ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη και υλοποίηση μιας πλατφόρμας κοινωνικής δικτύωσης, με έμφαση στη διαχείριση μεγάλου όγκου δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικηφόρου Ι. Μανδηλαρά

Επιβλέπων : **Ιάκωβος Βενιέρης**
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΥΛΙΚΩΝ

Μελέτη και υλοποίηση μιας πλατφόρμας κοινωνικής δικτύωσης, με έμφαση στη διαχείριση μεγάλου όγκου δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικηφόρου Ι. Μανδηλαρά

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Τρίτη, 6 Μαρτίου 2018

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ι. Βενιέρης	Δ.Θ. Κακλαμάνη	Γ. Ματσόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγήτρια Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

(Υπογραφή)

.....

Νικηφόρος Μανδηλαράς

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

Περίληψη

Μέσω αυτής της διπλωματικής εργασίας επιχειρούμε μια προσέγγιση πάνω στο πρόβλημα της διαχείρισης μεγάλου όγκου δεδομένων. Τα συστήματα όπου εμπλέκεται σημαντικός όγκος πληροφορίας είναι πάρα πολλά και αφορούν πλήθος διαφορετικών περιπτώσεων. Σαν παράδειγμα μελέτης επιλέξαμε αυτό των κοινωνικών δικτύων. Τα κοινωνικά δίκτυα την τελευταία δεκαετία έχουν αρχίσει να αποτελούν αναπόσπαστο κομμάτι της ζωής δισεκατομμυρίων ανθρώπων σε κάθε γωνιά του πλανήτη.

Καταπιανόμαστε λοιπόν κυρίως με τα ζητήματα που αφορούν την απόδοση ενός τέτοιου δικτύου κάτω από μεγάλο φορτίο, και εξερευνούμε τρόπους προκειμένου να δώσουμε απαντήσεις. Για το σκοπό αυτό σε πρώτο στάδιο γίνεται ο σχεδιασμός και η υλοποίηση μιας διαδικτυακής εφαρμογής κοινωνικού δικτύου προκειμένου να προσδιοριστούν οι λειτουργίες που πρέπει να διαθέτει ένα τέτοιο σενάριο και κατόπιν εντοπίζονται τα τμήματα αυτού που παρουσιάζουν αδυναμία ανταπόκρισης στο αυξανόμενο φορτίο. Στη συνέχεια χρησιμοποιώντας τις δυνατότητες του Apache Spark, που αποτελεί ένα προγραμματιστικό και υπολογιστικό μοντέλο για κατανεμημένα συστήματα, προσπαθούμε να αντιμετωπίσουμε τις ανάγκες του δικτύου, με χρήση συγκεκριμένων αρχιτεκτονικών και μεθόδων.

Τέλος για να εξασφαλίσουμε τους υπολογιστικούς πόρους που απαιτεί η χρήση του Apache Spark, δηλαδή μια διασυνδεδεμένη ομάδα υπολογιστών, μεταφέραμε την υλοποίηση μας και εκτελέσαμε τις μετρήσεις μας στην υπηρεσία υπολογιστικού νέφους της Google, την Google Cloud Platform.

Λέξεις Κλειδιά: Κοινωνικά δίκτυα, Μεγάλα δεδομένα, Εφαρμογή ιστού, Συστάδα υπολογιστών, Apache Spark, Μη σχεσιακή βάση δεδομένων, Υπολογιστικό νέφος

Abstract

Through this diploma thesis we are attempting an approach to the problem of managing a large volume of data. There are a lot of systems where a significant amount of information is involved and they belong to many different situations. As an example of a study we chose that of the social networks. Social networks in the last decade have become an integral part of the lives of billions of people all around the world.

Therefore, we mainly deal with issues concerning the performance of such a network under heavy load, and we are looking at ways to provide answers. To do this, in first place, we design and implement a social networking web application in order to determine the operations that such a scenario has to have, and then we try to identify those parts that are unable to respond to the growing load. Afterwards, using the capabilities of Apache Spark, which is a programming and computing model for distributed systems, we try to address the needs of the network, using specific architectures and methods.

Finally, to ensure the computing resources required by Apache Spark, a computer cluster, we transferred our implementation and executed our metrics to Google's cloud computing service, Google Cloud Platform.

Keywords: Social networks, Big data, Web app, Computer cluster, Apache Spark, NoSQL database, Cloud computing

Ευχαριστίες

Πρώτα απ’ όλα θα ήθελα να ευχαριστήσω θερμά τον καθηγητή του Ε.Μ.Π. κύριο Ιάκωβο Βενιέρη για την ευκαιρία που μου έδωσε να ασχοληθώ με το αντικείμενο της παρούσας εργασίας το οποίο πέρα από το μεγάλο ενδιαφέρον που παρουσιάζει βρίσκεται και στην αιχμή των τεχνολογικών εξελίξεων.

Ακόμη θα ήθελα ιδιαίτερα να ευχαριστήσω τον υποψήφιο Διδάκτορα Μανόλη Καραμανή για όλη τη βοήθεια και την καθοδήγηση που μου παρείχε κατά τους μήνες εκπόνησης αυτής της διπλωματικής εργασίας.

Πίνακας Περιεχομένων

Κεφάλαιο 1ο : Εισαγωγή

1.1 Γενικά

Με τον όρο κοινωνικά δίκτυα αναφερόμαστε στο σύνολο ενεργών οντοτήτων που αποτελούνται από άτομα, οργανισμούς, ομάδες κ.λ.π. καθώς και στο σύνολο των σχέσεων που αναπτύσσουν μεταξύ τους όπως σχέσεις φιλίας, δεσμοί ή χρηματικές συναλλαγές. Αυτή η κοινωνική κατασκευή που παρατηρείται σε όλες τις ιστορικές περιόδους, μεταφέρεται τον 21ο αιώνα και στο διαδίκτυο με εικονική πλέον μορφή και μετατρέπεται σε ένα παγκόσμιο και καθολικά αναγνωρισμένο θεσμό κοινωνικής αλληλεπίδρασης.

Από τη δημιουργία τους μέχρι και σήμερα, η εξάπλωση τους υπήρξε ραγδαία και εντάθηκε ακόμα περισσότερο τα τελευταία χρόνια με την επικράτηση των έξυπνων κινητών τηλεφώνων. Πλέον περίπου το ένα τρίτο του παγκόσμιου πληθυσμού χρησιμοποιεί ενεργά τα κοινωνικά δίκτυα, με το ποσοστό αυτό να φτάνει το 50% στη Δυτική Ευρώπη και το 60% στη Βόρεια Αμερική. Η ανοδική τους πορεία προβλέπεται να συνεχιστεί καθώς όλο και περισσότερες ηλικιακές ομάδες εξοικειώνονται με τις νέες τεχνολογίες και παράλληλα ένα νέο κύμα χρηστών έρχεται από τις αναπτυσσόμενες περιοχές του πλανήτη όπως είναι η νότια, η κεντρική Ασία και η Αφρική. Ακόμα τα κοινωνικά δίκτυα αποτελούν μια από τις κυριότερες δραστηριότητες στο διαδίκτυο αφού παραπάνω από το 70% των χρηστών αυτού τα χρησιμοποιούν ενεργά με το ποσοστό αυτό να βρίσκεται πίσω μόνο από το αντίστοιχο για το ηλεκτρονικό ταχυδρομείο.

Συνεπώς ο όγκος δεδομένων που καλείται να διαχειριστεί ένα κοινωνικό δίκτυο προσπέρασε γρήγορα αυτόν ενός συμβατικού ιστότοπου και αναδείχτηκε σε κύριο πρόβλημα μαζί με την ανάγκη για συνεχή προσαρμογή στο ολοένα και μεγαλύτερο φορτίο. Κάθε δευτερόλεπτο ένα κοινωνικό δίκτυο δέχεται δεκάδες χιλιάδες αιτήσεις για αναγνώσεις της αρχικής σελίδας καθώς και χιλιάδες νέες δημοσιεύσεις ανεβαίνουν. Η προσπάθεια για όσο το δυνατό αμεσότερη ανταπόκριση σε όλη αυτή την κίνηση οδήγησε τον κλάδο των social media στο να δημιουργήσει νέα πεδία μελέτης τα οποία απέφεραν την υλοποίηση νέων εργαλείων και συστημάτων.

Τα εργαλεία αυτά όπως θα δούμε και παρακάτω λειτουργούν σε συστάδες υπολογιστικών μηχανημάτων διαμοιράζοντας έτσι τον όγκο των εργασιών που έχουν να εκτελέσουν σε όλα τα επιμέρους μηχανήματα-κόμβους αθροίζοντας με αυτό τον

τρόπο τις δυνατότητες των υπολογιστικών τους πόρων. Η εξέλιξη τους είναι συνεχής και πλέον αναπτύσσουν δυνατότητες απόκρισης σε πραγματικό χρόνο, στοιχείο που στη σημερινή εποχή είναι το ζητούμενο. Τέλος ανταποκρίνονται αποτελεσματικά στον αυξανόμενο φόρτο καθώς και στα σφάλματα διασφαλίζοντας απρόσκοπτη και συνεχή λειτουργία στοιχεία απαραίτητα για τα κοινωνικά δίκτυα.

1.2 Αντικείμενο της διπλωματικής

Το αντικείμενο αυτής της διπλωματικής εργασίας είναι η μελέτη των προκλήσεων κατά την σχεδίαση και την κατασκευή μιας πλατφόρμας ικανής να διαχειρίζεται μεγάλο όγκο δεδομένων. Ως σενάριο για μια τέτοια πλατφόρμα επιλέχθηκε ένα κοινωνικό δίκτυο δημοσίευσης και ανάγνωσης σύντομων μηνυμάτων, που προσομοιώνει δηλαδή των τρόπο λειτουργίας του Twitter. Αρχικά λοιπόν για το σκοπό αυτό υλοποιήθηκε μια τέτοια πλατφόρμα με χρήση απλών τεχνολογιών και στη συνέχεια παρατηρήθηκε η λειτουργία της κάτω από συνθήκες αυξανόμενου φορτίου.

Έπειτα δόθηκε έμφαση στο με ποιο τρόπο - με χρήση ποιων αρχιτεκτονικών, εργαλείων και μεθόδων - θα μπορούσε να γίνει αποδοτική υλοποίηση της πλατφόρμας ώστε να ανταποκριθεί στις αυξημένες αυτές ανάγκες. Μέσα από αυτή τη διαδικασία αναδείχτηκαν τα πλεονεκτήματα που προσφέρει η χρήση συστάδων υπολογιστών για την κατανεμημένη και παράλληλη εκτέλεση των εφαρμογών. Η υλοποίηση βασίστηκε στη χρήση του εργαλείου Apache Spark το οποίο λειτουργεί κατά τον προαναφερθέντα τρόπο και επιτρέπει την επεξεργασία των δεδομένων τόσο ανά δέσμη –batch processing- όσο και σε πραγματικό χρόνο –stream processing.

Επιχειρήθηκε ο συνδυασμός των δύο αυτών τρόπων χειρισμού των δεδομένων με βάση το μοντέλο της λάμδα-αρχιτεκτονικής που θα αναλύσουμε παρακάτω, προκειμένου να υποστηριχτούν αποτελεσματικά οι λειτουργίες ενός κοινωνικού δικτύου. Ακόμα μελετήθηκαν οι δυνατότητες του Spark σαν αναλυτικό εργαλείο εξάγοντας κάποια μετρικά στοιχεία για τους χρήστες από τα δεδομένα του δικτύου. Τέλος εξερευνήσαμε τις δυνατότητες του υπολογιστικού νέφους -cloud computing- μεταφέροντας σε μια τέτοια υπηρεσία την υλοποίησή μας.

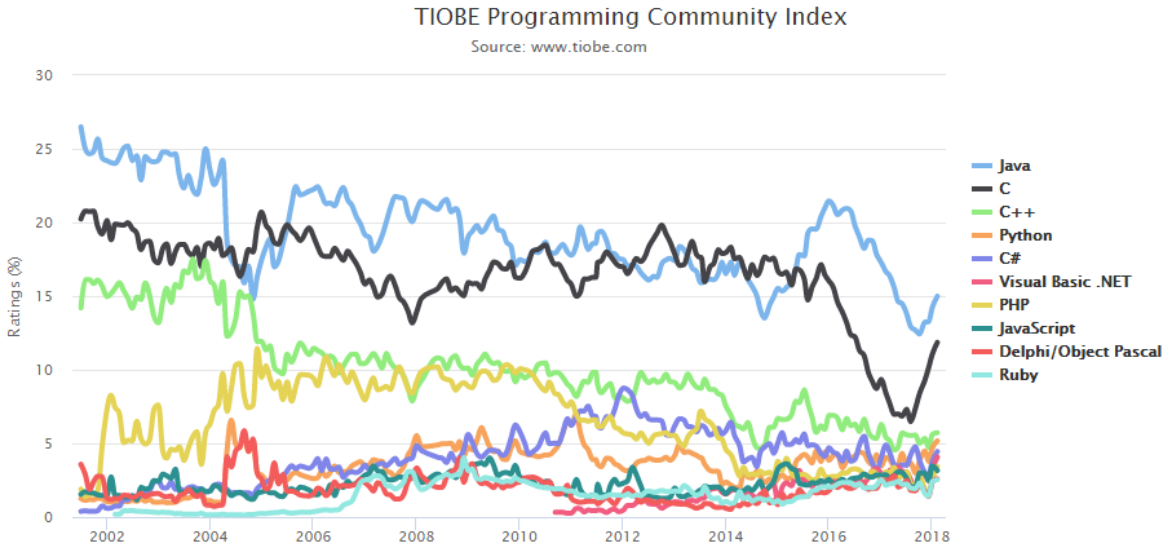
1.3 Οργάνωση κειμένου

Στο κεφάλαιο δύο παρουσιάζουμε όλες τις τεχνολογίες που μελετήθηκαν και χρησιμοποιήθηκαν στα πλαίσια αυτής της εργασίας. Στο κεφάλαιο τρία έχουμε αρχικά την ανάλυση των απαιτήσεων, των λειτουργιών και των σεναρίων χρήσης της εφαρμογής και στη συνέχεια ακολουθεί η σχεδίαση της όπου περιγράφεται η οργάνωση των σελίδων και των κλάσεων των ελεγκτών και των δεδομένων. Στο τέταρτο κεφάλαιο αναφέρουμε συγκεκριμένα στοιχεία που μας οδήγησαν στην επιλογή των εργαλείων και των υπηρεσιών που χρησιμοποιήσαμε για την τελική μας υλοποίηση. Στο κεφάλαιο πέντε περιγράφουμε αναλυτικά επιλεγμένα τμήματα της υλοποίησης μας για κάθε ένα από τα θέματα που προσεγγίσαμε καθώς και ένα σενάριο χρήσης της εφαρμογής. Τέλος στο έκτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα της μελέτης μας μαζί με τα συμπεράσματά μας. Κάνουμε ακόμα ένα συνολικό απολογισμό και προτείνουμε πιθανές επεκτάσεις.

Κεφάλαιο 2ο : Τεχνολογίες

2.1 Java

Η Java αποτελεί σταθερά τις τελευταίες δύο δεκαετίες μια από τις πιο ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού, είναι μία γλώσσα γενικού σκοπού που επιτρέπει τη χρήση πολλών διαφορετικών προγραμματιστικών προτύπων. Συν τοις άλλοις η Java είναι μια ζωντανή γλώσσα με την έννοια ότι εξελίσσεται διαρκώς, εμπλουτιζόμενη με νέα χαρακτηριστικά σε κάθε έκδοση. Επιπλέον με βάση τα στατιστικά που υπάρχουν χρησιμοποιείται από την πλειοψηφία των μεγάλων εταιρειών για την παραγωγή λογισμικού καθώς εξαρχής δημιουργήθηκε με αυτό το στόχο. Να αποτελέσει δηλαδή μία απλή, οικεία και αντικειμενοστραφή γλώσσα ώστε να επιτρέπει την παραγωγή μεγάλων έργων καθώς και την εύκολη συντήρησή τους . Επιπλέον το γεγονός πως η Java τρέχει σε εικονική μηχανή JVM καθιστά ευκολότερη τη χρήση της σε εταιρικό περιβάλλον, καθώς ο κώδικας που παράγεται αρχικά από τον μεταγλωττιστή αναφέρεται στο σύνολο εντολών της εικονικής μηχανής και μετέπειτα μεταφράζεται εκ νέου στο σύνολο εντολών της εκάστοτε αρχιτεκτονικής του υπολογιστικού συστήματος. Έτσι το λογισμικό σε Java τρέχει σε κάθε είδους συσκευή από μεγάλα data centers και υπερυπολογιστές έως κινητά τηλέφωνα και παιχνιδομηχανές.



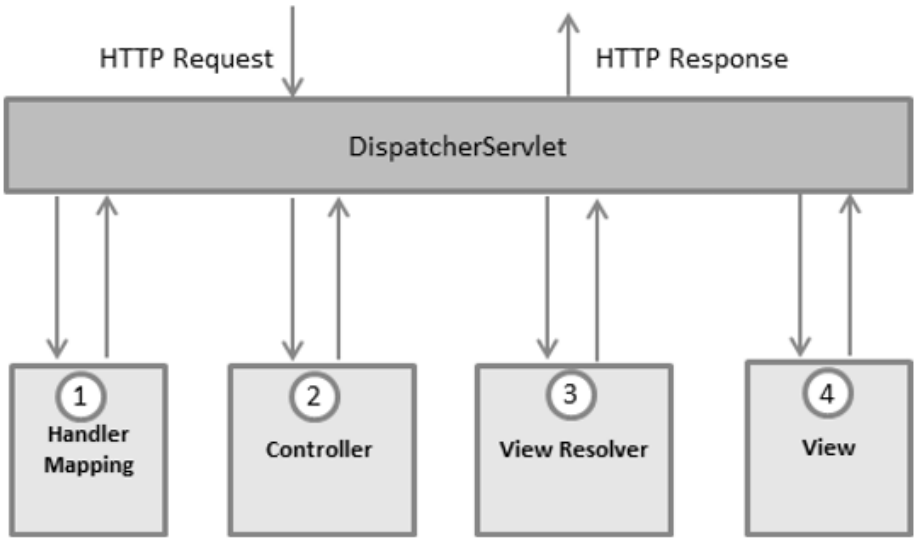
Εικόνα 1:Δημοφιλία γλωσσών προγραμματισμού με βάση αριθμό αναζητήσεων

Παράλληλα έγινε χρήση του προγραμματιστικού περιβάλλοντος Eclipse το οποίο έχει σχεδιαστεί για την ανάπτυξη κώδικα σε Java αν και με κατάλληλες επεκτάσεις

μπορεί να υποστηρίξει και άλλες γλώσσες προγραμματισμού. Το Eclipse ενσωματώνει πολλά χαρακτηριστικά που επιταχύνουν την ανάπτυξη κώδικα, όπως είναι το maven το οποίο είναι ένα εργαλείο για την αυτοματοποιημένη δημιουργία έργων λογισμικού. Συγκεκριμένα φροντίζει για όλες τις εξαρτήσεις που έχει η εφαρμογή μας, ενσωματώνοντας τα απαραίτητα jar αρχεία και παράγει επίσης το τελικό αρχείο της εφαρμογής. Ακόμα κάναμε χρήση της λειτουργίας git του Eclipse καθώς χρησιμοποιήθηκαν διαφορετικά μηχανήματα κατά τη διάρκεια της ανάπτυξης. Όλα τα παραπάνω αφορούν τους λόγους που μας οδήγησαν στην επιλογή της Java.

2.1.1 Spring Framework

Η χρήση της JAVA συνδέθηκε με αυτής του Spring MVC Framework. Το framework αυτό έχει ως επίκεντρό του ένα DispatcherServlet το οποίο είναι υπεύθυνο για το χειρισμό όλων των HTTP αιτήσεων και απαντήσεων.



