ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΧΡΗΣΗ MAP-REDUCE/HADOOP ΓΙΑ ΑΝΑΛΥΣΗ ΚΥΚΛΟΦΟΡΙΑΚΩΝ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΟΥ ΟΓΚΟΥ

Διπλωματική Εργασία

του

Αντωνίου Μπουτοβίνα

Θεσσαλονίκη, Απρίλιος 2019

ΧΡΗΣΗ MAP-REDUCE/HADOOP ΓΙΑ ΑΝΑΛΥΣΗ ΚΥΚΛΟΦΟΡΙΑΚΩΝ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΟΥ ΟΓΚΟΥ

Αντωνίος Μπουτοβίνα

Πτυχίο Μηχανικού Πληροφορικής,ΑΤΕΙ Θεσσαλονίκης,2013

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Παπαναστασίου Δημήτριος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21/06/2019

Παπαναστασίου Δημήτριος	Μαργαρίτης Κων/νος	Ευαγγελίδης Γεώργιος
.....	.....	.....

Αντώνιος Μπουτοβίνας

.....

Περίληψη

Στην παρούσα διπλωματική εργασία αναλύονται κυκλοφοριακά δεδομένα μεγάλου όγκου με την χρήση του προγραμματιστικού μοντέλου MapReduce και του οικοσυστήματος το Hadoop. Βασικός στόχος της εργασίας είναι ο υπολογισμός των τιμών όπως η ταχύτητα των κινούμενων οχημάτων, ο όγκος των οχημάτων και οι χρόνοι ταξιδιού, που βοηθούν στην αποτίμηση της κυκλοφοριακής κατάστασης μια πόλης. Η κυκλοφορική συμφόρηση, είναι μια κυκλοφοριακή κατάσταση η οποία είναι εξαιρετικά επιβλαβής τόσο για την ψυχική όσο και για την σωματική υγεία του ανθρώπου. Επιπλέον, η κυκλοφοριακή συμφόρηση έχει ένα τεράστιο κόστος στην οικονομία. Συνεπώς η οποιαδήποτε πληροφόρηση για τις τρέχουσες κυκλοφοριακές συνθήκες είναι εξαιρετικά χρήσιμη.

Στην εν λόγω διπλωματική εργασία χρησιμοποιήθηκαν δεδομένα που αφορούν την πόλη της Θεσσαλονίκης. Τα δεδομένα αυτά είναι δεδομένα μεγάλου όγκου τα οποία εμπεριέχουν και γεωγραφική πληροφορία. Τα δυο αυτά χαρακτηριστικά των δεδομένων είναι βασικά και όρισαν τις μεθόδους που χρησιμοποιούνται, καθώς και το θεωρητικό υπόβαθρο που παρουσιάζεται. Επιπρόσθετα, πρέπει να αναφερθεί ότι τα δεδομένα είναι τόσο δεδομένα ιστορικού όσο και δεδομένα πραγματικού χρόνου.

Αρχικά χρησιμοποιήθηκε το σύστημα αρχείων του Hadoop το οποίο είναι το HDFS, για να αποθηκευτούν τα δεδομένα μεγάλου όγκου που ανακτήθηκαν. Στην συνέχεια γίνεται η επεξεργασία των γεωχωρικών δεδομένων της εφαρμογής τα οποία στην συνέχεια εισάγονται στο προγραμματιστικό μοντέλο του MapReduce όπου λαμβάνουν χώρα όλοι οι υπολογισμοί των τιμών που αναφέρονται παραπάνω. Μετά το πέρας του προγραμματιστικού μοντέλου του MapReduce οι υπολογισμένες τιμές αποθηκεύονται σε μια σχεσιακή βάση δεδομένων. Επιπλέον, δημιουργούνται κυκλοφοριακά προφίλ παρόμοιων χρονικών περιόδων για τα δεδομένα πραγματικού χρόνου.

Το τελικό αποτέλεσμα της παρούσας, είναι μία διαδικτυακή πύλη όπου εκεί τα δεδομένα που υπολογίστηκαν παρουσιάζονται κατηγοριοποιημένα και παρέχεται πληροφόρηση για τις κυκλοφοριακές συνθήκες τόσο για τις τρέχουσες όσο και για τις παρελθοντικές χρονικές στιγμές.

Λέξεις Κλειδιά: Hadoop, Map Reduce, Floating Car Data, κυκλοφοριακή συμφόρηση

Abstract

This thesis analyzes large volume of traffic data using the MapReduce programming model and the Hadoop ecosystem. The main objective of this work is to calculate the values, such as the speed of moving vehicles, the volume of vehicles and the travel times, which help to assess the traffic situation of a city. Traffic congestion is a traffic situation that is extremely harmful to both the mental and physical health of the human being. In addition, congestion has a huge cost in the economy. Consequently, any information about current traffic conditions is extremely useful.

In this dissertation, data that used concerning the city of Thessaloniki. These data are large volume data and geographic information. These two characteristics of the data are basic and defined the methods used, as well as the theoretical background presented. Additionally, it should be noted that the data is both historical data and real-time data.

Initially, the Hadoop file system, which is HDFS, was used to store the large volume data that was retrieved. The geospatial data of the application is then processed and subsequently entered into the MapReduce programming model where all the calculations of the values mentioned above take place. At the end of the MapReduce programming model, the calculated values are stored in a relational database. In addition, traffic profiles of similar time periods are created for real-time data.

The final result of the present is a web portal where the calculated data are categorized and information is provided on the traffic conditions for both current and past times.

Keywords: Hadoop, Map Reduce, Floating Car Data, traffic condition

Ευχαριστίες

Μετά την ολοκλήρωση της παρούσας διπλωματικής εργασίας θα ήταν τεράστια παράλειψη να μην ευχαριστήσω τους ανθρώπους που υπήρξαν συμπαραστάτες μου τόσο πρακτικά, όσο και ηθικά καθ' όλη τη διάρκεια της εκπόνησης της.

Συγκεκριμένα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Παπαναστασίου Δημήτριο. Επιπλέον, ιδιαιτέρως ευχαριστώ τον καθηγητή μου κ. Μαργαρίτη Κωνσταντίνο για την πολύτιμη υποστήριξη και τη βοήθεια που μου παρείχε. Επίσης, δεν θα μπορούσα να παραλείψω το Ινστιτούτο Βιώσιμης Κινητικότητας & Δικτύων Μεταφορών (Ι.ΜΕΤ), μέλος του Εθνικού Κέντρου Έρευνας και Τεχνολογικής Ανάπτυξης (Ε.Κ.Ε.Τ.Α) για την άμεση ανταπόκρισή τους σε όποιο πρόβλημα παρουσιάστηκε αναφορικά με τα δεδομένα που χρησιμοποιήθηκαν για την διπλωματική εργασία.

Τέλος, ευχαριστώ θερμά την οικογένεια μου που πάντα στέκεται δίπλα μου σε κάθε μου προσπάθεια, αλλά και την Έλενα για την αμέριστη συμπαράσταση και τη βοήθειά της σε ό,τι κι αν χρειάστηκα.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

Βασικός στόχος της διπλωματική εργασία είναι να αναλυθούν κυκλοφοριακά δεδομένα μιας πόλης, ώστε να εντοπιστούν συμβάντα κυκλοφοριακής συμφόρησης αυτής και να εξαχθούν συμπεράσματα, ιδίως μέσω της επεξεργασίας του υπολογισμού του χρόνου ταξιδίων, όπως αυτός υποδεικνύεται από την εφαρμογή. Επιπλέον, να προσδιοριστεί το κυκλοφοριακό προφίλ της εκάστοτε χρονικής περιόδου της ημέρας ώστε να δοθεί η δυνατότητα, με επεξεργασία των προϊόντων της συγκεκριμένης έρευνας μελλοντικά, να γίνει συγκριτική μελέτη αυτών και να εντοπιστούν ομοιότητες ή ανομοιότητες μεταξύ τους. Τα δεδομένα που χρησιμοποιήθηκαν στην παρούσα διπλωματική εργασία αφορούν την πόλη της Θεσσαλονίκης. Τέλος, τα παραπάνω αποτελέσματα παρουσιάζονται σε μια διαδικτυακή πύλη, προκειμένου να είναι προσβάσιμα στο ευρύ κοινό .

Πολλοί οργανισμοί ανά την Ευρώπη παρέχουν διάφορα σετ ανοιχτών δεδομένων σε οποιονδήποτε ενδιαφερόμενο. Το δεδομένα που θα χρησιμοποιηθούν στην συγκεκριμένη έρευνα, είναι τέτοια ανοιχτά δεδομένα. Ποιο συγκεκριμένα το Ινστιτούτο Βιώσιμης Κινητικότητας & Δικτύων Μεταφορών (Ι.ΜΕΤ), μέλος του Εθνικού Κέντρου Έρευνας και Τεχνολογικής Ανάπτυξης (Ε.Κ.Ε.Τ.Α), διαθέτει 19 σετ ανοιχτών δεδομένων, κάποια από τα οποία χρησιμοποιήθηκαν στην συγκεκριμένη εργασία [1]. Αναλυτικότερα, τα σετ δεδομένων που θα χρησιμοποιηθούν είναι δεδομένα κινούμενων οχημάτων (FCD) και δεδομένα τμημάτων οδικού δικτύου της πόλης.

Τα εργαλεία τα οποία θα χρησιμοποιηθούν, στην προκειμένη περίπτωση, ώστε να εξαχθούν τα παραπάνω αποτελέσματα είναι στην πλειοψηφία τους εργαλεία ανοιχτού λογισμικού. Ο προσανατολισμός της εφαρμογής αυτή είναι να μπορεί να επεκταθεί και να είναι ικανή να εφαρμοστεί και σε μεγαλύτερες πόλεις ώστε να εξυπηρετήσει τις εκάστοτε ανάγκες. Τα βασικότερα εργαλεία είναι το Apache Hadoop, μια σχεσιακή βάση MySQL, ένας java application server WildFly που θα εξυπηρετεί την πλατφόρμα και τέλος cloud υπηρεσίες της Amazon που φιλοξενούν την πλατφόρμα καθώς και την βάση δεδομένων.

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Ένα από τα σημαντικότερα προβλήματα της εποχής, ιδίως στα μεγάλα αστικά κέντρα, είναι η κυκλοφοριακή συμφόρηση του οδικού δικτύου που προκαλεί την μείωση

των ταχυτήτων των οχημάτων, την αύξηση της διάρκειας των ταξιδιών και στις πιο ακραίες περιπτώσεις, δημιουργεί ακόμα και ουρές ακινητοποιημένων στην ουσία οχημάτων. Όσο ο πληθυσμός των οχημάτων αυξάνεται, τόσο αυξάνεται και η αλληλεπίδραση τους με αποτέλεσμα την δημιουργία κυκλοφοριακής συμφόρησης. Επιπρόσθετα, όσο η κυκλοφοριακή συμφόρηση μεγαλώνει, τόσο μεγαλώνει και ο χρόνος που οι πολίτες είναι αναγκασμένοι να ξοδεύουν μέσα στα οχήματα τους, χωρίς να καταφέρνουν να φτάσουν στον προορισμό σε ένα λογικό, με βάση την απόσταση που επιθυμούν να διανύσουν, χρόνο.

Καθώς η κυκλοφοριακή συμφόρηση αποτελεί ένα πρόβλημα που εμφανίζεται όλο και συχνότερα με την πάροδο του χρόνου, οι επιπτώσεις αυτής στην οικονομία, στο περιβάλλον, αλλά και στους ίδιους τους πολίτες αυξάνονται δραματικά. Σύμφωνα με μια έρευνα που δημοσίευσε η Ευρωπαϊκή Ένωση στο τέλος του 2018 και αφορά τις γενικότερες επιπτώσεις των μεταφορών, βρίσκουμε ότι το κόστος που προκαλείται στην οικονομία λόγο της κυκλοφοριακής συμφόρησης έχει υπολογιστεί στα 270 δισεκατομμύρια ευρώ (Περίπου το 2% του ακαθάριστου εθνικού προϊόντος της Ευρώπης των 28 μελών). [2]

Επιπλέον σύμφωνα με μία έρευνα που δημοσίευσε η Ευρωπαϊκή Επιτροπή Περιβάλλοντος (ΕΕΑ) αναφέρεται ότι πάνω από 100 εκατομμύρια πολίτες, στα 33 κράτη - μέλη του οργανισμού είναι εκτεθειμένοι σε μεγάλο θόρυβο, έντασης άνω των 55dB, που προκαλείται αποκλειστικά και μόνο από την κυκλοφοριακή συμφόρηση. Ο θόρυβος αυτός είναι ο δεύτερος πιο επιβλαβής και στρεσογόνος παράγοντας στην Ευρώπη, μετά από την ατμοσφαιρική ρύπανση που αποτελεί το πρώτο και κυριότερο πρόβλημα, σύμφωνα με τον Παγκόσμιο Οργανισμό Υγείας (WHO). [3]

Επιπρόσθετα, το Centre for Economics and Business Research (CEBR) συνέταξε μια έκθεση για το περιβαλλοντικό κόστος των σταματημένων οχημάτων σε συνθήκες συμφόρησης που αφορά στις Ηνωμένες Πολιτείες της Αμερικής, την Γαλλία, την Γερμανία και το Ηνωμένο Βασίλειο. Σύμφωνα με τη έκθεση αυτή, η πρόβλεψη των εκπεμπόμενων ρύπων για τις τέσσερις αυτές χώρες είναι ότι το 2030 θα έχουν αυξηθεί οι εκπομπές διοξειδίου του άνθρακα κατά 16%. Θα αναφέρουμε το παράδειγμα της Γαλλίας, όπου την χρονιά (2013) που συντάχθηκε η έκθεση, είχε τις χειρότερες κυκλοφοριακές συνθήκες από τις τέσσερις χώρες της έρευνας και παρατηρείται ότι οι Γάλλοι αυτοκινητιστές μετακινούμενοι σπατάλησαν τα περισσότερα καύσιμα και τις

περισσότερες ώρες σε συνθήκες συμφόρησης ετησίως, ήτοι 79 λίτρα και 44 ώρες αντίστοιχα ανά όχημα. [4]

Λαμβάνοντας υπόψιν τα παραπάνω δεδομένα, είναι ξεκάθαρο πως το να είναι δυνατόν να εντοπιστούν με εύκολο και λειτουργικό τρόπο τα σημεία που υπάρχει συμφόρηση σε ένα οδικό δίκτυο είναι μια πληροφορία χρήσιμη, τόσο για προσωπική πληροφόρηση του καθενός, όσο και για τον κεντρικό σχεδιασμό των πόλεων, προκριμένου να ληφθούν τα απαραίτητα μέτρα για την σταδιακή αποκλιμάκωση των συμφορημένων περιοχών, με τελικό στόχο την βελτίωση των συνθηκών ζωής, ιδιαίτερα στα μεγάλα αστικά κέντρα που μαστίζονται από το πρόβλημα αυτό.

1.2 Σκοπός – Στόχοι

Σκοπός αυτής της διπλωματικής εργασίας είναι η ανάλυση των κυκλοφοριακών δεδομένων και η δημιουργία της διαδικτυακής πύλης όπου θα παρουσιάζονται τα αποτελέσματα που θα αφορούν τις κυκλοφοριακές συνθήκες της Θεσσαλονίκης.

Οι επιμέρους στόχοι της διπλωματικής εργασίας είναι:

- Η απόκτηση ιστορικού των κυκλοφοριακών δεδομένων και ο υπολογισμός των τιμών που παρουσιάζουν ενδιαφέρον, όπως οι χρόνοι ταξιδιού, οι ταχύτητες των κινούμενων οχημάτων, και ο αριθμός των εγγραφών που εντοπίστηκαν ανά διαδρομή.
- Η απόκτηση των κυκλοφοριακών δεδομένων πραγματικού χρόνου και ο υπολογισμός των ίδιων στατιστικών τιμών όπως και στα ανωτέρω ιστορικά κυκλοφοριακά δεδομένα. Επιπλέον, η δημιουργία κυκλοφοριακών προφίλ παρόμοιων χρονικών περιόδων της ημέρας, ώστε να είναι δυνατή η καλύτερη κατανόηση των κυκλοφοριακών συνθηκών.
- Η αποθήκευση των παραπάνω αποτελεσμάτων σε μία σχεσιακή βάση δεδομένων, προκειμένου αυτά να είναι διαθέσιμα για παρουσίαση ή και περαιτέρω μελλοντική ανάλυση.

1.3 Ερωτήματα

Ζήτημα της εν λόγω διπλωματικής έρευνας αποτελεί το κατά πόσο μπορούν πράγματι να αξιοποιηθούν στην πράξη τα ανοιχτά δεδομένα μεγάλου όγκου σε συνδυασμό με τα εργαλεία ανοιχτού κώδικα, ώστε να αναλυθούν τα κυκλοφοριακά δεδομένα, με σκοπό την ενημέρωση των μετακινούμενων σχετικά με τις κυκλοφοριακές συνθήκες, εφόσον μάλιστα οι σχετικές πληροφορίες θα είναι πλέον εύκολα προσβάσιμες στον καθένα μέσω μιας εύχρηστης εφαρμογής.

1.4 Συνεισφορά

Ένα από τα κυριότερα προβλήματα που απασχολεί τους συγκοινωνιολόγους και στο οποίο προσπαθούν να δώσουν λύση είναι αυτό του εντοπισμού της κυκλοφοριακής συμφόρησης σε ένα οδικό δίκτυο. Για την επίλυση του προβλήματος αυτού έχουν αναπτυχθεί αρκετά συστήματα που χρησιμοποιούνται για τον σκοπό αυτό, όπως είναι οι κάμερες, οι ανιχνευτές επαγωγικού βρόχου, ανιχνευτές Bluetooth κ.α. Τα μειονέκτημα των συγκεκριμένων συστημάτων ωστόσο, είναι αφενός το τεράστιο κόστος εγκατάστασης και συντήρησης που συνεπάγονται, ώστε να είναι δυνατή η άρτια λειτουργία τους, με αποτέλεσμα τελικά να εγκαθίστανται σε περιορισμένα και συνήθως περισσότερο κεντρικά σημεία των περιοχών ενδιαφέροντος, αφετέρου το ότι οι μετρήσεις τους είναι περιορισμένων δυνατοτήτων, καθώς περιορίζονται κυρίως σε μετρήσεις απλώς του φόρτου των οχημάτων, και τέλος το ότι είναι δυνατόν να χρησιμοποιηθούν και να οδηγήσουν σε ασφαλή συμπεράσματα μόνο για την περιοχή στην οποία έχουν εγκατασταθεί, με αποτέλεσμα έτσι τελικά να καλύπτεται ένα πάρα πολύ μικρό μόνο τμήμα του συνολικού οδικού δικτύου, καθώς είναι πρακτικά αδύνατο, όπως άλλωστε προαναφέρθηκε, να εγκατασταθούν τα συστήματα αυτά σε κάθε περιοχή ενδιαφέροντος καλύπτοντας μάλιστα κάθε σημείο αυτής.

Λαμβανομένων υπόψη όλων των ανωτέρω και των περιορισμών που τα προαναφερόμενα συστήματα συνεπάγονται, όπως αυτά αναλύθηκαν, στην συγκεκριμένη διπλωματική εργασία προτείνεται η χρησιμοποίηση των GPS στιγμάτων που παρέχονται από πολλές συσκευές, όπως κινητά τηλεφώνα,tablet, αυτοκίνητα κ.α., και βρίσκονται σε πληθώρα, προκειμένου να είναι δυνατός ο υπολογισμός τόσο των τρεχουσών, όσο και των παρελθοντικών, υπό τη μορφή ιστορικού, κυκλοφοριακών συνθηκών, στην πόλη της Θεσσαλονίκης. Τελικός στόχος είναι να μπορούν να εξαχθούν έγκυρες πληροφορίες για την κυκλοφοριακή κατάσταση της πόλης, καθώς και να είναι δυνατή η άμεση διοχέτευσή

τους, μέσω της πλατφόρμας, στους μετακινούμενους αυτοκινητιστές. Οι αυτοκινητιστές που θα επιλέξουν να συμβουλεύονται την πλατφόρμα αναμένεται να έχουν μεγάλα οφέλη ως προς την ποιότητα των προσωπικών τους μετακινήσεων, αλλά και δραστική συμβολή στην αντιμετώπιση και την αποφυγή των προβλημάτων της κυκλοφοριακής συμφόρησης, όπως αυτά παρουσιάστηκαν ανωτέρω.

Συγκεκριμένα, θα έχουν τη δυνατότητα να εξοικονομούν πολύτιμο χρόνο επιλέγοντας εναλλακτικές διαδρομές στις οποίες δεν παρατηρείται έντονη κυκλοφοριακή συμφόρηση, με αποτέλεσμα να μπορούν να φτάσουν στον προορισμό τους συντομότερα και με μεγαλύτερη άνεση. Ταυτόχρονα δε, είναι δεδομένο ότι με τον τρόπο αυτό θα εξοικονομούν καύσιμα, με αποτέλεσμα την σταδιακή μείωση του κόστους μετακίνησής τους που σε βάθος χρόνου μπορεί να οδηγήσει στην εξοικονόμηση αρκετών χρημάτων από τις μετακινήσεις τους.

Επιπλέον, θα μπορούν να συμβάλλουν στην μείωση των περιβαλλοντικών ζητημάτων που πέραν του ότι αποτελούν σοβαρότατα αυτούσια προβλήματα, που χρήζουν άμεσης αντιμετώπισης, συνεπάγονται περαιτέρω και βαρύτατες συνέπειες τόσο για την σωματική, όσο και για την ψυχική υγεία των πολιτών. Έτσι, θα μπορούν να συμβάλλουν στη μείωση του θορύβου στις περιοχές έντονης κυκλοφοριακής συμφόρησης, που όπως σημειώθηκε ανωτέρω αποτελεί έναν από τους βασικότερους στρεσογόνους παράγοντες για τους κατοίκους, όπως ακόμα και στη μείωση της περιβαλλοντικής ρύπανσης, ουσιαστικά μέσω της μείωσης του χρόνου χρησιμοποίησης και λειτουργίας των οχημάτων.

1.5 Διάρθρωση της μελέτης

Η διάρθρωση που ακολουθείται από την διπλωματική εργασία είναι η ακόλουθη. Στο Κεφάλαιο 1 γίνεται μία εισαγωγή στο πρόβλημα που αναλύεται καθώς και στην σημαντικότητά του. Επιπλέον αναφέρονται τα κύρια ερωτήματα της διπλωματικής εργασίας καθώς και η συνεισφορά της. Στην συνέχεια στο Κεφάλαιο 2 κεφάλαιο 2 αναπτύσσεται το θεωρητικό υπόβαθρό και η βιβλιογραφική επισκόπηση που απαιτείται για να υλοποιηθεί η παρούσα. Το Κεφάλαιο 3 συζητά την μεθοδολογία που αναπτύχθηκε και ακολουθήθηκε για την υλοποίηση της εφαρμογής που αναπτύχθηκαν στα πλαίσια της διπλωματικής εργασίας. Τέλος στο Κεφάλαιο 4 αναπτύσσονται η σύνοψη και τα συμπεράσματα μετά την εφαρμογή της μεθοδολογίας, του περιορισμούς που

συναντήθηκαν σε αυτήν και τέλος οι μελλοντικές επεκτάσεις που μπορούν να υλοποιηθούν στα πλαίσια αυτής.

2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

Στο σημείο αυτό, κρίνεται σκόπιμο να γίνει αναφορά στα βασικά εργαλεία και στις τεχνολογίες που χρησιμοποιήθηκαν προκειμένου να δημιουργηθεί η εφαρμογή, η οποία αποτελεί το βασικότερο εργαλείο για την ανάπτυξη του αντικειμένου της εν λόγω διπλωματικής εργασίας. Πιο συγκεκριμένα, αρχικά θα αναλυθούν τα βασικά υποπρογράμματα του λογισμικού του του Apache Hadoop, τα οποία είναι το MapReduce και το HDFS. Στην συνεχεία, θα γίνει λόγος για το γεωδαιτικό σύστημα WGS84, αλλά και μια σύντομη αναφορά στο «Παγκόσμιο Σύστημα Στιγματοθέτησης» Global Positioning System (GPS). Τέλος, θα πραγματοποιηθεί μια μικρής έκτασης αναφορά στα «Μεγάλα δεδομένα» Big Data για να δούμε στην συνέχεια αν τα δεδομένα που χρησιμοποιούνται στην παρούσα εργασία τηρούν τις προϋποθέσεις για να θεωρηθούν Big Data.

2.1 Εισαγωγή στο Apache Hadoop

Το Apache Hadoop είναι μια συλλογή προγραμμάτων ανοιχτού λογισμικού γραμμένη σε γλώσσα προγραμματισμού Java, η οποία χρησιμοποιείται για την αποθήκευση και επεξεργασία τεράστιου όγκου δεδομένων, κλιμακούμενα και κατανεμημένα σε συστάδες συμβατικών υπολογιστών. Σύμφωνα με τους συνιδρυτές του Hadoop, Doug Cutting και Mike Cafarella, η γένεση του Apache Hadoop προήλθε από την ερευνητική εργασία που αφορούσε το σύστημα αρχείων της Google και δημοσιεύθηκε τον Οκτώβριο του 2003 [5]. Αυτή η ερευνητική εργασία έδωσε το έναυσμα για μια άλλη ερευνητική εργασία με τίτλο «MapReduce: Simplified Data Processing on Large Clusters». Η ανάπτυξη του Apache Hadoop ξεκίνησε στο έργο Apache Nutch, αλλά τελικά μεταφέρθηκε στο νέο έργο με την αντίστοιχη ονομασία Apache Hadoop στις αρχές του 2006. Ο Doug Cutting, όπως έχει αποκαλύψει ο ίδιος σε μια συνέντευξη του, εμπνεύστηκε την ονομασία του εν λόγω έργου από ένα παιχνίδι του γιού του και συγκεκριμένα από έναν λούτρινο ελέφαντα [6]. Ο αρχικός κώδικας που μεταφέρθηκε από το έργο Nutch αποτελείται από περίπου 5.000 γραμμές κώδικα για το

HDFS και περίπου 6.000 γραμμές κώδικα για το MapReduce. Από το 2006 οπότε και κυκλοφόρησε το Hadoop 0.1.0 το έργο είναι ενεργό έχοντας μια μεγάλη κοινότητα που συνεισφέρει σε αυτό, φτάνοντας μέχρι σήμερα όπου βρίσκεται ήδη στην έκδοση 3.1.x [7].

