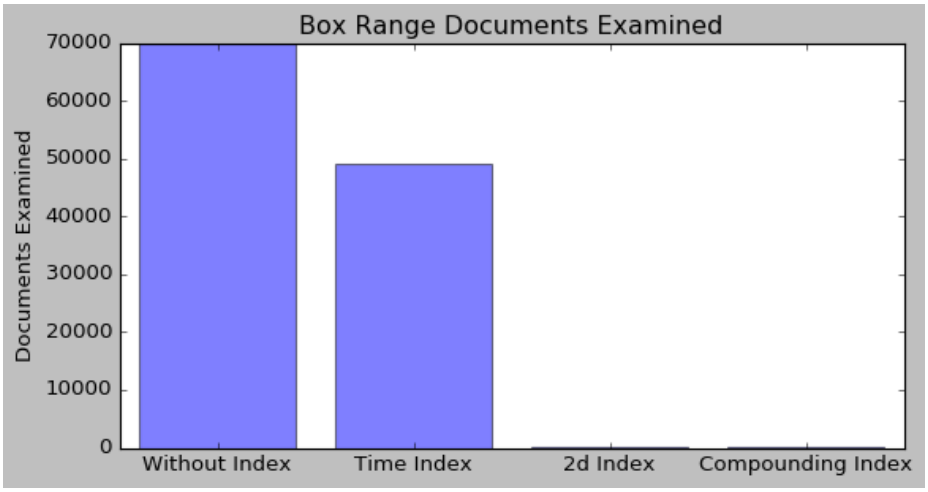
	Documents Examined	**Execution Time (ms)**	**Documents Returned**
Without index	70000	65	9
Time index	48981	60	9
2d index	28	3	9
Compounding index	22	1	9

Η μεγαλύτερη διαφορά σε σύγκριση με τις πρώτες μετρήσεις παρατηρείται όταν χρησιμοποιηθεί κάποιο index στο χρόνο, καθώς όσο αυξάνεται το χρονικό διάστημα τόσο περισσότερα έγγραφα εξετάζονται.



Εικόνα 34 Box Range: Χρόνος Εκτέλεσης Query (2ο Σύνολο Δεδομένων, 3ο Πείραμα)

Στο παραπάνω γράφημα παρατηρείται μεγάλη αύξηση του χρόνου εκτέλεσης στις δύο πρώτες περιπτώσεις (without index, time index), καθώς το χρονικό διάστημα είναι μεγαλύτερο.



Εικόνα 35 Box Range: Αριθμός Εγγράφων που εξετάστηκαν (2ο Σύνολο Δεδομένων, 3ο Πείραμα)

Επίσης, σε σύγκριση με τις πρώτες μετρήσεις παρατηρείται μεγάλη αύξηση των εγγράφων που εξετάζονται όταν χρησιμοποιηθεί time index.

5.3.5 Εξαγωγή όλου του trajectory

Όπως και στο πρώτο dataset εξετάζεται ο χρόνος εξαγωγής όλου του trajectory τόσο από το αρχικό collection όσο και από ένα δεύτερο που περιλαμβάνει ένα document με όλα τα στοιχεία για κάθε id.

	Αρχικό Collection Time (ms)	**Inverted Index Collection Time (ms)**
Box Range	49	0
Circle Range	56	0
k Nearest Neighbours	39	0

Στο δεύτερο collection όλοι οι χρόνοι είναι ίσοι με 0 και αυτό είναι απολύτως λογικό, καθώς το συγκεκριμένο dataset περιλαμβάνει μόνο 20 διαφορετικά id, δηλαδή 20 documents, που σε αντίθεση με τα 70000 documents του αρχικού collection, χρειάζονται πολύ λιγότερο χρόνο για να εξεταστούν.

6. ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτή την εργασία μελετήθηκαν κάποιοι αλγόριθμοι προσπέλασης χωροχρονικών δεδομένων που βρίσκονται αποθηκευμένα στη βάση δεδομένων Mongo και έγινε μια προσπάθεια βελτίωσης τους. Τα ερωτήματα που απαντήθηκαν αφορούσαν την εύρεση εγγράφων που βρέθηκαν μέσα σε ένα κύκλο ή ένα ορθογώνιο κάποια χρονική στιγμή, καθώς και την εύρεση των εγγράφων που βρέθηκαν κοντά σε κάποιο σημείο κάποια χρονική στιγμή.