Εικόνα 2: Τρόπος λειτουργίας του Spring MVC Framework

Μόλις φτάσει μια αίτηση στο DispatcherServlet, εκείνο θα ανατρέξει στον Handler Mapping προκειμένου να αναζητήσει τον κατάλληλο ελεγκτή. Έπειτα ο συγκεκριμένος ελεγκτής θα καλέσει την μέθοδο που αντιστοιχεί στην αίτηση και στον τρόπο που αυτή στάλθηκε (GET ή POST). Μετά την εκτέλεσή της η μέθοδος επιστρέφει το όνομα μιας όψης. Ο DispatcherServlet συμβουλεύεται τον View Resolver για να επιλέξει τη σωστή. Τέλος τα δεδομένα περνούν στην όψη και αυτή επιστρέφεται στο φυλλομετρητή. Μέσω αρχείων παραμετροποίησης καθορίζεται ποιες αιτήσεις θα διαχειρίζεται το framework και με ποιον τρόπο. Έτσι μπορούμε να έχουμε διαφορετική συμπεριφορά ανάλογα με το URI της αίτησης.

2.1.2 JavaServer Pages (JSP)

Αποτελούν μία τεχνολογία που συμβάλλει στην ανάπτυξη διαδραστικών σελίδων ιστού. Κύριο χαρακτηριστικό τους είναι ότι επιτρέπουν την συνύπαρξη κώδικα Java με στοιχεία HTML. Οι JSP συνιστούν αφαίρεση των Java servlets με την έννοια ότι κατά το χρόνο εκτέλεσης μεταφράζονται σε servlet. Σε αντίθεση όμως με αυτά, όπου η ενσωμάτωση HTML κώδικα εντός της Java είναι μία αρκετά επίπονη διαδικασία οι JSP καταφέρνουν να απλοποιήσουν αρκετά την ανάπτυξη παρέχοντας συγκεκριμένα στοιχεία οριοθέτησης του κώδικα σε Java και ετικέτες που ενσωματώνουν τη βασική λειτουργικότητα συγκεκριμένων ενεργειών που είναι κοινές σε πολλές εφαρμογές JSP. Οι σελίδες αυτές μπορούν είτε να χρησιμοποιηθούν ανεξάρτητα είτε να αποτελέσουν το σκέλος των όψεων στην αρχιτεκτονική MVC που αναλύουμε στη συνέχεια.

2.2 Model View Controller (MVC)

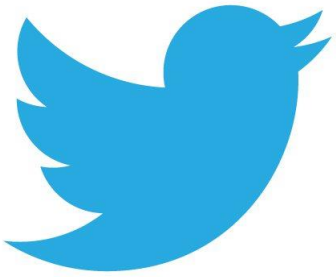
Πρόκειται ίσως για την πιο ευρέως χρησιμοποιούμενη αρχιτεκτονική τεχνοτροπία στην ανάπτυξη λογισμικού. Η λογική που ακολουθεί βασίζεται στο διαχωρισμό της εφαρμογής σε 3 διασυνδεδεμένα μεταξύ τους τμήματα. Την πρώτη συστάδα αυτών αποτελεί το μοντέλο. Με τον όρο αυτό αναφερόμαστε στον τρόπο με τον οποίο αναπαριστώνται τα δεδομένα της εφαρμογής, δηλαδή το πως είναι οργανωμένα άρα και τη μορφή τους όταν θα αποθηκευτούν στη βάση δεδομένων. Στη συνέχεια, η όψη αφορά τον τρόπο με τον οποίο η πληροφορία παρουσιάζεται στον τελικό χρήστη της εφαρμογής. Τέλος ο ελεγκτής καθορίζει το τι θα συμβεί σε κάθε ενέργεια του χρήστη εφαρμόζοντας ενέργειες στα δεδομένα είτε κατευθύνοντας αυτά στην όψη.

Η αρχιτεκτονική αυτή λόγω των πολλών πλεονεκτημάτων που συγκεντρώνει έχει καταφέρει να είναι ευρέως αποδεκτή για εφαρμογές παγκόσμιου ιστού σε πλήθος διαφορετικών γλωσσών προγραμματισμού. Συγκεκριμένα επιτρέπει την παραλληλοποίηση της ανάπτυξης μεταξύ των τριών διαφορετικών τμημάτων της. Τα διαφορετικά τμήματα μπορούν να επαναχρησιμοποιηθούν σε άλλες εφαρμογές. Τέλος η διάδοση της αρχιτεκτονικής είχε ως αποτέλεσμα τη δημιουργία μιας σειράς από frameworks για πλήθος γλωσσών προγραμματισμού, τα οποία προσφέρουν ένα στρώμα αφαίρεσης πάνω από τον πυρήνα της γλώσσας ώστε να διευκολύνουν την ανάπτυξη των τριών τμημάτων. Όλα τα παραπάνω αποτελούν σημαντικά χαρακτηριστικά για τη γρήγορη παραγωγή αξιόπιστου και οργανωμένου κώδικα.

2.3 JavaScript

Η JavaScript είναι μία γλώσσα σεναρίων υψηλού επιπέδου, ικανή για προγραμματισμό πολλών διαφορετικών προτύπων με κυριότερα τον οδηγούμενο από το γεγονός και τον αντικειμενοστραφή προγραμματισμό. Η JavaScript διαθέτει δυναμικό και ασθενές σύστημα τύπων που αποτιμάται κατά το χρόνο εκτέλεσης και χρησιμοποιεί διερμηνευτή. Επιπλέον είναι, μαζί με τα HTML και CSS που χρησιμοποιήσαμε επίσης, μία από τις τρεις κύριες τεχνολογίες που συνθέτουν τον παγκόσμιο ιστό. Χρησιμοποιείται κυρίως στην πλευρά του χρήστη –client side- μιας εφαρμογής, για να προσδώσει επιπλέον λειτουργικότητα στις στατικές HTML σελίδες. Τέλος χρησιμοποιήθηκε και το framework AJAX (Asynchronous JavaScript And XML) το οποίο επιτρέπει την ασύγχρονη επικοινωνία της πλευράς του χρήστη με το server της εφαρμογής και την ανταλλαγή πληροφοριών χωρίς να χρειάζεται η επαναφόρτωση κάποιας σελίδας προσφέροντας πιο διαδραστική και πλούσια πλοήγηση στην εφαρμογή.

2.4 Twitter



Εικόνα 3: Twitter logo

Το twitter είναι ένα από τα πιο δημοφιλή μέσα κοινωνικής δικτύωσης. Κύριο χαρακτηριστικό του είναι η αλληλεπίδραση των χρηστών μέσω σύντομων μηνυμάτων -tweets- που το μέγεθός τους μπορούσε να είναι μέχρι 140 χαρακτήρες κάτι που πρόσφατα άλλαξε σε 280. Έκανε την εμφάνιση του το 2006 και η διάδοσή του μεταξύ των χρηστών του διαδικτύου κατά τα επόμενα χρόνια υπήρξε πραγματικά ραγδαία.

Η χρήση του είναι συνυφασμένη με την επικαιρότητα και την ενημέρωση καθώς και με όλα τα μεγάλα γεγονότα είτε πρόκειται για κάποια αθλητική διοργάνωση, ένα πολιτικό γεγονός ή κάποιο τεχνολογικό συνέδριο. Έτσι έχει αποτελέσει σημείο αναφοράς της έκφρασης των χρηστών του διαδικτύου. Μέσω αυτού επιλέγει να εκφράζεται και μεγάλος αριθμός δημοσίων προσώπων, όπως καλλιτέχνες, ηθοποιοί αλλά και πολιτικοί και διπλωμάτες.

Όσο αφορά στον τρόπο λειτουργίας του, κάθε χρήστης μπορεί να ακολουθήσει άλλους λογαριασμούς χωρίς αυτή η σχέση να είναι αμφίδρομη. Αυτό το στοιχείο σε συνδυασμό με χρήση hashtags(#) -που συμβάλλουν στην κατηγοριοποίηση των tweets σε θεματικές και συζητήσεις- βοηθά σημαντικά στη γρήγορη διάδοση των ειδήσεων ή γενικότερα διαφόρων θεμάτων και στην εξάπλωση αυτής της τάσης σε

μεγάλο μέρος του δικτύου μέσα από συνεχή μηνύματα σχολιασμούς και αναδημοσιεύσεις.

Στόχος του twitter είναι η παράδοση κάθε μηνύματος εντός 5 δευτερολέπτων από τη στιγμή της δημοσίευσης γεγονός που το ανέδειξε στην πρώτη πηγή ενημέρωσης για έκτακτες ειδήσεις μπροστά τόσο από άλλα κοινωνικά δίκτυα αλλά και ειδησεογραφικούς ιστοτόπους. Για την επίτευξη της παραπάνω κατάστασης το twitter έχει δώσει ιδιαίτερη έμφαση στη λειτουργία σε πραγματικό χρόνο αναπτύσσοντας και χρησιμοποιώντας για την υποδομή του, το εργαλείο Apache Storm.

Όσο αφορά την λειτουργία του μπορούμε να πούμε πως διακρίνεται για την απλότητα του, τόσο στη χρήση όσο και στη διεπαφή που χρησιμοποιεί. Στην αρχική σελίδα κάθε χρήστη το κύριο περιεχόμενο είναι τα μηνύματα όλων των λογαριασμών που έχει επιλέξει να ακολουθεί, τοποθετημένα με χρονολογική σειρά και παράλληλα εμφανίζεται μία στήλη με τα πιο δημοφιλή θέματα εκείνη τη στιγμή ανάλογα και με την περιοχή που βρίσκεται ο χρήστης. Για να δούμε τα tweets κάποιου συγκεκριμένου λογαριασμού μπορούμε να αναζητήσουμε το προφίλ του και να εισέλθουμε σε αυτό. Λόγω αυτής της απλής δομής που όμως συγκεντρώνει το σύνολο των ιδιοτήτων και λειτουργιών ενός κοινωνικού δικτύου αποφασίσαμε το δίκτυο που θα χρησιμοποιήσουμε για την μελέτη μας στην παρούσα διπλωματική εργασία να προσομοιώνει το twitter.

2.5 MongoDB

Εικόνα 4: MongoDB Logo

Η MongoDB αποτελεί μία ανοικτού κώδικα μη σχεσιακή βάση δεδομένων υλοποιημένη στη γλώσσα C++. Οι συλλογές -collections- αποτελούν το αντίστοιχο των πινάκων στις σχεσιακές βάσεις και οι καταχωρήσεις εντός τους καλούνται έγγραφα -documents. Σε μία μη σχεσιακή βάση όπως η Mongo δεν υπάρχει ανάγκη για καθορισμό του σχήματος εξαρχής αλλά αυτό εξάγεται δυναμικά από τα δεδομένα που καταχωρούνται. Έτσι λοιπόν έγγραφα με διαφορετική δομή μπορούν να ανήκουν στην ίδια συλλογή και επίσης έχουμε την δυνατότητα είτε να προσθέσουμε είτε να διαγράψουμε πεδία από όσα έγγραφα επιθυμούμε. Πολύ εύκολα λοιπόν μπορούν να γίνουν αλλαγές στα δεδομένα μιας εφαρμογής καθώς αυτή εξελίσσεται.

Η αποθήκευση των δεδομένων στη MongoDB γίνεται σε μορφή JSON και σε

χαμηλότερο επίπεδο τα JSON έγγραφα αναπαρίστανται δυαδικά σε μία μορφή που αποκαλείται BSON. Η μορφή αυτή προσφέρει επιπλέον τύπους δεδομένων, μπορεί να ενσωματώνει αντικείμενα και πίνακες μέσα σε άλλα και μπορεί εύκολα να αναζητήσει αντικείμενα είτε αυτά βρίσκονται στο πρώτο επίπεδο είτε είναι εμφωλευμένα. Επίσης το μοντέλο ερωτημάτων της Mongo υλοποιείται σαν μέθοδοι ή συναρτήσεις μέσα στο API της κάθε γλώσσας προγραμματισμού, έτσι δεν υπάρχει μια γενική και διακριτή γλώσσα αντίστοιχη της SQL.

Κύριοι λόγοι για την επιλογή της Mongo έναντι των κλασσικών και πιο διαδεδομένων σχεσιακών βάσεων ήταν η δυνατότητά της να χειρίζεται καλύτερα πολύπλοκους τύπους δεδομένων που αλλιώς θα μεταφράζονταν σε έναν επιπλέον σχεσιακό πίνακα -denormalization. Το γεγονός αυτό προσφέρει πολύ καλή εγγύτητα των συγγενών δεδομένων κάτι που συντελεί σε αρκετά καλύτερη ταχύτητα έναντι σχεσιακών βάσεων. Τέλος σημαντική είναι και η δυνατότητα της να χειρίζεται αποτελεσματικά εφαρμογές σε καταστάσεις κλιμάκωσης καθώς και η λειτουργίας της σε κατανεμημένο περιβάλλον, υποστηρίζοντας τις τεχνολογίες Replication και Sharding που αναλύουμε στη συνέχεια.

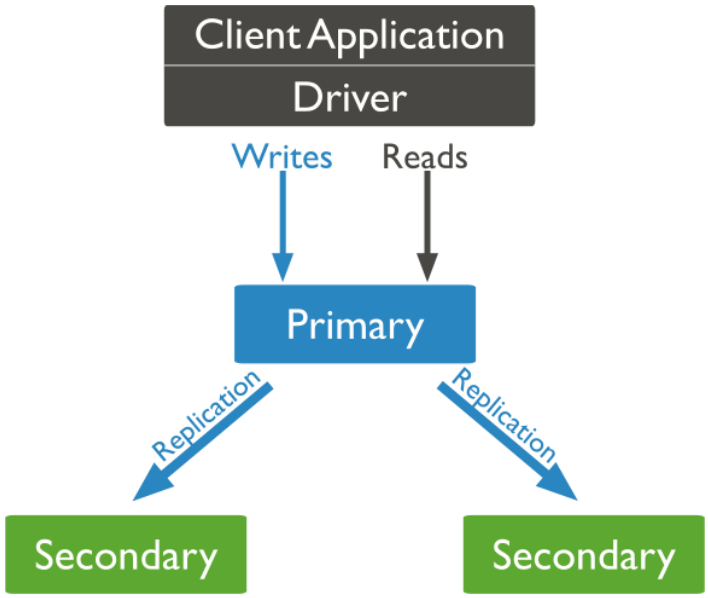
2.5.1 Replication

Με τον όρο replication –αντιγραφή, αναπαραγωγή- η Mongo αναφέρεται στην δυνατότητα που προσφέρει για διατήρηση των ίδιων δεδομένων σε πολλά διαφορετικά μηχανήματα. Στόχος αυτής της μεθόδου είναι να παρέχει κυρίως αυξημένη διαθεσιμότητα και ανοχή στα σφάλματα. Η αρχιτεκτονική που χρησιμοποιείται είναι η εξής: Αρχικά κάθε τέτοια διάταξη αποτελεί ένα replica set με περιττό αριθμό μελών. Ο αρχικός κόμβος που περιέχει τα δεδομένα ορίζεται ως αρχηγός -leader- ενώ τα υπόλοιπα μέλη ως δευτερεύοντα –secondary. Με την αρχικοποίηση τα δεδομένα της βάσης αντιγράφονται στους υπόλοιπους κόμβους. Μόλις ολοκληρωθεί αυτή η διαδικασία το set είναι έτοιμο για χρήση.

Το σύνολο αυτό λειτουργεί με συγκεκριμένους κανόνες. Αυτό που ξεχωρίζει κυρίως τον αρχηγό από τους άλλους κόμβους είναι πως όλες οι ενέργειες εγγραφής απευθύνονται σε αυτόν και μόνο. Στη συνέχεια και με ασύγχρονο τρόπο οι εγγραφές αυτές αναπαράγονται και στα δευτερεύοντα μέλη. Η ασύγχρονη αναπαραγωγή σημαίνει πως ο αρχικός κόμβος αναγγέλλει την ολοκλήρωση εγγραφής μόλις την πραγματοποιήσει ο ίδιος χωρίς να περιμένει να πράξουν το ίδιο και οι δευτερεύοντες. Παρόλα αυτά έχουμε την δυνατότητα να επιλέξουμε οι αναγνώσεις να γίνονται από τους δευτερεύοντες κόμβους. Κάτι τέτοιο προσφέρει παραλληλοποίηση των αναγνώσεων και ταυτόχρονα διευκολύνεται ο κόμβος αρχηγός που πλέον

διαχειρίζεται μόνο τις εγγραφές.

Σε περίπτωση που ο κόμβος-αρχηγός βγει εκτός λειτουργίας τότε γίνεται «ψηφοφορία» μεταξύ των υπόλοιπων για την ανάδειξη νέου, έτσι διασφαλίζεται ότι δε θα υπάρξει απώλεια δεδομένων ούτε και σημαντικός χρόνος μη εξυπηρέτησης της εφαρμογής. Επίσης με τη λειτουργία αναπαραγωγής μπορούμε να κατανείμουμε γεωγραφικά τους κόμβους και να επιλέξουμε οι αιτήσεις για αναγνώσεις των χρηστών να απευθύνονται στον κοντινότερο κόμβο κάτι που μειώνει τις καθυστερήσεις λόγω εγγύτητας.



Εικόνα 5: Αναπαράσταση δομής αναπαραγωγής τριών στοιχείων

Η Mongo λοιπόν, υλοποιεί την αναπαραγωγή χρησιμοποιώντας μια αρχιτεκτονική με κόμβο αρχηγό. Τέτοιες αρχιτεκτονικές είναι πιο αξιόπιστες έναντι αυτών με πολλούς ή καθόλου αρχηγούς. Παρόλα αυτά παρουσιάζουν συγκεκριμένα θέματα που χρήζουν προσοχής.

Κύριο πρόβλημα αποτελούν οι αναγνώσεις από τους δευτερεύοντες κόμβους. Δεδομένου ότι οι εγγραφές από τον αρχηγό γίνονται ασύγχρονα οι αναγνώσεις αυτές μπορούν να επιστρέψουν παλαιά δεδομένα. Λόγω αυτής της πραγματικότητας γεννιούνται και τα παρακάτω πιθανά προβλήματα.

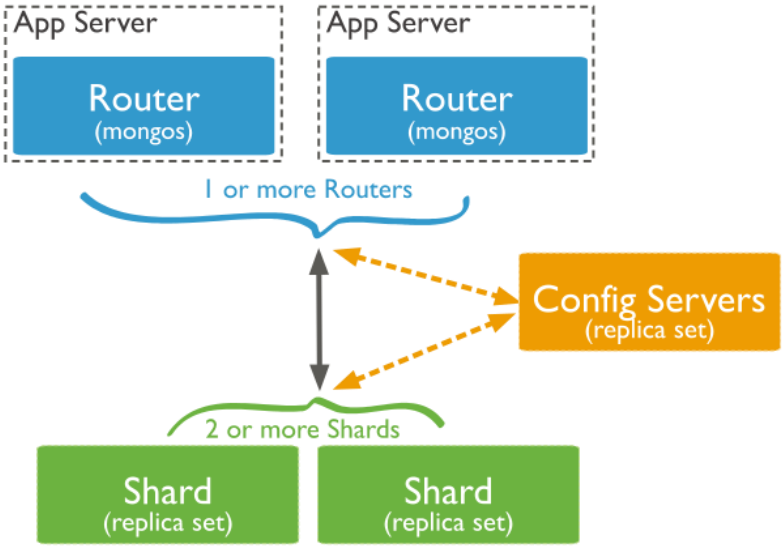
- Οι αναγνώσεις από διαφορετικές συσκευές είναι πιθανόν να επιστρέφουν διαφορετικά αποτελέσματα λόγω του ότι απευθύνονται σε διαφορετικούς δευτερεύοντες κόμβους όπου ο καθένας έχει ενσωματώσει σε διαφορετικό βαθμό τις εγγραφές από τον πρωταρχικό.
- Αδυναμία να εντοπίσει κάποιος χρήστης μία εγγραφή –δημοσίευση για την περίπτωση ενός κοινωνικού δικτύου- που μόλις έχει πραγματοποιήσει, λόγω

του ότι διαβάζει από ένα δευτερεύοντα κόμβο που βρίσκεται κάποια δευτερόλεπτα πίσω από τον αρχηγό.