2.1.1 Βασικές ενότητες του Apache Hadoop

Οι βασικές ενότητες από τις οποίες αποτελείται το Apache Hadoop σήμερα είναι οι εξής 4:

- **Hadoop Common** Είναι μία συλλογή από βιβλιοθήκες και βοηθητικά προγράμματα τα οποία είναι απαραίτητα για τις υπόλοιπες ενότητες του.
- **Hadoop Distributed File System (HDFS)** Είναι ένα κατανεμημένο σύστημα αρχείων που αποθηκεύει δεδομένα σε συστάδες συμβατικών υπολογιστών, παρέχοντας πολύ υψηλές ταχύτητες.
- **Hadoop YARN** Αρχικά, πρέπει να σημειωθεί ότι η συγκεκριμένη ενότητα εισήχθη μεταγενέστερα από τις υπόλοιπες, το 2012, και επί της ουσίας είναι μια πλατφόρμα η οποία είναι υπεύθυνη για τη διαχείριση των πόρων στις συστάδες των υπολογιστών και επιπλέον, είναι υπεύθυνη για τον προγραμματισμό της εκτέλεσης των εφαρμογών των χρηστών.
- **Hadoop MapReduce** Είναι μια υλοποίηση του προγραμματιστικού μοντέλου MapReduce [8] που χρησιμεύει για την επεξεργασία δεδομένων μεγάλης κλίμακας. [9]

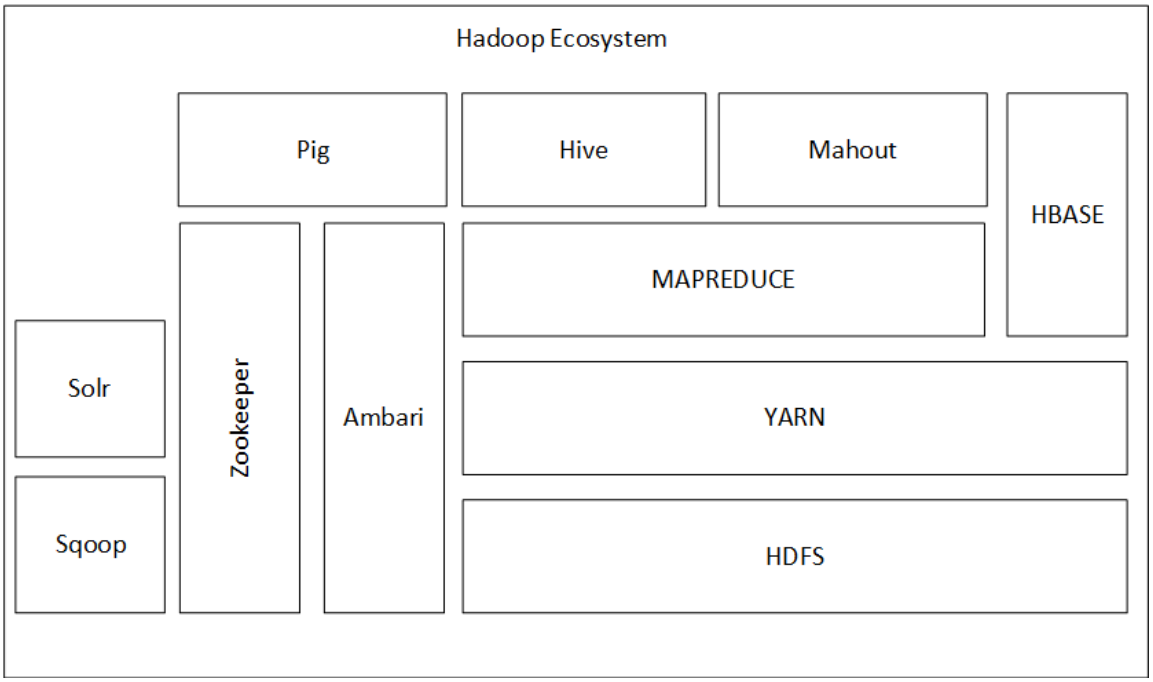
2.1.2 Υποενότητες του Apache Hadoop

Ο όρος «Hadoop Οικοσύστημα» (μτφ: Hadoop Ecosystem) χρησιμοποιείται τόσο για να περιγράψει τις βασικές ενότητες του Apache Hadoop, όσο όμως και για τις υποενότητες λογισμικού, οι οποίες μπορούν να λειτουργήσουν είτε συνεργατικά τόσο με τις βασικές ενότητες του Apache Hadoop, όσο και μεταξύ τους, είτε ως επεκτάσεις των κύριων ενοτήτων του Apache Hadoop. Στην συνέχεια, θα αναφερθούν μερικές από τις σημαντικότερες υποενότητες του «οικοσυστήματος» μέσα από μια σύντομη περιγραφή τους.[10]

- **Mahout:** Πρόκειται για μια δημοφιλή βιβλιοθήκη μηχανικής μάθησης. Αποτελείται από αλγόριθμους εξόρυξης δεδομένων και μηχανικής μάθησης για την εκτέλεση εργασιών, όπως η ομαδοποίηση, η ταξινόμηση. Επιπρόσθετα υποστηρίζει πολλαπλές βιβλιοθήκες κατανεμημένης επεξεργασίας. [11]
- **Apache HBase:** Είναι ένα κατανεμημένο αποθηκευτικό σύστημα για Big Data για τον Hadoop. Το HBase επιτρέπει την τυχαία πρόσβαση για εγγραφή ή ανάγνωση σε μεγάλου όγκου δεδομένα σε πραγματικό χρόνο. Έχει σχεδιαστεί ως μοντέλο αποθήκευσης δεδομένων που βασίζεται σε στήλες (column-oriented) και εμπνεύστηκε από την ερευνητική εργασία της Google με τίτλο «Bigtable: A Distributed Storage System for Structured Data».[12]
- **Hive:** Αποτελεί μια υποενότητα βασισμένη στο σύστημα αποθήκευσης αρχείων του Hadoop και αναπτύχθηκε από το Facebook και ενσωματώθηκε στο οικοσύστημα του Hadoop. Επιτρέπει στους χρήστες να πραγματοποιήσουν ερωτήματα σε γλώσσες τύπου SQL, όπως είναι και η HiveQL στα δεδομένα που είναι αποθηκευμένα στο HDFS είτε σε συστήματα αποθήκευσης όπως το HBase. Αυτό επιτρέπει στους προγραμματιστές SQL οι οποίοι χωρίς να έχουν εμπειρία πάνω στο MapReduce, να το χρησιμοποιούν τις δυνατότητες του και να επεξεργάζονται δεδομένα .[13]
- **Pig:** Είναι ένα έργο ανοιχτού λογισμικού που βασίζεται στο Hadoop για την ανάλυση των δεδομένων μεγάλου όγκου μέσω της δικής του γλώσσας τύπου SQL, η οποία ονομάζεται Pig Latin. Με την χρήση της Pig Latin μπορεί να επιτευχθούν διάφορες ενέργειες επάνω σε μεγάλα σετ δεδομένων, όπως για παράδειγμα η συγχώνευση 2 ή περισσοτέρων σετ δεδομένων. Ένας από τους κύριους χρήστες του Pig είναι η εταιρία Yahoo.[14]
- **Apache Sqoop:** Πρόκειται για ένα εργαλείο που σχεδιάστηκε για την αποτελεσματική και ταυτόχρονα μαζική εισαγωγή δεδομένων από τις σχεσιακές βάσεις δεδομένων στο HDFS και την εξαγωγή δεδομένων από HDFS σε σχεσιακές βάσεις δεδομένων. Συνεργάζεται με τις περισσότερες σύγχρονες σχεσιακές βάσεις δεδομένων, όπως είναι η

MySQL, η PostgreSQL, η Oracle, ο Microsoft SQL Server και η IBM DB2. Το Sqoop ανήκει και επίσημα στα έργα του Apache Foundation από τον Μάρτιο του 2012. [15]

- **Apache Solr:** Πρόκειται για ένα έργο το οποίο είναι εξαιρετικά επεκτάσιμο, αξιόπιστο, ανεκτικό σε λάθη και το οποίο υποστηρίζει την κατανεμημένη αναζήτηση και τη δημιουργία ευρετηρίων. Αυτό επιτρέπει την ανάπτυξη διαδικτυακών εφαρμογών με πολύ αποτελεσματική και πολύπλευρη αναζήτηση κειμένου, καθώς και αποτελεσματικό χειρισμό εγγράφων. Πολλές εταιρίες και οργανισμοί οι οποίοι διαθέτουν διαδικτυακές εφαρμογές με τεράστια διαδικτυακή κίνηση έχουν ενσωματώσει το Solr.[16]
- **Apache Ambari:** Είναι ένα εργαλείο που υποστηρίζει την ευκολότερη διαχείριση και παρακολούθηση του Apache Hadoop. Το Ambari χειρίζεται τις περισσότερες από τις ενότητες και υποενότητες του «Hadoop οικοσυστήματος», ως κεντρική διαχείριση. Ως μέρος αυτής της κεντρικής διαχείρισης παρέχεται μια διαδικτυακή εφαρμογή μέσω της οποίας εποπτεύεται η εύρυθμη λειτουργία του «Hadoop οικοσυστήματος». [17]
- **Apache Zookeeper** είναι επίσης ένα υποπρόγραμμα Hadoop που χρησιμοποιείται για τη διαχείριση των υποενοτήτων του Hadoop, Hive, Pig, HBase, Solr και πολλών άλλων. Το Zookeeper είναι μια υπηρεσία συντονισμού και συγχρονισμού κατανεμημένων εφαρμογών καθώς και της ρύθμισης των παραμέτρων αυτών. [18]



Εικόνα 2-1:Hadoop οικοσύστημα

2.2 Hadoop Distributed File System (HDFS)

Το HDFS είναι το σύστημα αρχείων του Hadoop, το οποίο βασίζεται σε UNIX. Το HDFS θα μπορούσε να λεχθεί ότι προέρχεται από το συστήματα αρχείων της Google καθώς έχει ακολουθήσει πολλές από τις από τις αρχές του. Ένα σημαντικό χαρακτηριστικό του Hadoop είναι η τμηματοποίηση και κατανομή των δεδομένων και ο υπολογισμός των τμηματοποιημένων δεδομένων σε πολλούς υπολογιστές ταυτόχρονα. Στο HDFS τα αρχεία δεδομένων τμηματοποιούνται σε blocks δεδομένων, τα οποία αποθηκεύονται στη συστάδα τον υπολογιστών που φιλοξενεί το Hadoop. Οι δυνατότητες υπολογισμού, αποθήκευσης, ταχύτητας εγγραφής και ανάγνωσης της εγκατάστασης Hadoop αυξάνονται προσθέτοντας επιπλέον υπολογιστές στην συστάδα. Το σύστημα αρχείων HDFS, μπορεί να είναι προσπελάσιμο με πολλούς διαφορετικούς τρόπους. Το HDFS παρέχει ένα API σε γλώσσα προγραμματισμού Java, ώστε να μπορεί να χρησιμοποιηθεί από τις εφαρμογές. Οι συστάδες του Hadoop στην Yahoo χρησιμοποιούν 40.000 διακομιστές και αποθηκεύουν 40 petabytes δεδομένων. [10] [19][20]

2.2.1 Χαρακτηριστικά του HDFS

Στο σημείο αυτό κρίνεται χρήσιμο να γίνει μία αναφορά στα βασικά χαρακτηριστικά του συστήματος αρχείων HDFS. Αρχικά, πρόκειται για ένα σύστημα αρχείων το οποίο είναι ανεκτικό σε σφάλματα, υπό την έννοια ότι παραμένει λειτουργικό ακόμη και κάτω από δυσμενείς συνθήκες. Στο σύστημα αυτό τα δεδομένα χωρίζονται σε τμήματα και εν συνεχεία δημιουργούνται πολλαπλά αντίγραφα του κάθε τμήματος δεδομένων και αποθηκεύονται σε διαφορετικούς υπολογιστές του cluster. Αυτό έχει ως αποτέλεσμα αν κάποιος υπολογιστής από το cluster χαλάσει, η πρόσβαση στα δεδομένα του να παραμένει ενεργή μέσω των αντιγράφων του που ήδη υπάρχουν. Περαιτέρω, το HDFS είναι ένα σύστημα εύκολα επεκτάσιμο, ενώ ταυτόχρονα έχει τη δυνατότητα να διαχειριστεί μεγάλο όγκο δεδομένων. Επιπρόσθετα, αξίζει να σημειωθεί ότι η αρχιτεκτονική που χρησιμοποιείται είναι τύπου συντονιστής / εργαζόμενος (master / worker), κατά την οποία το ρόλο του συντονιστή κατέχει πάντα ένας υπολογιστής που κατανέμει και συντονίζει τις εργασίες, ενώ στο ρόλο του εργαζόμενου μπορεί να βρίσκονται ένας ή περισσότεροι υπολογιστές που υλοποιούν το έργο που τους ανατίθεται από τον συντονιστή. Τέλος, βασικό χαρακτηριστικό είναι ότι χρησιμοποιεί την αρχή «Write once read many», σύμφωνα με την οποία από την στιγμή που μια πληροφορία αποθηκευτεί στο σύστημα δεν είναι δυνατή η τροποποίησή της, διασφαλίζοντας την με τον τρόπο αυτό από τον κίνδυνο αλλοίωσης της.[10][20]

2.2.2 Αρχιτεκτονική του HDFS

Όπως αναφέρεται παραπάνω, η αρχιτεκτονική του συστήματος αρχείων HDFS είναι τύπου συντονιστής / εργαζόμενος. Κατά την αρχιτεκτονική αυτή, ο διακομιστής που έχει τον ρόλο του συντονιστή ονομάζεται NameNode και ο ή οι διακομιστές που έχουν τον ρόλο του εργαζομένου ονομάζονται DataNodes. Ο NameNode είναι ένας διακομιστής που διαχειρίζεται το namespace του συστήματος αρχείων και επιπλέον διαχειρίζεται τα δικαιώματα πρόσβασης που έχουν οι χρήστες στα αρχεία. Μια ακόμη από τις βασικές αρμοδιότητες του NameNode είναι η διαίρεση των δεδομένων σε τμήματα και η κατανομή ως προς το ποια τμήματα θα αποθηκεύσει ο κάθε DataNode. Ο DataNode αποθηκεύει τα δεδομένα που του έχουν ανατεθεί και τα διαμοιράζει σύμφωνα με τα αιτήματα που δέχεται. Επιπρόσθετα, πραγματοποιεί και τις εργασίες δημιουργίας και διαγραφής των τμημάτων. Ταυτόχρονα, ένας τρίτος διακομιστής που βρίσκεται στο

σύστημα αρχείων του HDFS είναι ο Secondary NameNode ο οποίος είναι υπεύθυνος για δημιουργεί περιοδικά αντίγραφα του NameNode. Αν οποιαδήποτε στιγμή ο NameNode αποτύχει τότε αντικαθίσταται αυτομάτως από το πιο πρόσφατο αντίγραφο που έχει διαθέσιμο ο Secondary NameNode.[10][19]

Ο μηχανισμός του HDFS κάτω από φυσιολογικές συνθήκες δημιουργεί τρία αντίγραφα για κάθε τμήμα δεδομένων. Η στρατηγική αποθήκευσης των αντιγράφων είναι η ακόλουθη. Το πρώτο αντίγραφο αποθηκεύεται τοπικά, ενώ το δεύτερο και το τρίτο αποθηκεύονται σε διαφορετικούς DataNodes το καθένα τους. Το προκαθορισμένο μέγεθος των τμημάτων των δεδομένων είναι 128 MB όμως αυτό μπορεί να αλλάξει σύμφωνα με τις εκάστοτε ανάγκες [21].

2.3 Hadoop MapReduce

Το MapReduce είναι ένα προγραμματιστικό μοντέλο, το οποίο χρησιμοποιείται για την επεξεργασία μεγάλου όγκου δεδομένων τα οποία κατανέμονται σε μία συστάδα υπολογιστών. Το μοντέλο του MapReduce θεωρείται ότι είναι ο πυρήνας ολοκλήρου του «Hadoop οικοσυστήματος», δεδομένου ότι δίνει την δυνατότητα της μαζικής επεξεργασίας δεδομένων χρησιμοποιώντας μια πλειάδα υπολογιστών που συγκροτούν τη συστάδα υπολογιστών του Hadoop.

Το Hadoop MapReduce από την άλλη, είναι μια υλοποίηση του προγραμματιστικού μοντέλου MapReduce το οποίο παρέχει τη δυνατότητα της χρησιμοποίησης του MapReduce, για την υλοποίηση εφαρμογών. Οι εφαρμογές αυτές χρησιμοποιώντας το ανωτέρω μοντέλο έχουν την δυνατότητα να επεξεργάζονται μεγάλους όγκους δεδομένων παράλληλα σε μία συστάδα υπολογιστών με αξιόπιστο και κυρίως ιδιαίτερα ευέλικτο και ανθεκτικό σε λάθη τρόπο. Ολόκληρη η λογική και η μεθοδολογία που χρησιμοποιήθηκε προκειμένου να υλοποιηθεί το Hadoop MapReduce βασίστηκε στην επιστημονική εργασία της Google με τίτλο «MapReduce: Simplified Data Processing on Large Clusters» [7][10]

2.3.1 Χαρακτηριστικά του Hadoop MapReduce

Ο αλγόριθμός του MapReduce διαχωρίζεται σε δύο βασικές φάσεις, την φάση Map και την φάση Reduce. Οι δύο αυτές φάσεις εκτελούνται με αλληλουχία, η μία μετά την άλλη, με την φάση του Map να προηγείται αυτής του Reduce. Το αποτέλεσμα της πρώτης φάσης παράγει κάποια δεδομένα, τα οποία εισάγονται στην δεύτερη φάση ώστε

προχωρήσει η διαδικασία του αλγορίθμου. Τα δεδομένα που παράγονται έχουν κυρίως την μορφή ζευγών κλειδιού – τιμής (key- value).

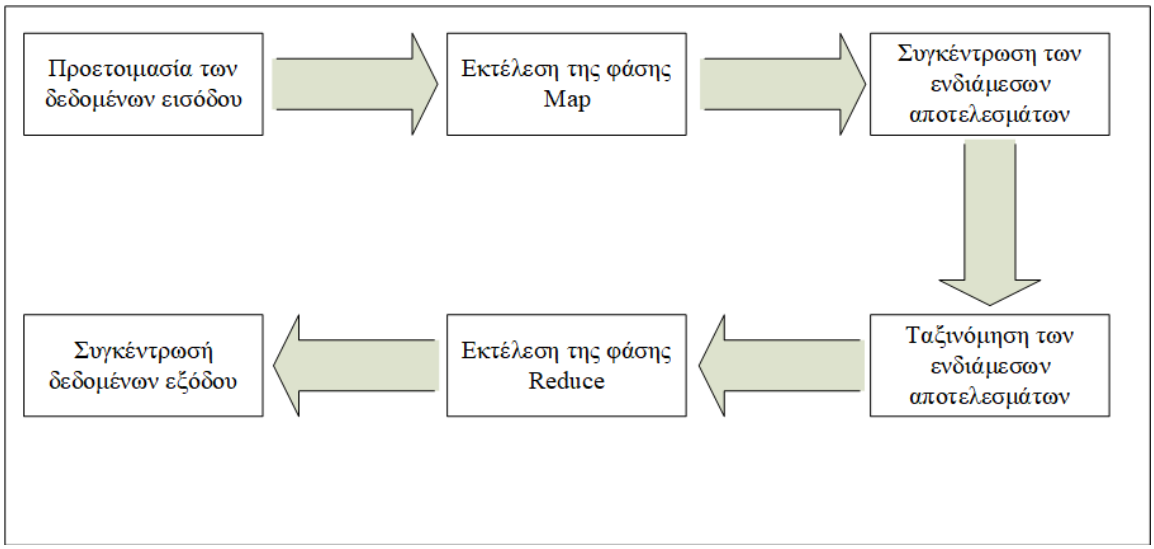
Ακολούθως αναλύονται με μια σύντομη περιγραφή οι δύο φάσεις του MapReduce:

- **Φάση Map:** Μόλις τα δεδομένα διαχωριστούν σε μικρότερα τμήματα, αυτά ανατίθενται στον task traker, όπως αναλύεται κατωτέρω [κεφάλαιο 2.3.3] για να εκτελεστεί η φάση του Map. Το τι ακριβώς θα επεξεργάζεται η φάση του Map είναι προκαθορισμένο και ορίζεται από τον εκάστοτε προγραμματιστή που έχει γράψει τον κώδικα της κάθε εφαρμογής που χρησιμοποιεί το μοντέλο MapReduce και εφαρμόζεται στα δεδομένα εισόδου της εφαρμογής. Τα δεδομένα εισόδου, απ' την άλλη, έχουν την μορφή ζευγών key-value, ενώ τα δεδομένα που παράγονται έχουν την μορφή διαφορετικών ζευγών (key-list(value)). Βασικός σκοπός της φάσης αυτής είναι η συλλογή όλων των τιμών με ίδιο key.
- **Φάση Reduce:** Όπως και στην προηγούμενη φάση έτσι και σ' αυτήν το τι ακριβώς λαμβάνει χώρα προκαθορίζεται από τον προγραμματιστή κατά τη γραφή του κώδικα της εφαρμογής. Βασικός σκοπός της φάσης αυτής είναι να συγκεντρώσει όλα τα αποτελέσματα της προηγούμενης φάσης και να εξάγει μια απάντηση για το πρόβλημα που καλέστηκε να λύσει το προγραμματιστικό μοντέλο του MapReduce.[7][10]

2.3.2 Βήματα MapReduce

Περαιτέρω, σε ακολουθία όλων των ανωτέρω, οι φάσεις που προαναφέρθηκαν θα μπορούσαν να προσεγγιστούν πιο αναλυτικά μέσω της παρουσίασης έξι διακριτών βημάτων. Αρχικά, ως πρώτο βήμα θα μπορούσε να νοηθεί η προετοιμασία των δεδομένων εισόδου της φάσης Map. Σε αυτό το πρώτο βήμα γίνεται η ανάγνωση των αρχικών δεδομένων που θα χρησιμοποιηθούν από την εφαρμογή. Στο σημείο αυτό μπορεί να γίνουν διάφορες επεξεργασίες των εισαγόμενων δεδομένων, όπως για παράδειγμα ένα φιλτράρισμα των πληροφοριών αυτών, ανάλογα πάντα με τον σκοπό που καλείται να εξυπηρετήσει η εφαρμογή. Εν συνεχεία, εφόσον ολοκληρωθεί η τυχόν επεξεργασία των δεδομένων δημιουργείται ένα ζεύγος τιμών ανά εγγραφή δεδομένων εισόδου.

Έπειτα ακολουθεί η εκτέλεση της φάσης Map καθώς και η παραγωγή των ενδιάμεσων αποτελεσμάτων, τα οποία στο επόμενο βήμα συγκεντρώνονται προκειμένου να δημιουργηθούν τα δεδομένα εισόδου της φάσης Reduce. Στη συνέχεια τα δεδομένα αυτά ταξινομούνται και η διαδικασία είναι έτοιμη να προχωρήσει στο επόμενο βήμα, ήτοι την εκτέλεση της φάσης Reduce. Με το πέρας της φάσης αυτής εξάγονται και πάλι νέα ενδιάμεσα αποτελέσματα, τα οποία στο τελικό βήμα συγκεντρώνονται και αποθηκεύονται σε ένα αρχείο κειμένου που βρίσκεται στο σύστημα αρχείων HDFS.[22]



Εικόνα 2-2: Βήματα MapReduce

2.3.3 Αρχιτεκτονική του MapReduce

Αντίστοιχα με την αρχιτεκτονική του συστήματος αρχείων HDFS, έτσι και η αρχιτεκτονική του MapReduce ακολουθεί τον τύπο συντονιστής / εργαζόμενος, όπως αυτός αναλύθηκε ανωτέρω. Στην κλασική εκδοχή του MapReduce ο διακομιστής που έχει τον ρόλο του συντονιστή ονομάζεται JobTracker, ενώ ο διακομιστής που έχει τον ρόλο του εργαζομένου ονομάζεται TaskTracker. Ο JobTracker διαχειρίζεται όλες τις εργασίες και τους πόρους της συστάδας υπολογιστών που χρησιμοποιείται. Ο JobTracker έχει ως στόχο να προγραμματίσει τις εκτελέσεις των φάσεων Map που πρόκειται να εκτελεστούν από τους TaskTrackers, υπολογίζοντας την μικρότερη δυνατή σπατάλη υπολογιστικών πόρων.

Υπάρχουν πολλές διαδικασίες που λαμβάνουν χώρα κατά την εκτέλεση του αλγορίθμου MapReduce. Αναλυτικότερα, αυτές είναι η υποβολή εργασίας, η

αρχικοποίηση εργασίας, η ανάθεση υποεργασιών, η εκτέλεση υποεργασιών, η ενημέρωση για την πρόοδο της εργασίας και οι διαδικασίες που σχετίζονται με το πέρας της υποβαλλόμενης εργασίας. Η διαχείριση των ανωτέρω εργασιών γίνεται κυρίως από τον JobTracker και η εκτέλεσή τους από τον εκάστοτε TaskTracker.

Όταν μια εργασία υποβάλλεται στο Hadoop MapReduce, αρχικά την αναλαμβάνει ο JobTracker. Στην συνέχεια, τα δεδομένα εισόδου τμηματοποιούνται και διαμοιράζονται στη συστάδα των υπολογιστών. Έπειτα, ο JobTracker υπολογίζει τον αριθμό των φάσεων Map και Reduce που είναι απαραίτητο να εκτελεστούν ώστε να διεκπεραιωθεί η εργασία που τους ανατέθηκε, όπως αυτές προκύπτουν από τα δεδομένα εισόδου. Ακολούθως, ένας ή περισσότεροι TaskTracker αναλαμβάνουν να εκτελέσουν τις εργασίες που τους ανατέθηκαν. Κατά την διάρκεια της εκτέλεσης των εργασιών αυτών οι TaskTracker στέλνουν ενημερώσεις στον JobTracker για την πρόοδο της εργασιών τους, αλλά και για τους πόρους που χρησιμοποιούν.[7][10][22]

2.4 Γεωδαιτικό σύστημα WGS 84

Ο όρος Γεωδαισία χρησιμοποιείται για να περιγράφει ένα γνωστικό αντικείμενο της επιστήμης της Γεωγραφίας, που ασχολείται με τον ακριβή προσδιορισμό του σχήματος της επιφάνειας της Γης. Πιο αναλυτικά, βασικό στόχο του εν λόγω ερευνητικού αντικειμένου αποτελεί ο υπολογισμός των υψομέτρων του γεωειδούς τόσο σε τοπικό, όσο και σε περιφερειακό και εν τέλει σε παγκόσμιο επίπεδο. Η θεωρητική, αλλά και η μαθηματική υποδομή της επιστήμης αυτής είναι τεράστια και κατά κύριο λόγο βασίζεται στην ανάλυση δεδομένων, όπως αυτά προκύπτουν από τα παρατηρούμενα μεγέθη. Προκειμένου να προσδιοριστεί κατά το δυνατόν πλησιέστερα η τιμή ενός γεωγραφικού μεγέθους, χρησιμοποιούνται ποικίλες μέθοδοι εκτίμησης του, οι οποίες σχετίζονται με το γήινο πεδίο βαρύτητας του, λαμβανομένων υπόψη πάντα και των μεταβολών που αυτό υφίσταται μέσα στο χρόνο. Πιο συγκεκριμένα μετρούνται γεωμετρικά κυρίως μεγέθη όπως για παράδειγμα μήκη, υψόμετρα, γωνίες και άλλα μεγέθη της γης.[23]

Από τα τέλη της δεκαετίας του 1950 και με την εκτόξευση των πρώτων δορυφόρων εξελίχθηκαν και τα συστήματα Γεωδαισίας Δορυφόρων, τα οποία βασίζονται στο φαινόμενο Doppler. Αδιαμφισβήτητα, ένα από τα πλέον διαδεδομένα συστήματα είναι αυτό του εντοπισμού θέσεων στο χώρο «Global Positioning System»

(GPS), το οποίο θα μπορούσε να λεχθεί ότι τέθηκε σε πλήρη λειτουργία περίπου στα τέλη της δεκαετίας του 1980.[24]

Το World Geodetic System (WGS) είναι ένα πρότυπο που χρησιμοποιείται στην χαρτογράφηση, στην επιστήμη της γεωδαισίας, αλλά και στα συστήματα πλοήγησης και εντοπισμού θέσεως στον χώρο. Το συγκεκριμένο πρότυπο περιέχει τον ορισμό των θεμελιωδών και σταθερών συντεταγμένων του συστήματος, το ελλειψοειδές (κανονικό) γήινο βαρυτικό μοντέλο (EGM), μια περιγραφή του σχετικού παγκόσμιου μαγνητικού μοντέλου (WMM) και μια τρέχουσα λίστα τοπικών μετασχηματισμών αναφοράς.[24][25]

Μετά από αλλεπάλληλες βελτιώσεις και τροποποιήσεις του εν λόγω προτύπου, η πιο πρόσφατη έκδοσή του είναι το WGS84 το οποίο συναντάται και ως WGS 1984, EPSG: 4326, το οποίο παγιώθηκε το 1984 και αναθεωρήθηκε και πάλι το 2004. Προγενέστερες μορφές του συγκεκριμένου προτύπου είναι τα WGS72, WGS66, και WGS60. Ωστόσο, αυτό που κάνει το WGS84 ιδιαιτέρως ξεχωριστό με αποτέλεσμα να το καθιστά ευρέως χρησιμοποιούμενο, είναι το ότι το GPS χρησιμοποίει το σύστημα συντεταγμένων του συγκεκριμένου προτύπου.[25]

2.5 Παγκόσμιο Σύστημα Στιγματοθέτησης Global Positioning System (GPS)

Το GPS (Global Positioning System), Παγκόσμιο Σύστημα εντοπισμού θέσης που είναι ένα από τα πιο γνωστά και ευρέως χρησιμοποιούμενα συστήματα εντοπισμού γεωγραφικής θέσης, τόσο κινητών, όσο και ακίνητων χρηστών, ξεκίνησε από το Υπουργείο Άμυνας των ΗΠΑ. Αρχικά, γύρω στη δεκαετία του 1980 έφερε διαφορετική ονομασία και συγκεκριμένα ονομαζόταν NAVSTAR (Navigation Signal Timing and Ranging Global Positioning System), όμως στη συνέχεια πήρε τη σημερινή του ονομασία. Το σύστημα αυτό βασίζεται κατά κύριο λόγο σε πολλούς δορυφόρους της γης, οι οποίοι είναι εφοδιασμένοι , με ειδικούς πομπούς και δέκτες εντοπισμού. Οι ειδικές αυτές συσκευές είναι ικανές να παρέχουν ακριβείς πληροφορίες σχετικά με τη θέση, το υψόμετρο, την ταχύτητα, αλλά και την κατεύθυνση προς την οποία κινείται το αντικείμενο ενδιαφέροντος. Στην ουσία, οι δέκτες GPS είναι ικανοί να αναπαράγουν με ακρίβεια το στίγμα οποιουδήποτε σημείου στον κόσμο χρησιμοποιώντας το δίκτυο των δορυφόρων που κινούνται σε σταθερή τροχιά γύρω από τον πλανήτη Γη. [27]

Από την πρώτη κιόλας εκτόξευση δορυφόρου, οι ερευνητές ήταν σε θέση να διαπιστώσουν με βεβαιότητα ότι όσο ο δορυφόρος πλησίαζε στη Γη και συγκεκριμένα στο σημείο παρατήρησης, τόσο το σήμα που λαμβανόταν απ' αυτόν αυξανόταν. Αντίστοιχα, όσο ο δορυφόρος απομακρυνόταν από τη Γη και συγκεκριμένα από το σημείο παρατήρησης, τόσο το σήμα μειωνόταν. Έτσι, οι ερευνητές παρατηρώντας το φαινόμενο αυτό οδηγήθηκαν στο συμπέρασμα ότι όπως ακριβώς η θέση ενός δορυφόρου ήταν δυνατό να εντοπιστεί με ακρίβεια βάσει της έντασης του σήματος που λαμβάνεται από αυτόν, αντίστοιχα θα υπήρχε η δυνατότητα να συμβεί και το ακριβώς αντίστροφο, ήτοι να εντοπιστεί από τον δορυφόρο η θέση ενός σημείου με την ίδια ακρίβεια. Βέβαια, είναι προφανές ότι ένας μόνο δορυφόρος δεν αρκεί για να παράξει το αποτέλεσμα αυτό, αλλά χρειάζεται η συνεισφορά τουλάχιστον τριών δορυφόρων.[24]

Το σύστημα εντοπισμού θέσης GPS αποτελεί ένα παγκόσμιο δίκτυο γεγονός που συνεπάγεται ότι είναι απαραίτητος ο διαχωρισμός του σε επιμέρους τμήματα στα οποία πραγματοποιούνται όλες οι λειτουργίες του και ο συντονισμός του, προκειμένου να εξυπηρετηθεί η τεράστια έκταση του με αποτελεσματικότητα. [26]

Αναλυτικά, τα τμήματα αυτά είναι:

Διαστημικό τμήμα: Αποτελείται από το δίκτυο των δορυφόρων που ήδη αναφέρεται παραπάνω. Οι δορυφόροι αυτοί, δημιουργώντας ένα πυκνό πλέγμα, καλύπτουν ομοιόμορφα με το σήμα τους ολόκληρή την επιφάνεια του πλανήτη. Όλοι οι δορυφόροι βρίσκονται σε ύψος περίπου 20 χιλιάδων χιλιομέτρων πάνω από την επιφάνεια της θάλασσας και εκτελούν δύο περιστροφές γύρω από τη Γη κάθε 24ωρο.