Όσο αφορά τα δύο πρώτα ερωτήματα (Circle Range, Box Range), τα καλύτερα αποτελέσματα παρουσιάζονται όταν χρησιμοποιηθεί ένα compounding index, δηλαδή ένα index που θα συνδυάζει το χώρο και το χρόνο. Σχετικά με τα άλλα δύο index (time index, 2d index), το πιο θα παρουσιάζει καλύτερα αποτελέσματα εξαρτάται κυρίως από το χρονικό διάστημα και το χώρο (Box, Circle) που θα επιλεχθούν και το πόσα έγγραφα θα ικανοποιούν αυτές τις παραμέτρους.

Για το τελευταίο ερώτημα, την εύρεση τον κοντινότερων εγγραφών σε ένα σημείο, ο πρώτος αλγόριθμος που λαμβάνει υπόψη μόνο το χρόνο είναι ο λιγότερο αποδοτικός, καθώς δεν μπορεί να χρησιμοποιηθεί κάποιο 2d index και το compounding index είναι λιγότερο αποδοτικό. Στα αποτελέσματα υπάρχει σημαντική βελτίωση όταν χρησιμοποιηθούν στατιστικά δεδομένα για τα έγγραφα της βάσης. Σχετικά με τα στατιστικά δεδομένα, σε όσο περισσότερα κελιά χωριστεί ο χώρος τόσο καλύτερα τα αποτελέσματα του αλγόριθμου. Αυτό συμβαίνει γιατί ουσιαστικά εκτελείται ο Circle Range. Συνεπώς, όσο περισσότερα τα κελιά, τόσο μικρότερος ο κύκλος και τόσο λιγότερα τα έγγραφα που θα εξεταστούν. Μια άλλη παράμετρος, που πρέπει να ληφθεί σοβαρά υπόψη είναι η συγκέντρωση των δεδομένων. Αν τα δεδομένα είναι συγκεντρωμένα σε ένα πολύ μικρό ορθογώνιο, όπως συμβαίνει στο πρώτο dataset, τότε ο χώρος ή πιο συγκεκριμένα το ορθογώνιο θα πρέπει να χωριστεί σε πολλά μικρότερα κελιά. Φυσικά, σε κάθε περίπτωση η χρήση κυρίως του compounding index επιστρέφει τα καλύτερα αποτελέσματα. Επομένως, για την εύρεση των κοντινότερων εγγράφων σε ένα σημείο, το βέλτιστο αποτέλεσμα παρουσιάζεται όταν χρησιμοποιηθούν στατιστικά δεδομένα σε συνδυασμό με compounding index.

7. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Pokorny J. “NoSQL databases: a step to database scalability in web environment” *International Journal of Web Information Systems* Vol. 9*, No. 1.: 69-82, 2013*

[2] Karande Nikhil D. “A Survey Paper on NoSQL Databases: Key-Value Data Stores and Document Stores” *International Journal of Research in Advent Technology*, *Vol. 6, No. 2, 2018*

[3] Vatika S., Meenu D. “SQL and NoSQL Databases” *International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, No. 8, 2012*

[4] Banker K. “MongoDB in Action” *Manning Publications, 2012*

[5] Xuefeng G., Cheng B., Zhenqiang L., Yaojin Y. “ST-Hash An Efficient Spatiotemporal Index for Massive Trajectory Data in a NoSQL Database” *25th International Conference on Geoinformatics, Geoinformatics 2017, Buffalo, NY, USA, 2017*

[6] Acharjya D. P., Kauser A. P. “A Survey on Big Data Analytics: Challenges, Open Research Issues and Tools” *(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 7, No. 2, 2016*

[7] Elgendy N., Elragal A. “Big Data Analytics: A Literature Review Paper” *Industrial Conference on Data Mining, 2014*

[8] Davoudian A., Chen L., Menghi L. “A survey on NoSQL stores” *ACM Computing Surveys, Vol. 51, No. 2, Article 40, 2018*

[9] Catell, Rick. “Scalable SQL and NoSQL data stores” *ACM SIGMOD Record Vol. 39, No. 4:12-27, 2010*