- Απώλεια πρόσφατων δεδομένων που δεν έχουν ακόμα διαδοθεί σε άλλους κόμβους λόγω αποτυχίας του αρχηγού.
- Διασφάλιση ότι κάθε νέα ανάγνωση συμβαίνει από τον ίδιο κόμβο ώστε να μην παρατηρούν οι χρηστές «αντιστροφή του χρόνου». Δηλαδή έπειτα από μία ανανέωση σελίδας αδυνατούμε να δούμε δημοσιεύσεις που προγενέστερα είχαμε εντοπίσει.
- Τέλος σε συνδυασμό με sharding που θα αναλύσουμε παρακάτω είναι πιθανό να δούμε ένα χρονικά μεταγενέστερο γεγονός να εμφανίζεται πρώτο.

2.5.2 Sharding

Sharding ονομάζει η Mongo, τη λειτουργία καταμερισμού των δεδομένων σε πολλά μηχανήματα. Κάτι τέτοιο συμβαίνει προκειμένου να εξυπηρετηθούν εφαρμογές με μεγάλο όγκο δεδομένων που χρειάζονται αθροιστικά μεγάλο throughput. Η λειτουργία αυτή είναι πιο σύνθετη καθώς είναι απαραίτητο να γνωρίζουμε σε ποιο κόμβο βρίσκονται τα διαμοιρασμένα δεδομένα. Το ρόλο αυτό αναλαμβάνουν τα mongos. Πρόκειται για διεργασίες που δρουν σαν query routers, ανακατευθύνοντας, ανάλογα με τη θέση των δεδομένων, τα ερωτήματα που έρχονται στη βάση στον κατάλληλο κόμβο. Μια άλλη μονάδα ενός sharded cluster είναι οι config servers που αποθηκεύουν μετα-δεδομένα και ρυθμίσεις διαμόρφωσης. Shards ονομάζονται οι διεργασίες που κατέχουν ένα τμήμα των δεδομένων. Τόσο τα shards όσο και οι config servers μπορούν να αναπτυχθούν σαν replica sets εξασφαλίζοντας έτσι και τα πλεονεκτήματα της αναπαραγωγής που είδαμε προηγουμένως. Τέλος τόσο οι εγγραφές όσο και οι αναγνώσεις πρέπει να γίνονται αποκλειστικά μέσω των mongos και όχι απευθείας από κάποιο shard.



Εικόνα 6: Αναπαράσταση δομής sharding

Ο καταμερισμός δεδομένων στους κόμβους μια συστάδας υπολογιστών γίνεται ανά συλλογή και με βάση κάποιο πεδίο -κλειδί- που υπάρχει σε κάθε έγγραφο. Έτσι λοιπόν μπορεί από μία βάση τα δεδομένα κάποιων συλλογών να έχουν διαμοιραστεί ενώ των υπόλοιπων να παραμένουν σε ένα κόμβο.

Σημαντική αποτελεί η επιλογή του πεδίου, βάση του οποίου θα γίνει ο καταμερισμός. Ας πάρουμε για παράδειγμα, μια εφαρμογή καιρού που συλλέγει δεδομένα διαρκώς από διάφορους αισθητήρες και έπειτα πραγματοποιεί ανάλυση τους ανά μήνα. Αν επιλέγαμε να κατανείμουμε τις μετρήσεις αυτές ανά μήνα επίσης στα διάφορα shards, δε θα είχαμε κανένα κέρδος από τον καταμερισμό καθώς κάθε φορά όλο το βάρος πέφτει σε έναν και μόνο κόμβο αφού αυτός έχει το απαραίτητο σύνολο δεδομένων ενώ όλοι οι υπόλοιποι θα βρίσκονται σε κατάσταση idle. Πρέπει λοιπόν να κοιτάμε προσεκτικά με ποιον τρόπο κάθε φορά θα αποφύγουμε την δημιουργία τέτοιων hot-spots.

Ακόμα προκειμένου ένα πεδίο να λειτουργήσει ως shard key πρέπει να ξεκινά με αυτό κάποιο δευτερεύων λεξικό -secondary index. Καλό είναι να επιλέγουμε ως shard key το πεδίο εκείνο που εμφανίζεται συχνότερα στα ερωτήματα προς τη βάση. Έτσι οι δρομολογητές γνωρίζουν σε ποιο κόμβο θα προωθήσουν το ερώτημα, ενώ σε διαφορετική περίπτωση θα το μετέφεραν σε όλους.

Υπάρχουν οι εξής δύο στρατηγικές για τον καταμερισμό των δεδομένων στους διάφορους κόμβους: hashed και ranged sharding. Στην πρώτη οι τιμές του πεδίου που έχουμε επιλέξει σαν κλειδί περνούν από μία hash συνάρτηση παράγοντας μία τιμή. Έπειτα σε κάθε shard ανατίθεται ένα εύρος τέτοιων τιμών. Αντίθετα με το ranged

sharding είναι η ίδια τιμή του πεδίου, που διαχωρίζει τα δεδομένα σε τμήματα. Μία ακόμα λειτουργία που μας παρέχει η Mongo είναι η αυτοματοποιημένη αναπροσαρμογή του όγκου των shards σε περίπτωση που συσσωρευτεί μεγάλος αριθμός δεδομένων σε κάποιο.

2.6 Apache Spark

Εικόνα 7: Apache Spark logo

Το Apache Spark αποτελεί ένα ανοικτού κώδικα γενικού σκοπού framework για κατανεμημένα συστήματα. Αναπτύχθηκε προκειμένου να αντιμετωπίσει συγκεκριμένα ελλείμματα του προγραμματιστικού μοντέλου MapReduce που υλοποιούταν σε άλλα εργαλεία με κυρίαρχο το Apache Hadoop. Οι περιπτώσεις που εστίασε ήταν εκείνες όπου ένα σύνολο δεδομένων χρειάζεται να επαναχρησιμοποιηθεί. Για παράδειγμα οι επαναληπτικές εργασίες, όπως αυτές που συμβαίνουν κατά κόρον στους αλγορίθμους μηχανικής μάθησης ή αλληλεπιδραστικές εφαρμογές και αναλύσεις που απαιτούν την φόρτωση ενός συνόλου δεδομένων στη μνήμη και έπειτα τη συνεχή υποβολή ερωτημάτων. Συνεπώς το Spark εστίασε στην ανάγκη συγκράτησης των δεδομένων στη μνήμη διατηρώντας όμως παράλληλα τα καίρια χαρακτηριστικά που ανέδειξαν τα προηγούμενα εργαλεία όπως η επεκτασιμότητα και η ανοχή στα σφάλματα. Έχει γραφτεί σε Scala, που είναι μία JVM αντικειμενοστραφής γλώσσα με δυνατότητες συναρτησιακού προγραμματισμού. Η ανάπτυξη εφαρμογών στο Spark πέρα από Scala μπορεί να γίνει σε Java, Python και R.

2.6.1 Μέρη του οικοσυστήματος

2.6.1.1 Κατανεμημένα Δεδομένα

Το Spark είναι στενά συνδεδεμένο με το οικοσύστημα του Hadoop. Χρησιμοποιεί βασικά το ίδιο σύστημα για την διαχείριση των κατανεμημένων δεδομένων το Hadoop Distributed File System (HDFS). Η λογική του είναι πως αποθηκεύοντας τα δεδομένα σε πολλές διαφορετικές μηχανές μπορώ να παρέχω πολύ μεγαλύτερο συνολικό bandwidth. Ταυτόχρονα κρατά πολλαπλά αντίγραφα αυτών προκειμένου να αντιμετωπίσει αστοχίες κάποιων κόμβων. Όπως θα δούμε και στη συνέχεια μπορούμε να χρησιμοποιήσουμε και άλλα συστήματα για την κατανεμημένη οργάνωση και διαχείριση των δεδομένων μας όπως βάσεις δεδομένων αλλά και αρχεία.

2.6.1.2 Κατανεμημένοι πόροι

Αυτό το επίπεδο αφορά τη διαχείριση των υπολογιστικών πόρων στο cluster. Τα συστήματα που λειτουργούν εδώ προσπαθούν να κατανείμουν τις εργασίες στα διάφορα μηχανήματα με τον καλύτερο δυνατό τρόπο ώστε να εκμεταλλευτούν τις επεξεργαστικές μονάδες και τη μνήμη κάθε μηχανήματος χωρίς να επιβαρύνουν κάποιο κόμβο. Μια άλλη παράμετρος που λαμβάνεται υπόψιν είναι η κατανομή των δεδομένων στους κόμβους ώστε ο καθένας να επεξεργάζεται όσο το δυνατό δεδομένα που ανήκουν στον ίδιο, αποφεύγοντας κατά αυτό τον τρόπο μεταφορά τους από τον έναν στον άλλο, κάτι που θα εισήγαγε μεγάλη καθυστέρηση. Καταλαβαίνουμε λοιπόν πόσο σημαντική είναι για την παράλληλη λειτουργία μια συστοιχίας υπολογιστών, η σωστή διαμέριση των δεδομένων.

Υπάρχουν ουσιαστικά τρεις διαφορετικοί τρόποι για το στήσιμο του Spark πάνω από το HDFS. Μπορεί να χρησιμοποιηθεί είτε το Hadoop YARN δηλαδή ο cluster manager που χρησιμοποιείται και από το Hadoop, είτε το Apache Mesos. Τέλος το Spark προσφέρει πλέον το δικό του cluster environment –Spark standalone cluster- χωρίς την ανάγκη χρήσης άλλου προγράμματος. Η τελευταία επιλογή είναι και η απλούστερη, με το YARN να εξυπηρετεί κυρίως τους προγραμματιστές προερχόμενους από το Hadoop και έχει σχεδιαστεί κυρίως για εργασίες map & reduce, ενώ το Mesos αποτελεί μια πιο γενική εκδοχή του και μπορεί να αντιμετωπίσει διαφορετικά είδη κατανεμημένων εργασιών.

2.6.1.3 Κατανεμημένη επεξεργασία

Αυτό το επίπεδο αποτελεί τον πυρήνα του Spark και είναι αυτό που φροντίζει για την αποστολή, τη δρομολόγηση μιας εργασίας και τις ενέργειες Εισόδου/Εξόδου που πρέπει να γίνουν. Όλα τα υπόλοιπα εργαλεία βασίζονται σε αυτό.

2.6.1.4 Διεπαφές

Πάνω από όλα τα προηγούμενα επίπεδα βρίσκονται διάφορες διεπαφές προγραμματισμού που διευκολύνουν την ανάπτυξη με το Spark.

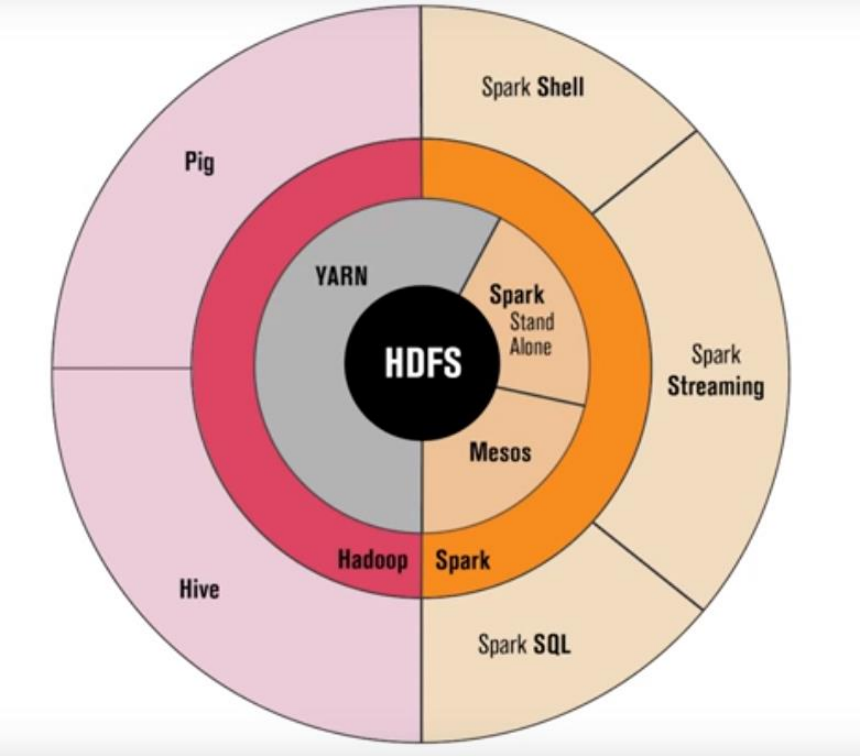
Spark SQL: Με το module αυτό μπορούμε να δουλέψουμε με δομημένα δεδομένα και να θέσουμε σε αυτά SQL ερωτήματα.

Spark MLlib: Παρέχει έτοιμους γνωστούς αλγορίθμους για μηχανική μάθηση όπως ταξινόμησης, ομαδοποίησης, φιλτραρίσματος και προεπεξεργασίας.

Spark Streaming: Το Spark δημιουργήθηκε για την επεξεργασία δεδομένων σε μεγάλες ομάδες -batches. Το σκέλος αυτό προστέθηκε προκειμένου να δοθούν δυνατότητες επεξεργασίας πραγματικού χρόνου. Έτσι δημιουργεί μικρές ομάδες -mini-batches- για να προσομοιώσει ροές δεδομένων -streams- και εφαρμόζει ανάλυση σε αυτές. Αυτός ο σχεδιασμός επιτρέπει ο κώδικας εφαρμογής που έχει γραφτεί για batch επεξεργασία να μπορεί να εφαρμοστεί και για την ανάλυση σε πραγματικό χρόνο.

Spark GraphX: Framework για κατανεμημένη επεξεργασία γράφων.

Spark Shell: Μας παρέχει ένα περιβάλλον γραμμής εντολών όπου μπορούμε να εξερευνήσουμε τα διάφορα χαρακτηριστικά του Spark πολύ άμεσα, είτε να τρέξουμε απλές εφαρμογές.



Εικόνα 8: Αναπαράσταση σε στρώματα του οικοσυστήματος Hadoop – Spark

2.6.2 Δομές δεδομένων

Η πρωταρχική δομή δεδομένων που χρησιμοποιεί το Spark είναι τα RDDs - Resilient distributed datasets. Οι δομές αυτές κατανέμονται σε τμήματα -partitions- τα οποία βρίσκονται στη μνήμη διαφορετικών κόμβων του cluster. Επιπλέον τα RDDs είναι read-only fault-tolerant δομές. Τα RDDs υποστηρίζουν δύο είδη ενεργειών: τα

transformations όπου παράγεται ένα νέο RDD από κάποιο ήδη υπάρχων (δεν μπορεί να τροποποιηθεί ένα RDD καθώς πρόκειται για αμετάβλητη δομή) και τα actions, τα οποία επιστρέφουν μια τιμή ως αποτέλεσμα αφού εκτελέσουν κάποιον υπολογισμό. Όλες οι μετατροπές είναι οκνηρές -lazy- με την έννοια ότι δε θα πραγματοποιηθούν αν δεν ακολουθήσει κάποιο action. Κατά αυτό τον τρόπο βελτιστοποιείται η συνολική επεξεργασία των δεδομένων. Η ανεκτικότητα στα λάθη αυτών των δομών οφείλεται στο γεγονός πως όλες οι μετατροπές καταγράφονται σε ένα ακυκλικό γράφημα –Directed Acyclic Graph (DAG). Τα RDDs αποτελούν τις κορυφές ενός τέτοιου γραφήματος ενώ οι ενέργειες πάνω στα RDDs αναπαρίστανται με ακμές. Σε περίπτωση απώλειας ενός μηχανήματος το τμήμα του RDD που κατείχε μπορεί να ανακτηθεί από το DAG χωρίς να επηρεαστεί η υπόλοιπη διαδικασία.

Στις εκδόσεις του Spark που ακολούθησαν προστέθηκαν δύο επιπλέον δομές τα dataframes και τα datasets. Στα dataframes τα δεδομένα οργανώνονται σε στήλες όπως θα γινόταν και σε μία σχεσιακή βάση δεδομένων. Τα κοινά χαρακτηριστικά με τα RDDs είναι ότι παραμένουν επίσης αμετάβλητα, διατηρούνται στη μνήμη και πως κατανέμονται σε μια συστάδα υπολογιστών. Επιπλέον αυτών τα dataframes διαθέτουν λειτουργία που βελτιστοποιεί το πλάνο εκτέλεσης των queries. Όσο αφορά τα Datasets προσφέρουν σε σχέση με τα Dataframes ασφάλεια τύπων κατά το compile time και όταν γίνονται cache καταναλώνουν λιγότερη μνήμη λόγω του ότι η μορφή των δεδομένων είναι γνωστή.

Πέρα από τα παραπάνω το Spark διαθέτει δύο διαφορετικούς τύπους μοιραζόμενων μεταβλητών, τις broadcast variables και τους accumulators. Οι broadcast μεταβλητές χρησιμοποιούνται για να μοιραστεί σε όλα τα μηχανήματα ένα μεγάλο σύνολο δεδομένων. Παρ’ όλα αυτά το Spark φροντίζει έτσι κι αλλιώς να στείλει σε όλους τους κόμβους δεδομένα που χρειάζονται για την εκτέλεση ενός stage. Συνεπώς οι broadcast μεταβλητές μπορεί να είναι χρήσιμες μόνο όταν αυτά τα δεδομένα χρησιμοποιούνται από περισσότερα από ένα stages, όπου και θέλουμε να αποφύγουμε τον συνεχή διαμοιρασμό τους στου κόμβους, καθώς και όταν είναι σημαντικό να κρατιούνται σε μη σειριοποιημένη μορφή.

Απ’ την άλλη ένας accumulators είναι μια μεταβλητή που κατανέμεται παράλληλα σε όλους τους κόμβους. Οι κόμβοι εκτέλεσης μπορούν να προσθέτουν στην μεταβλητή αυτή, αλλά δεν μπορούν να τη διαβάσουν κάτι που επιτρέπεται μόνο στο πρόγραμμα οδηγό καθώς εκείνο συγκεντρώνει τις τιμές από όλους τους κόμβους.

2.6.3 Το Spark σε κατανεμημένο περιβάλλον

2.6.3.1 Ορολογία

Spark Application: Το πρόγραμμα του χρήστη. Κάθε εφαρμογή αποτελείται από το πρόγραμμα οδηγό –driver program- και τις διεργασίες –executors- που τρέχουν στη συστοιχία υπολογιστών.

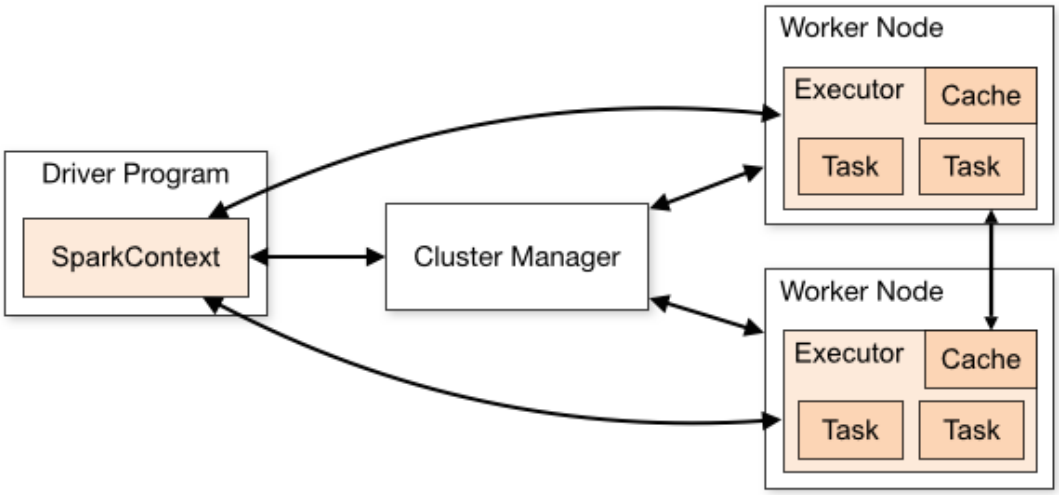
Driver program: Αποκαλείται η διεργασία που εκτελεί τη συνάρτηση main της εφαρμογής και δημιουργεί το SparkContext.