Επίγειο τμήμα ελέγχου: Είναι δεδομένο ότι οι δορυφόροι μπορούν ανά πάσα στιγμή να αντιμετωπίσουν οποιοδήποτε πρόβλημα ως προς τη σωστή λειτουργία τους. Οι έλεγχοι που γίνονται σε αυτούς αφορούν κυρίως την σωστή τους ταχύτητα, το σωστό τους υψόμετρο και τη σωστή παροχή σε ηλεκτρική ενέργεια. Επιπρόσθετα, εφαρμόζονται όλες οι απαραίτητες διορθωτικές ενέργειες που αφορούν στο σύστημα χρονομέτρησης των δορυφόρων, ώστε να αποτρέπεται η παροχή λανθασμένων πληροφοριών στους χρήστες του συστήματος.

Το τμήμα τελικού χρήστη: Απαρτίζεται από τους εκατομμύρια χρήστες δεκτών GPS ανά την υφήλιο. Για να προσφέρουν όσο το δυνατόν περισσότερες πληροφορίες, οι δέκτες συνδυάζονται με ειδικό λογισμικό, που συνήθως προβάλλει ένα χάρτη στην οθόνη της συσκευής. Πρόκειται, δηλαδή, για λογισμικό που λαμβάνει από

τους δορυφόρους τις πληροφορίες για το στίγμα του σημείου στο οποίο βρίσκεται ο δέκτης και τις μετατρέπει σε μια κατανοητή μορφή.

2.6 Δεδομένα μεγάλου όγκου (big data)

Όλο και συχνότερα τα τελευταία χρόνια γίνεται λόγος τόσο στους επιχειρηματικούς, όσο και στους επιστημονικούς κλάδους για τα δεδομένα μεγάλου όγκου (big data). Σύμφωνα με τον ορισμό της Gartner τα big data είναι δεδομένα μεγάλου όγκου (volume) , τα οποία παράγονται με μεγάλη ταχύτητα (velocity) και/ή μεγάλης ποικιλίας (variety) τα οποία απαιτούν οικονομικά αποδοτικές και καινοτόμες μορφές επεξεργασίας πληροφοριών που επιτρέπουν την βελτίωση της γνώσης, της λήψης αποφάσεων και της αυτοματοποίησης διαδικασιών [28]. Η παραπάνω θεωρία είναι γνωστή σαν την θεωρία των τριών V.

2.6.1 Τα τρία V των Big Data

Παρακάτω δίνεται μια λεπτομερής περιγραφή για τους τρείς βασικούς άξονες χαρακτηριστικών της θεωρίας. [29]

- **Όγκος (Volume):** Ο όγκος των δεδομένων αποτελεί αναντίρρητα ένα από τα βασικότερα και πιο εύκολα αντιληπτά χαρακτηριστικά τους. Στον όγκο των δεδομένων είναι δυνατόν να υποκρύπτεται τεράστια γνώση, η οποία συνεπάγεται δυνητικά μεγάλη αξία για όποιον μπορεί να επεξεργαστεί τα δεδομένα και να την εξάγει. Με τη ραγδαία ανάπτυξη των κοινωνικών δικτύων και των έξυπνων κινητών τηλεφώνων, σε συνδυασμό με την σταθερή ανάπτυξη του διαδικτύου, είναι προφανές ότι τα τελευταία χρόνια ο όγκος των δεδομένων που πιθανότατα να διαχειρίζεται ένας οργανισμός αυξήθηκε εκθετικά και μπορεί να κυμαίνεται από μερικά terabyte έως και εκατοντάδες petabyte .
- **Ταχύτητα (Velocity):** Ο πρώτος βασικός άξονας που εξετάζεται σχετικά με την ταχύτητα, είναι η ταχύτητα παραγωγής των δεδομένων. Εν συνεχεία, ένας δεύτερος άξονας που παρουσιάζει έντονο ενδιαφέρον είναι η ταχύτητα με την οποία επεξεργάζονται τα εισερχόμενα δεδομένα οι εκάστοτε εφαρμογές. Αξίζει να σημειωθεί ότι πολλές εφαρμογές επεξεργάζονται τα δεδομένα σε πραγματικό χρόνο ή σε σχεδόν πραγματικό χρόνο.

- **Ποικιλία (Variety):** Μέχρι πρότινος η αναφορά στα δεδομένα αφορούσε δεδομένα συγκεκριμένων τύπων και αποθηκεύονταν σε σχεσιακές βάσεις δεδομένων. Ωστόσο, πλέον, εκτός της προαναφερόμενης δομημένης μορφής, τα δεδομένα υπάρχουν και σε ημιδομημένες ή ακόμα και σε αδόμητες μορφές. Τα δεδομένα αυτά μπορεί να είναι αρχεία κειμένου, αρχεία ήχου ή βίντεο τα οποία χρειάζονται ειδικό χειρισμό.

Με την πάροδο των χρόνων στην ανωτέρω αναφερόμενη αρχική θεωρία των τριών V τείνουν να προσθέτουν ακόμη δύο V που δεν είναι άλλα από την αξία (Value) και την εγκυρότητα (Veracity) [9]. Η αξία του παραγόμενου αποτελέσματος από τα δεδομένα μεγάλου όγκου είναι ιδιαίτερα σημαντική, καθώς η αποθήκευση και η συντήρηση μεγάλου όγκου δεδομένων έχει ιδιαίτερα υψηλό κόστος. Συνεπώς, θα ήταν δυσβάσταχτη μια επένδυση σε υποδομή για την διαχείριση μεγάλου όγκου δεδομένων χωρίς το αντίστοιχό αντίκρισμα. Ωστόσο, ένα σημαντικό θέμα που ανακύπτει είναι η εγκυρότητα των δεδομένων, καθώς καθίσταται σχεδόν αδύνατο τα δεδομένα με τα ανωτέρω περιγραφόμενα χαρακτηριστικά να είναι απολύτως έγκυρα. Περαιτέρω, η ποιότητα των δεδομένων μπορεί να ποικίλει σημαντικά και ως εκ τούτου να ποικίλει και η αξία του παραγόμενου αποτελέσματος. [22]

3 Μεθοδολογία

Σε αυτό το κεφάλαιο της διπλωματικής εργασία παρουσιάζονται τα βήματα που ακολουθηθήκαν για την ανάλυση των κυκλοφοριακών δεδομένων με τελικό σκοπό τον εντοπισμό συμβάντων κυκλοφοριακής συμφόρησης και την δημιουργία κυκλοφοριακών προφίλ. Αναλυτικότερα, παρουσιάζονται τα δεδομένα που χρησιμοποιήθηκαν αλλά και η μορφολογία τους, η διαδικασία συλλογής και η προετοιμασία των δεδομένων. Περαιτέρω, αναλύεται η εισαγωγή και η διαδικασία αντιστοίχισης των δεδομένων των συντεταγμένων πάνω στις διαδρομές, οι φάσεις των map και reduce που εφαρμόστηκαν, καθώς και η δημιουργία προφίλ παρόμοιων χρονικών περιόδων για τα δεδομένα πραγματικού χρόνου. Επιπλέον, παρουσιάζεται η διαδικασία αποθήκευσης των αποτελεσμάτων και τέλος, η διαδικτυακή πύλη στην οποία βρίσκονται διαθέσιμα τα αποτελέσματα αυτά.

3.1 Δεδομένα

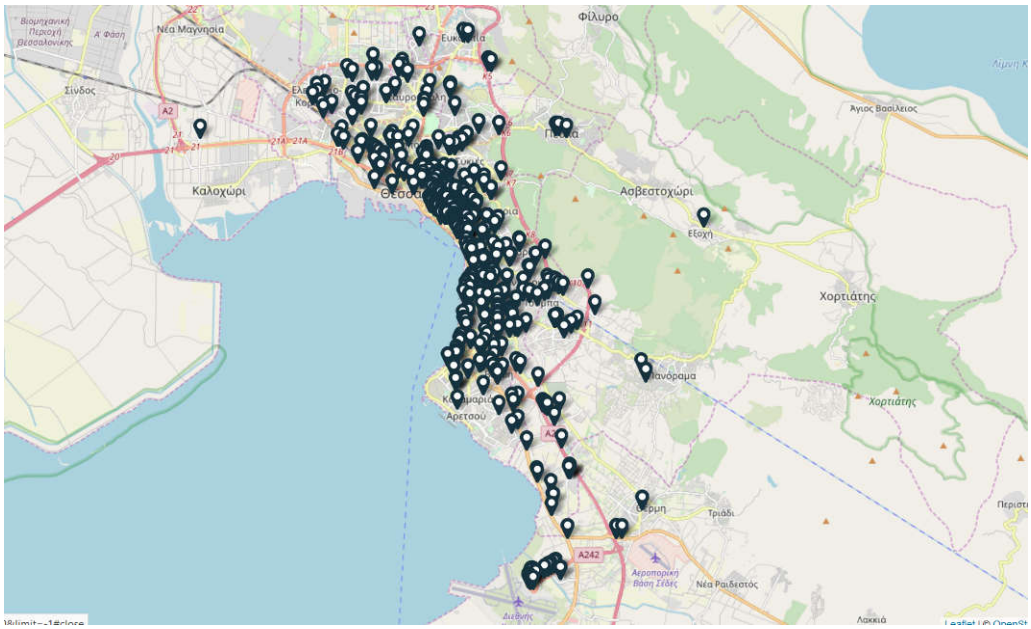
Όπως προαναφέρθηκε και στην εισαγωγή της παρούσας διπλωματικής εργασίας, τα δεδομένα που χρησιμοποιούνται στην εν λόγω διπλωματική εργασία είναι ανοιχτά δεδομένα και παρέχονται από το Ινστιτούτο Βιώσιμης Κινητικότητας & Δικτύων Μεταφορών [1]. Από το σύνολό των σετ δεδομένων που παρέχονται, χρησιμοποιούνται τα ακόλουθα δυο:

- Δεδομένα κινούμενων οχημάτων (FCD) τα οποία παρέχονται τόσο ως δεδομένα ιστορικού, όσο και ως δεδομένα πραγματικού χρόνου.
- Δεδομένα που αναπαράγουν διαδρομές αποτελούμενες από τμήματα του οδικού δικτύου της πόλης της Θεσσαλονίκης.

Κάνοντας μια μικρή ανασκόπηση στα δεδομένα που θα χρησιμοποιηθούν συμπεραίνεται ότι πληρούν τις προϋποθέσεις για να θεωρηθούν δεδομένα μεγάλου όγκου. Ο συνολικός όγκος των δεδομένων είναι κοντά στα 140 GB ως αναφορά τα δεδομένα ιστορικών οχημάτων, επιπλέον τα δεδομένα πραγματικού χρόνου παράγονται με μεγάλη ταχύτητα, για κάθε ορολογική ώρα συλλέγονται περίπου 7MB δεδομένων. Τέλος τα δεδομένα που χρησιμοποιούνται έχουν μια ποικιλομορφία ως αναφορά την δομή τους.

3.1.1 Δεδομένα κινούμενων οχημάτων (FCD)

Το σετ των δεδομένων των κινούμενων οχημάτων δημιουργείται συλλέγοντας τις πληροφορίες που το απαρτίζουν, είτε από αυτοκίνητα που έχουν δέκτες GPS, είτε από άλλες συσκευές με δέκτες GPS, όπως για παράδειγμα κινητά τηλέφωνα. Οι πληροφορίες που συλλέγονται αποτελούνται συνήθως από δεδομένα γεωγραφικής φύσης όπως είναι οι συντεταγμένες, η ταχύτητα, ο προσανατολισμός και η χρονική στιγμή που συλλέχθηκαν οι πληροφορίες.



Εικόνα 3-3: Απεικόνιση δεδομένων κινούμενων οχημάτων

Επιπρόσθετα, τόσο τα δεδομένα ιστορικού όσο και τα δεδομένα πραγματικού χρόνου αποτελούνται από τις ίδιες ακριβώς πληροφορίες και αυτές παρουσιάζονται στον πίνακα 3-1. Ωστόσο, η δομή των δεδομένων ιστορικού και των δεδομένων πραγματικού χρόνου διαφέρει. Τα δεδομένα ιστορικού βρίσκονται σε αρχεία κειμένου και η πληροφορία που περιέχεται σε αυτά είναι οριοθετημένη με τον χαρακτήρα του στηλοθέτη (tab). Τα αρχεία αυτά βρίσκονται σε συμπιεσμένη μορφή λόγω του μεγέθους τους. Ενδεικτικά αναφέρεται ότι τα δεδομένα ενός μήνα σε ασυμπίεστη μορφή έχουν μέγεθός περίπου 5 GB, ενώ συμπιεσμένα έχουν μέγεθος περίπου 600 MB

Πίνακας 3-1: Μορφολογία δεδομένων κινούμενων οχημάτων

Όνομα πεδίου	Περιγραφή
recorded_timestamp	Η χρονική στιγμή που έγινε η καταγραφή της πληροφορίας. Η ακρίβεια της χρονικής στιγμής είναι σε χιλιοστά του δευτερολέπτου.
lon	Γεωγραφικό μήκος
lat	Γεωγραφικό πλάτος
altitude	Υψόμετρο από το επίπεδο της θάλασσας

speed	Ταχύτητα σε χιλιόμετρα ανά ώρα
orientation	Προσανατολισμός σε σχέση με τον βορρά

Τα δεδομένα πραγματικού χρόνου από την άλλη, διατίθενται προς χρήση με 5 διαφορετικές μορφές. Αυτές είναι η μορφή JSON, η μορφή XML, η μορφή CSV, η μορφή KML και τέλος η μορφή MAP. Για τις ανάγκες της παρούσας διπλωματικής εργασίας τα δεδομένα πραγματικού χρόνου χρησιμοποιούνται με την μορφή JSON.

```
1  2018-02-01 00:00:00.197 22.8963 40.6722416666667    0.69999999999999996 28.0    266.07998657226602
2  2018-02-01 00:00:00.323 22.952911666666701  40.631435000000003  3.2000000000000002  0.0 280.79000854492199
3  2018-02-01 00:00:00.467 22.955175000000001  40.636971666666703  4.2999999999999998  18.0    140.82000732421901
4  2018-02-01 00:00:00.517 22.9666216666667    40.631986666666698  8.4000000000000004  27.0    120.300003051758
5  2018-02-01 00:00:00.590 22.9587133333333    40.617878333333302  4.5 42.0    343.57000732421898
6  2018-02-01 00:00:00.607 22.929040000000001  40.662129999999998  3.5 38.0    235.60000610351599
7  2018-02-01 00:00:00.623 22.94341    40.636215   2.0 0.0 246.63000488281301
8  2018-02-01 00:00:00.637 22.928128333333301  40.643606666666699  0.40000000000000002 37.0    27.4799995422363
9  2018-02-01 00:00:00.667 22.959050000000001  40.641539999999999  15.699999999999999  0.0 237.61999511718801
10 2018-02-01 00:00:00.677 22.959050000000001  40.641539999999999  15.699999999999999  0.0 237.61999511718801
11 2018-02-01 00:00:00.763 22.950521666666699  40.604356666666703  0.0 42.0    189.80000305175801
12 2018-02-01 00:00:00.863 22.940288333333299  40.633143333333301  0.90000000000000002 0.0 202.41000366210901
13 2018-02-01 00:00:00.880 22.951136666666699  40.632211666666699  1.8 0.0 0.0
14 2018-02-01 00:00:00.897 22.962895   40.6287916666667    5.0 53.0    333.29998779296898
```

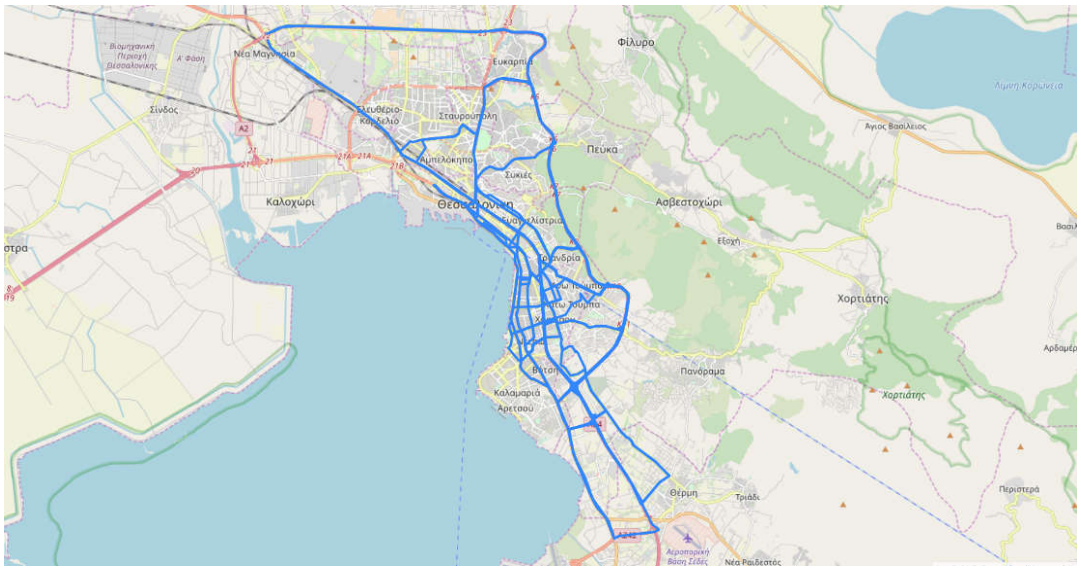
Εικόνα 3-4:Δείγμα δεδομένων ιστορικού


```
[
  {
    "recorded_timestamp": "2019-05-04 15:22:02.647",
    "lon": "22.9675216666667",
    "lat": "40.579665",
    "altitude": "9.4",
    "speed": "55",
    "orientation": "146.589996337891"
  },
  {
    "recorded_timestamp": "2019-05-04 15:22:02.790",
    "lon": "22.9350433333333",
    "lat": "40.63537",
    "altitude": "-0.1",
    "speed": "33",
    "orientation": "150.869995117188"
  },
```


Εικόνα 3-5:Δείγμα δεδομένων πραγματικού χρόνου

3.1.2 Δεδομένα διαδρομών

Τα συγκεκριμένα δεδομένα αποτελούνται από προκαθορισμένες διαδρομές στο οδικό δίκτυο της Θεσσαλονίκης. Οι διαδρομές αυτές καλύπτουν την πλειονότητα των οδικών αξόνων της πόλης, όπως απεικονίζεται και στην εικόνα 3-4. Επιπλέον, έχουν δημιουργηθεί για την παραγωγή χρόνων διαδρομών με την χρήση ανιχνευτών Bluetooth από ένα σημείο σε ένα άλλο [30].



Εικόνα 3-6:Απεικόνιση διαδρομών

Εξετάζοντας την μορφή τον δεδομένων των διαδρομών διαπιστώνεται ότι καλύπτονται οι ανάγκες της παρούσας διπλωματικής εργασίας. Στα δεδομένα που παρέχονται περιλαμβάνεται για κάθε διαδρομή ένας πίνακας συντεταγμένων. Χρησιμοποιώντας τον πίνακα αυτό είναι δυνατόν να αναπαρασταθεί γεωγραφικά η διαδρομή, ώστε να καταστεί δυνατή η αντιστοίχιση των δεδομένων των κινούμενων οχημάτων πάνω στις διαδρομές. Θα πρέπει να σημειωθεί ότι η ακρίβεια των σχετικών προβλέψεων θα μπορούσε να αυξηθεί ακόμα περισσότερο εάν στα υπάρχοντα δεδομένα υπήρχε και ο προσανατολισμός της διαδρομής σε σχέση με το βορρά.

Πίνακας 3-2:Μορφολογία δεδομένων διαδρομών

Όνομα πεδίου	Περιγραφή
Path_Id	Ένα μοναδικό αναγνωριστικό για την διαδρομή.
Path_Name	Το όνομα της διαδρομής.
polyline	Ο πίνακας των συντεταγμένων της διαδρομής.

3.2 Συλλογή και προετοιμασία δεδομένων

Το πρώτο βήμα για την ανάλυση των δεδομένων που αναφέρονται ανωτέρω είναι η ανάκτηση τους και η προετοιμασία τους όπου αυτό είναι αναγκαίο, καθώς και η εισαγωγή τους στο σύστημα αρχείων του Hadoop, ήτοι στο HDFS. Παρ' όλο που ο χειρισμός των δεδομένων πραγματικού χρόνου σε σχέση με τα δεδομένα ιστορικού διαφέρει, τελικά τα δεδομένα και των δύο καταλήγουν στο HDFS. Επιπλέον, όσον αφορά τα δεδομένα των διαδρομών, αυτά ανακτώνται και αποθηκεύονται σε μια σχεσιακή βάση δεδομένων.

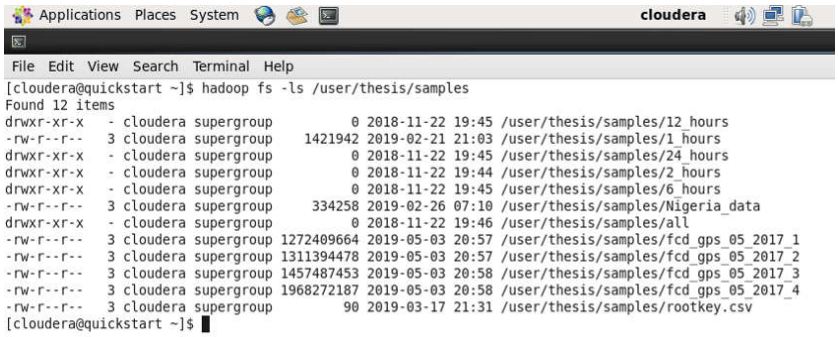
3.2.1 Συλλογή και προετοιμασία δεδομένων ιστορικού

Για την συλλογή και την εισαγωγή των δεδομένων ιστορικού στο HDFS ακολουθούνται τα παρακάτω βήματα. Αρχικά, γίνεται λήψη του συμπιεσμένου αρχείου με τα δεδομένα ενός συγκεκριμένου μήνα. Στην συνέχεια, αποσυμπιέζεται το αρχείο στην αρχική του μορφή. Το αρχείο την δεδομένη χρονική στιγμή βρίσκεται στο σύστημα αρχείων του λειτουργικού συστήματος. Στο επόμενο βήμα αντιγράφεται το αρχείο από το σύστημα αρχείων του λειτουργικού συστήματος στο HDFS με την παρακάτω εντολή στο τερματικό του λειτουργικού συστήματος.

hadoop fs – copyFromLocal «διαδρομή του αρχείου στο λειτουργικό σύστημα» «διαδρομή του αρχείου στο HDFS»

Η παραπάνω διαδικασία επιβεβαιώνεται για την επιτυχία της, εκτελώντας την ακόλουθη εντολή στο τερματικό του λειτουργικού συστήματος και επιστρέφοντας ένα αποτέλεσμα αντίστοιχο με αυτό της εικόνας 3-5.

hadoop fs -ls «διαδρομή του αρχείου στο HDFS»



Εικόνα 3-7:Αποτελέσματα περιεχομένων φακέλου HDFS

3.2.2 Συλλογή και προετοιμασία δεδομένων πραγματικού χρόνου

Η συλλογή των δεδομένων πραγματικού χρόνου διαφέρει από την παραπάνω περιγραφόμενη διαδικασία που αφορά στα δεδομένα ιστορικού κυρίως λόγω της δομής της διαδικτυακής υπηρεσίας που διαθέτει τα δεδομένα. Τα δεδομένα πραγματικού χρόνου ανανεώνονται ανά ένα λεπτό. Συνεπώς, δημιουργήθηκε μια διαδικασία η οποία εκτελείται ανά ένα λεπτό και καλεί προγραμματιστικά τη διαδικτυακή υπηρεσία για να αποκτήσει τα δεδομένα της προηγούμενης χρονικής περιόδου. Στην συνέχεια, η διαδικασία εξάγει τις τιμές από την απάντηση της διαδικτυακής υπηρεσίας και τις τοποθετεί σε ένα προσωρινό αρχείο.

Η εγγραφή των τιμών στο αρχείο γίνεται με τον ίδιο ακριβώς τρόπο όπως και στα αρχεία των δεδομένων ιστορικού, ώστε να υπάρχει μία συνάφεια στα δεδομένα που χρησιμοποιεί η εφαρμογή που αναπτύχθηκε στο πλαίσιο της διπλωματικής εργασίας. Μόλις συμπληρωθεί μία ωρολογιακή ώρα το προσωρινό αρχείο αντιγράφεται αυτόματα από το σύστημα αρχείων του λειτουργικού συστήματος στο HDFS. Κάθε μία ωρολογιακή ώρα δημιουργείται ένα νέο προσωρινό αρχείο και η διαδικασία επαναλαμβάνεται με τον τρόπο που περιγράφεται ανωτέρω. Η πλήρης υλοποίηση της διαδικασίας βρίσκεται στο παράρτημα Α.1

3.2.3 Συλλογή και προετοιμασία δεδομένων των διαδρομών

Στην συλλογή και προετοιμασία των δεδομένων των διαδρομών ακολουθήθηκε ένας διαφορετικός τρόπος, γιατί τα δεδομένα δεν αποθηκεύονται στο HDFS αλλά σε μια σχεσιακή βάση δεδομένων από όπου και ανακτώνται όποτε αυτό είναι αναγκαίο. Αρχικά, γίνεται λήψη των δεδομένων των διαδρομών και αυτά αποθηκεύονται με την μορφή αρχείου κειμένου στο σύστημα αρχείων του λειτουργικού συστήματος. Έπειτα, με την βοήθεια μιας άλλης διαδικασίας, διαβάζεται το περιεχόμενο του αρχείου και αποθηκεύονται οι τιμές των διαδρομών στην σχεσιακή βάση δεδομένων.

Επιπρόσθετα, υπολογίζεται και το μήκος της διαδρομής σε μέτρα. Ο υπολογισμός του μήκους της διαδρομής είναι καθοριστικός για την μετέπειτα εξέλιξη της εφαρμογής που παρουσιάζεται στη διπλωματική εργασία, γιατί αποτελεί βασικό παράγοντα των μελλοντικών υπολογισμών που λαμβάνουν χώρα. Ο υπολογισμός γίνεται με την βοήθεια της βιβλιοθήκης Spatial4j, [31] η οποία είναι μια βιβλιοθήκη ανοιχτού κώδικά σε γλώσσα προγραμματισμού Java για γεωγραφικούς υπολογισμούς. Στην εικόνα 3-6 απεικονίζεται το απόσπασμα της διαδικασίας κατά την οποία από τον πίνακα με τις

συντεταγμένες που αποτελεί την διαδρομή υπολογίζεται η απόσταση. Η πλήρης υλοποίηση της συγκεκριμένης διαδικασίας βρίσκεται στο παράρτημα Α.2

```
private static Double calculateDistance(BufferedLineString polyline) {
  double returnDistance=0;
  for(int i =0;i<polyline.getPoints().size()-1;i++){
    returnDistance = returnDistance +
        DistanceUtils.degrees2Dist(SpatialContext.GEO.getDistCalc()
                .distance(polyline.getPoints().get(i),
                        polyline.getPoints().get(i+1)),
                DistanceUtils.EARTH_MEAN_RADIUS_KM) * 1000;
  }
return returnDistance;
}
```

Εικόνα 3-8:Υπολογισμός μήκους διαδρομής

3.3 Εισαγωγή και η διαδικασία αντιστοίχισης των δεδομένων

Για την εισαγωγή των δεδομένων στην βασική εφαρμογή που υλοποιήθηκε στο πλαίσιο της εν λόγω διπλωματικής εργασίας, κρίθηκε αναγκαίο να δημιουργηθεί ένας ξεχωριστός τρόπος εισαγωγής αποκλειστικά για την εφαρμογή αυτή, καθώς οι υπάρχουσες δομές δεδομένων που παρέχει το Hadoop MapReduce δεν επαρκούσαν για να καλύψουν τα υπάρχοντα δεδομένα [32].

Αρχικά, δημιουργούνται 2 αντικείμενα το αντικείμενο GeoKey και το αντικείμενο GeoValue, οι οποίες κληρονομούν την κλάση WritableComparable. Τα δύο αυτά αντικείμενα θα περιέχουν τις τιμές των αρχικών δεδομένων σε μορφή ζευγών (key-value). Ο μετασχηματισμός αυτός είναι απαραίτητος γιατί η διαδικασία του Map απαιτεί τα δεδομένα εισόδου να πληρούν τα παραπάνω.

Έπειτα, δημιουργείται το αντικείμενο InputFormat το οποίο χρησιμοποιείται για να περάσει τα δεδομένα που αντιγράφηκαν στο HDFS, κατά την παραπάνω διαδικασία, μέσα στην εφαρμογή και να δημιουργήσουν τα δεδομένα εισόδου της διαδικασίας Map. Η δημιουργία των αντικειμένων GeoKey και GeoValue γίνεται σε ένα άλλο αντικείμενο το RecordReader, το οποίο δημιουργήθηκε για τον σκοπό αυτό. Οι δύο βασικές μέθοδοι που περιέχει το αντικείμενο RecordReader είναι η μέθοδος initialize και η μέθοδος nextKeyValue.

3.3.1 Αντικείμενο RecordReader

Αρχικά καλείται η μέθοδος initialize του RecordReader, για μία φορά. Στην κλήση αυτή αρχικοποιείται το αντικείμενο του reader με την διαδρομή του αρχείου που

περιέχει τα δεδομένα και έπειτα φορτώνονται οι διαδρομές σε μια λίστα για να χρησιμοποιηθούνε στο επόμενη μέθοδο.

```
reader.initialize(is, tac);
try {
    String data = readFileAsString("/home/cloudera/Downloads/paths");
    myPaths = MyPath.listfromJSON(data);
} catch (Exception ex) {
    LOG.debug(ex.toString());
}
```

Εικόνα 3-9:Η Μέθοδος initialize

Στην συνέχεια καλείται η μέθοδος nextKeyValue επαναληπτικά τόσες φορές όσες και οι έγγραφές του αρχείου εισόδου. Η μέθοδος τερματίζεται μόλις προσπελαστούν όλες οι εγγραφές του αρχείου. Η ακολουθία των ενεργειών που εκτελείται στην κάθε επανάληψη της μεθόδου είναι η ακόλουθη:

- Αρχικά ελέγχεται εάν υπάρχει επομένη εγγραφή στο αρχείο. Έπειτα ελέγχονται αν τα δυο αντίγραφα των αντικείμενων GeoKey και GeoValue είναι κενά. Στην περίπτωση που είναι κενά αρχικοποιούνται, ειδάλλως η διαδικασία προχωράει στο επόμενο βήμα.

```
boolean gotNextKeyValue = reader.nextKeyValue();
if (gotNextKeyValue) {
    if (key == null) {
        key = new GeoKey();
    }
    if (value == null) {
        value = new GeoValue();
    }
```

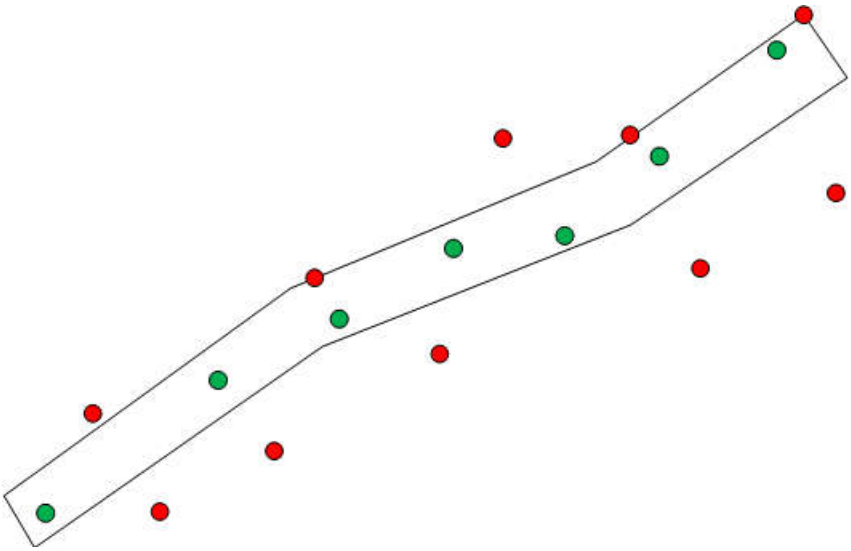
Εικόνα 3-10:Οι έλεγχοι της μεθόδου nextKeyValue

- Στην συνέχεια διαβάζεται η τρέχουσα εγγραφή, τμηματοποιείται βάσει του χαρακτήρα διαχωρισμού που έχει οριστεί από τον δημιουργό του αρχείου εισαγωγής και τέλος, τα αντικείμενα που παράγονται από την τμηματοποίηση αποθηκεύονται σε έναν πίνακα που περιέχει αλφαριθμητικά. Έπειτα, δημιουργείται ένα προσωρινό γεωγραφικό σημείο από το γεωγραφικό μήκος και πλάτος που βρίσκονται στις θέσεις 1 και 2 του παραπάνω πίνακα. Το προσωρινό αυτό σημείο, θα χρησιμοποιηθεί στην επόμενο βήμα για να διαπιστωθεί σε ποια διαδρομή αντιστοιχεί.

```
Text line = reader.getCurrentValue();
LOG.debug(line.toString());
String[] tokens = line.toString().split("\t");
PointImpl tmp = new PointImpl(Double.
        parseDouble(tokens[1]), Double.
        parseDouble(tokens[2]), SpatialContext.GEO);
```

Εικόνα 3-11:Δημιουργία προσωρινού γεωγραφικού σημείου

- Περαιτέρω, ελέγχεται αν το γεωγραφικό σημείο αντιστοιχεί σε κάποια διαδρομή και αν ναι σε ποια. Αυτό το στάδιο είναι ένα από τα σημαντικότερα σημεία της εφαρμογής, γιατί όσο πιο σωστή αντιστοίχιση επιτευχθεί, τόσο καλύτερα δεδομένα θα υπάρχουν στα επόμενα βήματα. Για κάθε διαδρομή έχει δημιουργηθεί ένα γεωγραφικό σχήμα πλάτους 7 μέτρων για να προσομοιωθεί μια πραγματική τυπική οδός. Στην συνέχεια με μία επαναληπτική διαδικασία ελέγχεται αν το σημείο συσχετίζεται με το γεωγραφικό σχήμα της διαδρομής.



Εικόνα 3-12:Αναπαράσταση συσχέτισης γεωγραφικών σημείων με μια διαδρομή

Αν επιτευχθεί συσχέτιση τότε στα αντίγραφα των αντικείμενων GeoKey και GeoValue ανατίθενται τιμές. Από τον πίνακα των αλφαριθμητικών που δημιουργήθηκαν παραπάνω κάποιες τιμές ανατίθενται στο αντικειμένου GeoValue και κάποιες τιμές από το αντικείμενο της διαδρομής ανατίθενται στον GeoKey. Επιπρόσθετα, υπολογίζονται και κάποιες βοηθητικές τιμές που θα χρησιμοποιηθούν στην συνεχεία, όπως είναι η χρονοσφραγίδα σε δευτερόλεπτα. Αν το γεωγραφικό σημείο δεν συσχετιστεί με καμία διαδρομή η εγγραφή αγνοείται από την εφαρμογή. Μόλις εντοπιστεί συσχετισμός ενός γεωγραφικού σημείου με μια διαδρομή τερματίζεται η επαναληπτική διαδικασία προσπέλασης των διαδρομών, προκειμένου το γεωγραφικό σημείο να είναι συσχετισμένο μόνο με μία διαδρομή. Η πλήρης υλοποίηση της συγκεκριμένης διαδικασίας βρίσκεται στο παράρτημα Α.3

```
for (MyPath path : myPaths) {
    SpatialRelation relation = path.getPolyline().relate(tmp);
    if (relation.equals(SpatialRelation.CONTAINS)) {
        key.setLocation(new Text(path.getPathId()));
        key.setLatitude(new DoubleWritable(path.getPolyline().getPoints().get(0).getX()));
        key.setLongitude(new DoubleWritable(path.getPolyline().getPoints().get(0).getY()));
        key.setDistance(new DoubleWritable(path.getDistance()));
        value.setTimestamp(new Text(tokens[0]));
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
        Date date;
        long unixTime = 0;
        try {
            date = dateFormat.parse(tokens[0]);
            unixTime = (long) date.getTime() / 1000;

        } catch (ParseException ex) {

            java.util.logging.Logger.getLogger(GeoRecordReader.class.getName()).log(Level.SEVERE, null, ex);
        }
        value.setUnixTimestamp(new LongWritable(unixTime));
        value.setLatitude(new DoubleWritable(Double.
                parseDouble(tokens[2])));
        value.setLongitude(new DoubleWritable(Double.
                parseDouble(tokens[1])));

        value.setAltitude(new DoubleWritable(Double.
                parseDouble(tokens[3])));
        value.setSpeed(new DoubleWritable(Double.
                parseDouble(tokens[4])));
        value.setOrientation(new DoubleWritable(Double.
                parseDouble(tokens[5])));
    }
    break;
}
```

Εικόνα 3-13:Συσχέτισης γεωγραφικών σημείων με μια διαδρομή

3.4 Η Φάση Map

Εν συνεχεία και με το πέρας της παραπάνω διαδικασίας, ξεκινάει η φάση του Map που είναι ένα από τα κύρια μέρη του αλγορίθμου MapReduce. Η μοναδική μέθοδος που υπάρχει στην φάση του Map είναι η μέθοδος map. Στην μέθοδο map λαμβάνουν χώρα όλες οι διαδικασίες που περιγράφονται παρακάτω.

Όπως προαναφέρθηκε, τα δεδομένα εισόδου σε αυτήν τη μέθοδο έχουν την μορφή ζευγών (key-value), και στην ουσία έχουν συσχετιστεί γεωγραφικά τα δεδομένα κινούμενων οχημάτων με τα δεδομένα διαδρομών. Τα παραπάνω ζεύγη τιμών πρέπει να ομαδοποιηθούν και με βάση μία χρονική περίοδο για να είναι εφικτή η ανάλυση τους αλλά και για να έχουν νόημα τα αποτελέσματα που παράγονται. Τα δεδομένα επιβάλλεται να ομαδοποιηθούν σε χρονικές περιόδους γιατί είναι δεδομένο ότι οι κυκλοφοριακές συνθήκες μεταβάλλονται μέσα στην διάρκεια της ημέρας. Στην παρούσα διπλωματική εργασία χρησιμοποιείται μία χρονική περίοδος μιας ωρολογιακής ώρας, καθώς μετά από δοκιμές κρίθηκε ότι είναι το ιδανικό χρονικό διάστημα για το σετ δεδομένων που θα εξεταστεί.

Για να επιτευχθεί η ομαδοποίηση κρίνεται αναγκαίος ένας μετασχηματισμός του αρχικού κλειδιού GeoKey σε ένα νέο κλειδί το GeoKeyMap. Το αντικείμενο GeoKeyMap αποτελείται από το αρχικό κλειδί GeoKey που εμπεριέχει τις πληροφορίες τις διαδρομής, και τη χρονική περίοδο στην οποία ανήκει η εκάστοτε τιμή του ζεύγους τιμών. Έτσι με το πέρας της φάσης Map τα δεδομένα είναι ομαδοποιημένα βάση της γεωγραφικής πληροφορίας και του χρονικού παραθύρου και την μορφή (key- list(value)) που αποτελεί και είσοδο για την φάση Reduce. Η πλήρης υλοποίηση της συγκεκριμένης διαδικασίας βρίσκεται στο παράρτημα Α.3

```
@Override
protected void map(GeoKey key, GeoValue value, Mapper.Context
context) throws IOException, InterruptedException {
    long bucket = 0;
    if(key.getLocation() != null){
String location = key.getLocation().toString();
bucket =  value.getUnixTimestamp().get() - (value.getUnixTimestamp().get() % 3600);
GeoKeyMap mapKey = new GeoKeyMap(key,new LongWritable(bucket));

context.write(mapKey,value);

}
}
}
```

Εικόνα 3-14:Η μέθοδος map

3.5 Η φάση Reduce

Το δεύτερο κύριο μέρος του αλγορίθμου MapReduce είναι η φάση του Reduce. Σε αυτήν την φάση υπολογίζονται στατιστικές τιμές που αφορούν στις διαδρομές ανά χρονική περίοδο μιας ωρολογιακής ώρας. Οι τιμές που υπολογίζονται είναι το πλήθος των εγγραφών των κινούμενων οχημάτων που αντιστοιχήθηκαν πάνω στην διαδρομή, ο

μέσος όρος των ταχυτήτων, η ελάχιστη ταχύτητα, η μέγιστη ταχύτητα, η διάμεσος των ταχυτήτων και ο μέσος όρος του χρόνου ταξιδιού για αυτήν την διαδρομή.

Για την παραγωγή των παραπάνω τιμών ακολουθούνται τα κατωτέρω αναφερόμενα βήματα. Αρχικά, η λίστα των τιμών προσπελάζεται επαναληπτικά και υπολογίζεται το πλήθος των εγγραφών που αντιστοιχήθηκαν και το άθροισμα όλων των τιμών των ταχυτήτων. Επιπλέον, δημιουργείται μια λίστα τιμών με όλες τις τιμές των ταχυτήτων. Οι παραπάνω τιμές χρησιμοποιούνται για τους υπολογισμούς των τιμών που αφορούν την ταχύτητα.

Στην συνέχεια υπολογίζονται οι τιμές που αφορούν την ταχύτητα. Οι αρχικές τιμές της ταχύτητας από τις εγγραφές που έχουν αντιστοιχιστεί στην εκάστοτε διαδρομή είναι και οι τιμές βάσει των οποίων παράγονται οι περισσότερες τιμές των αποτελεσμάτων της εφαρμογής που αποτελεί το κύριο αντικείμενο της διπλωματικής αυτής εργασίας. Οι υπολογισμοί του μέσου όρου των ταχυτήτων, της εύρεσης της διαμέσου των, αλλά και του υπολογισμού του ελάχιστου και του μέγιστου των ταχυτήτων είναι απλοί δεν χρίζουν περαιτέρω ανάλυσης και απεικονίζονται στην εικόνα 3-13.

```
public Double min(List<Double> array) {
if(array.isEmpty()) return 0.0;
 Double min = array.get(0);
 for(int i=0; i<array.size(); i++) {
    if(array.get(i).compareTo(min)<0) {
       min = array.get(i);
    }
 }
 return min;
}
public Double getMedian(List<Double> array) {
if(array.isEmpty()) return  0.0;
Collections.sort(array);
int middle = array.size() / 2;
middle = middle > 0 && middle % 2 == 0 ? middle - 1 : middle;
return array.get(middle);

 public Double max(List<Double> array) {
 if(array.isEmpty()) return 0.0;
 Double max = array.get(0);
 for(int i=0; i<array.size(); i++) {
    if(array.get(i).compareTo(max)>0) {
       max = array.get(i);
    }
 }
 return max;
}
```

Εικόνα 3-15:Οι υπολογισμοί των τιμών που αφορούν την ταχύτητα

Ο τελευταίος υπολογισμός τιμής που θα πραγματοποιηθεί στην φάση του Reduce είναι ο υπολογισμός του χρόνου ταξιδιού για αυτήν την διαδρομή. Ο χρόνος ταξιδιού υπολογίζεται αν διαιρεθεί το μήκος της διαδρομής με την ταχύτητα. Βασικό σημείο στο οποίο πρέπει να δοθεί ιδιαίτερη προσοχή είναι οι μονάδες μέτρησης των εμπλεκόμενων τιμών. Έτσι, αν η απόσταση είναι σε μέτρα τότε και η ταχύτητα πρέπει να είναι σε μ/δ και τότε ο παραγμένος χρόνος είναι σε δευτερόλεπτα. Στη προκειμένη περίπτωση χρειάστηκε η μετατροπή του μέσου όρου της ταχύτητας που χρησιμοποιήθηκε από χλμ./ω σε μ/δ. Επιπλέον, ελέγχεται η τιμή της ταχύτητας ώστε να μην είναι μηδενική, για τον προφανή λόγο ότι δεν παράγεται κάποιος χρόνος ταξιδιού εφόσον η ταχύτητα είναι μηδενική, ήτοι το όχημα είναι ακινητοποιημένο. Επιπλέον, σε αυτό το σημείο θα πρέπει να αναφερθεί ότι υπολογίζονται και οι τιμές που χρησιμοποιούνται για την ομαδοποίηση και την παρουσίαση των αποτελεσμάτων.

Στο τελευταίο βήμα της παρούσας φάσης δημιουργείται το σχήμα των εγγραφών των αποτελεσμάτων εξόδου. Οι εγγραφές εξόδου αποτελούνται τόσο από τιμές και από το ζεύγος τιμών (key- list(value)), όσο και από τις τιμές που υπολογίζονται στην φάση του Reduce. Τέλος, όλες οι εγγραφές αποθηκεύονται σε ένα αρχείο κειμένου το οποίο βρίσκεται στο σύστημα αρχείων του HDFS. Η πλήρης υλοποίηση της συγκεκριμένης διαδικασίας βρίσκεται στο παράρτημα Α.3

```
context.write(keyMap.getKey().getLocation(),
        new Text(String.valueOf(keyMap.getTimeslot())+"\t"+String.valueOf(count)+"\t"+
        String.valueOf(speed)+"\t"+String.valueOf(time)+"\t"+
        String.valueOf(keyMap.getKey().getDistance().get())+"\t"+
        String.valueOf(max)+"\t"+String.valueOf(min)+"\t"+String.valueOf(median)+"\t"+
        String.valueOf(day)+"\t"+String.valueOf(month)+"\t"+String.valueOf(year)));
```

Εικόνα 3-16:Το σχήμα των δεδομένων εξόδου της φάσης Reduce

3.6 Δημιουργία προφίλ παρόμοιων χρονικών περιόδων για τα δεδομένα πραγματικού χρόνου

Μετά το πέρας και της φάσης του Reduce έχει ολοκληρωθεί η εφαρμογή του Apache MapReduce στα δεδομένα. Εφόσον η εφαρμογή αυτή είναι επιτυχής, το επόμενο στάδιο της εφαρμογής αφορά την διαχείριση των αποτελεσμάτων που παράχθηκαν. Τα αποτελέσματα που παράχθηκαν προέρχονται είτε από δεδομένα ιστορικού, είτε από δεδομένα πραγματικού χρόνου και καταλήγουν να αποθηκεύονται σε μια σχεσιακή βάση δεδομένων. Η διαδικασία της αποθήκευσης παρουσιάζεται σε επόμενο κεφάλαιο.

Όταν γίνεται διαχείριση δεδομένων πραγματικού χρόνου, εκτός από την αποθήκευση των τιμών στην σχεσιακή βάση δεδομένων, δημιουργείται και το προφίλ της χρονικής περιόδου σε σχέση με τις παρόμοιες χρονικές περιόδους των προηγούμενων ετών. Με αυτόν τον τρόπο παρέχεται μια τάξη μεγέθους σχετικά με το αν οι τιμές της τρέχουσας χρονικής περιόδου είναι βελτιωμένες η όχι σε σχέση με παρελθόντα έτη. Οι τιμές που εξετάζονται κατά την διαδικασία αυτή είναι οι μέσες τιμές των ταχυτήτων, ο χρόνος ταξιδιού, καθώς και ο αριθμός των εγγραφών που αντιστοιχήθηκαν στην εκάστοτε διαδρομή. Η πλήρης υλοποίηση της συγκεκριμένης διαδικασίας βρίσκεται στο παράρτημα Α.3.

3.6.1 Συλλογή δεδομένων παρελθόντων χρονικών περιόδων

Το πρώτο βήμα της παρούσας διαδικασίας είναι ο εντοπισμός των παρόμοιων χρονικών περιόδων και η συλλογή των δεδομένων από την σχεσιακή βάση δεδομένων. Αρχικά, δημιουργήθηκε η μέθοδος getSameDayOfYearD η οποία επιστρέφει ένα αντικείμενο Date το οποίο περιέχει την ημερομηνία που αφορά μία χρονική περίοδο παρελθόντος χρόνου. Με την βοήθεια του αντικειμένου Calendar που είναι μέρος των βασικών βιβλιοθηκών της γλώσσας προγραμματισμού java εντοπίζονται τα χαρακτηριστικά της τρέχουσας χρονικής περιόδου. Πιο συγκεκριμένα χρησιμοποιώντας ως παράμετρο στην μέθοδο get την τιμή *Calendar.DAY_OF_WEEK_IN_MONTH* επιστρέφει τον αριθμό των εμφανίσεων της συγκεκριμένης ημέρας μέσα στον μήνα. Όταν χρησιμοποιείται η παράμετρος *Calendar.DAY_OF_WEEK* η μέθοδος get του αντικειμένου Calendar επιστρέφει έναν αριθμό που αντιπροσωπεύει μία από τις μέρες της εβδομάδας.

Οι δύο παραπάνω τιμές αυτές μαζί με την τιμή του μήνα, καθώς και την χρονιά που επιλέγεται να εντοπιστεί δίδονται ως ορίσματα στην μέθοδο getSameDayOfYearD. Για παράδειγμα, έστω ότι πρέπει να εντοπιστούν οι παρόμοιες χρονικές περίοδοι, των προηγούμενων 2 ετών, για το χρονικό παράθυρο της Τρίτης 12 Απριλίου 2019 22:00:00-22:59:59. Στον παρακάτω πίνακα δίδονται τα ορίσματα της μεθόδου getSameDayOfYearD καθώς και τα επιστρεφόμενα αποτελέσματα.

Πίνακας 3-3:Ορίσματα και αποτελέσματα μεθόδου getSameDayOfYearD

Year	Month	Calendar.DAY_OF_WEEK_IN_MONTH	Calendar.DAY_OF_WEEK	Αποτέλεσμα
2018	3	2	3	Τρίτης 10 Απρίλιου 2018 22:00:00
2017	3	2	3	Τρίτης 11 Απρίλιου 2017 22:00:00

Στην συνέχεια και εφόσον έχουν ανακτηθεί οι ημερομηνίες που αντιπροσωπεύουν τις παρόμοιες χρονικές περιόδους δημιουργείται ένα ερώτημα και υποβάλλεται στην σχεσιακή βάση δεδομένων. Έπειτα, γεμίζονται τα αντικείμενα με τις τιμές που επέστρεψαν από την σχεσιακή βάση δεδομένων, και είναι έτοιμα να δοθούν ως όρισμα στην επόμενη μέθοδο η οποία θα υπολογίσει τις αποκλίσεις των τιμών ανάμεσα στις παρόμοιες χρονικές περιόδους.

```
Calendar cal = Calendar.getInstance();
int dayOfWeekInMonth = cal.get(Calendar.DAY_OF_WEEK_IN_MONTH);
int dayOfWeek = cal.get(Calendar.DAY_OF_WEEK);
int year = cal.get(Calendar.YEAR);
int month = cal.get(Calendar.MONTH);
long previousYearTimestamp = getSameDayOfYearD(year - 1, month,
        dayOfWeek, dayOfWeekInMonth).getTime() / 1000;
long previous2YearTimestamp = getSameDayOfYearD(year - 2, month,
        dayOfWeek, dayOfWeekInMonth).getTime() / 1000;
String queryProfileSameDay = " select * from FILTER_DATA T WHERE"
        + " (T.TIMESTAMP = ? OR T.TIMESTAMP = ?)"
        + " AND T.PATH_ID = ? ORDER BY T.TIMESTAMP ";
PreparedStatement pStmtpProfileDay = conn.prepareStatement(queryProfileSameDay);
pStmtpProfileDay.setLong(1, previousYearTimestamp);
pStmtpProfileDay.setLong(2, previous2YearTimestamp);
pStmtpProfileDay.setLong(3, pathId);

ResultSet rs = pStmtpProfileDay.executeQuery();

while (rs.next()) {
    FilterData filterData = new FilterData();
    filterData.setPathId(rs.getInt(1));
    filterData.setTimestamp(rs.getLong(2));
    filterData.setTime(rs.getInt(3));
    filterData.setCount(rs.getInt(4));
    filterData.setSpeed(rs.getDouble(6));
    filterData.setDay(rs.getInt(7));
    filterData.setMonth(rs.getInt(8));
    filterData.setYear(rs.getInt(9));
    filterData.setMedianSpeed(rs.getInt(10));
    filterData.setMaxSpeed(rs.getInt(11));
    filterData.setMinSpeed(rs.getInt(12));
    profileDayData.add(filterData);

}
```

Εικόνα 3-17:Ερώτημα στην βάση και ανάκτηση δεδομένων

3.6.2 Υπολογισμός τιμών χρονικών περιόδων

Για τον υπολογισμό των αποκλίσεων των τιμών της τρέχουσας χρονικής περιόδου σε σχέση με τις παρελθοντικές χρονικές περιόδους, έχει δημιουργηθεί η μέθοδος createProfileForCurrentDayTime. Η παρούσα μέθοδος δέχεται ως ορίσματα μια

λίστα αντικειμένων που περιέχουν τις τιμές των παρελθοντικών χρονικών περιόδων, την χρονοσφραγίδα, την τιμή της ταχύτητας, το χρόνο ταξιδιού, τον αριθμό των αντιστοιχίσεων στην εκάστοτε διαδρομή της τρέχουσας χρονικής περιόδου και το αντικείμενο σύνδεσης για την σχεσιακή βάση δεδομένων. Έπειτα, για κάθε αντικείμενο που βρίσκεται μέσα στην λίστα υπολογίζονται οι διαφορές των δυο χρονικών περιόδων σχετικά με την τιμή της ταχύτητας, τον χρόνο ταξιδιού και τον αριθμό των αντιστοιχίσεων. Οι τιμές που υπολογίζονται σε κάθε σύγκρισή δύο χρονικών περιόδων αποθηκεύονται σε ένα προσωρινό αντικείμενο.