Executor: Μία διεργασία που ανήκει σε συγκεκριμένη εφαρμογή και τρέχει σε κάποιον worker node, εκτελεί tasks και κρατά δεδομένα στη μνήμη ή το δίσκο.

Task: Αποτελεί το μικρότερο τμήμα εργασίας και στέλνεται σε κάποιον executor.

Job: Πρόκειται για παράλληλο υπολογισμό που αποτελείται από πολλαπλά tasks, και δημιουργείται για να εξυπηρετήσει δράσεις –actions.

Stage: Κάθε εργασία -job- χωρίζεται σε μικρότερα σύνολα από tasks τα οποία αποκαλούνται stages και αλληλεξαρτώνται.



Εικόνα 9: Λειτουργία του Spark σε cluster

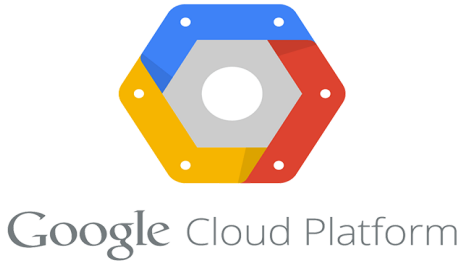
2.6.3.2 Περιγραφή λειτουργίας

Σε κάθε συστοιχία υπολογιστών που τρέχει το Spark ένας από αυτούς έχει το ρόλο του Master, ενώ οι υπόλοιποι ονομάζονται Worker nodes. Το πρόγραμμα οδηγός -Driver Program- τρέχει πάντα στο Master κόμβο ενώ οι κόμβοι «Εργάτες» εκτελούν

τον κώδικα της εφαρμογής. Ο διαχειριστής -Cluster Manager- είναι μία από τις υπηρεσίες που είδαμε προηγουμένως (Standalone, Mesos, Yarn).

Το SparkContext που βρίσκεται στο πρόγραμμα οδηγό αποτελεί την καρδιά του Spark. Αρχικά συνδέεται στον cluster manager και εξασφαλίζει executors στους κόμβους του cluster. Οι λειτουργίες στις κατανεμημένες δομές δεδομένων αποτυπώνονται σε ένα ακυκλικό γράφημα. Η κλήση κάποιου action στέλνει το δημιουργημένο γράφημα στο DAG scheduler που φροντίζει για το διαχωρισμό σε stages. Οι μετασχηματισμοί που εφαρμόζονται στις δομές χωρίζονται σε δύο κατηγορίες (narrow και wide transformations). Οι μετασχηματισμοί της πρώτης κατηγορίας είναι απλούστεροι και μπορούν να ενταχθούν στο ίδιο stage, αντίθετα οι wide μετασχηματισμοί όπως μία reduce λειτουργία που απαιτούν το διαμοιρασμό δεδομένων μεταξύ διαφορετικών κόμβων οδηγούν στην δημιουργία ενός νέου σταδίου. Αφού λοιπόν καθοριστεί το πλάνο στέλνεται ο κώδικας στους Executors όπου και γίνεται η εκτέλεση. Τέλος να σημειώσουμε πως κάθε εφαρμογή τρέχει ανεξάρτητα στο cluster έχοντας τις δικές της διεργασίες. Αυτό από τη μία σημαίνει πως κάθε πρόγραμμα οδηγός κάνει τον δικό του προγραμματισμό εκτέλεσης και από την άλλη πως δεδομένα δεν μπορούν να μοιραστούν μεταξύ διαφορετικών εφαρμογών.

2.7 Google Cloud Platform



Εικόνα 10: Google Cloud Platform logo

Το Google Cloud Platform αποτελεί μια σειρά υπηρεσιών υπολογιστικού νέφους. Οι υπηρεσίες του καλύπτουν ένα ευρύ φάσμα αντικειμένων όπως είναι οι περιοχές των big data, της μηχανικής μάθησης, της ασφάλειας, των ευφυών δικτύων, του διαδικτύου των πραγμάτων, της αποθήκευσης και ανάλυσης δεδομένων καθώς και άλλων πολλών. Παρέχεται από τη Google και αποτελεί την αντίστοιχη υπηρεσία της εταιρίας των Amazon Web Services και Microsoft Azure, που προσφέρονται από την Amazon και τη Microsoft αντίστοιχα. Για τη χρήση της υπηρεσίας απαιτείται σύνδεση και εγγραφή με τραπεζικό λογαριασμό ή κάρτα. Για τις χρεώσεις των υπηρεσιών ακολουθείται η πολιτική «pay as you go» δηλαδή καταβάλλουμε ένα ποσό ανάλογα με το χρόνο που κρατάμε δεσμευμένους κάποιους πόρους αλλά και το βαθμό χρήσης

που κάνουμε στους πόρους αυτούς.

Χρησιμοποιήσαμε κυρίως τρεις από αυτές τις υπηρεσίες:

Dataproc: Η υπηρεσία αυτή μας επιτρέπει να δημιουργήσουμε και να διαχειριστούμε cluster που τρέχουν τα Apache Spark και Hadoop. Η δημιουργία ενός cluster για αυτά τα frameworks η οποία κατά κανόνα αποτελεί μια χρονοβόρα διαδικασία γίνεται εντελώς αυτοματοποιημένα σε μηδαμινό χρόνο επιτρέποντας στο χρήστη να επικεντρωθεί στην ουσία της εργασίας του, παρά στα εργαλεία που χρησιμοποιούνται. Σημαντικά στοιχεία αποτελούν ακόμα η δυνατότητα της υπηρεσίας για αναπροσαρμογή του μεγέθους της συστοιχίας μας, με την προσθήκη ή και την αφαίρεση κόμβων καθώς και ο συνδυασμός της υπηρεσίας με άλλες.

Compute Engine: Η υπηρεσία αυτή επιτρέπει την εκκίνηση, τον τερματισμό την επαναφορά και την σύνδεση μέσω ssh σε κάποια από τις εικονικές μηχανές που μας ανήκουν. Ακόμα μπορούμε να αναπροσαρμόσουμε τους πόρους μιας μηχανής και φροντίζει να μας ενημερώνει αν κάτι τέτοιο κριθεί κατάλληλο. Τέλος μας παρέχει αναπαράσταση της χρήσης που κάνουν οι μηχανές στις διάφορες μονάδες τους.

Storage: Καθιστά δυνατή την αποθήκευση αρχείων τα οποία μπορούν άμεσα να μεταφερθούν σε κάποιο από τα έργα των χρηστών. Έτσι αποφεύγεται η συνεχής μεταφόρτωση δεδομένων από ένα τοπικό μηχάνημα στο σύννεφο, γεγονός αρκετά χρονοβόρο για μεγάλα αρχεία όπως τα δεδομένα της βάσης κάποιας εφαρμογής. Επιπλέον οι χρήστες μπορούν να κρατήσουν στο Storage «σενάρια» (scripts) που χρησιμοποιούνται για παράδειγμα στην περαιτέρω παραμετροποίηση των cluster κατά τη δημιουργία τους μέσω του Dataproc.

Κεφάλαιο 3ο : Ανάλυση - Σχεδίαση

3.1 Ανάλυση

3.1.1 Απαιτήσεις κοινωνικού δικτύου

Για να είναι χρήσιμο και λειτουργικό ένα κοινωνικό δίκτυο θα πρέπει να διαθέτει κάποια βασικά χαρακτηριστικά και λειτουργίες:

- Θα πρέπει να επιτρέπει την δικτύωση μεταξύ των χρηστών μέσω της σύναψης κάποιας σχέσης.
- Θα πρέπει ο χρήστης να μπορεί να κάνει δημοσιεύσεις και αυτές να εμφανίζονται σε όλους τους χρήστες που τον ακολουθούν.
- Να προσφέρει τη δυνατότητα αναζήτησης χρηστών όσο και δημοσιεύσεων.
- Να διαθέτει μια λειτουργική και φιλική προς το χρήστη διεπαφή.

3.1.2 Το κοινωνικό δίκτυο Jitter

Η web εφαρμογή που αναπτύχθηκε για την παρούσα διπλωματική εργασία ονομάστηκε Jitter. Λειτουργεί σαν κοινωνικό δίκτυο και προσφέρει τις κύριες λειτουργίες που αναφέρθηκαν παραπάνω. Στόχος μας ήταν εξαρχής η υλοποίηση των κύριων χαρακτηριστικών ενός δικτύου προκειμένου να μετρηθεί η αποδοτικότητα τους υπό φορτίο.

Όπως αναφέραμε και προηγουμένως πρόκειται για μια πλατφόρμα που προσομοιώνει τη λειτουργία του γνωστού μας Twitter. Συγκεκριμένα διαθέτει μία σελίδα για είσοδο - log in - και μία για εγγραφή - sign up -, όπου οι χρήστες εισάγουν τα στοιχεία τους, αυτά ελέγχονται και έπειτα μεταφέρονται στην κεντρική σελίδα όπου παρουσιάζονται τα μηνύματα των χρηστών εκείνων, που έχουν επιλέξει να ακολουθούν, με χρονολογική σειρά. Επίσης από αυτή τη σελίδα ο χρήστης έχει τη δυνατότητα να δημοσιεύσει ο ίδιος κάποιο προσωπικό μήνυμα. Έπειτα ο κάθε χρήστης έχει την προσωπική σελίδα του, όπου απεικονίζονται μόνο τα μηνύματα που έχει δημοσιεύσει εκείνος. Από αυτές τις σελίδες έχουμε τη δυνατότητα να ακολουθήσουμε - follow - είτε να σταματήσουμε να ακολουθούμε - unfollow - ένα χρήστη. Για κάθε προσωπικό μήνυμα υποστηρίζονται ο σχολιασμός και η αναδημοσίευση καθώς και η διαγραφή αν πρόκειται φυσικά για μήνυμα του ίδιου χρήστη. Ένα ακόμα χαρακτηριστικό είναι η δυνατότητα για αναζήτηση η οποία είναι εφικτή από κάθε σελίδα του δικτύου. Η αναζήτηση μπορεί να γίνει με βάση το όνομα

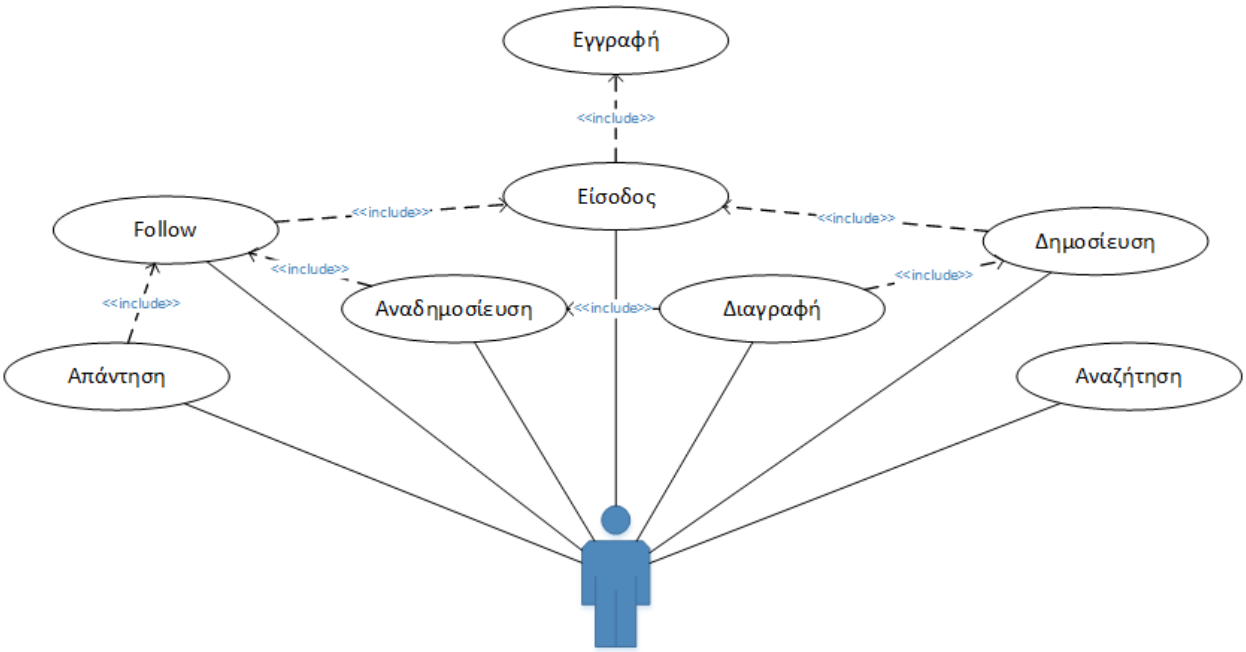
κάποιου χρήστη - username -είτε κάποιο "hashtag" στοιχείο που εισήγαγε πρώτο το ίδιο το Twitter άλλωστε. Όλα τα παραπάνω συνθέτουν το βασικό σκελετό του κοινωνικού δικτύου που χρειαζόμαστε προκειμένου να ξεκινήσουμε την μελέτη μας.

3.1.3 Σενάρια χρήσης

Οι λειτουργίες που υποστηρίζονται στην εφαρμογή είναι οι εξής:

- Εγγραφή και σύνδεση
- Δημοσίευση μηνύματος με δυνατότητα εισαγωγής hashtags
- Σύναψη σχέσης με άλλο χρήστη
- Αναζήτηση άλλων χρηστών και δημοσιεύσεων
- Αναδημοσίευση μηνύματος
- Απάντηση σε μήνυμα
- Διαγραφή μηνύματος
- Προβολή της αρχικής σελίδας του συνδεδεμένου χρήστη
- Προβολή του προφίλ για κάθε χρήστη
- Αποσύνδεση από το δίκτυο

Στο παρακάτω διάγραμμα UML αναλύονται τα κυριότερα σενάρια χρήσης που χρησιμοποιήθηκαν για την εξαγωγή των απαιτήσεων.



Εικόνα 11: Διάγραμμα σεναρίων χρήσης

3.1.3.1 Εγγραφή χρήστη

3.1.3.1.1 Ροή γεγονότων

Χρήστης	Client side	Server side
1. Εκκίνηση browser 2. Πληκτρολόγηση διεύθυνσης ιστοτόπου		
		3.Μετάβαση στη φόρμα εισόδου
4. Επιλογή συνδέσμου ‘Sign up’		
		5.Μετάβαση στη φόρμα εγγραφής.
6. Συμπλήρωση πεδίων i. Όνομα χρήστη ii. Πραγματικό όνομα iii. Διεύθυνση email iv. Κωδικός πρόσβασης		
	7. Έλεγχος στοιχείων 8. Αποστολή στο Server	
		9. Έλεγχος μοναδικότητας στοιχείων 10. Δημιουργία εγγραφής χρήστη στη βάση δεδομένων. 11. Επιστροφή μηνύματος επιτυχίας
	12. Αίτηση στο Server για την αρχική σελίδα	

3.1.3.1.2 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση φόρμας

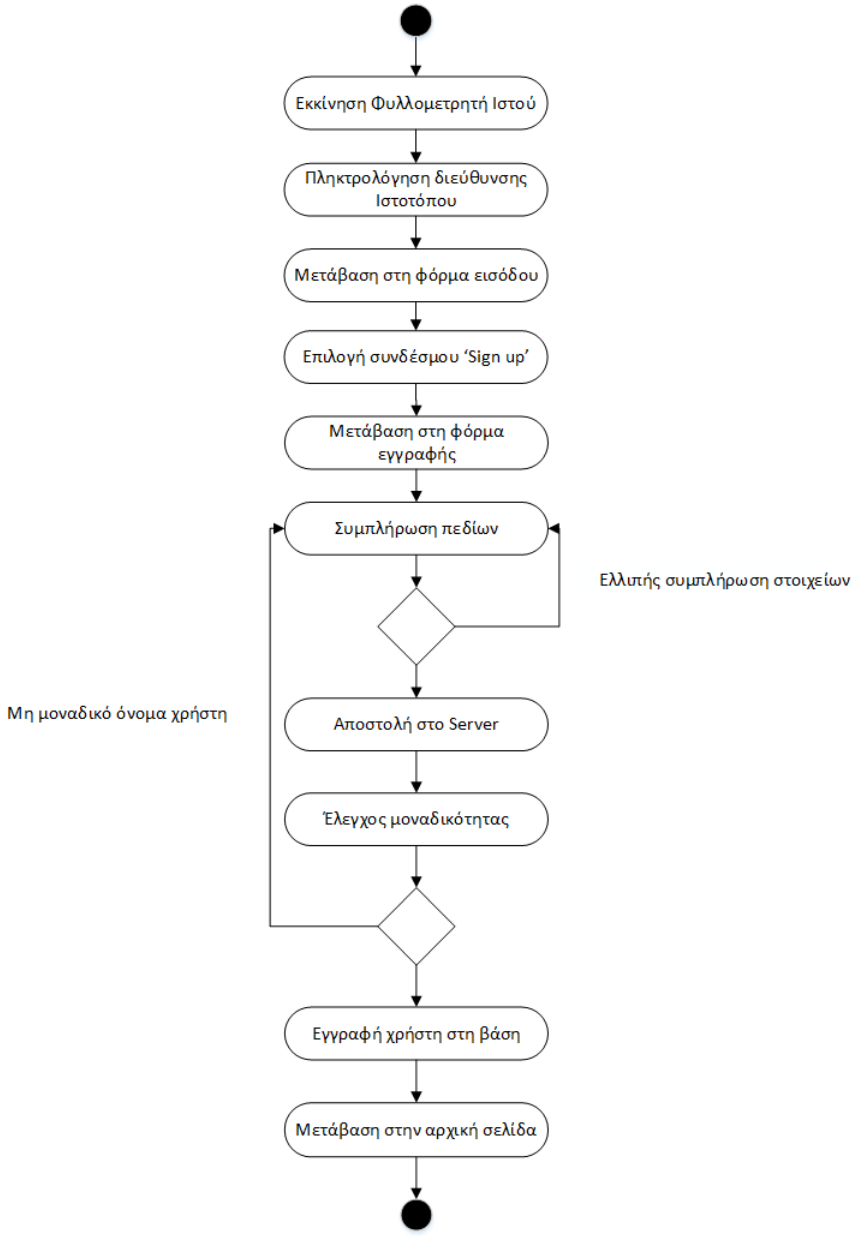
Χρήστης	Client Side
	8. Προβολή συγκεκριμένου μηνύματος σφάλματος στην ίδια σελίδα

3.1.3.1.3 Ροή διαχείρισης σφάλματος – Μη μοναδικό όνομα χρήστη

Χρήστης	Client Side	Server Side
		10. Επιστροφή κατηγορίας σφάλματος username

	11. Προβολή συγκεκριμένου μηνύματος στη ίδια σελίδα	

3.1.3.1.4 Διάγραμμα δραστηριοτήτων



Εικόνα 12: Εγγραφή χρήστη

3.1.3.2 Σύνδεση χρήστη

3.1.3.2.1 Ροή γεγονότων

Χρήστης	Client Side	Server Side
1. Εκκίνηση browser 2. Πληκτρολόγηση		

διεύθυνσης		
		3. Επιστροφή φόρμας εισόδου
4. Συμπλήρωση πεδίων i. Όνομα χρήστη ii. Κωδικός πρόσβασης		
	5. Έλεγχος στοιχείων 6. Αποστολή στο Server	
		7. Έλεγχος διαπιστευτηρίων 8. Επιστροφή μηνύματος επιτυχίας
	9. Αίτηση στο Server για την αρχική σελίδα	