Μετά το πέρας των υπολογισμών, όλα τα προσωρινά αρχεία που περιέχουν τα αποτελέσματα, δίνονται ως όρισμα σε μία άλλη μέθοδο που έχει ως σκοπό την αποθήκευση των αποτελεσμάτων στην σχεσιακή βάση δεδομένων. Τα αποτελέσματα αυτά παρουσιάζονται στη διαδικτυακή πύλη που δημιουργήθηκε για τις ανάγκες της παρουσίασης των αποτελεσμάτων της παρούσας διπλωματικής εργασίας.

3.7 Αποθήκευση αποτελεσμάτων

Ένας από τους βασικούς στόχους της παρούσας διπλωματικής εργασίας είναι και η αποθήκευση του συνόλου των αποτελεσμάτων που παράγονται, είτε για την περαιτέρω ανάλυση τους, είτε για την παρουσίαση τους στη σχετική διαδικτυακή πλατφόρμα για την εξαγωγή συμπερασμάτων. Για την αποθήκευση των αποτελεσμάτων έχει επιλεγεί η χρήση μιας σχεσιακής βάσης δεδομένων MySQL η οποία φιλοξενείται στις could υπηρεσίες της Amazon. Επιλέχθηκε ο παραπάνω συνδυασμός γιατί προσφέρει αξιοπιστία, είναι επεκτάσιμος και εύχρηστος.

3.7.1 Αποθήκευση αποτελεσμάτων του αλγορίθμου MapReduce

Για την αποθήκευση των αποτελεσμάτων που παράχθηκαν από τον αλγόριθμο MapReduce, πρέπει πρώτα να ανακτηθεί από το HDFS το αρχείο με τα αποτελέσματα. Αρχικά, πρέπει να δημιουργηθεί ένα αντικείμενο το οποίο είναι υπεύθυνο με την επικοινωνία με τον NameNode και κατ' επέκταση και με το σύστημα αρχείων του HDFS. Έπειτα, δημιουργείται ένα αντικείμενο με την διαδρομή στο αρχείο που μας ενδιαφέρει.

```
Configuration hadoopConfig = new Configuration();
hadoopConfig.set("fs.defaultFS", "hdfs://quickstart.cloudera:8020");
hadoopConfig.set("fs.file.impl", org.apache.hadoop.fs.LocalFileSystem.class.getName());
FileSystem fs = FileSystem.get(hadoopConfig);
Path inputPath2 = new Path(outputPath.trim().replace("\n", "") + "/part-r-00000");
openFileFromHDFS(fs, inputPath2, realTime);
```

Εικόνα 3-18:Αντικείμενο για επικοινωνία με NameNode και HDFS

Στην συνέχεια, καλείται η μέθοδος openFileFromHDFS όπου αρχικοποιείται το αντικείμενο που είναι υπεύθυνο για την σύνδεση στην βάση δεδομένων. Ακολούθως, διαβάζεται το αρχείο εισόδου γραμμή γραμμή και δίδεται σαν παράμετρος σε άλλες μεθόδους που θα πραγματοποιήσουν την αποθήκευση, οι οποίες διαφέρουν ανάλογα με το αν πρόκειται για δεδομένα πραγματικού χρόνου ή για δεδομένα ιστορικού. Με το τέλος των εγγράφων του αρχείου εισόδου τερματίζεται και σύνδεση με την βάση δεδομένων.

```
public static void openFileFromHDFS(FileSystem fs, Path path, boolean realTime) throws IOException,
        ClassNotFoundException, InstantiationException, IllegalAccessException, SQLException {

    InputStream is = fs.open(path);
    String readLine;
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
            Class.forName("com.mysql.jdbc.Driver").newInstance();
    Connection conn = DriverManager.getConnection(
            "jdbc:mysql://geofilterdb.c4nkehdtywwc.us-east-2.rds.amazonaws.com:3306/geofilterdb",
            "cloudera", "cloudera");
    if (realTime) {
        while (((readLine = br.readLine()) != null)) {
            saveItemToDBRealTime(readLine, conn);
            System.out.println(readLine);

        }

    } else {
        while (((readLine = br.readLine()) != null)) {
            saveItemToDB(readLine, conn);
            System.out.println(readLine);

        }
    }
    conn.close();
}
```

Εικόνα 3-19:Η μέθοδος openFileFromHDFS

Στο τελευταίο βήμα που πραγματοποιείται, οι δύο μέθοδοι προετοιμάζουν και εισάγουν τα δεδομένα στην βάση δεδομένων. Οι δύο αυτές μέθοδοι κάνουν παρόμοια πράγματα αλλά προτιμήθηκε για την λογική ροή της εφαρμογής να διαχειρίζονται χωριστά τα δεδομένα πραγματικού χρόνου από τα ιστορικά δεδομένα. Θα παρουσιαστεί η μέθοδος saveItemToDB η οποία παίρνει ως όρισμα μια έγγραφή και την σύνδεση στην βάση δεδομένων και πραγματοποιεί την εισαγωγή. Αρχικά, δημιουργείται ερώτημα που εισάγει τα δεδομένα στην βάση δεδομένων. Ακολούθως, ανακτώνται οι τιμές ενδιαφέροντος από την εγγραφή και δίδονται ως παράμετροι στην ερώτημα που θα εκτελεστεί. Εφόσον έχουν ανακτηθεί όλες οι τιμές από την εγγραφή και το ερώτημα είναι έτοιμο, εκτελείται και εισάγει τα δεδομένα στην σχεσιακή βάση.

```
// the mysql insert statement
String query = " insert into FILTER_DATA (PATH_ID,TIMESTAMP ,COUNT, SPEED,"
        + " TIME, MAX_SPEED, MIN_SPEED, MEDIAN_SPEED, DAY, MONTH, YEAR)"
        + " values (?, ?, ?, ?, ?,?, ?, ?, ?,?,?)";

// create the mysql insert preparedstatement
String[] parts = readLine.split("\t");
PreparedStatement preparedStmt = conn.prepareStatement(query);
//PATH_ID
if (!parts[0].equals("") && !parts[0].equals("NaN")) {
    preparedStmt.setInt(1, Integer.parseInt(parts[0]));
} else {
    preparedStmt.setInt(1, 0);
}
```

Εικόνα 3-20:Εισαγωγή δεδομένων στην βάση

3.7.2 Αποθήκευση τιμών παρόμοιων χρονικών παραθύρων

Για την αποθήκευση των τιμών των παρόμοιων χρονικών παραθύρων ακολουθείται μια διαδικασία παρόμοια με αυτήν που περιγράφεται στο προηγούμενο κεφάλαιο. Η διαφορά είναι ότι τα αποτελέσματα δεν ανακτώνται από κάποιο αρχείο, αλλά βρίσκονται ήδη σε μια λίστα μετά το πέρας του υπολογισμού των τιμών. Στην συνέχεια, καλείται η μέθοδος που θα προετοιμάσει και θα εισάγει τα δεδομένα στην βάση δεδομένων. Η μέθοδος αυτή ονομάζεται saveProfileDataToDB. Στην μέθοδο αυτή δημιουργείται το ερώτημα που θα εισάγει τα δεδομένα στην βάση. Έπειτα, δίνονται οι τιμές εισόδου παραμετρικά στο ερώτημα. Τέλος, εκτελείται το ερώτημα και τα δεδομένα εισάγονται στην βάση.

3.8 Ενοποίηση, εκκίνηση και αυτοματοποίηση διαδικασιών

Τα ανωτέρω επιμέρους βήματα που περιγράφονται, τα οποία αποτελούν και τον πυρήνα της υπολογιστικής διαδικασίας της παρούσας διπλωματικής εργασίας πρέπει να ενοποιηθούν σε μία παραμετροποιήσιμη εφαρμογή η οποία θα εκτελείται σύμφωνα με τις απαιτήσεις του εκάστοτε χρήστη. Επιπρόσθετα, για της ανάγκες της παρούσας εργασίας αναπτύχθηκαν αυτοματοποιημένες διαδικασίες οι οποίες εκτελούνται με την βοήθεια του χρονοπρογραμματιστή του λειτουργικού συστήματος. Οι διαδικασίες αυτές επιτρέπουν την αδιάκοπη λειτουργία της εφαρμογής χωρίς την παρέμβαση του ανθρώπινου παράγοντα.

3.8.1 Ενοποίηση διαδικασιών

Για την ενοποίηση των παραπάνω βημάτων σε μια εφαρμογή που μπορεί να εκτελείται και να διανέμεται ανεξάρτητα, επιλέχθηκε η δημιουργία ενός αρχείου τύπου

jar. Το αρχείο jar είναι ένα συμπιεσμένο αρχείο που περιλαμβάνει αρχεία κλάσεων java καθώς και βοηθητικών αρχείων όπως εικόνες και αρχεία κειμένου. Τέλος, περιλαμβάνει και ένα αρχείο ρυθμίσεων, το MANIFEST.MF. Το αρχείο jar είναι εκτελέσιμο όταν συμπεριλαμβάνει μια κλάση main η οποία δέχεται παραμέτρους, εκτελεί τις απαραίτητες ενέργειες και τέλος παράγει το τελικό αποτέλεσμα. Η κλάση main δηλώνεται κατά την εκτέλεση της εφαρμογής.

Τα βήματα που ακολουθούνται για την δημιουργία της εκτελέσιμης εφαρμογής είναι δύο. Το πρώτο αφορά την μεταγλώττιση των αρχείων της java για να παραχθούν τα εκτελέσιμα αρχεία της java, ενώ το δεύτερο αφορά την δημιουργία του αρχείου jar. Για το πρώτο βήμα προκειμένου να γίνει η μεταγλώττιση πρέπει να εκτελεστεί η εντολή που απεικονίζεται στην εικόνα 3-19. Παρατηρείται ότι κατά την μεταγλώττιση συμπεριλαμβάνονται οι βιβλιοθήκες που χρησιμοποιούνται από την εφαρμογή, αυτές είναι οι βιβλιοθήκες του Hadoop, του Map-Reduce, του driver για την σύνδεση στην βάση δεδομένων και δευτερεύουσες βοηθητικές βιβλιοθήκες. Στην συνέχεια, υπάρχει η διαδρομή προς τα αρχεία java και η διαδρομή αποθήκευσης των μεταγλωττισμένων αρχείων. Τέλος, υπάρχουν οι διάφορες επιλογές και ρυθμίσεις που ορίζονται στον μεταγλωττιστή. Εκτελώντας την μεταγλώττιση και εάν δεν υπάρχει κάποιο σφάλμα τα μεταγλωττισμένα αρχεία δημιουργούνται στην διαδρομή που έχει οριστεί. Εάν υπάρχει κάποιο σφάλμα τότε αυτό εμφανίζεται στην κονσόλα του λειτουργικού συστήματος, όπου εκτελέστηκε η εντολή.

```
[cloudera@quickstart ~]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/*:/home/cloudera/.m2/repository/org/json/
json/20180813/*:/home/cloudera/.m2/repository/com/spatial4j/spatial4j/0.5/*:/home/cloudera/Downloads/mysql-connector-jav
a-8.0.15/* /home/cloudera/NetBeansProjects/GeoMapReduceJob/src/main/java/com/anmpout/geomapreducejob/* -d build -Xlint
```

Εικόνα 3-21: Μεταγλώττιση εφαρμογής

Το επόμενο βήμα αφορά την δημιουργία του αρχείου jar. Όπως και στην μεταγλώττιση έτσι και εδώ εκτελείται μια εντολή στην κονσόλα του λειτουργικού συστήματος. Η εντολή αυτή παρουσιάζεται στην εικόνα 3-20. Η εντολή αποτελείται από τις επιλογές της βιβλιοθήκης που χρησιμοποιείται για την δημιουργία του αρχείου jar. Επιπλέον, αποτελείται από το όνομα που δίνεται στο αρχείο jar καθώς και την διαδρομή που θα αποθηκευτεί. Στην συνέχεια δίνεται η διαδρομή των μεταγλωττισμένων αρχείων που συμπεριλαμβάνονται σε αυτό. Εάν δημιουργηθεί κάποιο σφάλμα κατά την εκτέλεση της εντολής, αυτό εμφανίζεται στην κονσόλα του λειτουργικού συστήματος. Εάν δεν εμφανιστεί κάποιο σφάλμα το αρχείο jar βρίσκεται στην διαδρομή όπου το έχουμε δηλώσει.

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ jar -cvf geomapfilter.jar -C build/ .
```

Εικόνα 3-22: Διμιουργία jar αρχείου

3.8.2 Εκκίνηση εφαρμογής

Για την εκκίνηση της εφαρμογής αρχικά πρέπει να γίνουν κάποιες προπαρασκευαστικές ενέργειες στο λειτουργικό σύστημα όπου εκτελείται η εφαρμογή καθώς επίσης και στο λειτουργικό σύστημα να υπάρχουν εγκατεστημένες και παραμετροποιημένες καταλλήλως οι εφαρμογές που απαρτίζουν το οικοσύστημα Hadoop. Στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε το Cloudera CHD (Cloudera Distributed Hadoop), το όποιο είναι μια πλατφόρμα ανοιχτού κώδικα, η οποία περιλαμβάνει όλες τις βασικές λειτουργίες του Hadoop οικοσυστήματος εγκατεστημένες σε ένα Unix λειτουργικό σύστημα [35]. Το CHD διανέμεται ως εικονική μηχανή. Με την εκκίνηση της εικονικής μηχανής παρέχεται ένα έτοιμο περιβάλλον όπου μπορούμε να χρησιμοποιήσουμε όλες τις δυνατότητες του Hadoop οικοσυστήματος που αναφέρονται παραπάνω.

Επιπλέον για την εκκίνηση της εφαρμογής θα πρέπει να αντιγράψουμε τις βιβλιοθήκες που χρησιμοποιήθηκαν κατά την μεταγλώττιση των java κλάσεων στην διαδρομή του λειτουργικού συστήματος */usr/lib/hadoop/lib* έτσι ώστε να είναι προσβάσιμες από την εφαρμογή κατά την εκτέλεση της.

Εφόσον τηρούνται οι παραπάνω προϋποθέσεις μπορεί να εκκινήσει με επιτυχία η εφαρμογή εκτελώντας την εντολή της εικόνας 3-21 στην κονσόλα του λειτουργικού συστήματος. Η εντολή αποτελείται από την δήλωση του τύπου της εκτελέσιμης εφαρμογής που στην συγκεκριμένη περίπτωση είναι τύπου jar. Στη συνέχεια παρουσιάζεται η διαδρομή προς το αρχείο jar. Έπειτα, δηλώνεται το όνομα της main κλάσης μαζί με το πακέτο της. Τέλος, υπάρχουν οι τρείς παράμετροι με τις οποίες μπορεί να παραμετροποιηθεί η εφαρμογή. Η πρώτη παράμετρος αφορά την διαδρομή του αρχείου εισόδου με τα δεδομένα, η δεύτερη ενημερώνει την εφαρμογή για τον τύπο των δεδομένων, δηλαδή εάν είναι δεδομένα πραγματικού χρόνου ή δεδομένα ιστορικού. Η τρίτη και τελευταία παράμετρος ενημερώνει την εφαρμογή για την αποθήκευση των δεδομένων στην βάση δεδομένων όπως περιγράφεται παραπάνω.

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ hadoop jar geomapfilter.jar com.anmpout.geomapreducejob.GeoFilter
 /user/thesis/samples/fcd_gps_06_2017_1 false true
```

Εικόνα 3-23: Εντολή εκκίνησης jar εφαρμογής

3.8.3 Αυτοματοποίηση διαδικασιών

Για τον υπολογισμό των τιμών που αφορούν τα δεδομένα πραγματικού χρόνου επιβάλλεται η εκκίνηση της εφαρμογής jar κάθε μια ώρα για τον υπολογισμό των τιμών της τελευταίας χρονικής περιόδου από τα δεδομένα που συλλέγονται. Για τον σκοπό αυτόν αναπτύχθηκε ένα shell script. Το shell script είναι ένα πρόγραμμα το οποίο εκτελείται στο κέλυφος (shell) του Unix λειτουργικού συστήματος. Το πρόγραμμα που αναπτύχθηκε πραγματοποιεί τις ακόλουθες ενέργειες. Αρχικά, παίρνει την τρέχουσα ημερομηνία από το λειτουργικό σύστημα και στην συνέχεια αναζητά το αρχείο της αμέσως προηγουμένης χρονικής περιόδου. Μόλις εντοπιστεί το αρχείο αντιγράφεται από σύστημα αρχείων του λειτουργικού συστήματος στο HDFS. Στην συνέχεια, ξεκινάει η εφαρμογή με τις κατάλληλες τιμές των παραμέτρων για τον υπολογισμό των τιμών. Μετά το πέρας των υπολογισμών το προσωρινό αρχείο διαγράφεται από το HDFS και τερματίζεται το πρόγραμμα.

```
1   #!/bin/bash
2   #retrieve date
3   timestamp=$( date +"%Y-%m-%d" )
4   #lookup current date file
5   for entry in "$search_dir"/home/cloudera/Downloads/realTime/$timestamp/*
6   do
7   #create file name
8   file=$(basename "$entry")
9   #create timestamp
10  timestamp=$( date +%H )
11  #convert timestamp
12  run=$((10#$timestamp-1))
13  #if we have a match excecute the following commands
14  if [ "$run" = "$file" ]
15  then
16  #copy tmp file from local fs to HDFS
17  hadoop fs -copyFromLocal $entry /user/thesis/samples/$file
18  hadoop fs -ls /user/thesis/samples
19  #execute jar for tmp file
20  hadoop jar /home/cloudera/geomapfilter.jar
    com.anmpout.geomapreducejob.GeoFilter /user/thesis/samples/$file true
    true
21  #remove tmp file
22  hadoop fs -rm /user/thesis/samples/$file
23  hadoop fs -ls /user/thesis/samples
24  fi
25  done
```

Εικόνα 3-24: Shell script για αυτοματοποίηση διαδικασίας

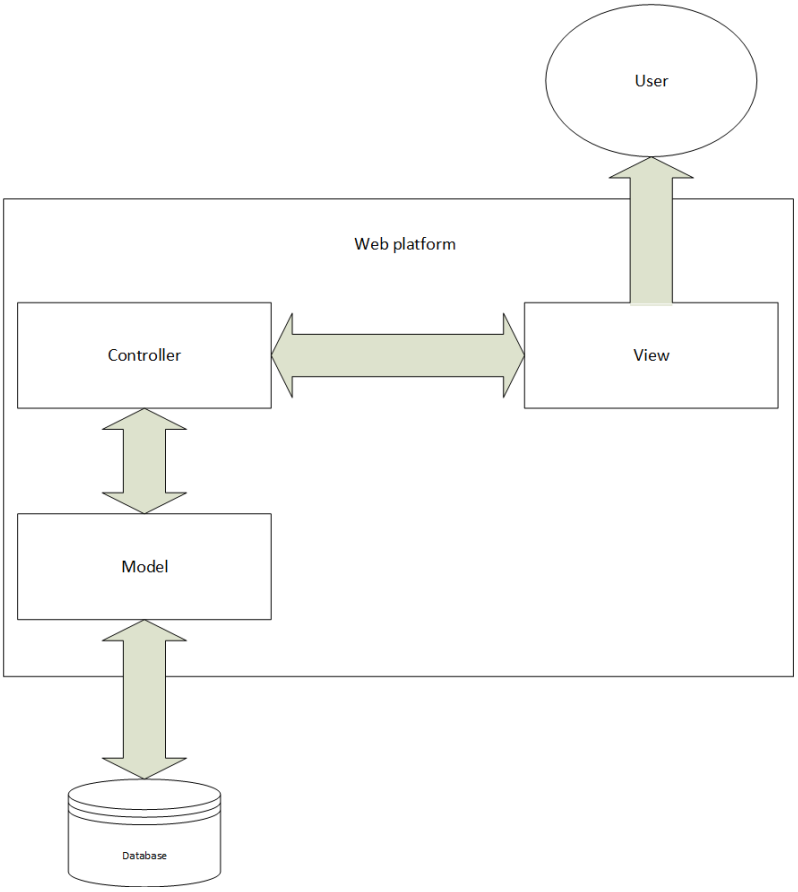
Το παραπάνω πρόγραμμα θα πρέπει να εκτελείται κάθε ώρα. Για το λόγο αυτό έχει δηλωθεί στο χρονοπρογραμματιστή του λειτουργικού συστήματος μια εργασία η οποία το εκτελεί κάθε μια ωρολογιακή ώρα.

3.9 Παρουσίαση διαδικτυακής πύλης

Στο πλαίσιο της παρούσας διπλωματικής εργασίας και για την ευκολότερη και πιο κατανοητή παρουσίαση των αποτελεσμάτων δημιουργήθηκε μια διαδικτυακή πύλη. Η διαδικτυακή πύλη αναπτύχθηκε με τεχνολογίες Java και πιο συγκεκριμένα με Java, Enterprise Edition όσον αφορά το κομμάτι του προγραμματισμού στην μεριά του διακομιστή. Για το κομμάτι του προγραμματισμού στο κομμάτι του client έχουν χρησιμοποιηθεί JavaServer Faces, xhtml, css και javascript. Η διαδικτυακή πύλη σχεδιάστηκε βάση του προτύπου που είναι διαθέσιμο στο έργο ανοιχτού κώδικά AdminFaces [33]. Ως application server για την φιλοξενία της διαδικτυακής πύλης χρησιμοποιείται ο WildFly. Ο WildFly που παλαιότερα ήταν γνωστός ως JBoss αναπτύσσεται από την Red Hat και την κοινότητα που την περιστοιχίζει, διατίθεται δωρεάν και είναι ένα έργο ανοιχτού λογισμικού [34]. Η πλήρης υλοποίηση της διαδικασίας βρίσκεται στο GitHub (https://github.com/ampoutovinas/fcd-dashboard).

3.9.1 Αρχιτεκτονική διαδικτυακής πλατφόρμας

Το αρχιτεκτονικό πρότυπο που επιλέχθηκε να ακολουθηθεί για την ανάπτυξή της διαδικτυακής πλατφόρμας είναι το πρότυπο MVC (Model-View-Controller). Στο συγκεκριμένο πρότυπο υπάρχουν τρία διακριτά τμήματα τα οποία αλληλοεπιδρούν μεταξύ τους, καθώς και με το εξωτερικό περιβάλλον της πλατφόρμας. Το τμήμα model απαρτίζεται από τα αντικείμενα εκείνα που εκτελούν ενέργειες που αφορούν τα δεδομένα της διαδικτυακής πλατφόρμας. Οι ενέργειες αυτές είναι η ανάκτηση, η αποθήκευση και η τροποποίηση των δεδομένων από την σχεσιακή βάση δεδομένων που χρησιμοποιεί η πλατφόρμα. Το τμήμα view απαρτίζεται από τις οντότητές αυτές που είναι υπεύθυνες για την παρουσίαση της πληροφορίας στον χρήση. Τέλος το τμήμα του controller απαρτίζεται από εκείνες τις ενότητες που παίζουν τον συνδετικό κρίκο ανάμεσα στο view και στο model αναλαμβάνοντας τις επικοινωνία και την ανταλλαγή δεδομένων.



Εικόνα 3-25: Αρχιτεκτονική διαδικτυακής πλατφόρμας

3.9.2 Παρουσίαση βασικών τμημάτων της διαδικτυακής πλατφόρμας

Στην παρούσα ενότητα θα παρουσιαστούν τα τμήματα της πλατφόρμας σύμφωνα με το πρότυπο της αρχιτεκτονικής που ακολουθήθηκε στην ανάπτυξη της παρούσας διαδικτυακής πλατφόρμας. Αρχικά το τμήμα του model αποτελείται από τα αντικείμενα των Java κλάσεων τα οποία θα υποδεχθούν τα δεδομένα από την βάση δεδομένων. Αξίζει να σημειωθεί πως τα συγκεκριμένα αντικείμενα αποτυπώνουν ακριβώς τις συσχετίσεις μεταξύ των οντοτήτων της βάσης δεδομένων. Έτσι έχουμε πλήρη πρόσβαση στα αντικείμενα της βάσης δεδομένων συμπεριλαμβανομένων και των συσχετίσεων τους. Επιπλέον, στο τμήμα του model εντάσσεται η διεπαφή (interface), η οποία ορίζει όλες εκείνες τις ενέργειες που θα εκτελεστούν πάνω στα δεδομένα. Η συγκεκριμένη διεπαφή συνήθως ονομάζεται DAO (Data Access Object). Η χρήση της διεπαφής κρίνεται αναγκαία για λογούς ασφάλειας της βάσης δεδομένων, γιατί με αυτόν το τρόπο διασφαλίζεται ότι δεν αποκαλύπτονται οι πληροφορίες που αφορούν την σύνδεση στην βάση δεδομένων. Τέλος, στο τμήμα του model εντάσσεται και το αντικείμενο που υλοποιεί την διεπαφή και είναι το συνδετικό αντικείμενο μεταξύ του τμήματος model και το τμήματος controller.

```
@Entity
@IdClass(FilterDataKey.class)
@Table(name = "FILTER_DATA")
public class FilterData implements Serializable{
@Id
@Column(name = "PATH_ID")
private Integer pathId;
@Id
@Column(name = "TIMESTAMP")
private Long  timestamp;

@Column(name = "COUNT")
private Integer  count;
```

Εικόνα 3-26: Αντικείμενο αποθήκευσης οντότητας βάσης δεδομένων.

```java
@Remote
public interface PathDao {
    /**
     * Return a single path
     * @param pathId
     * @return Path
     */
    public Path getPath(Integer pathId);
    /**
     * Returns all available paths
     * @return List of Paths
     */
    public List<Path> getAllPaths();
    /**
     * Returns all paths for a region
     * @param regionId
     * @return List of Paths
     */
    public List<Path> getAllRegionPaths(Integer regionId);
    /**
     * Return data for a day
     * @param pathId
     * @param timestampFrom
     * @param timestampTo
     * @return List of FilterData
     */
```

Εικόνα 3-27: Η διεπαφή DAO

Επιπρόσθετα, τμήμα του view απαρτίζεται από όλες τις οντότητες οι οποίες δημιουργούν την τελική διεπαφή στον χρήστη. Πρόκειται για τα αρχεία xhtml που βρίσκονται στην διαδικτυακή πλατφόρμα τα οποία παρουσιάζουν τα δεδομένα στον χρήστη καθώς επίσης μεταβιβάζουν και όλες τις αλληλεπιδράσεις του χρήστη προς το τμήμα του controller. Αυτό που αξίζει να σημειωθεί για τα αρχεία xhtml είναι ότι διαφέρουν σε σχέση με τα ευρέως διαδεδομένα html αρχεία. Η δομή τους είναι πιο αυστηρή και πρέπει να είναι καλοσχηματισμένα συμφώνα με αυτά που ορίζουν οι κατευθύνσεις των xml αρχείων. Αυτό είναι επιβεβλημένο για την ορθή λειτουργεία της πλατφόρμας, καθώς το framework JavaServer Faces που χρησιμοποιήθηκε για την ανάπτυξή της διεπαφής της διαδικτυακής πλατφόρμας απαιτεί αρχεία που εναρμονίζονται με τους κανόνες των xml αρχείων.

Το τμήμα controller απαρτίζεται από αρχεία κλάσεων java τα οποία είναι επιφορτιζόμενα τόσο με την τροφοδότηση του τμήματος view με δεδομένα και πιο συγκεκριμένα με την τροφοδότηση των αρχείων xhtml, όσο και με την διαχείριση των

αλληλεπιδράσεων του χρήση. Επιπλέον, ο controller επικοινωνεί αμφίδρομα με το τμήμα του model, είτε ζητώντας να ανακτηθούν κάποια δεδομένα, είτε λαμβάνοντας ενημέρωση για μια αλλαγή στα δεδομένα. Για παράδειγμα, εάν ένας χρήστης πατήσει ένα κουμπί σε μια διεπαφή της διαδικτυακής πλατφόρμας το οποίο ανανεώνει τα δεδομένα που έχει μπροστά του, τότε αυτό το event αφού φτάσει στην κλάση java που παίζει τον ρόλο του controller για το συγκεκριμένο view, το μεταβιβάζει με την σειρά του στο τμήμα του model για να του επιστρέψει τα κατάλληλα δεδομένα. Μόλις τα δεδομένα ανακτηθούν από το model μεταβιβάζονται στον controller και αυτός με την σειρά τους τα μεταβιβάζει στο view.