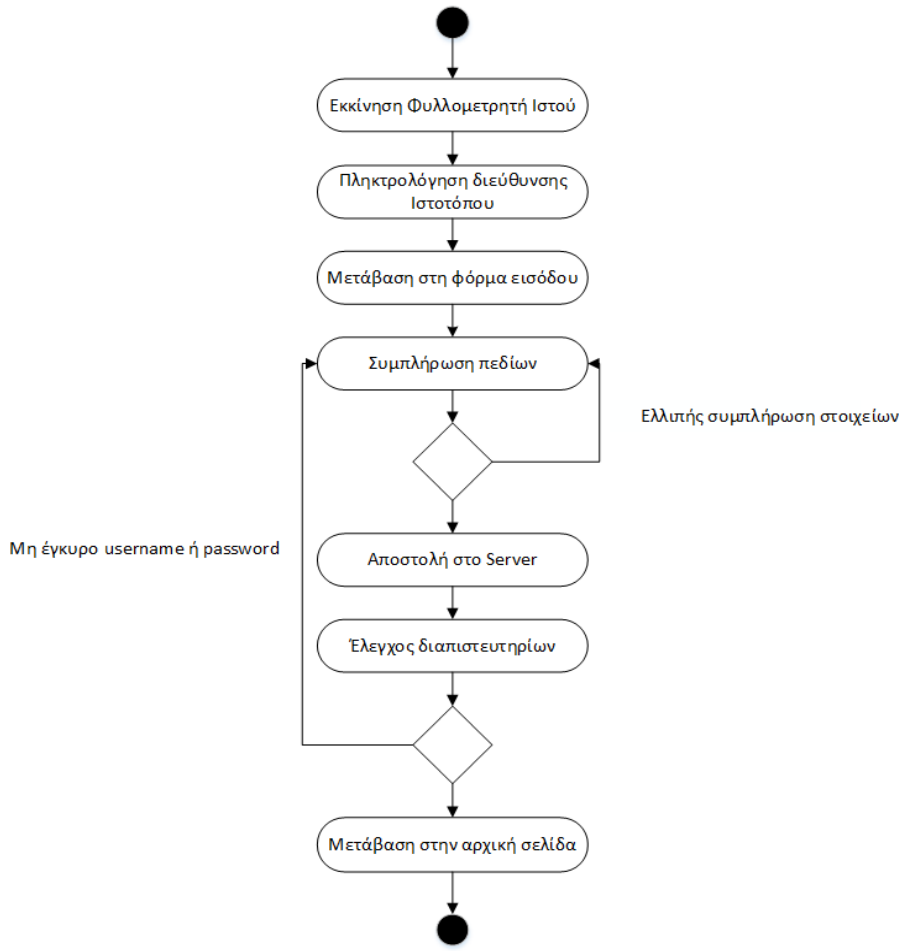
3.1.3.2.2 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση στοιχείων

Χρήστης	Client Side
	6. Επιστροφή συγκεκριμένου μηνύματος σφάλματος στην ίδια σελίδα

3.1.3.2.3 Ροή διαχείρισης σφάλματος – Μη έγκυρα διαπιστευτήρια

Χρήστης	Client Side	Server Side
		8. Επιστροφή κατηγορίας προβλήματος (username ή password)
	9. Προβολή συγκεκριμένου μηνύματος στη ίδια σελίδα	

3.1.3.2.4 Διάγραμμα δραστηριοτήτων



Εικόνα 13: Είσοδος χρήστη

3.1.3.3 Δημοσίευση μηνύματος

3.1.3.3.1 Ροή γεγονότων

Χρήστης	Client Side	Server Side
1. Πληκτρολόγηση μηνύματος		
	2. Έλεγχος για κενό περιεχόμενο 3. Αποστολή στο Server	
		4. Έλεγχος session 5. Δημιουργία νέας εγγραφής μηνύματος 6. Εύρεση hashtags στο tweet 7. Δημιουργία νέων

		hashtags ή προσθήκη του tweet σε ήδη υπάρχοντα 8. Επιστροφή επιβεβαίωσης
	9. Αίτηση ανανέωσης της αρχικής σελίδας	
		10. Επιστροφή αρχικής σελίδας

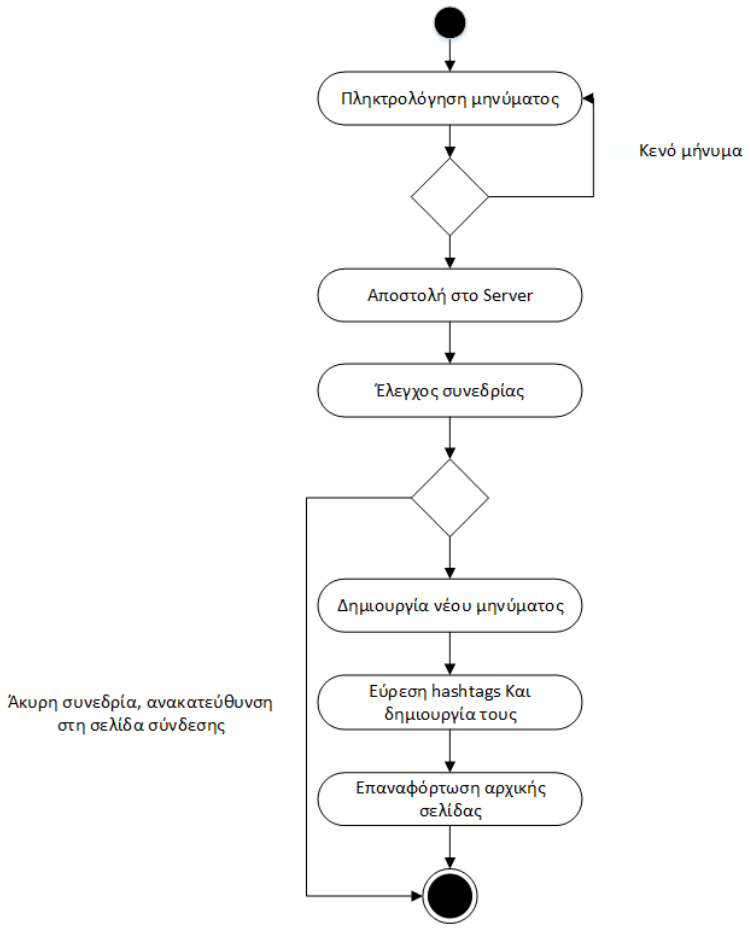
3.1.3.3.2 Ροή διαχείρισης σφάλματος – Ελλιπής συμπλήρωση στοιχείων

Χρήστης	Client Side
	3. Επιστροφή μηνύματος σφάλματος στην ίδια σελίδα

3.1.3.3.3 Ροή διαχείρισης σφάλματος – Λήξη συνεδρίας

Χρήστης	Server Side
	5. Ανακατεύθυνση στη σελίδα σύνδεσης

3.1.3.3.4 Διάγραμμα δραστηριοτήτων



Εικόνα 14: Δημοσίευση Μηνύματος

3.1.3.4 Απάντηση σε μήνυμα

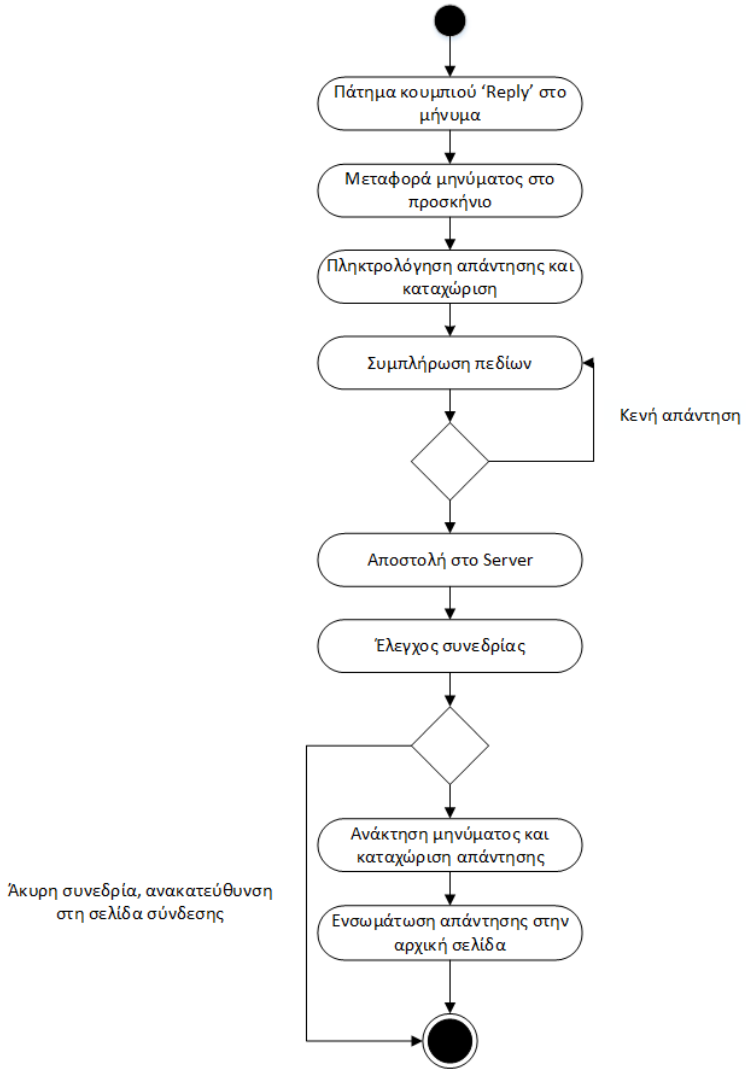
3.1.3.4.1 Ροή γεγονότων

Χρήστης	Client Side	Server Side
1. Πάτημα κουμπιού ‘Reply’ σε κάποιο μήνυμα		
	2. Μεγέθυνση μηνύματος και μεταφορά του στο προσκήνιο	
3. Πληκτρολόγηση απάντησης και καταχώριση		
	4. Έλεγχος συμπλήρωσης πεδίων 5. Αποστολή στο Server	
		6. Έλεγχος session

		7. Ανάκτηση tweet, δημιουργία απάντησης και καταχώριση 8. Επιστροφή της απάντησης
	9. Ενσωμάτωση της απάντησης στην όψη	

Η ροή σφάλματος αφορά τις ίδιες περιπτώσεις που αναλύσαμε και παραπάνω (έλεγχος συμπλήρωσης και έλεγχος συνεδρίας) οπότε και παραλείπεται. Το ίδιο συμβαίνει και σε όλα τα σενάρια που ακολουθούν.

3.1.3.4.2 Διάγραμμα Δραστηριοτήτων



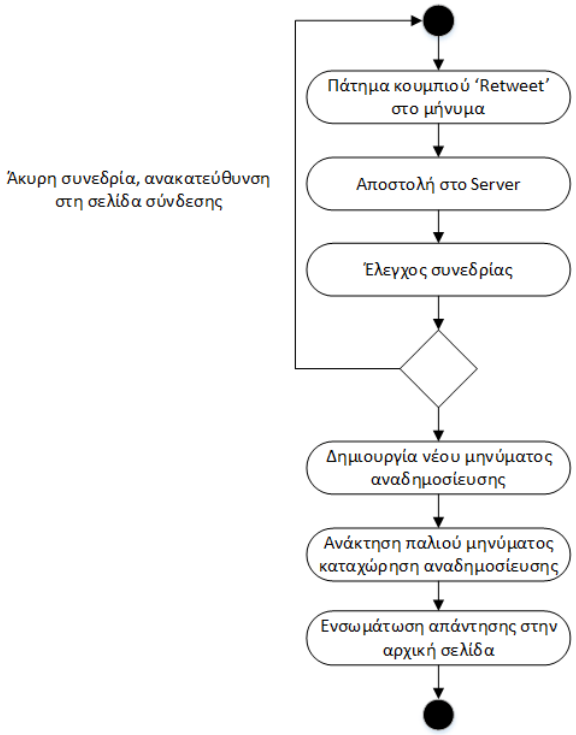
Εικόνα 15: Απάντηση σε μήνυμα

3.1.3.5 Αναδημοσίευση μηνύματος

3.1.3.5.1 Ροή Γεγονότων

Χρήστης	Client Side	Server Side
1. Επιλογή μηνύματος και πάτημα κουμπιού ‘Retweet’		
	2. Αποστολή στοιχείων στο Server	
		3. Έλεγχος session 4. Δημιουργία νέου μηνύματος αναδημοσίευσης 5. Ανάκτηση παλιού μηνύματος και καταχώρηση αναδημοσίευσης 6. Επιστροφή επιβεβαίωσης
	7. Αίτημα επαναφόρτωσης αρχικής σελίδας	
		8. Αποστολή αρχικής σελίδας

3.1.3.5.2 Διάγραμμα δραστηριοτήτων



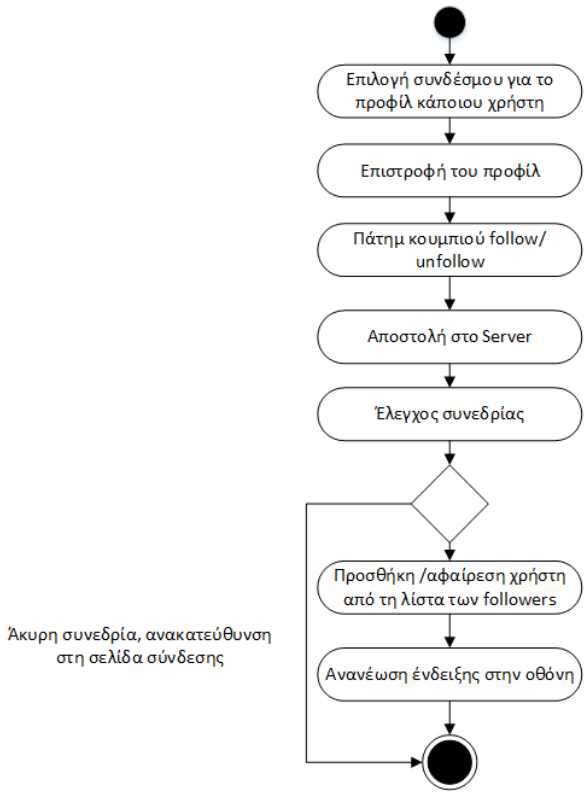
Εικόνα 16: Αναδημοσίευση μηνύματος

3.1.3.6 Λειτουργία Follow/ Unfollow

3.1.3.6.1 Ροή Γεγονότων

Χρήστης	Client Side	Server Side
1. Επιλογή συνδέσμου για το προφίλ κάποιου χρήστη		
		2. Επιστροφή προφίλ του χρήστη
3. Πάτημα του κουμπιού follow/unfollow		
	4. Αποστολή στο Server	
		5. Έλεγχος session 6. Προσθήκη/αφαίρεση του επιλεγμένου χρήστη στη λίστα followers 7. Επιστροφή
	8. Ανανέωση ένδειξης	

3.1.3.6.2 Διάγραμμα Δραστηριοτήτων



Εικόνα 17: Λειτουργία Follow

3.1.3.7 Διαγραφή Δημοσίευσης

3.1.3.7.1 Ροή γεγονότων

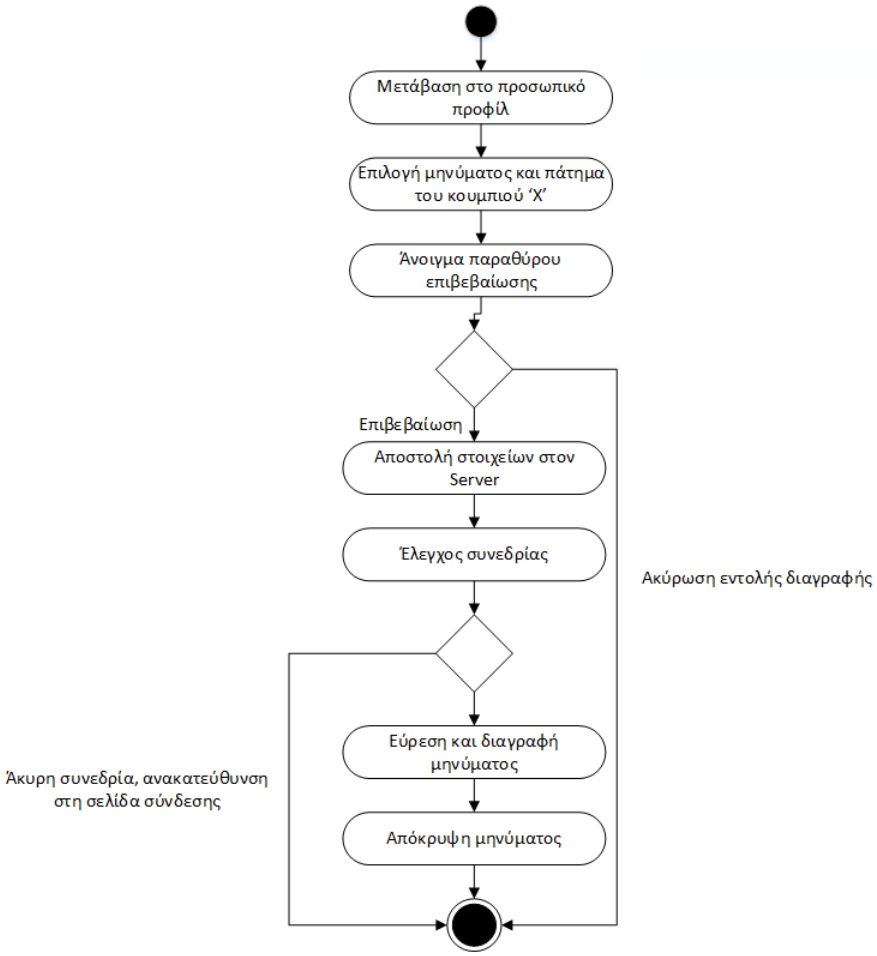
Χρήστης	Client Side	Server Side
1. Μετάβαση στο προφίλ του από οποιαδήποτε σελίδα 2. Επιλογή μηνύματος και πάτημα κουμπιού 'X'		
	3. Άνοιγμα παραθύρου επιβεβαίωσης	
4. Επιβεβαίωση		
	5. Αποστολή στοιχείων της δημοσίευσης στο Server	
		6. Εύρεση και διαγραφή μηνύματος 7. Αποστολή

		επιβεβαίωσης
	8. Απόκρυψη μηνύματος	

3.1.3.7.2 Εναλλακτική ροή

Χρήστης	Client Side
4. Ακύρωση εντολής διαγραφής	

3.1.3.7.3 Διάγραμμα δραστηριοτήτων



Εικόνα 18: Διαγραφή δημοσίευσης

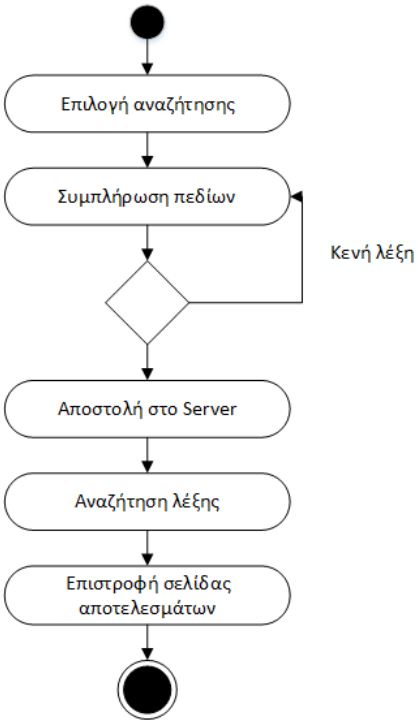
3.1.3.8 Αναζήτηση

3.1.3.8.1 Ροή Γεγονότων

Χρήστης	Client Side	Server Side
1. Επιλογή της μπάρας αναζήτησης και καταχώριση λέξης (όνομα		

χρήστη ή hashtag)		
	3. Έλεγχος για κενή λέξη 4. Αποστολή στο Server	
		4. Αναζήτηση λέξης 6. Συγκέντρωση και επιστροφή αποτελεσμάτων

3.1.3.8.1 Διάγραμμα δραστηριοτήτων



Εικόνα 19: Αναζήτηση

3.1.3.9 Αποσύνδεση από το δίκτυο

Χρήστης	Server Side
1. Πάτημα κουμπιού 'Log Out' από οποιαδήποτε σελίδα	
	2. Ακύρωση της συνεδρίας 3. Επιστροφή της σελίδας σύνδεσης

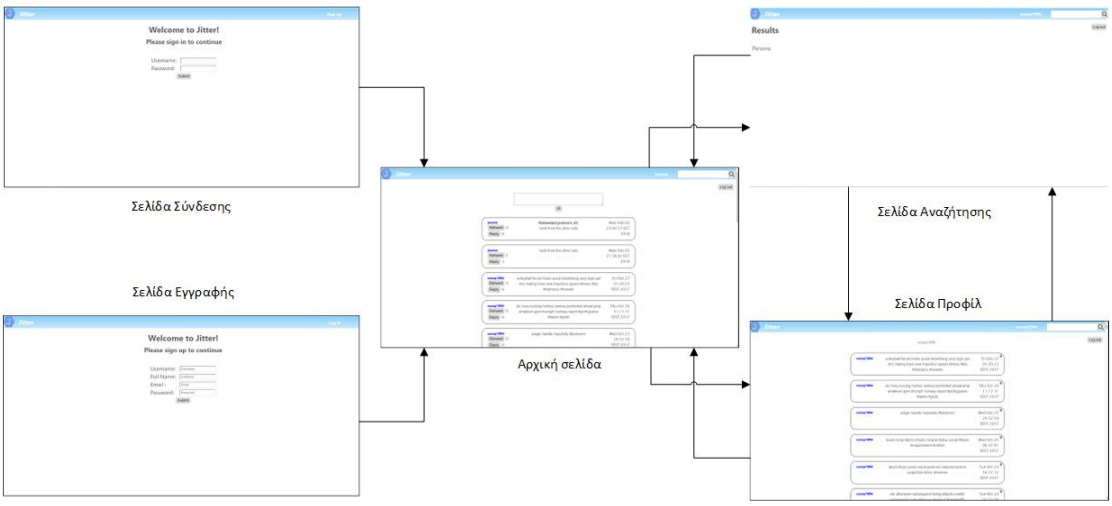
3.2 Σχεδίαση

3.2.1 Η διεπαφή του κοινωνικού δικτύου

Η διεπαφή του κοινωνικού δικτύου αποτελείται από τις παρακάτω σελίδες.

- **Η σελίδα σύνδεσης,** ο χρήστης θα μπορεί να συνδεθεί σε υπάρχοντα λογαριασμό, είτε να επιλέξει να μεταβεί στη σελίδα εγγραφής. Θα εμφανίζεται όταν κάποιος πληκτρολογεί τη διεύθυνση του δικτύου στο φυλλομετρητή ιστού.
- **Η σελίδα εγγραφής,** ο χρήστης θα μπορεί να δημιουργήσει νέο λογαριασμό είτε να επιστρέψει στη σελίδα σύνδεσης.
- **Η αρχική σελίδα,** αποτελεί την πιο σημαντική σελίδα του δικτύου. Από εδώ ο χρήστης θα μπορεί να κάνει κάποια δημοσίευση και να παρακολουθεί όλες τις δημοσιεύσεις των χρηστών που ακολουθεί. Επίσης θα έχει τη δυνατότητα να κάνει κάποια αναζήτηση όπως και να αποσυνδεθεί από το δίκτυο κάτι που θα ισχύει και για τις επόμενες σελίδες. Αν έχει προηγηθεί είσοδος στο δίκτυο τότε ο χρήστης εισάγεται κατευθείαν σε αυτή τη σελίδα.
- **Η σελίδα προφίλ,** αποτελεί την προσωπική σελίδα κάθε χρήστη όπου εμφανίζονται όλες οι δημοσιεύσεις του. Από εδώ μπορούμε να ακολουθήσουμε κάποιον χρήστη, είτε να επεξεργαστούμε τις δημοσιεύσεις μας αν πρόκειται για τη δική μας σελίδα προφίλ.
- **Η σελίδα αναζήτησης,** εδώ εμφανίζονται όλα τα αποτελέσματα μιας αναζήτησης είτε για όνομα χρήστη είτε για κάποιο hashtag.

Ακολουθεί διάγραμμα όπου φαίνονται όλες οι σελίδες της εφαρμογής και η μεταξύ τους συσχέτιση.



Εικόνα 20: Οι μεταβάσεις μεταξύ των σελίδων του ιστοτόπου

3.2.2 Ελεγκτές

Ακολουθώντας το πρότυπο της MVC αρχιτεκτονικής δημιουργήσαμε τέσσερις κλάσεις controller, που ουσιαστικά αντιστοιχούν μία σε κάθε σελίδα της εφαρμογής. Λόγω της ομοιότητας των σελίδων εγγραφών και σύνδεσης, ο χειρισμός τους εντάχθηκε σε έναν ελεγκτή.

Όνομα κλάσης	CredController
Μέθοδοι	• login Έλεγχος διαπιστευτηρίων και σύνδεση • signup Έλεγχος διπλής εγγραφής και σύνδεση • logout Ακύρωση της συνεδρίας

Συνεχίζουμε με τον ελεγκτή για την αρχική σελίδα.

Όνομα κλάσης	HomeController	
Πεδία	myQuery	Query
Μέθοδοι	• home Σχηματίζει την αρχική σελίδα • jit Καταχωρεί είτε ένα νέο μήνυμα είτε νέα αναδημοσίευση • reply Καταχωρεί μία νέα απάντηση στο tweet στο οποίο έγινε • fetch Ανταποκρίνεται σε κλήση AJAX για να φέρει νέα σελίδα με tweets • tweetsPage Επιστρέφει τη σελίδα με tweets που ζητείται	

Ακολουθεί ο ελεγκτής του προφίλ.

Όνομα κλάσης	ProfilController
Μέθοδοι	• profil Επιστρέφει τα μηνύματα του χρήστη • follow Απαντά σε κλήση AJAX και ανανεώνει τη σχέση μεταξύ δύο χρηστών • delete Απαντά σε κλήση AJAX και διαγράφει ένα μήνυμα

Τέλος έχουμε τον controller για την αναζήτηση.

Όνομα κλάσης	SearchController
Μέθοδοι	• search Πραγματοποιεί αναζήτηση είτε για όνομα χρήστη είτε για κάποιο hashtag

3.2.3 Μοντέλο - Σχήμα βάσης δεδομένων

Τα δεδομένα του κοινωνικού δικτύου οργανώθηκαν σε 3 κλάσεις που αποτέλεσαν και τις αντίστοιχες οντότητες στη βάση δεδομένων μας. Αυτές ήταν οι κλάσεις User, Tweet και Hashtag για την αναπαράσταση αντίστοιχα, των χρηστών, των μηνυμάτων που δημοσιεύονται και των hashtags που εμπεριέχονται στα μηνύματα αυτά.

Όνομα κλάσης	User	
Πεδία	• id • realName • userName • lastLoginDate • firstLoginDate • password • email • followers	ObjectId String String Date Date Int String List<String>
Μέθοδοι	• getRealName • setRealName • getUserName • setUserName • getLastLoginDate • setLastLoginDate • getFirstLoginDate • setFirstLoginDate • getPassword • setPassword • setFollowers • getFollowers • addFollower • deleteFollower • print	

Ενδεικτικά θα σχολιάσουμε κάποιες επιλογές που παρουσιάζουν ενδιαφέρον. Για κάποιο χρήστη μαζί με όλα τα χαρακτηριστικά που τον αφορούν κρατάμε επιπλέον τα ονόματα όλων των χρηστών που ακολουθεί. Αυτό γίνεται προκειμένου εύκολα να

μπορούμε να σχηματίζουμε την αρχική σελίδα κάθε χρήστη συγκεντρώνοντας όλα τα tweets, αυτών που ακολουθεί. Το γεγονός ότι κρατάμε μία λίστα με ονόματα είναι κάτι που μας επιτρέπει η JSON αναπαράσταση των δεδομένων στη MongoDB, πετυχαίνοντας έτσι καλύτερο "locality" και άρα ταχύτητα. Εκμεταλλευόμαστε έτσι τον πιο φυσικό τρόπο με τον οποίο μπορούν να αναπαραστήσουν οι μη σχεσιακές βάσεις δεδομένων τις ένα προς πολλά σχέσεις - one to many relationships, αποφεύγοντας τον κατακερματισμό σε πολλά tables και τα σύνθετα - join - queries. Κατά αυτόν τον τρόπο μειώνεται κιόλας η αναντιστοιχία μεταξύ του κώδικα εφαρμογής και του στρώματος αποθήκευσης.

Όνομα κλάσης	Tweet	
Πεδία	• id • userName • text • date • location • replies • retweets	ObjectId String String Date Int[] List<String[]> List<ObjectId>
Μέθοδοι	• getUserName • setUserName • getText • setText • getDate • setDate • getLocation • setLocation • getReplies • addReply • getRetweets • addRetweet	

Για κάθε μήνυμα τώρα, έχουμε επιλέξει και πάλι με την ίδια λογική να κρατήσουμε τις απαντήσεις στο ίδιο έγγραφο, όπως επίσης διατηρούμε και μια λίστα με τα κλειδιά των μηνυμάτων που αποτελούν αναδημοσιεύσεις αυτού. Από τα παραπάνω μπορεί λοιπόν να καταλάβει κανείς πως κάθε αναδημοσίευση αποτελεί ένα νέο μήνυμα - tweet - σε αντίθεση με μια απάντηση που είναι αποκλειστικά συνδεδεμένη με το μήνυμα στο οποίο αναφέρεται.

Όνομα κλάσης	Hashtag	
	• id	ObjectId

Πεδία	• text • jits	String List<ObjectId>
Μέθοδοι	• getText • setText • getJits • addJit • removeJit	

Για κάθε hashtag κρατάμε και μία λίστα με τα κλειδιά των μηνυμάτων στα οποία εμφανίζεται, κάτι που μας διευκολύνει αρκετά στην λειτουργία της αναζήτησης μέσω κάποιου hashtag. Τέλος παραθέτουμε την κλάση που αφορά τη σύνδεση του server με τη MongoDB.

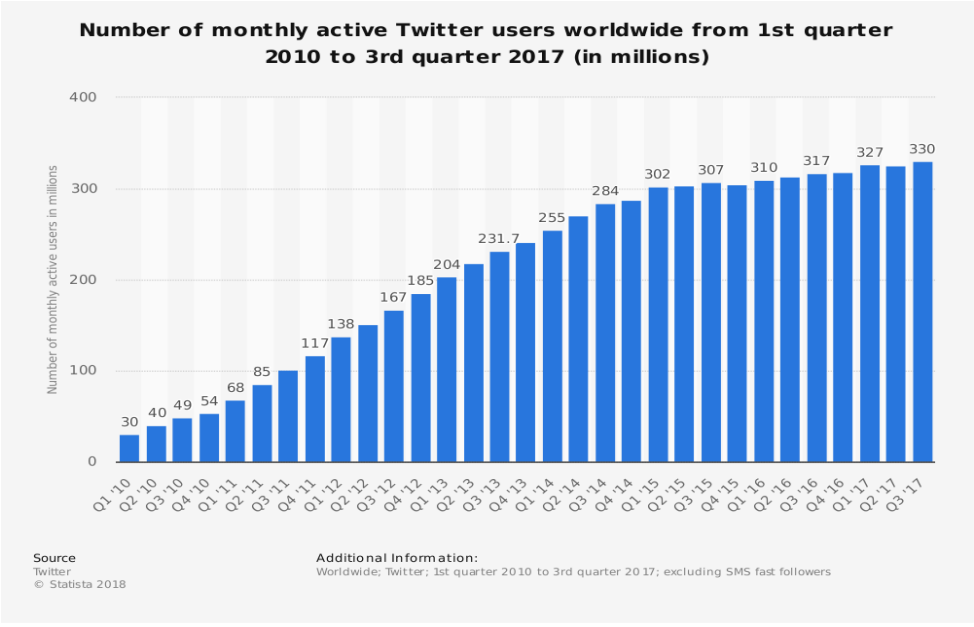
Όνομα κλάσης	MongoMyClient
Μέθοδοι	• mongoDbFactory • mongoTemplate

Κεφάλαιο 4ο : Υπόβαθρο

Σε αυτό το κεφάλαιο αναλύουμε καταρχήν το πρόβλημα στην απόδοση των κοινωνικών δικτύων, αναφέρουμε που εντοπίζεται αυτό και κατόπιν αναλύουμε κάποιες τεχνοτροπίες που μπορούν να δώσουν λύσεις.

4.1 Προσδιορισμός προβλήματος

Έως αυτό το σημείο έχουμε σχεδιάσει την εφαρμογή και έχουμε βεβαιωθεί πως λειτουργεί όπως θα θέλαμε. Το γεγονός όμως ότι μπορούμε να αντιμετωπίσουμε αποτελεσματικά τις ανάγκες που έχει ένα μικρό σύνολο χρηστών δε μας εξασφαλίζει καθόλου ότι κάτι τέτοιο θα είναι εφικτό και στο μέλλον λόγω μιας αύξησης των ενεργών χρηστών της εφαρμογής μας. Συνεπώς κρίνεται επιτακτικό να γνωρίζουμε ποιο είναι το όριο στο φορτίο που μπορούμε να διαχειριστούμε καθώς και ποιες είναι οι επιλογές μας προκειμένου το σύστημα μας να ανταποκριθεί σε ενδεχόμενη αύξηση του φορτίου. Ουσιαστικά ασχολούμαστε με την επεκτασιμότητα δηλαδή την ικανότητα ενός συστήματος να ανταπεξέρχεται στο αυξανόμενο φορτίο με ανάλογη αύξηση των πόρων του. Το πρόβλημα είναι κάτι παραπάνω από υπαρκτό. Ενδεικτικά αναφέρουμε την αύξηση που καλείται να αντιμετωπίσει το Twitter. Όπως βλέπουμε στην παρακάτω εικόνα μέσα σε διάστημα 7 ετών οι ενεργοί χρήστες του δικτύου εντεκαπλασιάστηκαν. Από 30 εκατομμύρια το πρώτο τρίμηνο του 2010 σε 330 εκατομμύρια το τρίτο τρίμηνο του 2017.



Εικόνα 21: Μηναία ενεργοί χρήστες του Twitter παγκοσμίως την περίοδο 2010-2017

Οι δύο λειτουργίες του δικτύου που συγκεντρώνουν το μεγαλύτερο ενδιαφέρον είναι η δημοσίευση μηνυμάτων και η παρουσίαση της αρχικής σελίδας -timeline- όπου ο κάθε χρήστης βλέπει τα μηνύματα των λογαριασμών που έχει επιλέξει να ακολουθεί. Με βάση στοιχεία του 2012 η ανάρτηση μηνύματος συγκέντρωνε 4,6 χιλιάδες αιτήσεις το δευτερόλεπτο, ενώ οι αναγνώσεις της αρχικής σελίδας έφταναν τις 300 χιλιάδες/δευτερ. Βλέπουμε λοιπόν πως οι αναγνώσεις του timeline δεν είναι απλά πιο σύνθετες λόγω του ότι χρειάζεται να συγκεντρώσουμε και να ταξινομήσουμε ως προς το χρόνο μηνύματα διαφορετικών χρηστών έναντι απλά μιας εισαγωγής στη βάση που απαιτεί μία δημοσίευση, αλλά είναι και σαν ενέργεια δύο τάξεις μεγέθους πιο συχνή. Από όλα αυτά τα δεδομένα έχουμε σχηματίσει πλέον μια πρώτη άποψη για το ποιο πρόκειται να είναι το πιο απαιτητικό κομμάτι που θα συναντήσουμε και εμείς.

Αρχικά το Twitter είχε υιοθετήσει μια συμβατική υλοποίηση όπου κάθε νέο μήνυμα προκαλούσε απλά μία εγγραφή στη βάση, ενώ για τις αιτήσεις των χρηστών για το timeline τους πραγματοποιείτο ένα σύνθετο query. Πρώτα εντοπίζονταν όλοι οι χρήστες που ακολουθούσε κάποιος και έπειτα επιστρεφόντουσαν τα tweets αυτών ταξινομημένα ως προς το χρόνο. Η δυσκολία αυτού του συστήματος να ανταποκριθεί στο αυξανόμενο φορτίο οδήγησε σε μια πρακτική όπου για κάθε χρήστη διατηρείται μια προϋπολογισμένη όψη του timeline του. Οποτεδήποτε δημοσιεύεται ένα νέο tweet αναζητείται σε ποιους χρήστες πρέπει να παραδοθεί. Με αυτό τον τρόπο επιβαρύνεται η λειτουργία του posting που είναι απλούστερη και πολύ πιο σπάνια. Αν και στη μέση περίπτωση μια τέτοια προσέγγιση λειτουργεί ικανοποιητικά δε συμβαίνει το ίδιο και για δημοσιεύσεις δημοφιλών ατόμων. Για ένα χρήστη με εκατομμύρια ακολούθους μια δημοσίευση μεταφράζεται αυτόματα σε εκατομμύρια εγγραφές στη βάση. Έτσι λοιπόν το Twitter οδηγήθηκε σε μια υβριδική προσέγγιση, όπου εξαίρεσε δημοφιλείς κόμβους από την παραπάνω διαδικασία, τα μηνύματα των οποίων εντάσσονται στα timeline των ακολούθων τους την ώρα που αυτά ζητούνται όπως συνέβαινε και στην πρώτη προσέγγιση.

4.2 Οριζόντια και κατακόρυφη κλιμάκωση

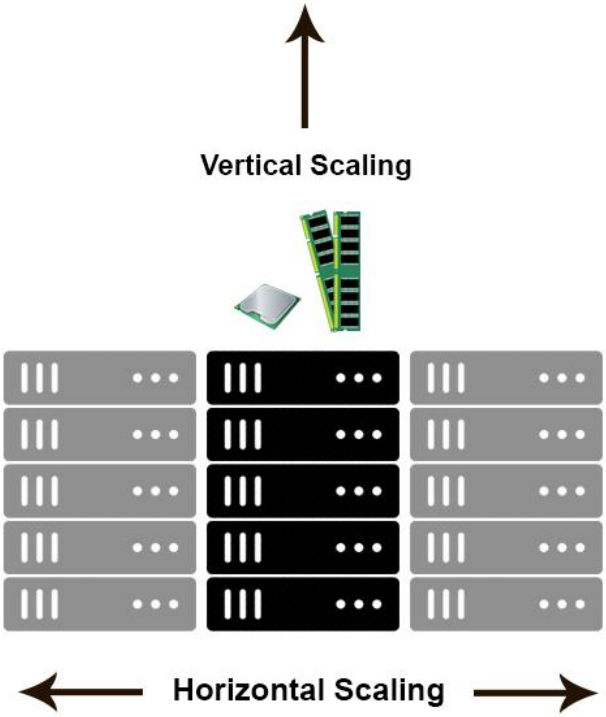
Μία πρώτη σκέψη για επίτευξη καλύτερων επιδόσεων θα ήταν η μετάβαση σε ένα ισχυρότερο υπολογιστικό μηχάνημα. Έτσι σε ενδεχόμενο διπλασιασμό του φορτίου θα έπρεπε να υπάρχει ανάλογος διπλασιασμός των ικανοτήτων (CPU, RAM, Δίσκοι) του υπολογιστικού συστήματος. Αν και εξαρχής αυτή η συλλογιστική μοιάζει απλή και λογική υπάρχουν συγκεκριμένα προβλήματα. Καταρχάς λόγω συμφόρησης –

bottleneck- διπλασιασμός των ικανοτήτων δεν σημαίνει αναγκαία και διαχείριση διπλάσιου βάρους. Κυριότερα το κόστος της αναβάθμισης δεν αυξάνεται γραμμικά. Δηλαδή ένα μηχάνημα διπλάσιων ικανοτήτων υπερβαίνει το κόστος δύο απλών μηχανημάτων. Συνεπώς ένα τέτοιο σύστημα δε θα ήταν επεκτάσιμο, δε θα μπορούσε να διαχειριστεί δηλαδή επιπλέον φορτίο με ανάλογα επιπλέον πόρους.