Εκτός από τα αρχεία που αποτελούν μέρος της λογικής αρχιτεκτονικής της πλατφόρμας, υπάρχουν και πολλά ακόμη αρχεία που αποτελούν σημαντικό κομμάτι της και χωρίς αυτά δεν θα ήταν λειτουργική. Τα αρχεία αυτά τα οποία διαδραματίζουν σημαντικό ρόλο για την εύρυθμή λειτουργία της πλατφόρμας είναι τρία αρχεία ρυθμίσεων και συγκεκριμένα το web.xml, το persistence.xml και το pom.xml. Το web.xml είναι ένα αρχείο ρυθμίσεων που αφορά την εγκατάστασή της εφαρμογής στο application server. Σε αυτό περιλαμβάνονται όλες οι ρυθμίσεις που πρέπει να ισχύουν για την διαδικτυακή πλατφόρμα, όπως για παράδειγμα η αρχική σελίδα της εφαρμογή, η γραμματοσειρά στην οποία θα παρουσιάζεται η εφαρμογή, η διαχείριση των σφαλμάτων από την διαδικτυακή πλατφόρμα κ.α. Στην εικόνα 3-26 παρουσιάζεται ένα τμήμα του web.xml.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee
3            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5       <context-param>
6           <param-name>javax.faces.PROJECT_STAGE</param-name>
7           <param-value>Development</param-value>
8       </context-param>
9       <context-param><param-name>primefaces.THEME</param-name>
10      <param-value>admin</param-value>
11      </context-param>
12          <context-param>
13          <param-name>primefaces.FONT_AWESOME</param-name>
14          <param-value>true</param-value>
15      </context-param>
16      <servlet>
17          <servlet-name>Faces Servlet</servlet-name>
18          <servlet-class>javax.faces.webapp.FacesServlet</servlet-
19          <load-on-startup>1</load-on-startup>
20      </servlet>
21      <servlet-mapping>
22          <servlet-name>Faces Servlet</servlet-name>
23          <url-pattern>*.xhtml</url-pattern>
24      </servlet-mapping>
25      <session-config>
26          <session-timeout>
27              30
28          </session-timeout>
29      </session-config>
```

Εικόνα 3-28: Τμήμα του web.xml

Το αρχείο persistence.xml είναι ένα αρχείο ρυθμίσεών σχετικό με την διασύνδεση της βάσης δεδομένων με την εφαρμογή καθώς επίσης και με τις παραμέτρους του JPA. Το JPA είναι μια προγραμματιστική διεπαφή για την διαχείριση σχεσιακής βάσης δεδομένων από μία εφαρμογή γραμμένη σε γλώσσα java. Υπάρχουν πολλές βιβλιοθήκες που υλοποιούν την προγραμματιστική διεπαφή του JPA, στην παρούσα εργασία επιλέχθηκε η βιβλιοθήκη Hibernate. Στην εικόνα 3-27 παρουσιάζεται το αρχείο persistence.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             version="2.0"
             xsi:schemaLocation=" http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="primary">
    <jta-data-source>java:/MySqlDS</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="false"/>
      <property name="hibernate.transaction.flush_before_completion" value="true"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect" />
      <property name="hibernate.enable_lazy_load_no_trans" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

Εικόνα 3-29:Το αρχείο persistence.xml

Το αρχείο pom.xml είναι ένα αρχείο ρυθμίσεων το οποίο περιλαμβάνει παραμετροποιήσεις του project της διαδικτυακής πλατφόρμας. Πιο συγκεκριμένα, είναι το βασικό αρχείο για την χρησιμοποίηση των εργαλείων του maven στο project. Τα εργαλεία maven είναι εργαλεία για την αυτοματοποίηση της δόμησης του εκτελέσιμου αρχείου του project. Τα maven tools χρησιμοποιούνται κυρίως σε project πού είναι γραμμένα σε γλώσσα java. Οι σημαντικότερες παράμετροι οι οποίες δηλώνονται στο pom.xml αρχείο είναι η παράμετρος *packaging* η οποία ορίζει τον τύπο του εκτελέσιμου αρχείου το οποίο θα παραχθεί για να εγκατασταθεί στο application server και οι παράμετροι *groupId & artifactId* οι όποιες συνδυαστικά παράγουν ένα μοναδικό αναγνωριστικό για το εκτελέσιμο αρχείο που παράγεται. Τέλος, στο αρχείο pom.xml πραγματοποιείται η διαχείριση των βιβλιοθηκών που χρησιμοποιεί το project, προσθέτοντας ή αφαιρώντας μια βιβλιοθήκη στο project το maven εκτελεί όλες εκείνες τις ενέργειες για να είναι διαθέσιμη η λειτουργικότητα της στο project.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
3            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5            http://maven.apache.org/xsd/maven-4.0.0.xsd">
6       <modelVersion>4.0.0</modelVersion>
7
8       <groupId>com.anmpout</groupId>
9       <artifactId>FCDDashboard</artifactId>
10      <version>1.0</version>
11      <packaging>war</packaging>
12
13      <name>FCDDashboard</name>
14
15      <properties>
16          <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
17          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
18      </properties>
19
20      <dependencies>
21  <dependency>
22      <groupId>com.github.adminfaces</groupId>
23      <artifactId>admin-template</artifactId>
24      <version>1.0.0-RC21</version>
25  </dependency>
26
27  <dependency>
28          <groupId>org.primefaces.themes</groupId>
29          <artifactId>all-themes</artifactId>
30          <version>1.0.10</version>
31  </dependency>
32
```

Εικόνα 3-30:Τμήμα xml αρχείου

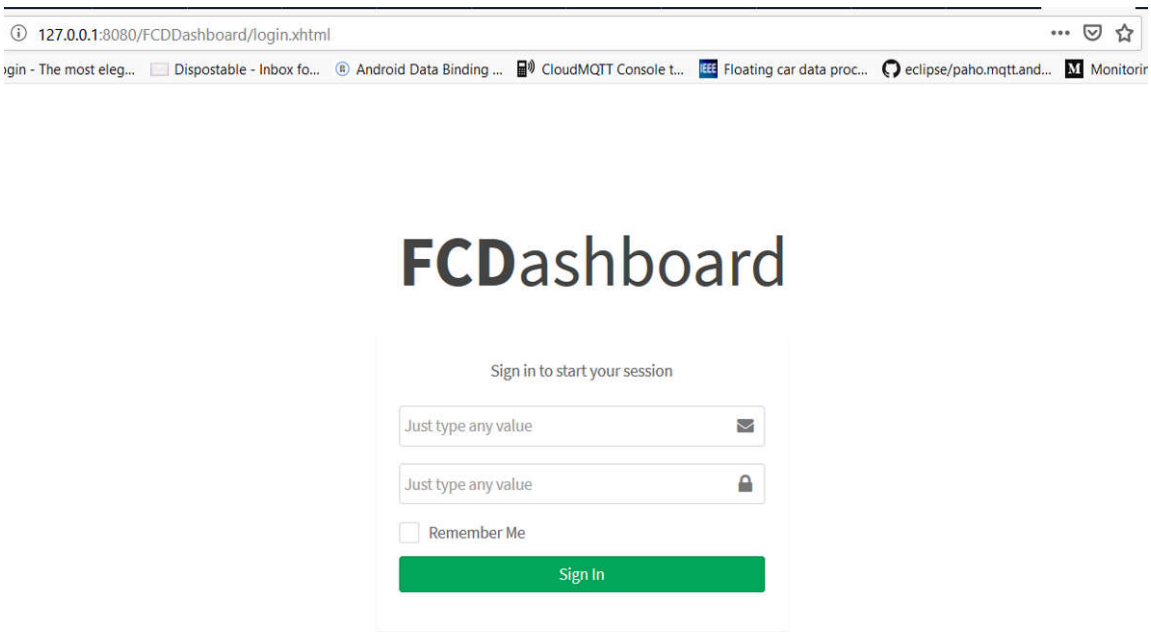
3.10 Παρουσίαση αποτελεσμάτων

Σε σχέση και με όσα παρουσιάζονται ανωτέρω αναλυτικά, γίνεται εύκολα κατανοητό ότι στόχος της εν λόγω εφαρμογής είναι η δημιουργία μιας εύχρηστης πλατφόρμας, η οποία θα είναι εύκολα προσβάσιμη και εύχρηστη για τον χρήστη, με αποτέλεσμα να μπορεί πράγματι να λάβει γνώση των αποτελεσμάτων της εφαρμογής και να τα αξιοποιήσει για τον σκοπό που επιθυμεί. Στη συνέχεια παρουσιάζεται αναλυτικά η πορεία και τρόπος παρουσίασης των αποτελεσμάτων από την εφαρμογή.

3.10.1 Δομή διαδικτυακής πύλης αποτελεσμάτων

Η δομή της διαδικτυακής πύλης που δημιουργήθηκε επιλέχθηκε να είναι απλή και εύκολα κατανοητή από τον μέσω χρήστη. Η δομή της διαδικτυακής πύλης είναι η ακόλουθη.

- Η πρώτη σελίδα της διαδικτυακής πύλης είναι η σελίδα εισόδου, όπου ο χρήσης θα πρέπει να εισάγει το όνομα χρήστη και τον κωδικό του για να εισέλθει στην πλατφόρμα. Χρησιμοποιήθηκε αυτή η σελίδα για να υπάρχει ελεγχόμενη πρόσβαση στην πλατφόρμα για το λόγο ότι η πλατφόρμα είναι ακόμα σε δοκιμαστική λειτουργία.



Εικόνα 3-31:Σελίδα εισόδου στην διαδικτυακή πύλη

- Μετά την είσοδο του χρήση με τα αναγνωριστικά του, ανακατευθύνεται στην αρχική σελίδα της διαδικτυακής πύλης. Στα αριστερά της σελίδας

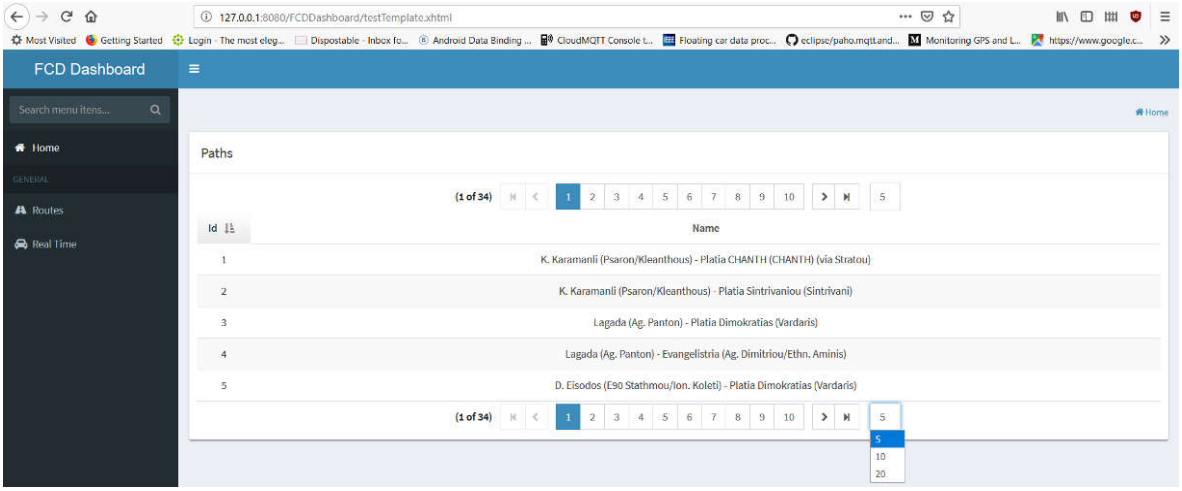
υπάρχει ένα μενού επιλογών για τις υπόλοιπες σελίδες της πλατφόρμας. Στο κεντρικό κομμάτι της διαδικτυακής πύλης φορτώνονται οι σελίδες που επιλέγει ό χρήστης της πλατφόρμας.

- Οι επιλογές που έχει ο χρήστης στο μενού της διαδικτυακής πύλης αφορούν τα στατιστικά των δεδομένων ιστορικού για τις διαδρομές, καθώς και για τα στατιστικά που υπολογίζονται για τα δεδομένα πραγματικού χρόνου. Επιπλέον, παρέχονται στον χρήστη όλες οι διαθέσιμες πληροφορίες για της διαδρομές καθώς και η απεικόνιση τους στον χάρτη.

3.10.2 Παρουσίαση αποτελεσμάτων δεδομένων ιστορικού

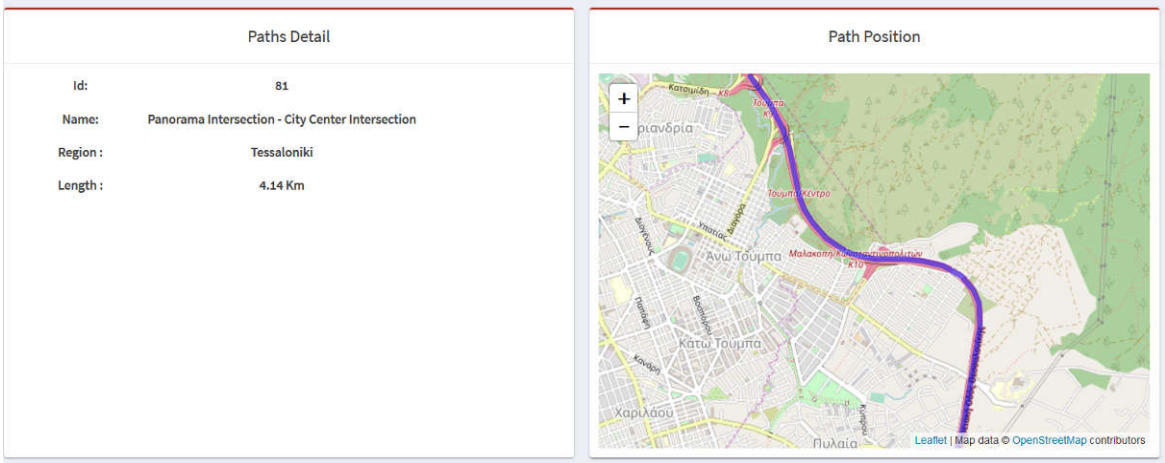
Η παρουσίαση των δεδομένων ιστορικού γίνεται με την βοήθεια 2 σελίδων που δημιουργήθηκαν για αυτόν τον σκοπό. Η πρώτη σελίδα περιέχει όλες τις διαδρομές που έχουν χρησιμοποιηθεί στην παρούσα διπλωματική εργασία. Η δεύτερη σελίδα που έχει δημιουργηθεί στο επάνω μισό μέρος της αποτελείται από πληροφορίες της διαδρομής που έχει επιλεχθεί από τον χρήστη και στο κάτω μισό της έχει τις επιλογές για τον χρήστη που μπορεί να δει τα στατιστικά για τις παραγόμενες τιμές που αναφέρονται παραπάνω είτε ανά ημέρα είτε ανά μήνα.

Επιπρόσθετα, στην πρώτη σελίδα οι διαδρομές βρίσκονται σε έναν πίνακα. Τα στοιχεία που υπάρχουν στον πίνακα για την κάθε διαδρομή είναι ένας μοναδικός κωδικός καθώς και το όνομα της διαδρομής. Ο πίνακας έχει την δυνατότητα να ταξινομεί τις διαδρομές με βάση τον μοναδικό κωδικό, είτε σε αύξουσα, είτε σε φθίνουσα σειρά. Επιπλέον, για καλύτερη διαχείριση των διαδρομών υπάρχει σελιδοποίηση στον πίνακα. Τέλος, δίδεται και η δυνατότητα επιλογής του μεγέθους της σελίδας σε 5, 10 ή 20 αντικείμενα.



Εικόνα 3-32:Πίνακας διαδρομών

Επιλέγοντας μια διαδρομή από τον πίνακα ανακατευθυνόμαστε στην δεύτερη σελίδα που περιέχει τα αποτελέσματα των ιστορικών δεδομένων, καθώς και τις πληροφορίες για την συγκεκριμένη διαδρομή.

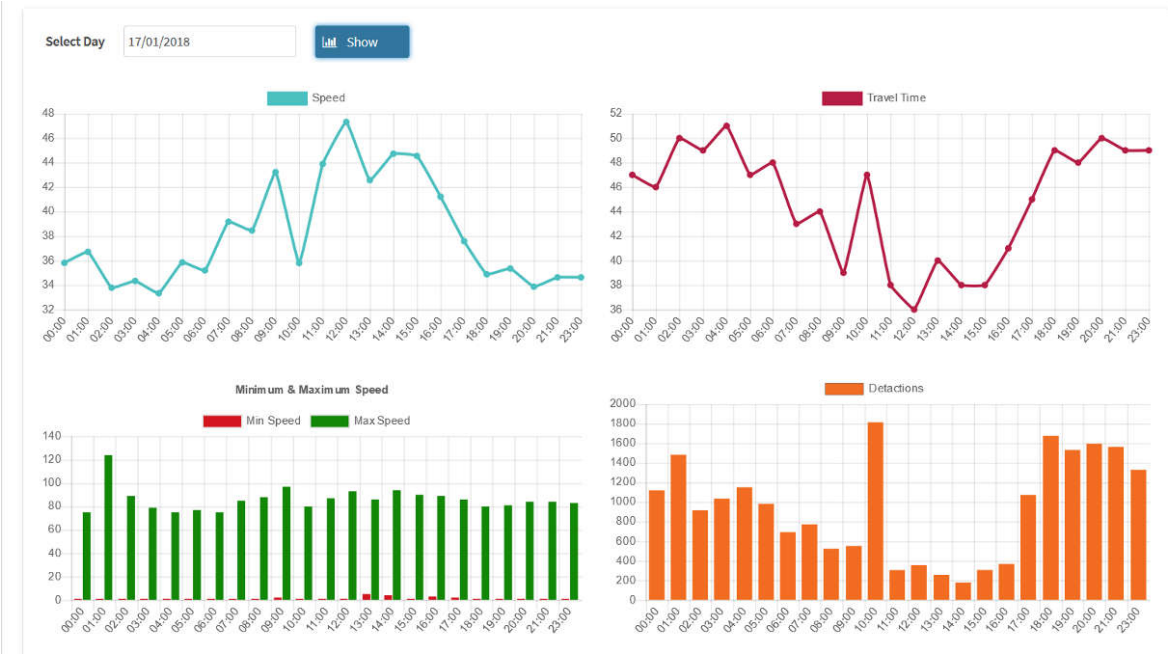


Εικόνα 3-33:Πληροφορίες και αναπαράσταση διαδρομής

Τα στατιστικά των ιστορικών δεδομένων για την διαδρομή βρίσκονται στο κάτω μέρος της οθόνης. Εκεί είναι διαθέσιμα για επιλογή τα στατιστικά ανά ημέρα και τα στατιστικά ανά μήνα.

Εάν επιλεγούν από τον χρήστη τα στατιστικά ανά ημέρα, εμφανίζεται σε αυτόν ένας επιλογέας ημερομηνίας. Επιλέγοντας την ημέρα που τον ενδιαφέρει και πατώντας το διπλανό κουμπί, εάν υπάρχουν διαθέσιμα στατιστικά εμφανίζονται με την μορφή

διαγραμμάτων, εάν δεν υπάρχουν διαθέσιμες τιμές τότε η πλατφόρμα ενημερώνει σχετικά τον χρήστη.

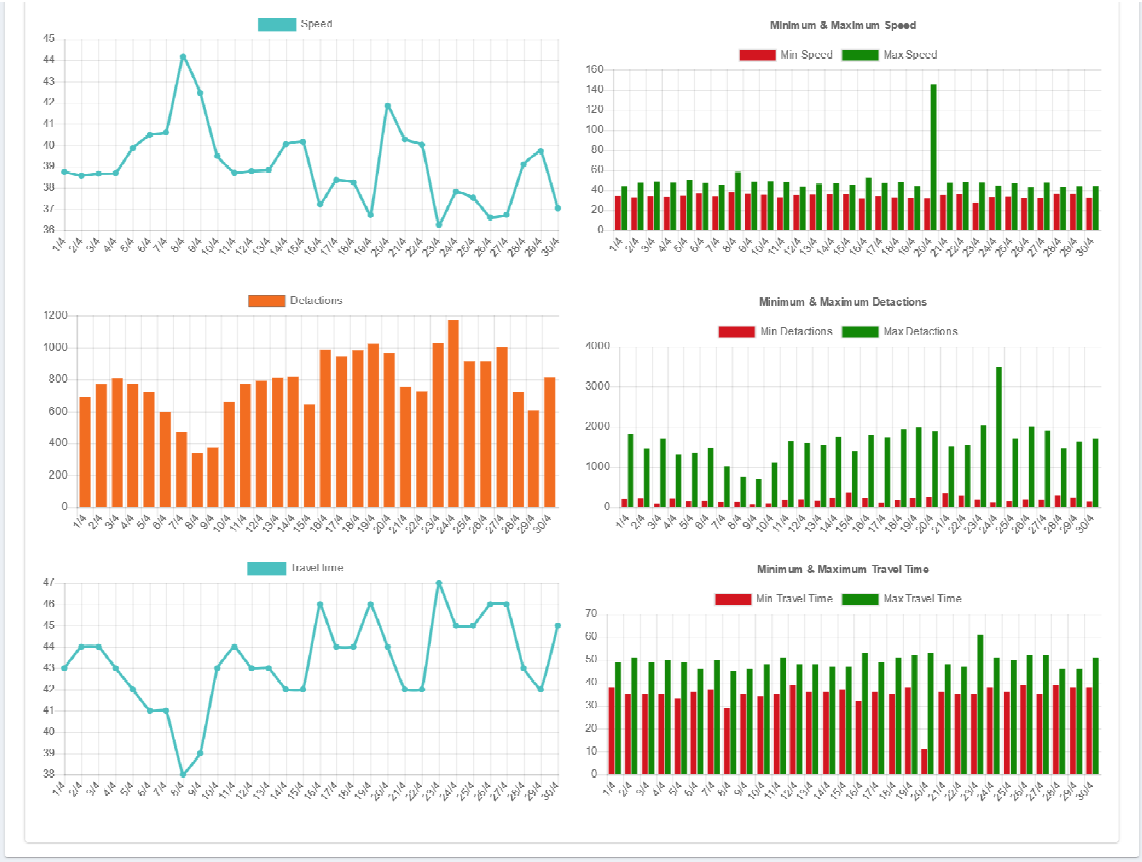


Εικόνα 3-34:Στατιστικά ημέρας δεδομένων ιστορικού

Οι πληροφορίες που δίνει η πλατφόρμα στον χρήστη για τα ιστορικά δεδομένα μιας ημέρας μέσω των διαγραμμάτων όπως φαίνεται και στην παραπάνω εικόνα είναι:

- Ο μέσος όρος των ταχυτήτων ανά ώρα. Από το διάγραμμα βλέπουμε την διακύμανση τους στο πέρασμα μιας ημέρας.
- Οι χρόνοι ταξιδιών ανά ώρα.
- Οι μέγιστές και ελάχιστες τιμές που εντοπιστήκαν ανά ώρα.
- Ο αριθμός των αντιστοιχίσεων που έγιναν για αυτήν την διαδρομή ανά ώρα.

Αντίστοιχα εάν επιλεγούν από τον χρήστη τα στατιστικά ανά μήνα εμφανίζεται ένας επιλογέας με τους διαθέσιμους μήνες. Ο χρήστης επιλέγει τον μήνα που επιθυμεί και πατάει το κουμπί για να εμφανιστούν τα αποτελέσματα εάν είναι διαθέσιμα. Εάν δεν είναι διαθέσιμα τότε εμφανίζεται ένα σχετικό ενημερωτικό μήνυμα στον χρήστη. Τα αποτελέσματα παρουσιάζονται με την μορφή διαγραμμάτων στον χρήστη και έχουν την μορφή σύμφωνα με την εικόνα 3-23



Εικόνα 3-35:Στατιστικά μηνός δεδομένων ιστορικού

Οι πληροφορίες που παρέχονται στον χρήστη είναι οι ακόλουθες

- Ο μέσος όρος της ταχύτητας ανά ημέρα
- Η ελάχιστη και η μέγιστη ταχύτητα που εντοπίστηκαν ανά ημέρα
- Τον μέσο όρο των αντιστοιχίσεων ανά ημέρα
- Η ελάχιστη και η μέγιστη τιμή των αντιστοιχίσεων ανά ημέρα
- Ο μέσος όρος των χρόνων ταξιδιού ανά ημέρα
- Η ελάχιστη και η μέγιστη τιμή χρόνου ταξιδιού ανά ημέρα

Με τον παραπάνω τρόπο παρατίθενται τα ιστορικά δεδομένα προς παρουσίαση στην παρούσα διπλωματική εργασία.

3.10.3 Παρουσίαση αποτελεσμάτων πραγματικού χρόνου

Η παρουσίαση των δεδομένων πραγματικού χρόνου μοιάζει αρκετά στη δομή της με τη παρουσίαση των δεδομένων ιστορικού. Αυτό επιλέχθηκε για να υπάρχει συνοχή

στο τρόπο που παρουσιάζονται οι πληροφορίες αλλά και για να διατηρηθεί ο απλός τρόπος που προσφέρονται οι πληροφορίες στους χρήστες.

Στην πρώτη σελίδα των αποτελεσμάτων πραγματικού χρόνου υπάρχει ένας πίνακας ίδιος με αυτόν που αναφέρεται παραπάνω, όσον αφορά τις λειτουργίες και τις δυνατότητες. Αυτό όμως που διαφέρει είναι τα περιεχόμενα του πίνακα. Ο πίνακας περιέχει μόνο τις διαδρομές για τις οποίες έχουν υπολογιστεί χρόνοι διαδρομών και τιμές για το χρονικό παράθυρο που παρουσιάζεται. Επιπλέον, στο πάνω δεξί τμήμα της σελίδας αναγράφεται η τιμή του χρονικού παραθύρου για το οποίο παρουσιάζονται τα δεδομένα. Επιλέγοντας μία διαδρομή ο χρήστης ανακατευθύνεται στην δεύτερη σελίδα των αποτελεσμάτων.

Στην δεύτερη σελίδα των αποτελεσμάτων, πάλι στο πάνω μέρος της σελίδας υπάρχουν οι πληροφορίες που αφορούν την διαδρομή που επιλέχθηκε. Στο κάτω μέρος της σελίδας υπάρχει ένας πίνακας που παρουσιάζει τα δεδομένα που υπολογίστηκαν για την τρέχουσα χρονική περίοδο, καθώς και τις τιμές που υπολογίστηκαν για τις παρόμοιες χρονικές περιόδους των προηγούμενων ετών.

4 Επίλογος

Εν κατακλείδι, στην εν λόγω διπλωματική εργασία παρουσιάστηκε το θεωρητικό υπόβαθρο, καθώς η μεθοδολογία που ακολουθήθηκε προκειμένου να υλοποιηθεί η κρίσιμη εφαρμογή, αλλά και η τελική υλοποίηση της. Όπως προαναφέρθηκε, η υλοποίηση της εφαρμογής αυτής σκοπεί στην συνεισφορά στην επίλυση καίριων ζητημάτων που σχετίζονται με τα κυκλοφοριακά φαινόμενα, όπως αυτά παρουσιάστηκαν αναλυτικά, αλλά και στην εξαγωγή γενικότερων συμπερασμάτων που μπορούν να εξαχθούν απ' όλα τα στάδια της υλοποίησης της και παρουσιάζονται κατωτέρω.

4.1 Σύνοψη και συμπεράσματα

Σύμφωνα με όλα τα ανωτέρω, αλλά και σε σχέση με το ερώτημα που τέθηκε, αρχικά καθίσταται σαφές ότι η αξιοποίηση ανοιχτών δεδομένων μεγάλου όγκου σε συνδυασμό με τα εργαλεία ανοιχτού κώδικα είναι εφικτή και πραγματοποιήσιμη, όπως άλλωστε αποδεικνύεται από την υλοποίηση της εν λόγω εφαρμογής. Είναι προφανές, ότι

μέσω της εφαρμογής αυτής καθίσταται πράγματι τελικά εφικτή η ανάλυση κυκλοφοριακών δεδομένων, τόσο πραγματικού, όσο και παρελθοντικού χρόνου, μιας οποιασδήποτε πόλης, και στη συγκεκριμένη περίπτωση της πόλης της Θεσσαλονίκης, δίνοντας τη δυνατότητα δημιουργίας κυκλοφοριακών προφίλ τα οποία μπορούν να φανούν χρήσιμα για την εξαγωγή πολύτιμων συμπερασμάτων.