Όταν μιλάμε τώρα για horizontal scaling αναφερόμαστε στην προσθήκη επιπλέον μηχανημάτων, κόμβων, σε μια ήδη υπάρχουσα συστοιχία υπολογιστών -cluster- που εξυπηρετεί μία εφαρμογή. Έτσι το φορτίο κατανέμεται σε όλους τους κόμβους. Συνήθως πρόκειται για απλά μηχανήματα χωρίς εξεζητημένες δυνατότητες ώστε το κόστος να διατηρηθεί χαμηλό.

Ακόμα τα μηχανήματα αυτά μπορεί να βρίσκονται διασκορπισμένα σε διαφορετικές γεωγραφικές τοποθεσίες ώστε να μη χρειάζεται αιτήματα χρηστών να ταξιδεύουν σημαντικές αποστάσεις μέσω του δημόσιου διαδικτύου, εισάγοντας έτσι σημαντικές καθυστερήσεις στην εξυπηρέτηση τους. Επιπλέον υπάρχει η δυνατότητα νέα μηχανήματα να τίθενται διαθέσιμα είτε να παύουν την λειτουργία τους με δυναμικό τρόπο ανάλογα με τις ανάγκες της εφαρμογής. Κατά αυτόν τον τρόπο μπορούν να αντιμετωπιστούν αποτελεσματικά περίοδοι υψηλής κίνησης στην εφαρμογή, όπως συμβαίνει συχνά στα κοινωνικά δίκτυα λόγω κάποιου γεγονότος. Τέλος είναι δυνατόν μηχανήματα να θέτονται περιοδικά εκτός λειτουργίας προκειμένου να γίνει κάποια αναβάθμιση λογισμικού ή συντήρηση, χωρίς να απειλείται η συνεχιζόμενη εξυπηρέτηση της εφαρμογής. Στον αντίποδα τώρα η οριζόντια κλιμάκωση εισάγει μεγάλη πολυπλοκότητα καθώς και καθυστέρηση λόγω πρώτα της ανάγκης εκτέλεσης μια ενέργειας σε όλους τους κόμβους και έπειτα της σύνθεσης και παράδοσης του αποτελέσματος.

Η τάση τα τελευταία χρόνια δείχνει να ευνοεί την οριζόντια κλιμάκωση καθώς πέρα του χαμηλότερου κόστους, έχουν αναπτυχθεί και τα συστήματα εκείνα που επιτρέπουν την αποτελεσματικότερη διαχείριση τέτοιων κατανεμημένων τοπολογιών.

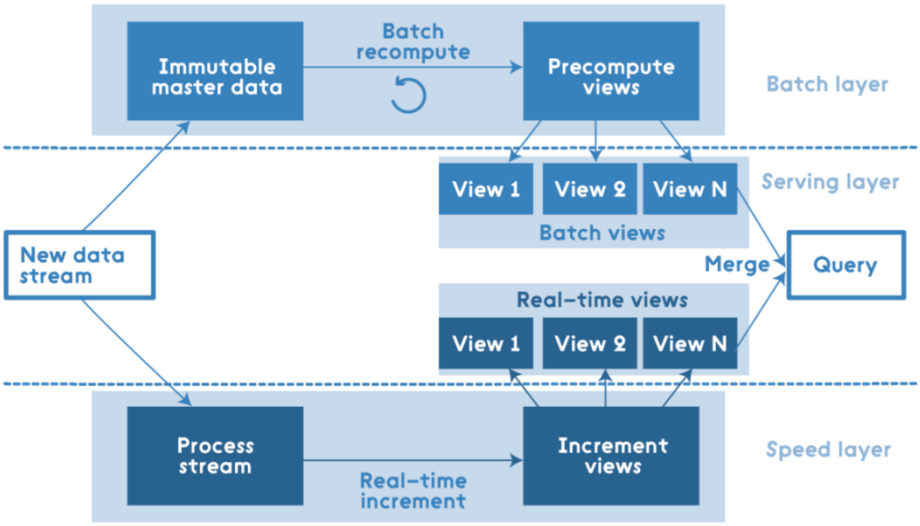


Εικόνα 22: Horizontal vs Vertical Scaling

4.3 Λάμδα αρχιτεκτονική

Πρόκειται για μία αρχιτεκτονική επεξεργασίας μεγάλου όγκου δεδομένων η οποία ενσωματώνει τα πλεονεκτήματα τόσο της μαζικής επεξεργασίας όσο και αυτής του πραγματικού χρόνου. Στόχοι αυτής είναι η δημιουργία συστημάτων γενικού σκοπού με δυνατότητες κλιμακωσιμότητας και ανθεκτικότητας στα σφάλματα.

Ουσιαστικά αποτελείται από τρία διαφορετικά στρώματα, το στρώμα μαζικής επεξεργασίας, το στρώμα πραγματικού χρόνου και το στρώμα εξυπηρέτησης. Το πρώτο αναλύει προϋπάρχοντα δεδομένα και παράγει κάποιο αποτέλεσμα, συμπληρώνεται έπειτα με το αποτέλεσμα της επεξεργασίας των δεδομένων πραγματικού χρόνου. Το στρώμα εξυπηρέτησης αναλαμβάνει να συγχωνεύσει αυτές τις όψεις και να απαντήσει σε κάποιο εισερχόμενο ερώτημα.



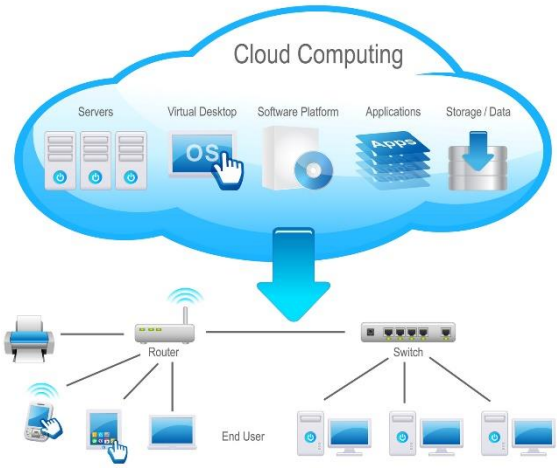
Εικόνα 23: Στρώματα της λάμδα αρχιτεκτονικής

Όσο αφορά τον υπολογισμό των αρχικών σελίδων των χρηστών, ο μεγάλος όγκος δεδομένων από ένα σημείο και έπειτα δε μας επιτρέπει να συνθέτουμε τις αρχικές σελίδες τη στιγμή που αυτές ζητηθούν. Απ' την άλλη, η batch επεξεργασία επιστρέφει πεπαλαιωμένα αποτελέσματα. Έτσι λοιπόν στρεφόμαστε στη λάμδα αρχιτεκτονική. Το στρώμα μαζικής επεξεργασίας που παρουσιάζει μεγαλύτερη ανοχή στα σφάλματα αναλαμβάνει την επεξεργασία των ιστορικών δεδομένων, ενώ το στρώμα ταχύτητας συλλέγει τα νέα tweets που καταφθάνουν και φροντίζει για το καθένα να βρει όλους τους χρήστες στους οποίους πρέπει να φτάσει δηλαδή όλους εκείνους που ακολουθούν το χρήστη που έκανε τη δημοσίευση.

Τέλος αρνητικά για αυτή την αρχιτεκτονική θεωρούνται η ανάγκη για ανάπτυξη και συντήρηση πολλών διαφορετικών συστημάτων τα οποία όμως εκτελούν την ίδια ουσιαστικά λειτουργία. Κάτι τέτοιο αυξάνει την πολυπλοκότητα του συστήματος και τα κόστη συντήρησής του.

4.4 Υπολογιστικό Νέφος

Με τον όρο αυτό αναφερόμαστε στη διάθεση υπολογιστικών πόρων ή και υπηρεσιών υψηλού επιπέδου μέσω του διαδικτύου. Υπηρεσίες υπολογιστικού νέφους προσφέρουν μεγάλοι πάροχοι με δικά τους



Εικόνα 24: Υπολογιστικό νέφος

ιδιόκτητα κέντρα υπολογιστών. Άλλες εταιρίες/οργανισμοί επιλέγουν να μεταφέρουν εκεί τα υπολογιστικά τους συστήματα προκειμένου να αποφύγουν τις αναγκαίες δαπάνες για την υποδομή και τη συντήρηση τους. Επιπλέον κάτι τέτοιο επιτρέπει στους πελάτες του υπολογιστικού νέφους να επικεντρωθούν στις κύριες δραστηριότητες τους και όχι στα συστήματα που τις υποστηρίζουν.

Η παροχή τέτοιων υπηρεσιών καθίσταται εφικτή λόγω της εικονοποίησης. Με την εικονοποίηση υπολογιστών αναφερόμαστε στη δημιουργία άλλων ισοδύναμων μέσω λογισμικού, που έχουν τη δυνατότητα να εξυπηρετούν τις ίδιες λειτουργίες όπως και ένα φυσικό μηχάνημα. Χάρη σε αυτή πολλοί μικροί φυσικοί εξυπηρετητές εφαρμογών μπορούν να αντικατασταθούν από ένα μεγαλύτερο όπου κάθε ένας τρέχει σε μια διαφορετική και απομονωμένη εικονική μηχανή. Κατά αυτό τον τρόπο επιτυγχάνεται αυξημένη χρήση ακριβών υπολογιστικών πόρων όπως οι επεξεργαστικές μονάδες, και συνεπώς μικρότερα κόστη λειτουργίας για όλους τους ενοποιημένους εξυπηρετητές. Αυτό το γεγονός εκμεταλλεύονται οι πάροχοι υπηρεσιών υπολογιστικού νέφους επιτυγχάνοντας μέσω της λειτουργίας τους σε μεγάλη κλίμακα να μειώσουν σημαντικά το μοναδιαίο κόστος.

Άλλα πλεονεκτήματα που προκύπτουν από την εικονοποίηση είναι η εξοικονόμηση ενέργειας λόγω της ενοποιημένης λειτουργίας, η ευελιξία των εικονικών μηχανών όπου η αναπροσαρμογή των πόρων μπορεί να αλλάξει άμεσα καθώς και η ευκολία μεταφοράς τους μεταξύ διαφορετικών φυσικών μηχανημάτων.

Το cloud computing είναι συνδεμένο με την παροχή συγκεκριμένων μοντέλων υπηρεσιών. Αυτά είναι:

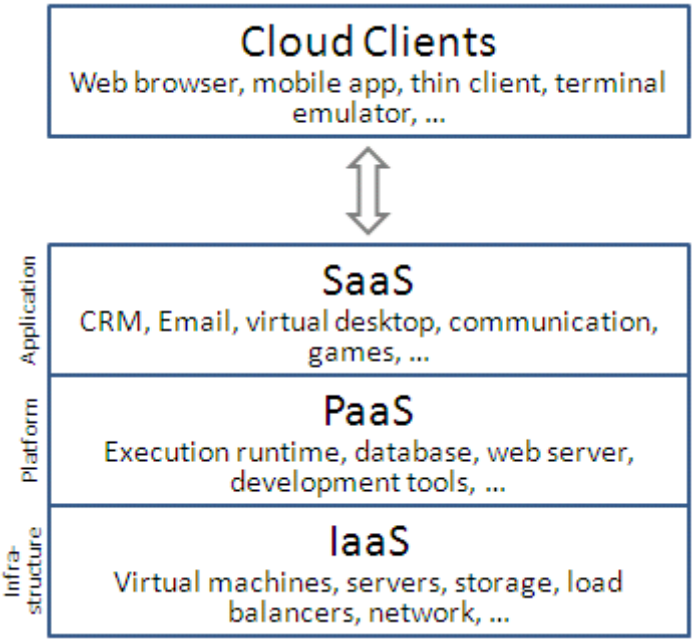
Infrastructure as a service (Iaas): Πρόκειται για την παροχή μόνο των υπολογιστικών πόρων (εικονικές μηχανές) όπου ο χρήστης μπορεί να ανεβάσει και να εκτελέσει οποιοδήποτε λογισμικό επιθυμεί.

Platform as a service (PaaS): Παρέχεται μια ολοκληρωμένη πλατφόρμα εργαλείων την οποία χρησιμοποιεί ο πελάτης για να εκτελέσει την εφαρμογή του. Μια τέτοια πλατφόρμα μπορεί να είναι μία βάση δεδομένων, μία στοίβα εργαλείων για μηχανική μάθηση ή για μεγάλα δεδομένα.

Software as a service (SaaS): Παρέχεται ολοκληρωμένο λογισμικό που μπορεί να χρησιμοποιεί ο πελάτης. Ο πάροχος αναλαμβάνει τη διαχείριση τόσο της υποδομής όσο και της εφαρμογής.

Mobile "backend" as a service (MBaaS): Πρόκειται για ένα νέο μοντέλο παροχής υπηρεσιών για web και κινητές εφαρμογές. Αποτελείται από συμπληρωματικές

υπηρεσίες που μπορούν να συνδυαστούν για να ικανοποιήσουν τις ανάγκες των εφαρμογών.



Εικόνα 25: Αναπαράσταση των υπηρεσιών νέφους ως στοίβα

Κεφάλαιο 5ο : Υλοποίηση

5.1 Υλοποίηση Εφαρμογής

Σε αυτή την ενότητα θα προσπαθήσουμε μέσα από την ανάλυση κάποιων μεθόδων να δείξουμε τον τρόπο που λειτουργεί η εφαρμογή και την επίδραση που έχει στην ανάπτυξη η χρήση του Spring Framework.

```
@Controller
public class HomeController {

  private static final Logger logger =
Logger.getLogger(HomeController.class.getName());
  private static Query myQuery = null;
  ConfigurableApplicationContext ctx = new
AnnotationConfigApplicationContext(MongoMyClient.class);
  MongoOperations mongoOperation = (MongoOperations)
ctx.getBean("mongoTemplate");

  @RequestMapping(value="/Home",method=RequestMethod.GET)
  public String  home(Model m, HttpServletRequest request,
HttpServletRequest response){
      HttpSession session = request.getSession(false);
      if (session == null || session.getAttribute("userName") ==
null) return "restrict";
      String userName = (String) session.getAttribute("userName");
      session.setAttribute( "scroll", 1);
      m.addAttribute("userName", userName);
      List<Tweet> jits = new ArrayList<Tweet>();
      new Criteria();
      Query query = new
Query(Criteria.where("userName").is(userName));
      logger.info("Start getting user and followers");
      User user = mongoOperation.findOne(query, User.class);
      List<String> followers = user.getFollowers();
      logger.info("Start getting jits for all followers");
      myQuery = new
Query(Criteria.where("userName").in(followers));
      myQuery.with(new Sort(Sort.Direction.DESC, "date"));
      jits = tweetsPage(myQuery, 0, mongoOperation);
      logger.info("Finished collecting jits and sorting");
      m.addAttribute("results", jits);
      return "home";
  }

  @RequestMapping(value="/Fetch",method=RequestMethod.GET)
  @ResponseBody
  public String  fetch(Model m, HttpServletRequest request,
@RequestParam(value = "reply", required = false) String reply ,
@RequestParam(value = "reId", required = false) ObjectId id){
```

```
    HttpSession session = request.getSession(false);
    String userName = (String) session.getAttribute("userName");
    int z = (Integer)session.getAttribute(  "scroll" );
    List<Tweet> jits = new ArrayList<Tweet>();
    jits = tweetsPage(myQuery, z, mongoOperation);
    session.setAttribute( "scroll", ++z);
    return new Gson().toJson(jits);
  }
 //…
```

Τμήμα κώδικα του ελεγκτή αρχικής σελίδας

Αρχικά να αναφέρουμε ότι το παραπάνω τμήμα κώδικα ελέγχει για την ύπαρξη session και σε περίπτωση που δε το βρει επιστρέφει σελίδα άρνησης εισόδου. Διαφορετικά υπολογίζει το προφίλ του χρήστη του οποίου η αρχική σελίδα ζητείται επιστρέφοντας την πρώτη σελίδα που αποτελείται από 20 μηνύματα. Ο αριθμός της σελίδας που έχει επιστραφεί αποθηκεύεται στο session έτσι μία ενέργεια 'scroll down' από πλευράς χρήστη δημιουργεί μια ασύγχρονη κλήση AJAX που εξυπηρετείται από τη μέθοδο fetch που ακολουθεί. Η μέθοδος αυτή συμβουλεύεται το session για να βρει ποια σελίδα πρέπει να επιστρέψει. Κατά τα άλλα λειτουργεί όπως και η προηγούμενη.

Η επίδραση του Spring framework στο παραπάνω τμήμα κώδικα είναι καταλυτική. Αρχικά με το annotation @Controller η κλάση μας δηλώνεται ως ελεγκτής. Εντός της κλάσης τώρα έχουμε μεθόδους που δηλώνουν σε ποιο URI και σε ποια μέθοδο αίτησης αντιστοιχούν. Το παραπάνω απλοποιεί σημαντικά τη διαδικασία δρομολόγησης και εξοικονομεί χρόνο για τον προγραμματιστή. Τέλος η πρώτη μέθοδος επιστρέφει απλά μια συμβολοσειρά η οποία αντιστοιχεί στο όνομα της όψης που θέλουμε να επιστραφεί. Το Spring framework κάνει το ταίριασμα αυτόματα και προσθέτει στην επιλεγμένη όψη τα δεδομένα που έχουμε στείλει από τον ελεγκτή.

Όσο αφορά τη δεύτερη μέθοδο τώρα παρατηρούμε ότι σημειώνεται με το annotation @ResponseBody το οποίο δηλώνει πως ότι επιστρέψει αυτή η μέθοδος αποτελεί και το σώμα του response που επιστρέφουμε χωρίς να εμπλέκεται κάποια όψη. Έτσι μπορούμε να χειριστούμε τις κλήσεις AJAX. Πέρα από αυτά το Spring μας διευκολύνει αρκετά και στο χειρισμό των δομών της βάσης δεδομένων.

```
@Document(collection = "users")
public class User {

      @Id
      private ObjectId id;
      private String realName;
      private String userName;
      @DateTimeFormat(iso = ISO.DATE_TIME)
```

```
        private Date lastLoginDate;
        @DateTimeFormat(iso = ISO.DATE_TIME)
        private Date firstLoginDate;
        private int password;
        private String email;
        private List<String> followers;

        public User(){
              this.followers = new ArrayList<String>();
        }
 //…
```

Κλάση χρήστη

Από το παραπάνω τμήμα κώδικα βλέπουμε πως συσχετίσουμε την κλάση User με τη συλλογή users στη βάση χρησιμοποιώντας το annotation @Document. Τα πεδία της κλάσης αντιστοιχούν στα πεδία των εγγράφων της συλλογής. Επιπλέον προκειμένου να προσδιορίσουμε σε ποια συλλογή απευθύνεται ένα query απλώς περνάμε την αντίστοιχη κλάση σαν παράμετρο.

```
@RequestMapping(value="/Search",method=RequestMethod.POST)
public String  search(Model m, HttpServletRequest request,
@ModelAttribute("search") String searchString ){

//…

  if (searchString.startsWith("#")){
    m.addAttribute("title", searchString);
    new Criteria();
    Query query = new
Query(Criteria.where("text").is(searchString.substring(1)));
    Hashtag hashtag = mongoOperation.findOne(query, Hashtag.class);
    List<Tweet> jits = new ArrayList<Tweet>();
    if (hashtag != null){
      List<ObjectId> jitsToRemove = new ArrayList<ObjectId>();
      for (ObjectId jitId : hashtag.getJits()){
        new Criteria();
        query = new Query(Criteria.where("id").is(jitId));
        Tweet tweet = mongoOperation.findOne(query, Tweet.class);
        if (tweet != null)
          jits.add(tweet);
        else
          jitsToRemove.add(jitId);
      }
      for (ObjectId jitId : jitsToRemove)
        hashtag.removeJit(jitId);
        mongoOperation.save(hashtag);
      }
      m.addAttribute("jits", jits);
    }else{
      m.addAttribute("title", "Persons");
      Pattern pattern = Pattern.compile("^" + searchString);
```

```
        new Criteria();
        Query query = new
Query(Criteria.where("userName").regex(pattern));
        List<User> users = mongoOperation.find(query, User.class);
        m.addAttribute("results", users);
    }
//...
```

Υλοποίηση αναζήτησης

Παραπάνω βλέπουμε τον τρόπο που διαχωρίζεται η αναζήτηση μεταξύ χρηστών και hashtags ανάλογα με το αν προηγείται ένας χαρακτήρας '#'. Κατά την επιστροφή των μηνυμάτων που αντιστοιχούν σε ένα hashtag ανανεώνεται και η σχετική λίστα σβήνοντας τις αναφορές στα μηνύματα που έχουν διαγραφεί από το δίκτυο. Για την αναζήτηση χρηστών κατασκευάζουμε μία κανονική έκφραση που αντιστοιχεί σε όλα τα ονόματα που ξεκινούν με το όρισμα που έδωσε ο χρήστης και αναζητούμε αυτή στη βάση.