Περαιτέρω, αξίζει να σημειωθεί ότι η σωστή χρήση και αξιοποίηση των εξαγόμενων από την εφαρμογή αυτή πληροφοριών από τους μετακινούμενους αυτοκινητιστές θα μπορούσε να βοηθήσει στην επίλυση πολλών καίριων ζητημάτων απ' αυτά που αναλύθηκαν ανωτέρω. Για παράδειγμα, με τη χρήση των πληροφοριών αυτών θα μπορούσαν οι μετακινούμενοι να ενημερώνονται εκ των προτέρων για τα σημεία στα οποία υπάρχει κυκλοφοριακή συμφόρηση, οδηγούμενοι στο συμπέρασμα αυτό στην ουσία από τον χρόνο της διαδρομής, με αποτέλεσμα να είναι σε θέση να επιλέξουν κάποια εναλλακτική διαδρομή, εξοικονομώντας πολύτιμο χρόνο, αλλά και χρήματα από τα καύσιμα και συνεισφέροντας συγχρόνως στην αποσυμφόρηση των προβληματικών διαδρομών, αλλά και στην επίλυση άλλων ζητημάτων, όπως τα περιβαλλοντικά ζητήματα, τα οποία όπως είναι γνωστό επιβαρύνονται ιδιαίτερα από τα καυσαέρια που παράγονται από τα οχήματα.

Επίσης, είναι σημαντικό να αναφερθεί ότι μέσω της αξιοποίησης των κυκλοφοριακών προφίλ που δημιουργεί και διατηρεί η εφαρμογή, οι χρήστες της μπορούν εύκολα και γρήγορα να ενημερώνονται για το αν κάποιοι δρόμοι της πόλης παραμένουν «κλειστοί», ήτοι αν έχει διακοπεί προσωρινά η κυκλοφορία σ' αυτούς εξαιτίας κάποιου τυχαίου ή προγραμματισμένου γεγονότος. Η πληροφορία αυτή είναι ιδιαίτερης σημασίας δεδομένου ότι αποτελεί σχεδόν καθημερινό φαινόμενο στις μεγάλες πόλεις να δημιουργείται κυκλοφοριακό κομφούζιο από την αιφνίδια διακοπή της κυκλοφορίας σε κάποιον από τους δρόμους της.

Εν συνεχεία, θα πρέπει αν αναφερθούν και κάποια αρνητικά συμπεράσματα τα οποία εξάγονται από την εκπόνηση της συγκεκριμένης διπλωματικής εργασίας. Πιο αναλυτικά, δεν θα μπορούσε να μην υπογραμμιστεί το γεγονός ότι η λειτουργία της εν λόγω εφαρμογής είναι άρρηκτα εξαρτώμενη, όπως άλλωστε προαναφέρθηκε, από τις πληροφορίες που λαμβάνει από μία μόνο πηγή, ήτοι από το Ινστιτούτο Βιώσιμης Κινητικότητας & Δικτύων Μεταφορών (Ι.ΜΕΤ), μέλος του Εθνικού Κέντρου Έρευνας και Τεχνολογικής Ανάπτυξης (Ε.Κ.Ε.Τ.Α). Αυτό συνεπάγεται το αυτονόητο συμπέρασμα ότι αν για κάποιο λόγο υπάρξει κάποια δυσλειτουργία στην υποδομή του

οργανισμού που παρέχει τα δεδομένα και σταματήσει η ροή αυτών ή ακόμα περισσότερο αν τα δεδομένα σταματήσουν να συλλέγονται από τον οργανισμό, θα υπάρξει αυτόματη αδυναμία της εφαρμογής να εξάγει τα σχετικά αποτελέσματα, όσον αφορά πάντα τα αποτελέσματα πραγματικού χρόνου, δεδομένου ότι δεν υπάρχει το ίδιο πρόβλημα και για τα αποτελέσματα που αφορούν τον παρελθόντα χρόνο.

4.2 Μελλοντικές Επεκτάσεις

Όπως αναφέρθηκε ανωτέρω, η εφαρμογή επί του παρόντος λαμβάνει δεδομένα μόνο από μία πηγή. Ως εκ τούτου, μία από τις σημαντικότερες μελλοντικές επεκτάσεις της εν λόγω εφαρμογής αποτελεί η αύξηση των πηγών από τις οποίες λαμβάνονται τα κρίσιμα δεδομένα με σκοπό να βελτιωθεί ακόμα περισσότερο η ποιότητα και η αξιοπιστία των παραγόμενων αποτελεσμάτων, αλλά και να μην διακόπτεται η αποτελεσματική λειτουργία της εφαρμογής σε περίπτωση που η πηγή των δεδομένων αντιμετωπίσει οποιοδήποτε πρόβλημα ως προς τη συλλογή τους.

Επιπλέον, δεν μπορεί να παραβλεφθεί το γεγονός ότι κάποιες φορές παρατηρούνται ακραίες τιμές των δεδομένων που λαμβάνονται και κατά συνέπεια και των εξαγόμενων αποτελεσμάτων. Αυτό οφείλεται στη προσωρινή δυσλειτουργία κάποιων συσκευών απ' αυτές που αποτελούν μέρος του στόλου από τον οποίο λαμβάνονται τα αρχικά δεδομένα κινούμενων οχημάτων. Ως εκ τούτου, καθίσταται σαφές ότι σε περίπτωση που τα δεδομένα που καλείται να επεξεργαστεί η εφαρμογή δεν είναι ακριβή, είναι δυνατόν να προκύψουν λανθασμένα αποτελέσματα. Το πρόβλημα αυτό βέβαια θα βελτιωνόταν δραστικά στην περίπτωση που στα δεδομένα που λαμβάνει η εφαρμογή υπήρχε και το μοναδικό αναγνωριστικό νούμερο της κάθε συσκευής (id), καθώς στην περίπτωση αυτή θα ήταν δυνατό να ομαδοποιηθούν οι εγγραφές εισόδου ανά μοναδικό αναγνωριστικό νούμερο, να ελεγχθεί η ορθότητά τους ως προς τις τιμές που λαμβάνονται, καθώς και ο αριθμός των εγγραφών που λήφθηκαν ανά συσκευή. Έτσι, αν εντοπιζόταν κάποια ανωμαλία είτε στις τιμές που λήφθηκαν, όπως για παράδειγμα κάποια ακραία ταχύτητα, είτε στο μέγεθος των εγγραφών για κάποια συσκευή, τότε οι εγγραφές αυτές θα αποκλείονταν από το δείγμα για τη συγκεκριμένη χρονική περίοδο και ως εκ τούτου τα εξαγόμενα αποτελέσματα θα ήταν πιο αξιόπιστα.

Ακόμα, αξίζει να σημειωθεί ότι στην συγκεκριμένη εφαρμογή θα μπορούσαν να εφαρμοστούν δεδομένα από οποιαδήποτε πόλη και να εξαχθούν τα αντίστοιχα αποτελέσματα. Έτσι, καθίσταται σαφές ότι μέσω της χρήσης της εφαρμογής αυτής θα ήταν δυνατή η εκπόνηση και συγκριτικών μελετών ως προς τα κυκλοφοριακά δεδομένα αλλά και τα κυκλοφοριακά προφίλ που παρουσιάζουν περισσότερες πόλεις ή ακόμα περισσότερο και ως προς την αξιοπιστία των αποτελεσμάτων σε σχέση με τις λαμβανόμενα δεδομένα, ήτοι τόσο την ποιότητα τους, όσο και των αριθμό των πηγών ή και οτιδήποτε άλλο.

Παράρτημα Α - Διάφορα

Α.1 Υλοποίηση διαδικασίας συλλογής δεδομένων πραγματικού χρόνου

RealTimeGrabber.java

```
package com.anmpout.realtimemapreduce;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.hadoop.util.ToolRunner;
import org.json.JSONArray;
import org.json.JSONObject;

public class RealTimeGrabber {
   private static int MINUTES = 1; // The delay in minutes
   private final String USER_AGENT = "Mozilla/5.0";

   public static void main(String[] args) throws Exception {
      RealTimeGrabber grabber = new RealTimeGrabber();

      Timer timer = new Timer();
      timer.schedule(new TimerTask() {
         @Override
         public void run() {
            try {
               grabber.callRealtimeData();
            } catch (Exception ex) {
               ex.printStackTrace();
               Logger.getLogger(RealTimeGrabber.class.getName()).log(Level.SEVERE, null, ex);
            }
         }
      }, 0, 1000 * 60 * MINUTES);

   }

   private void callRealtimeData() throws Exception {
      String url = "http://feed.opendata.imet.gr:23577/fcd/gps.json?offset=0&limit=-1";
      URL obj = new URL(url);
```

```
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();
    // optional default is GET
    con.setRequestMethod("GET");
    //add request header
    con.setRequestProperty("User-Agent", USER_AGENT);
    int responseCode = con.getResponseCode();
    BufferedReader in = new BufferedReader(
            new InputStreamReader(con.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();
    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();
    parseResposeAndSaveData(response.toString());
}

private void parseResposeAndSaveData(String toString) throws IOException {
    long unixTimestamp = System.currentTimeMillis() / 1000L;
    long bucket = unixTimestamp - (unixTimestamp % 3600);
    FileWriter writer = null;
    JSONArray jSONArray = new JSONArray(toString);
    for (int i = 0; i < jSONArray.length(); i++) {
        JSONObject jSONObject = new JSONObject(jSONArray.get(i).toString());
        List<String> rowElements = new ArrayList<>();

        rowElements.add(jSONObject.optString("recorded_timestamp", ""));
        rowElements.add(jSONObject.optString("lon", ""));
        rowElements.add(jSONObject.optString("lat", ""));
        rowElements.add(jSONObject.optString("altitude", ""));
        rowElements.add(jSONObject.optString("speed", ""));
        rowElements.add(jSONObject.optString("orientation", ""));
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
        Date date;
        long unixTime = 0;

        try {
            date = dateFormat.parse(jSONObject.optString("recorded_timestamp", ""));
        } catch (Exception ex) {
            continue;

        }
        String dateDir = createDirectoryOptional(date);
        String csvFile = "/home/cloudera/Downloads/realTime/" + dateDir;
        writer = new FileWriter(csvFile, true);
        CSVUtils.writeLine(writer, rowElements);
        writer.flush();
    }

    writer.close();
}

long unixTimestamp = System.currentTimeMillis() / 1000L;

private String createDirectoryOptional(Date myDate) {
    String dateDir = new SimpleDateFormat("yyyy-MM-dd").format(myDate);

    String directoryName = "/home/cloudera/Downloads/realTime/" + dateDir;
    File directory = new File(directoryName);
```

```
        if (!directory.exists()) {
            directory.mkdir();
        }
        return dateDir + "/" + new SimpleDateFormat("H").format(myDate) + "/";
    }
}
```

CSVUtils.java

```
package com.anmpout.realtimemapreduce;

import java.io.IOException;
import java.io.Writer;
import java.util.List;

public class CSVUtils {

private static final char DS= '\t';
      /**
     * Writes line to  output file
     * @param w
     * @param values
     * @throws IOException
     */
public static void writeLine(Writer w, List<String> values) throws IOException {
        writeLine(w, values, DS, ' ');
    }

 private static String followformat(String value) {

        String result = value;
        if (result.contains("\"")) {
            result = result.replace("\"", "\"\"");
        }
        return result;

    }

 public static void writeLine(Writer w, List<String> values, char separators, char customQuote) throws
IOException {

        boolean first = true;
        if (separators == ' ') {
            separators = DS;
        }

        StringBuilder sb = new StringBuilder();
        for (String value : values) {
            if (!first) {
                sb.append(separators);
            }
            if (customQuote == ' ') {
                sb.append(followformat(value));
            } else {
                sb.append(customQuote).append(followformat(value)).append(customQuote);
            }

            first = false;
```

```
        }
        sb.append("\n");
        w.append(sb.toString());

    }
}
```

A.2 Υλοποίηση διαδικασίας αποθήκευσης δεδομένων διαδρομής

MyPath.java

```
package com.mycompany.awstest;
import com.spatial4j.core.context.SpatialContext;
import com.spatial4j.core.distance.DistanceUtils;
import com.spatial4j.core.shape.impl.BufferedLineString;
import com.spatial4j.core.shape.impl.PointImpl;
import java.util.ArrayList;
import java.util.List;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class MyPath {

private String pathId;
private String pathName;
private String pathOrignDeviceId;
private String pathDestinationDeviceId;
private List<Point> polyline;
private BufferedLineString  polylinePoints;
private Double distance;
public MyPath() {
}
public static MyPath fromJSON(String jsonString) {
if (jsonString == null) {
return null;
}
MyPath returnPath = new MyPath();
try {
JSONObject jSONObject = new JSONObject(jsonString);
if (jSONObject.has("Path_id")
&& jSONObject.optString("Path_id") != null) {
returnPath.setPathId(jSONObject.optString("Path_id"));
}
if (jSONObject.has("Path_Name")
&& jSONObject.optString("Path_Name") != null) {
returnPath.setPathName(jSONObject.optString("Path_Name"));
}
if (jSONObject.has("Path_origin_device_id")
&& jSONObject.optString("Path_origin_device_id") != null) {
returnPath.setPathOrignDeviceId(
jSONObject.optString("Path_origin_device_id"));
}
if (jSONObject.has("Path_destination_device_id")
&& jSONObject.optString("Path_destination_device_id") != null) {
returnPath.setPathDestinationDeviceId(
jSONObject.optString("Path_destination_device_id"));
}
```

```
if(jSONObject.has("polyline")){
returnPath.setPolyline(createPolyline(jSONObject.optString("polyline")));
returnPath.setPolylinePoints(createPolyline2(jSONObject.optString("polyline")));
returnPath.setDistance(calculateDistance(returnPath.getPolylinePoints()));
}
} catch (JSONException ex) {
ex.printStackTrace();
}
return returnPath;
}
public static List<MyPath> listfromJSON(String jsonString) {
List<MyPath> returnList = new ArrayList<>();
try {
JSONArray JSONpaths = new JSONArray(jsonString);
for (int i = 0; i < JSONpaths.length(); i++) {
MyPath tmpPath = MyPath.fromJSON(JSONpaths.get(i).toString());
if (tmpPath != null) {
returnList.add(tmpPath);
}
}
} catch (JSONException ex) {
ex.printStackTrace();
}
return returnList;
}
private static List<Point> createPolyline(String jsonString) {
if (jsonString == null) {
return null;
}
List<Point> pointsList = new ArrayList<>();
try {
String[] pointsArray = jsonString.split("\\s+");
for (String point : pointsArray) {
String[] Latlon =  point.split(",");
Point tmp = new Point(Double.parseDouble(Latlon[0]),
Double.parseDouble(Latlon[1]));
pointsList.add(tmp);
}
} catch (JSONException ex) {
ex.printStackTrace();
}
return pointsList;
}
private static BufferedLineString createPolyline2(String jsonString) {
if (jsonString == null) {
return null;
}
BufferedLineString  ls = null;
List<com.spatial4j.core.shape.Point> pointsList = new ArrayList<>();
try {
String[] pointsArray = jsonString.split("\\s+");
for (String point : pointsArray) {
String[] Latlon =  point.split(",");
PointImpl tmp = new PointImpl(Double.parseDouble(Latlon[0]),
Double.parseDouble(Latlon[1]), SpatialContext.GEO);
pointsList.add(tmp);
}
ls = new BufferedLineString(pointsList, 0.001, SpatialContext.GEO);
} catch (JSONException ex) {
```

```
ex.printStackTrace();
}
return ls;
}
public String getPathId() {
return pathId;
}
public void setPathId(String pathId) {
this.pathId = pathId;
}
public String getPathName() {
return pathName;
}
public void setPathName(String pathName) {
this.pathName = pathName;
}
public String getPathOrignDeviceId() {
return pathOrignDeviceId;
}
public void setPathOrignDeviceId(String pathOrignDeviceId) {
this.pathOrignDeviceId = pathOrignDeviceId;
}
public String getPathDestinationDeviceId() {
return pathDestinationDeviceId;
}
public void setPathDestinationDeviceId(String pathDestinationDeviceId) {
this.pathDestinationDeviceId = pathDestinationDeviceId;
}
public List<Point> getPolyline() {
return polyline;
}
public void setPolyline(List<Point> polyline) {
this.polyline = polyline;
}
public Double getDistance() {
return distance;
}
public void setDistance(Double distance) {
this.distance = distance;
}
public BufferedLineString getPolylinePoints() {
return polylinePoints;
}
public void setPolylinePoints(BufferedLineString polylinePoints) {
this.polylinePoints = polylinePoints;
}
@Override
public String toString() {
return "MyPath{" + "pathId=" + pathId + ", pathName=" + pathName + ","
+ " pathOrignDeviceId=" + pathOrignDeviceId + ","
+ " pathDestinationDeviceId=" + pathDestinationDeviceId + ","
+ " polyline=" + polyline + '}';
}
private static Double calculateDistance(BufferedLineString polyline) {
double returnDistance=0;
for(int i =0;i<polyline.getPoints().size()-1;i++){
returnDistance = returnDistance +
DistanceUtils.degrees2Dist(SpatialContext.GEO.getDistCalc()
.distance(polyline.getPoints().get(i),
```

```
polyline.getPoints().get(i+1)),
DistanceUtils.EARTH_MEAN_RADIUS_KM) * 1000;
}
return returnDistance;
}
}
```

Point.java

```
package com.mycompany.awstest;
public class Point {
private double latitude;
private double longitude;
Point(double latitude, double longitude) {
this.latitude = latitude;
this.longitude = longitude;

}

public double getLatitude() {
return latitude;
}

public void setLatitude(double latitude) {
this.latitude = latitude;
}

public double getLongitude() {
return longitude;
}

public void setLongitude(double longitude) {
this.longitude = longitude;
}

}
```

DBConnector.java

```
package com.mycompany.awstest;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;
public class DBConnector {
public static void main(String[] args) throws ClassNotFoundException, SQLException, Exception{
Class.forName("com.mysql.cj.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://geofilterdb.c4nkehdtywwc.us-east-2.rds.amazonaws.com:3306/geofilterdb",
```

```
"cloudera","cloudera");
String data = readFileAsString("C:\\Users\\anmpout\\Downloads\\paths.txt");
List<MyPath> myPaths = MyPath.listfromJSON(data);
String query = " insert into PATH (PATH_ID,PATH_NAME ,PATH_DEST_DEV_ID,"
+ " PATH_ORIGIN_DEV_ID, PATH_DISTANCE)"
+ " values (?, ?, ?, ?, ?)";

PreparedStatement preparedStmt = con.prepareStatement(query);
String query2 = " insert into POINT (LATITUDE,LONGITUDE, PATH_ID)"
+ " values (?, ?, ?)";
PreparedStatement preparedStmt2 = con.prepareStatement(query2);
for (MyPath myPath : myPaths) {
for(Point point: myPath.getPolyline()){
preparedStmt2.setDouble(1, point.getLatitude());
preparedStmt2.setDouble(2, point.getLongitude());
preparedStmt2.setInt(3, Integer.parseInt(myPath.getPathId()));
preparedStmt2.execute();
}
}
}
public static String readFileAsString(String fileName)throws Exception
{
String data = "";
data = new String(Files.readAllBytes(Paths.get(fileName)));
return data;
}
}
```

A.3 Υλοποίηση διαδικασίας MapReduce και αποθήκευσης αποτελεσμάτων

GeoValue.java

```
package com.anmpout.geomapreducejob;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.*;

public class GeoValue implements WritableComparable {
private Text timestamp;
private DoubleWritable latitude;
private DoubleWritable longitude;
private DoubleWritable altitude;
private DoubleWritable speed;
private DoubleWritable orientation;
private LongWritable unixTimestamp;

public GeoValue() {
this.timestamp = null;
this.latitude = null;
this.longitude = null;
this.altitude = null;
this.speed = null;
this.orientation = null;
```

```
this.unixTimestamp = null;
}

public GeoValue(Text timestamp, DoubleWritable latitude,
DoubleWritable longitude,DoubleWritable altitude, DoubleWritable speed,
DoubleWritable orientation,LongWritable unixTimestamp) {
this.timestamp = timestamp;
this.latitude = latitude;
this.longitude = longitude;
this.altitude = altitude;
this.speed = speed;
this.orientation = orientation;
this.unixTimestamp = unixTimestamp;
}

public Text getTimestamp() {
return timestamp;
}

public void setTimestamp(Text timestamp) {
this.timestamp = timestamp;
}

public DoubleWritable getLatitude() {
return latitude;
}

public void setLatitude(DoubleWritable latitude) {
this.latitude = latitude;
}

public DoubleWritable getLongitude() {
return longitude;
}

public void setLongitude(DoubleWritable longitude) {
this.longitude = longitude;
}

public DoubleWritable getAltitude() {
return altitude;
}

public void setAltitude(DoubleWritable altitude) {
this.altitude = altitude;
}

public DoubleWritable getSpeed() {
return speed;
}

public void setSpeed(DoubleWritable speed) {
this.speed = speed;
}

public DoubleWritable getOrientation() {
return orientation;
}
```

```
public void setOrientation(DoubleWritable orientation) {
this.orientation = orientation;
}

public LongWritable getUnixTimestamp() {
return unixTimestamp;
}

public void setUnixTimestamp(LongWritable unixTimestamp) {
this.unixTimestamp = unixTimestamp;
}
@Override
public void write(DataOutput d) throws IOException {
timestamp.write(d);
longitude.write(d);
latitude.write(d);
altitude.write(d);
speed.write(d);
orientation.write(d);
unixTimestamp.write(d);
}

@Override
public void readFields(DataInput di) throws IOException {
if (timestamp == null) {
timestamp = new Text();
}
if (latitude == null) {
latitude = new DoubleWritable();
}
if (longitude == null) {
longitude = new DoubleWritable();
}
if (altitude == null) {
altitude = new DoubleWritable();
}
if (speed == null) {
speed = new DoubleWritable();
}
if (orientation == null) {
orientation = new DoubleWritable();
}
if (unixTimestamp == null) {
unixTimestamp = new LongWritable();
}
timestamp.readFields(di);
latitude.readFields(di);
longitude.readFields(di);
altitude.readFields(di);
speed.readFields(di);
orientation.readFields(di);
unixTimestamp.readFields(di);
}

@Override
public int compareTo(Object o) {
GeoValue other = (GeoValue)o;
int cmp = timestamp.compareTo(other.timestamp);
if (cmp != 0) {
```

```
return cmp;
}
cmp = latitude.compareTo(other.latitude);
if (cmp != 0) {
return cmp;
}
cmp = longitude.compareTo(other.longitude);
if (cmp != 0) {
return cmp;
}
cmp = altitude.compareTo(other.altitude);
if (cmp != 0) {
return cmp;
}
cmp = speed.compareTo(other.speed);
if (cmp != 0) {
return cmp;
}
cmp = unixTimestamp.compareTo(other.unixTimestamp);
if (cmp != 0) {
return cmp;
}
return orientation.compareTo(other.orientation);
}
}
```

GeoRecordReader.java

```
package com.anmpout.geomapreducejob;

import com.spatial4j.core.context.SpatialContext;
import com.spatial4j.core.shape.*;
import com.spatial4j.core.shape.impl.*;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.logging.Level;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.LineRecordReader;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.log4j.Logger;

public class GeoRecordReader extends RecordReader<GeoKey, GeoValue> {

private static final Logger LOG = Logger.getLogger(GeoRecordReader.class);

private GeoKey key;
```

```
private GeoValue value;
private LineRecordReader reader = new LineRecordReader();
List<MyPath> myPaths = new ArrayList<MyPath>();

public GeoRecordReader() {

}

@Override
public void initialize(InputSplit is, TaskAttemptContext tac)
throws IOException,InterruptedException {

reader.initialize(is, tac);
try {
String data = readFileAsString("/home/cloudera/Downloads/paths");
myPaths = MyPath.listfromJSON(data);
} catch (Exception ex) {
LOG.debug(ex.toString());
}

}

@Override
public boolean nextKeyValue() throws IOException,
 InterruptedException {

boolean gotNextKeyValue = reader.nextKeyValue();
if (gotNextKeyValue) {
if (key == null) {
key = new GeoKey();
}
if (value == null) {
value = new GeoValue();
}

Text line = reader.getCurrentValue();
LOG.debug(line.toString());
String[] tokens = line.toString().split("\t");
PointImpl tmp = new PointImpl(Double.
parseDouble(tokens[1]), Double.
parseDouble(tokens[2]), SpatialContext.GEO);

for (MyPath path : myPaths) {
SpatialRelation relation = path.getPolyline().relate(tmp);
if (relation.equals(SpatialRelation.CONTAINS)) {
key.setLocation(new Text(path.getPathId()));
key.setLatitude(new DoubleWritable(path.getPolyline()
.getPoints().get(0).getX()));
key.setLongitude(new DoubleWritable(path.getPolyline()
.getPoints().get(0).getY()));
key.setDistance(new DoubleWritable(path.getDistance()));
value.setTimestamp(new Text(tokens[0]));
DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
Date date;
long unixTime = 0;
try {
date = dateFormat.parse(tokens[0]);
unixTime = (long) date.getTime() / 1000;
```

```
} catch (ParseException ex) {

java.util.logging.Logger.getLogger(GeoRecordReader.class.getName())
.log(Level.SEVERE, null, ex);
}
value.setUnixTimestamp(new LongWritable(unixTime));
value.setLatitude(new DoubleWritable(Double.
parseDouble(tokens[2])));
value.setLongitude(new DoubleWritable(Double.
parseDouble(tokens[1])));

value.setAltitude(new DoubleWritable(Double.
parseDouble(tokens[3])));
value.setSpeed(new DoubleWritable(Double.
parseDouble(tokens[4])));
value.setOrientation(new DoubleWritable(Double.
parseDouble(tokens[5])));
}
}
} else {
key = null;
value = null;
}
return gotNextKeyValue;
}

@Override
public GeoKey getCurrentKey() throws IOException,
 InterruptedException {
return key;
}

@Override
public GeoValue getCurrentValue() throws IOException,
 InterruptedException {
return value;
}

@Override
public float getProgress() throws IOException,
 InterruptedException {
return reader.getProgress();
}

@Override
public void close() throws IOException {
reader.close();
}

public static String readFileAsString(String fileName)
 throws Exception {
String data = "";
data = new String(Files.readAllBytes(Paths.get(fileName)));
return data;
}

}
```

GeoKeyMap.java

```
package com.anmpout.geomapreducejob;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.*;

public class GeoKeyMap implements WritableComparable{
private GeoKey key;
private LongWritable timeslot;

public GeoKeyMap(GeoKey key, LongWritable timeslot) {
this.key = key;
this.timeslot = timeslot;
}

public GeoKeyMap() {
key= null;
timeslot = null;
}

public GeoKey getKey() {
return key;
}

public void setKey(GeoKey key) {
this.key = key;
}

public LongWritable getTimeslot() {
return timeslot;
}

public void setTimeslot(LongWritable timeslot) {
this.timeslot = timeslot;
}
@Override
public void write(DataOutput d) throws IOException {
key.write(d);
timeslot.write(d);
}

@Override
public void readFields(DataInput di) throws IOException {

if (key == null) {
key = new GeoKey();
}
if (timeslot == null) {
timeslot = new LongWritable();
}

key.readFields(di);
timeslot.readFields(di);

}
```

```
@Override
public int compareTo(Object o) {
GeoKeyMap other = (GeoKeyMap)o;
int cmp = key.compareTo(other.key);
if (cmp != 0) {
return cmp;
}
return timeslot.compareTo(other.timeslot);
}
}
```

GeoKey.java

```
package com.anmpout.geomapreducejob;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.*;

public class GeoKey implements WritableComparable{

private Text location;
private DoubleWritable latitude;
private DoubleWritable longitude;
private DoubleWritable distance;
public GeoKey() {
location = null;
latitude = null;
longitude = null;
distance = null;
}

public GeoKey(Text location, DoubleWritable latitude,
DoubleWritable longitude,DoubleWritable distance) {
this.location = location;
this.latitude = latitude;
this.longitude = longitude;
this.distance = distance;
}



@Override
public void write(DataOutput d) throws IOException {
location.write(d);
latitude.write(d);
longitude.write(d);
distance.write(d);
}

@Override
public void readFields(DataInput di) throws IOException {

if (location == null) {
```

```
location = new Text();
}
if (latitude == null) {
latitude = new DoubleWritable();
}
if (longitude == null) {
longitude = new DoubleWritable();
}
if (distance == null) {
distance = new DoubleWritable();
}
location.readFields(di);
latitude.readFields(di);
longitude.readFields(di);
distance.readFields(di);

}

@Override
public int compareTo(Object o) {
GeoKey other = (GeoKey)o;
int cmp = location.compareTo(other.location);
if (cmp != 0) {
return cmp;
}
cmp = distance.compareTo(other.distance);
if (cmp != 0) {
return cmp;
}
cmp = latitude.compareTo(other.latitude);
if (cmp != 0) {
return cmp;
}
return longitude.compareTo(other.longitude);

}

public Text getLocation() {
return location;
}

public void setLocation(Text location) {
this.location = location;
}

public DoubleWritable getLatitude() {
return latitude;
}

public void setLatitude(DoubleWritable latitude) {
this.latitude = latitude;
}

public DoubleWritable getLongitude() {
return longitude;
}

public void setLongitude(DoubleWritable longitude) {
this.longitude = longitude;
```

```
}

public DoubleWritable getDistance() {
return distance;
}

public void setDistance(DoubleWritable distance) {
this.distance = distance;
}
}
```

GeoInputFormat.java

```
package com.anmpout.geomapreducejob;

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.CompressionCodecFactory;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class GeoInputFormat extends FileInputFormat<GeoKey,GeoValue> {
   @Override
   public RecordReader<GeoKey, GeoValue> createRecordReader(InputSplit is,
           TaskAttemptContext tac)
           throws IOException, InterruptedException {
      return new GeoRecordReader();
   }
   @Override
   protected boolean isSplitable(JobContext context, Path filename) {
   CompressionCodec codec =
   new CompressionCodecFactory(context.
   getConfiguration()).getCodec(filename);
   return codec == null;
   }

}
```

GeoFilterReducer.java

```
package com.anmpout.geomapreducejob;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class GeoFilterReducer extends Reducer<GeoKeyMap,
GeoValue, Text, Text> {

@Override
protected void reduce(GeoKeyMap keyMap,
java.lang.Iterable<GeoValue> values, Context context)
 throws IOException, InterruptedException {
double speed = 0;
int count = 0;
int time = 0;
String timeString ="";
List<Double> allSpeeds = new ArrayList<>();


for (GeoValue value : values) {
speed = value.getSpeed().get() + speed;
count = count+1;
allSpeeds.add(value.getSpeed().get());
}

Double max = max(allSpeeds);
Double min = min(allSpeeds);
Double median = getMedian(allSpeeds);
speed= speed/count;

if(speed != 0){
time = (int) (keyMap.getKey()
.getDistance().get()/convertSpeedFromKMHTOMS(speed));
}

long timestamp = keyMap.getTimeslot().get()*1000;
Date d = new Date(timestamp);
Calendar cal = Calendar.getInstance();
cal.setTime(d);
int year = cal.get(Calendar.YEAR);
int month = cal.get(Calendar.MONTH)+1;
int day = cal.get(Calendar.DAY_OF_MONTH);

context.write(keyMap.getKey().getLocation(),
new Text(String.valueOf(keyMap.getTimeslot())+"\t"+
String.valueOf(count)+"\t"+
String.valueOf(speed)+"\t"+String.valueOf(time)+"\t"+
String.valueOf(keyMap.getKey().getDistance().get())+"\t"+
String.valueOf(max)+"\t"+String.valueOf(min)+"\t"+
String.valueOf(median)+"\t"+
String.valueOf(day)+"\t"+String.valueOf(month)+"\t"+
String.valueOf(year)));

}

public Double min(List<Double> array) {
if(array.isEmpty()) return 0.0;
Double min = array.get(0);
for(int i=0; i<array.size(); i++) {
if(array.get(i).compareTo(min)<0) {
min = array.get(i);
```

```
}
}
return min;
}
public Double getMedian(List<Double> array) {
if(array.isEmpty()) return  0.0;
Collections.sort(array);
int middle = array.size() / 2;
middle = middle > 0 && middle % 2 == 0 ? middle - 1 : middle;
return array.get(middle);
}
public Double max(List<Double> array) {
if(array.isEmpty()) return 0.0;
Double max = array.get(0);
for(int i=0; i<array.size(); i++) {
if(array.get(i).compareTo(max)>0) {
max = array.get(i);
}
}
return max;
}

private double convertSpeedFromKMHTOMS(double speed) {
return (0.277778 * speed);
}
}
```

GeoFilterMapper.java

```
package com.anmpout.geomapreducejob;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

  public class GeoFilterMapper extends Mapper<GeoKey,
GeoValue, GeoKeyMap, GeoValue> {

@Override
protected void map(GeoKey key, GeoValue value,
 Mapper.Context context) throws IOException, InterruptedException {
    long bucket = 0;
    if(key.getLocation() != null){
String location = key.getLocation().toString();
bucket =  value.getUnixTimestamp().get() -
 (value.getUnixTimestamp().get() % 3600);
GeoKeyMap mapKey = new GeoKeyMap(key,new LongWritable(bucket));

context.write(mapKey,value);
}
}
}
```

GeoFilter.java

```
package com.anmpout.geomapreducejob;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.log4j.Logger;

public class GeoFilter extends Configured implements Tool {

private static final Logger LOG = Logger.getLogger(GeoFilter.class);

private static void saveItemToDBRealTime(String readLine, Connection conn) {
try {

double speed = 0.0;
int count = 0;
int time = 0;
int pathId = 0;
long currentTimestamp = 0L;
List<FilterData> profileDayData = new ArrayList<>();
// the mysql insert statement
String query = " insert into FILTER_DATA_RT (PATH_ID,TIMESTAMP ,COUNT, SPEED,"
        + " TIME, MAX_SPEED, MIN_SPEED, MEDIAN_SPEED, DAY, MONTH, YEAR)"
        + " values (?, ?, ?, ?, ?,?, ?, ?, ?,?,?)";

// create the mysql insert preparedstatement
String[] parts = readLine.split("\t");
if (!parts[2].equals("") && !parts[2].equals("NaN")) {
count = Integer.parseInt(parts[2]);
}
if (!parts[3].equals("") && !parts[3].equals("NaN")) {
```

```
speed = Double.parseDouble(parts[3]);
}
if (!parts[0].equals("") && !parts[0].equals("NaN")) {
pathId = Integer.parseInt(parts[0]);

}
//TIME
if (!parts[4].equals("") && !parts[4].equals("NaN")) {
time =  Integer.parseInt(parts[4]);
}
//TIMESTAMP
if (!parts[1].equals("") && !parts[1].equals("NaN")) {
currentTimestamp = Long.parseLong(parts[1]);
} else {

}
if (pathId != 0) {


Calendar cal = Calendar.getInstance();
int dayOfWeekInMonth = cal.get(Calendar.DAY_OF_WEEK_IN_MONTH);
int dayOfWeek = cal.get(Calendar.DAY_OF_WEEK);
int year = cal.get(Calendar.YEAR);
int month = cal.get(Calendar.MONTH);
long previousYearTimestamp = getSameDayOfYearD(year - 1, month,
                    dayOfWeek, dayOfWeekInMonth).getTime() / 1000;
long previous2YearTimestamp = getSameDayOfYearD(year - 2, month,
                    dayOfWeek, dayOfWeekInMonth).getTime() / 1000;
String queryProfileSameDay = " select * from FILTER_DATA T WHERE"
                    + " (T.TIMESTAMP = ? OR T.TIMESTAMP = ?)"
                    + " AND T.PATH_ID = ? ORDER BY T.TIMESTAMP ";
PreparedStatement pStmtpProfileDay = conn
.prepareStatement(queryProfileSameDay);
pStmtpProfileDay.setLong(1, previousYearTimestamp);
pStmtpProfileDay.setLong(2, previous2YearTimestamp);
pStmtpProfileDay.setLong(3, pathId);

ResultSet rs = pStmtpProfileDay.executeQuery();

while (rs.next()) {
        FilterData filterData = new FilterData();
        filterData.setPathId(rs.getInt(1));
        filterData.setTimestamp(rs.getLong(2));
        filterData.setTime(rs.getInt(3));
        filterData.setCount(rs.getInt(4));
        filterData.setSpeed(rs.getDouble(6));
        filterData.setDay(rs.getInt(7));
        filterData.setMonth(rs.getInt(8));
        filterData.setYear(rs.getInt(9));
        filterData.setMedianSpeed(rs.getInt(10));
        filterData.setMaxSpeed(rs.getInt(11));
        filterData.setMinSpeed(rs.getInt(12));
        profileDayData.add(filterData);

}

}
PreparedStatement preparedStmt = conn.prepareStatement(query);
//PATH_ID
```

```
if (!parts[0].equals("") && !parts[0].equals("NaN")) {
preparedStmt.setInt(1, Integer.parseInt(parts[0]));
} else {
preparedStmt.setInt(1, 0);
}
//TIMESTAMP
if (!parts[1].equals("") && !parts[1].equals("NaN")) {
preparedStmt.setLong(2, Long.parseLong(parts[1]));
} else {
preparedStmt.setLong(2, 0);
}
//COUNT
if (!parts[2].equals("") && !parts[2].equals("NaN")) {
preparedStmt.setInt(3, Integer.parseInt(parts[2]));
} else {
preparedStmt.setInt(3, 0);
}
//SPEED
if (!parts[3].equals("") && !parts[3].equals("NaN")) {
preparedStmt.setDouble(4, Double.parseDouble(parts[3]));
} else {
preparedStmt.setDouble(4, 0);
}
//TIME
if (!parts[4].equals("") && !parts[4].equals("NaN")) {
preparedStmt.setInt(5, Integer.parseInt(parts[4]));
} else {
preparedStmt.setInt(5, 0);
}
//MAX
if (!parts[6].equals("") && !parts[6].equals("NaN")) {
preparedStmt.setDouble(6, Double.parseDouble(parts[6]));
} else {
preparedStmt.setDouble(6, 0);
}
//min
if (!parts[7].equals("") && !parts[7].equals("NaN")) {
preparedStmt.setDouble(7, Double.parseDouble(parts[7]));
} else {
preparedStmt.setDouble(7, 0);
}
//median
if (!parts[8].equals("") && !parts[8].equals("NaN")) {
preparedStmt.setDouble(8, Double.parseDouble(parts[8]));
} else {
preparedStmt.setDouble(8, 0);
}
//day
if (!parts[9].equals("") && !parts[9].equals("NaN")) {
preparedStmt.setInt(9, Integer.parseInt(parts[9]));
} else {
preparedStmt.setInt(9, 0);
}
//month
if (!parts[10].equals("") && !parts[10].equals("NaN")) {
preparedStmt.setInt(10, Integer.parseInt(parts[10]));
} else {
preparedStmt.setInt(10, 0);
}
```

```
//year
if (!parts[11].equals("") && !parts[11].equals("NaN")) {
preparedStmt.setInt(11, Integer.parseInt(parts[11]));
} else {
preparedStmt.setInt(11, 0);
}

// execute the preparedstatement
preparedStmt.execute();
//create profile for current record
createProfileForCurrentDayTime(profileDayData, currentTimestamp,
 speed, count,time,conn);

} catch (Exception e) {
System.err.println(e.getMessage());
}
}
private static void createProfileForCurrentDayTime(List<FilterData> profileDayData,
 long currentTimestamp, double speed, int count, int time, Connection conn) {
System.out.println("================================");
System.out.println("createProfileForCurrentDay");
System.out.println(profileDayData.size());
for (FilterData pData : profileDayData) {
double difrenceCount  = 0.0;
double difrenceSpeed = 0.0;
double difrenceTime = 0.0;
System.out.println(pData.getYear());
if (count > 1) {
difrenceCount = (pData.getCount() - count) / count;
System.out.println("count");
System.out.println(count);
System.out.println(pData.getCount());
System.out.println(difrenceCount);
System.out.println(difrenceCount * 100);

System.out.println("--------------------");

difrenceSpeed  = (pData.getSpeed() - speed) / speed;
System.out.println("speed");
System.out.println(speed);
System.out.println(pData.getSpeed());
System.out.println(difrenceSpeed);
System.out.println(difrenceSpeed * 100);

System.out.println("--------------------");

difrenceTime  = (pData.getTime() - time) / time;
System.out.println("time");
System.out.println(time);
System.out.println(pData.getTime());
System.out.println(difrenceTime);
System.out.println(difrenceTime * 100);

saveProfileDataToDB(pData,currentTimestamp,difrenceSpeed,
difrenceCount,difrenceTime,conn);

} else {
System.out.println("small sample");
}
```

```
System.out.println("##################");
}
}
private static void saveProfileDataToDB(FilterData pData, long currentTimestamp,
double difSpeed, double difCount, double difTime, Connection conn) {

String query = " insert into PROFILE_DATA (PATH_ID,CURRENT_TIMESTAMP"+
" ,OLD_TIMESTAMP, DIF_SPEED,"
            + " DIF_COUNT, DIF_TIME)"
            + " values (?, ?, ?, ?, ?,?)";
try{
            PreparedStatement preparedStmt = conn.prepareStatement(query);

preparedStmt.setInt(1, pData.getPathId());
preparedStmt.setLong(2, currentTimestamp);
preparedStmt.setLong(3, pData.getTimestamp());
preparedStmt.setDouble(4, difSpeed);
preparedStmt.setDouble(5, difCount);
preparedStmt.setDouble(6, difTime);
preparedStmt.execute();
}catch(Exception ex){

}


}

@Override
public int run(String[] args) throws Exception {
String outputPath = "/user/thesis/sample1/output/"
+ Long.toString(Calendar.getInstance().getTimeInMillis());

Job job = Job.getInstance(getConf(), "GeoFilter");
job.setJarByClass(this.getClass());
try (Writer writer = new BufferedWriter(new OutputStreamWriter(
new FileOutputStream("/home/cloudera/Downloads/mapperoutput.txt"),
 "utf-8"))) {
writer.write(outputPath);
}
boolean realTime = Boolean.parseBoolean(args[1]);
boolean saveStats = Boolean.parseBoolean(args[2]);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(outputPath));
job.setMapperClass(GeoFilterMapper.class);
job.setMapOutputKeyClass(GeoKeyMap.class);
job.setMapOutputValueClass(GeoValue.class);
job.setInputFormatClass(GeoInputFormat.class);
job.setReducerClass(GeoFilterReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setOutputFormatClass(TextOutputFormat.class);

if (job.waitForCompletion(true) && job.isSuccessful() && saveStats) {

Configuration hadoopConfig = new Configuration();
hadoopConfig.set("fs.defaultFS",
 "hdfs://quickstart.cloudera:8020");
hadoopConfig.set("fs.file.impl",
 org.apache.hadoop.fs.LocalFileSystem.class.getName());
```

```
FileSystem fs = FileSystem.get(hadoopConfig);
Path inputPath2 = new Path(outputPath.trim()
.replace("\n", "") + "/part-r-00000");
openFileFromHDFS(fs, inputPath2, realTime);

}

return job.waitForCompletion(true) ? 0 : 1;

}

public static void main(String[] args) throws Exception {
int exitCode = ToolRunner.run(new GeoFilter(), args);
System.exit(exitCode);

}

private static void saveItemToDB(String readLine, Connection conn) {
try {
// the mysql insert statement
String query = " insert into FILTER_DATA (PATH_ID, "
        +"TIMESTAMP ,COUNT, SPEED,"
        + " TIME, MAX_SPEED, MIN_SPEED, MEDIAN_SPEED, DAY, MONTH, YEAR)"
        + " values (?, ?, ?, ?, ?,?, ?, ?, ?,?,?)";

// create the mysql insert preparedstatement
String[] parts = readLine.split("\t");
PreparedStatement preparedStmt = conn.prepareStatement(query);
//PATH_ID
if (!parts[0].equals("") && !parts[0].equals("NaN")) {
preparedStmt.setInt(1, Integer.parseInt(parts[0]));
} else {
preparedStmt.setInt(1, 0);
}
//TIMESTAMP
if (!parts[1].equals("") && !parts[1].equals("NaN")) {
preparedStmt.setLong(2, Long.parseLong(parts[1]));
} else {
preparedStmt.setLong(2, 0);
}
//COUNT
if (!parts[2].equals("") && !parts[2].equals("NaN")) {
preparedStmt.setInt(3, Integer.parseInt(parts[2]));
} else {
preparedStmt.setInt(3, 0);
}
//SPEED
if (!parts[3].equals("") && !parts[3].equals("NaN")) {
preparedStmt.setDouble(4, Double.parseDouble(parts[3]));
} else {
preparedStmt.setDouble(4, 0);
}
//TIME
if (!parts[4].equals("") && !parts[4].equals("NaN")) {
preparedStmt.setInt(5, Integer.parseInt(parts[4]));
} else {
preparedStmt.setInt(5, 0);
}
//MAX
```

```
if (!parts[6].equals("") && !parts[6].equals("NaN")) {
preparedStmt.setDouble(6, Double.parseDouble(parts[6]));
} else {
preparedStmt.setDouble(6, 0);
}
//min
if (!parts[7].equals("") && !parts[7].equals("NaN")) {
preparedStmt.setDouble(7, Double.parseDouble(parts[7]));
} else {
preparedStmt.setDouble(7, 0);
}
//median
if (!parts[8].equals("") && !parts[8].equals("NaN")) {
preparedStmt.setDouble(8, Double.parseDouble(parts[8]));
} else {
preparedStmt.setDouble(8, 0);
}
//day
if (!parts[9].equals("") && !parts[9].equals("NaN")) {
preparedStmt.setInt(9, Integer.parseInt(parts[9]));
} else {
preparedStmt.setInt(9, 0);
}
//month
if (!parts[10].equals("") && !parts[10].equals("NaN")) {
preparedStmt.setInt(10, Integer.parseInt(parts[10]));
} else {
preparedStmt.setInt(10, 0);
}
//year
if (!parts[11].equals("") && !parts[11].equals("NaN")) {
preparedStmt.setInt(11, Integer.parseInt(parts[11]));
} else {
preparedStmt.setInt(11, 0);
}

// execute the preparedstatement
preparedStmt.execute();

} catch (Exception e) {
System.err.println(e.getMessage());

}

}

public static void openFileFromHDFS(FileSystem fs,
 Path path, boolean realTime) throws IOException,
ClassNotFoundException, InstantiationException, IllegalAccessException, SQLException {

InputStream is = fs.open(path);
String readLine;
BufferedReader br = new BufferedReader(new InputStreamReader(is));
Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection conn = DriverManager.getConnection(
"jdbc:mysql://geofilterdb.c4nkehdtywwc.us-east-2.rds.amazonaws.com:3306/geofilterdb",
"cloudera", "cloudera");
if (realTime) {
while (((readLine = br.readLine()) != null)) {
```

```
saveItemToDBRealTime(readLine, conn);
System.out.println(readLine);

}

} else {
while (((readLine = br.readLine()) != null)) {
saveItemToDB(readLine, conn);
System.out.println(readLine);

}
}
conn.close();
}

public static Date getSameDayOfYearD(int year, int month, int day, int week) {
Calendar cal = Calendar.getInstance();
cal.set(Calendar.DAY_OF_WEEK, day);
cal.set(Calendar.DAY_OF_WEEK_IN_MONTH, week);
cal.set(Calendar.MONTH, month);
cal.set(Calendar.YEAR, year);
cal.set(Calendar.HOUR, cal.get(Calendar.HOUR) - 1);
cal.set(Calendar.MINUTE, 0);
cal.set(Calendar.SECOND, 0);
cal.set(Calendar.MILLISECOND, 0);
return cal.getTime();
}
}
```

FilterData.java

```
package com.anmpout.geomapreducejob;

public class FilterData {
private int pathId;
private long timestamp;
private int time;
private int count;
private double speed;
private int day;
private int month;
private int year;
private int medianSpeed;
private int maxSpeed;
private int minSpeed;

public FilterData() {
}


public int getPathId() {
return pathId;
}

public void setPathId(int pathId) {
this.pathId = pathId;
```

```
}

public long getTimestamp() {
return timestamp;
}

public void setTimestamp(long timestamp) {
this.timestamp = timestamp;
}

public int getTime() {
return time;
}

public void setTime(int time) {
this.time = time;
}

public int getCount() {
return count;
}

public void setCount(int count) {
this.count = count;
}

public double getSpeed() {
return speed;
}

public void setSpeed(double speed) {
this.speed = speed;
}

public int getDay() {
return day;
}

public void setDay(int day) {
this.day = day;
}

public int getMonth() {
return month;
}

public void setMonth(int month) {
this.month = month;
}

public int getYear() {
return year;
}

public void setYear(int year) {
this.year = year;
}

public int getMedianSpeed() {
```

```
return medianSpeed;
}

public void setMedianSpeed(int medianSpeed) {
this.medianSpeed = medianSpeed;
}

public int getMaxSpeed() {
return maxSpeed;
}

public void setMaxSpeed(int maxSpeed) {
this.maxSpeed = maxSpeed;
}

public int getMinSpeed() {
return minSpeed;
}

public void setMinSpeed(int minSpeed) {
this.minSpeed = minSpeed;
}

@Override
public String toString() {
return "FilterData{" + "pathId=" + pathId + ", timestamp=" +
timestamp + ", time=" + time + ", count=" + count +
", speed=" + speed + ", day=" + day + ", month=" +
month + ", year=" + year + ", medianSpeed=" +
medianSpeed + ", maxSpeed=" + maxSpeed +
", minSpeed=" + minSpeed + '}';
}
}
```

Παράρτημα Β - Βιβλιογραφία

[1] I.MET-E.K.E.T.A, *CERTH-HIT OpenData Hub Greece*, [online] Available at: http://opendata.imet.gr [Accessed 04 December 2018].

[2] Huib van Essen/CE Delft, December 2018. *Sustainable Transport Infrastructure Charging and Internalisation of Transport Externalities.* [pdf] Available at: https://ec.europa.eu/transport/sites/transport/files/2018-year-multimodality-external-costs-ce-delft-preliminary-results.pdf [Accessed 13 April 2019].

[3] European Environment Agency, April 2017. *Road traffic remains biggest source of noise pollution in Europe* [pdf] Available at: https://www.eea.europa.eu/downloads/7579a27ece7344089a0a24baf86f9267/1493020977/road-traffic-remains-biggest-source.pdf [Accessed 13 April 2019].

[4] CERB, July 2014. *The future economic and environmental costs of gridlock in 2030. An assessment of the direct and indirect economic and environmental costs of idling in road traffic congestion to households in the UK, France, Germany and*

the USA [pdf] Διαθέσιμο: https://www.ibtta.org/sites/default/files/documents/MAF/Costs-of-Congestion-INRIX-Cebr-Report%20(3).pdf [Accessed 14 April 2019].

[5] Ghemawat, S., Gobioff, H., Leung, S. (2003) *The Google File System,* Proceedings of the 19th ACM Symposium on Operating Systems Principles, ACM, Bolton Landing, NY (2003), pp. 20-43.

[6] Doug, C., 2013. *Hadoop: Toddler Talk Provides Big Data Name.* Interview from Chris Morris. CNBC Tuesday, 28 May 2013 11:59.

[7] Apache Software Foundation, *Apache Hadoop Releases*, [online] Available at: https://hadoop.apache.org/old/releases.html [Accessed 20 April 2019].

[8] Ghemawat, S., Dean, J. (2004) MapReduce: *Simplified Data Processing on Large Clusters,* OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150

[9] Apache Software Foundation, *The Apache™ Hadoop® project,* [online] Available at: https://hadoop.apache.org/ [Accessed 20 April 2019].

[10] Prajapati, V. (2013). *Big Data Analytics with R and Hadoop*. Birmingham - Mumbai: Packt Publishing, pp 28-58.

[11] Apache Software Foundation, *Apache Mahout project,* [online] Available at: https://mahout.apache.org/ [Accessed 21 April 2019].

[12] Apache Software Foundation, *Apache HBase™ project,* [online] Available at: https://hbase.apache.org/ [Accessed 21 April 2019].

[13] Apache Software Foundation, *Apache Hive™ project,* [online] Available at: https://hive.apache.org/ [Accessed 21 April 2019].

[14] Apache Software Foundation, *Apache Pig project,* [online] Available at: https://pig.apache.org/index.html [Accessed 21 April 2019].

[15] Apache Software Foundation, *Apache Sqoop project,* [online] Available at: https://sqoop.apache.org/ [Accessed 21 April 2019].

[16] Apache Software Foundation, *Apache Solr project,* [online] Available at: https://lucene.apache.org/solr/ [Accessed 21 April 2019].

[17] Apache Software Foundation, *Apache Ambari project,* [online] Available at: https://ambari.apache.org/ [Accessed 21 April 2019].

[18] Apache Software Foundation, *Apache Zookeeper project,* [online] Available at: https://zookeeper.apache.org/ [Accessed 21 April 2019].

[19] Sajwan, V., Yadav, V., Dr. Haider, M. (2015) *The Hadoop Distributed File System:Architecture and Internals,* International Journal of Combined Research & Development (IJCRD), 4(3).

[20] Shvachko, K., Kuang, H., Radia, S., Chansler, r. (2010) *The Hadoop Distributed File System,* IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA (2010)

[21] Apache Software Foundation, *HDFS default configuration,* [online] Available at: https://hadoop.apache.org/docs/r3.1.0/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml [Accessed 23 April 2019].

[22] Sarkar, A., Ghosh, A., Dr. Nath A. (2015) *MapReduce: A Comprehensive Study on Applications, Scope and Challenges,* International Journal of Advance Research in Computer Science and Management Studies, July 2015, 3(7).

[23] Μητσακάκη, Χ., Μπιλλήρης, Χ. (2007) *Εισαγωγή στη Γεωδαισία,* ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ Σχολή Αγρονόμων και Τοπογράφων Μηχανικών Τομέας Τοπογραφίας-Εργαστήριο Ανώτερης Γεωδαισίας, Αθήνα ,σελίδες 55-70

[24] Smith, J.R. (1997) *Introduction to Geodesy,* New York: John Wiley and Sons, Inc, p 83-150 & 155-170

[25] NGA, *National Geospatial-Intelligence Agency portal,* [online] Available at: https://web.archive.org/web/20120402143802/https://www1.nga.mil/ProductsServices/GeodesyandGeophysics/WorldGeodeticSystem/Pages/default.aspx [Accessed 25 April 2019].

[26] U.S. government, *GPS: The Global Positioning System A global public service brought to you by the U.S. government* , [online] Available at: https://www.gps.gov [Accessed 25 April 2019].

[27] Rizos, C. (2014) *Trends in GPS Technology & Applications*, Satellite Navigation & Positioning Group, School of Surveying and Spatial Information Systems The University of New South Wales, Sydney.

[28] Gartner (2001) *Big data Definition,* [online] Available at: https://www.gartner.com/it-glossary/big-data/ [Accessed 27 April 2019].

[29] Ishwarappa, Anuradha, J. (2015) *A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology* , International Conference on Intelligent Computing,Communication & Convegence (ICCC-2015).

[30] Mitsakis, E., Salanova Grau, J.M., Chrysohoou, E., Aifadopoulou, G. (2015) *A robust method for real time estimation of travel times for dense urban road networks using point-to-point detectors*, Transport, 30:3, 264-272, DOI:10.3846/16484142.2015.1078845.

[31] LocationTech, *LocationTech Spatial4j project,* [online] Available at: https://projects.eclipse.org/projects/locationtech.spatial4j [Accessed 10 February 2019].

[32] Owens, J. R., Lentz, J., Femiano B. (2013). *Hadoop Real-World Solutions Cookbook*. Birmingham - Mumbai: Packt Publishing, pp 98-104.

[33] Adminfaces, *AdminFaces open source project,* [online] Available at*:* https://adminfaces.github.io/docs/latest/ [Accessed 12 March 2019].

[34] Red Hat, *Wildfily project,* [online] Available at*:* *https://wildfly.org/* [Accessed 15 March 2019].

[35] Cloudera, *Cloudera CHD,* [online] Available at*:* https://www.cloudera.com/about.html [Accessed 5 june 2019].