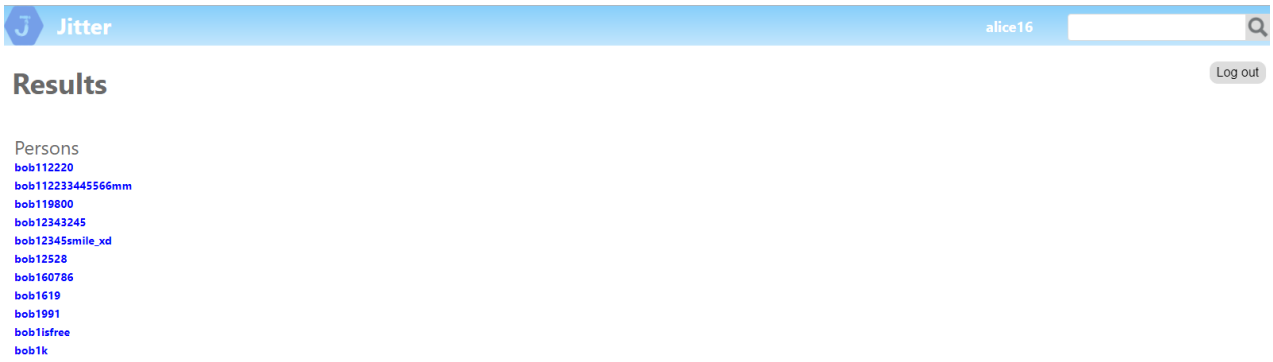
5.2 Παράδειγμα Χρήσης

Στη συνέχεια θα παρουσιάσουμε ένα σενάριο χρήσης του κοινωνικού δικτύου προκειμένου να γίνει κατανοητός ο τρόπος λειτουργίας του. Για το σκοπό αυτό θα χρειαστούμε δύο χρήστες. Ο Bob έχει ήδη λογαριασμό και χρησιμοποιεί τον ιστότοπο ενώ η Alice θα χρειαστεί να πραγματοποιήσει εγγραφή.

Εικόνα 26: Είσοδος του χρήστη Bob

Εικόνα 27: Εγγραφή του χρήστη Alice

Αφού η Alice εγγραφεί, αναζητά τον Bob και τον κάνει ‘follow’. Πλέον μπορεί να δει τα μηνύματά του στην αρχική της σελίδα.



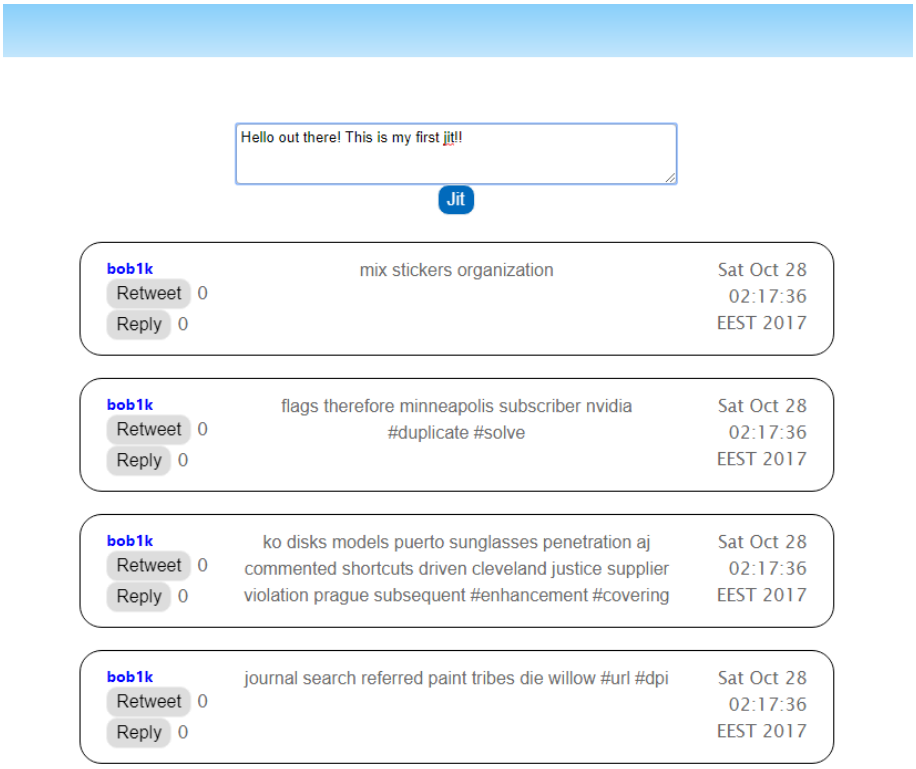
Εικόνα 28: Αναζήτηση του ονόματος χρήστη για τον Bob

Εικόνα 29: Προφίλ του χρήστη bob

Βρίσκει ένα μήνυμα του Bob που την ενδιαφέρει και πραγματοποιεί μία απάντηση. Η απάντηση καταχωρείται και εμφανίζεται αμέσως στην οθόνη της.

Εικόνα 30: Αρχική σελίδα της Alice, απάντηση σε μήνυμα του Bob

Έπειτα δημοσιεύει το πρώτο της μήνυμα, γνωστοποιώντας την παρουσία της.



Εικόνα 31: Δημοσίευση μηνύματος από την Alice

Όταν ο Bob συνδεθεί βλέπει την απάντηση της Alice, μπαίνει στο προφίλ της και την κάνει και εκείνος ‘follow’. Κατόπιν αποφασίζει να αναδημοσιεύσει το μήνυμά της για να την βοηθήσει να βρει ακολούθους.

Εικόνα 32: Αναδημοσίευση του μηνύματος της Alice

5.3 Δημιουργία των δεδομένων της βάσης

Προκειμένου να εκτιμήσουμε την απόδοση του συστήματος που υλοποιήσαμε πρέπει να δημιουργήσουμε συνθήκες φόρτου και να ελέγξουμε πως λειτουργεί υπό αυτές. Για το σκοπό αυτό κατασκευάσαμε μία βάση 100 χιλιάδων χρηστών όπου ο καθένας με τυχαίο τρόπο ακολουθεί έως και 10 χιλιάδες άλλους χρήστες. Κάθε χρήστης και πάλι τυχαία δημοσιεύει έως 1000 μηνύματα και ο συνολικός τους αριθμός ανήλθε στα 50 εκατομμύρια μηνύματα. Τα δεδομένα που χρησιμοποιήθηκαν για τη βάση προήλθαν από δύο αρχεία, ένα που περιείχε usernames και ένα άλλο που περιείχε 10 χιλιάδες αγγλικές λέξεις εκ των οποίων δημιουργήθηκαν τυχαία μηνύματα μήκους μέχρι 15 λέξεων. Κάθε μήνυμα επίσης περιείχε από 0 έως 4 hashtags. Τέλος σε περιορισμένο αριθμό μηνυμάτων προστέθηκαν έως 5 απαντήσεις και retweets.

Έγινε προσπάθεια βελτιστοποίησης των προγραμμάτων που υλοποίησαν το συγκεκριμένο κομμάτι προκειμένου να μειωθεί ο απαιτούμενος χρόνος και οι απαιτήσεις σε μνήμη που διαφορετικά οδήγησαν σε σφάλματα και εξαιρέσεις. Αυτό συνέβη διότι ο συνολικός όγκος της βάσης ανήλθε σε 25GB.

```
public static void main(String[] args) throws IOException,
FileNotFoundException {

  //..σύνδεση στη βάση, άνοιγμα αρχείων, αρχικοποίηση μεταβλητών

  User userNew;
  List<User> users = new ArrayList<User>();
  int j=0;
  while ((userName = reader.readLine()) != null ){
      if ((++j)%1000 == 0){
        mongoOperation.insertAll(users);
        users = new ArrayList<User>();
      }
      userNew = new User();
      Date date = new Date();
      password = generateWord();
      realName =  generateWord() + " " + generateWord() ;
      email = generateWord() + "@" + generateWord() + ".com";
      userNew.setUsername(userName);
      userNew.setRealName(realName);
      userNew.setPassword(password.hashCode());
      userNew.setEmail(email);
      userNew.setFirstLoginDate(date);
      userNew.setLastLoginDate(date);
      userNew.addFollower(userName);
      numOfFollowers = randNumOfFollowers.nextInt(10000);
      for (int i=0; i< numOfFollowers; i++){
```

```
            raf.seek(randPos.nextInt(fixedLength));
            raf.readLine();
              userNew.addFollower(raf.readLine());
        }
        users.add(userNew);
    }
    mongoOperation.insertAll(users);

    //.. κλείσιμο συνδέσεων και αρχείων
}
```

Δημιουργία χρηστών της εφαρμογής

Παραπάνω βλέπουμε τον κώδικα για τη δημιουργία των χρηστών. Αντιμετωπίσαμε εδώ δύο προβλήματα, αρχικά η εισαγωγή των χρηστών τον ένα μετά τον άλλο υπήρξε εξαιρετικά χρονοβόρα, οπότε επιλέχτηκε να κρατιούνται πρώτα σε μία λίστα και έπειτα να γίνεται η μαζική εισαγωγή τους. Κάτι τέτοιο όμως είχε πολύ υψηλές απαιτήσεις σε μνήμη καθώς η συλλογή των χρηστών φτάνει τα 10GB. Έτσι η μαζική εισαγωγή επιλέχτηκε να γίνεται ανά 1000 χρήστες. Ακολουθεί ο κώδικας για τη δημιουργία των tweets και των hashtags.

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

  //..σύνδεση στη βάση, άνοιγμα αρχείων, δηλώσεις μεταβλητών

  HashMap<String, Hashtag> map = new HashMap<String, Hashtag>(13334);
  while ((userName = reader.readLine()) != null ){
    if ((++k)%10000 == 0){
      commit(map,mongoOperation);
      map = new HashMap<String, Hashtag>(13334);
      System.gc();
    }
    numOfTweets = randNumOfTweets.nextInt(1000);
    for(int i=0; i<numOfTweets; i++){
      List<String> hashtags = new ArrayList<String>();
      jitText = "";
      Date date = new Date();
      numOfWords = 1 + randNumOfWords.nextInt(16);
      for(int j=0; j<numOfWords; j++){
            raf.seek(randPos.nextInt(limit));
            raf.readLine();
            jitText += raf.readLine() + " ";
      }
      numOfHashtags = randNumOfHashtags.nextInt(5);
      for(int j=0; j<numOfHashtags; j++){
            raf.seek(randPos.nextInt(limit));
            raf.readLine();
            Shashtag = raf.readLine();
            jitText += "#" + Shashtag + " ";
            hashtags.add(Shashtag);
      }
```

```
        jit = new Tweet();
        jit.setUserName(userName);
        jit.setDate(date);
        jit.setText(jitText);
        mongoOperation.insert(jit);
        for(String hashtagName : hashtags){
              Hashtag value = map.get(hashtagName);
              if (value != null){
                    value.addJit(jit.getId());
              }else{
                    value = new Hashtag();
                    value.setText(hashtagName);
                    value.addJit(jit.getId());
              }
              map.put(hashtagName, value);
        }
      }
    }
    commit(map,mongoOperation);

    //..
}

private static void commit(HashMap<String, Hashtag> map,
MongoOperations mongoOperation) {

        Iterator it = map.entrySet().iterator();
        Map.Entry pair;
        String key;
        Hashtag myValue;
        while (it.hasNext()) {
              pair = (Map.Entry)it.next();
              key = (String) pair.getKey();
              myValue = (Hashtag) pair.getValue();
              Query query = new Query(Criteria.where("text").is(key));
              Hashtag hashtag = mongoOperation.findOne(query,
Hashtag.class);
              if (hashtag == null){
                    hashtag = new Hashtag();
                    hashtag.setText(key);
                    for (ObjectId myId: myValue.getJits())
                          hashtag.addJit(myId);
                    mongoOperation.insert(hashtag);
              }else{
                    for (ObjectId myId: myValue.getJits())
                          hashtag.addJit(myId);
                    mongoOperation.save(hashtag);
              }
        }
        return;
}
```

Δημιουργία tweets και hashtags

Στον παραπάνω κώδικα παρατηρούμε πως κάθε tweet πρέπει να εισαχθεί πρώτα στη βάση προκειμένου να αποθηκεύσουμε το id στα hashtags τα οποία βρίσκονται σε αυτό. Όσο αφορά τα hashtags έχουμε δημιουργήσει μια δομή HashMap με κλειδί τον όρο του hashtag και value ένα αντικείμενο hashtag. Για κάθε νέο hashtag ελέγχουμε αν υπάρχει ήδη στη δομή οπότε προσθέτουμε το tweet στη λίστα του, διαφορετικά το προσθέτουμε στη δομή. Τέλος επειδή και πάλι αυτή η δομή δεν χωρούσε στη μνήμη την αποθηκεύουμε περιοδικά στη βάση και κάθε φορά ελέγχουμε αν κάποια λέξη υπάρχει ήδη στη βάση ώστε να μην δημιουργήσουμε διπλότυπα. Τέλος στον παρακάτω κώδικα παραθέτουμε τη δημιουργία των απαντήσεων και των retweets.

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

  //.. σύνδεση στη βάση

  String userName ;
  int j=0;
  Random randRetw = new Random(), randRepl = new Random();
  List<Tweet> tweets = new ArrayList<Tweet>();
  while ((userName = reader.readLine()) != null ){
      if ((++j)%1000 == 0){
            for(Tweet tweet : tweets) mongoOperation.save(tweet);
            tweets = new ArrayList<Tweet>();
      }
      Query query=new Query(Criteria.where("userName").is(userName));
      List<Tweet> jits = mongoOperation.find(query, Tweet.class);
      for(Tweet jit: jits){
        if ( randRetw.nextInt(10) == 0) addRetweets(jit, tweets,
mongoOperation);
        if (randRepl.nextInt(5) == 0) addReplies(jit, tweets,
mongoOperation);
  }
}
for(Tweet tweet : tweets) mongoOperation.save(tweet);

  // ..
}
```

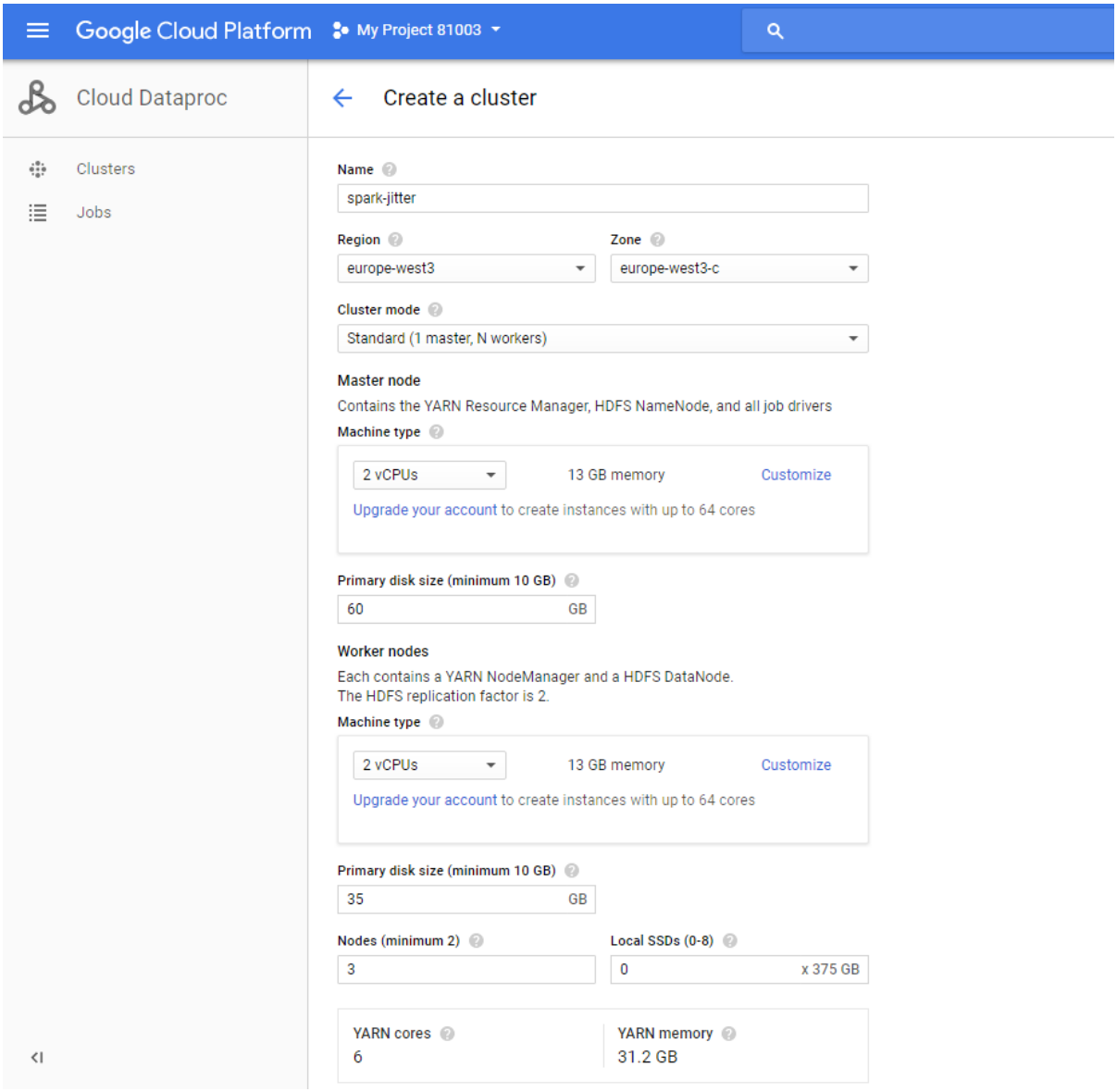
Δημιουργία απαντήσεων και retweets

5.4 Δημιουργία τοπολογίας στο Google Cloud Platform

Αφού συνδεθούμε στην υπηρεσία πρέπει να δημιουργήσουμε το cluster όπου θα λειτουργήσουν το Apache Spark και η βάση μας. Μεταβαίνουμε στην υπηρεσία Cloud Dataproc η οποία μας παρέχει την αυτοματοποιημένη δημιουργία τοπολογιών που τρέχουν τα Apache Spark και Apache Hadoop. Οι τοπολογίες αυτές χρησιμοποιούν ως διαχειριστή πόρων το Hadoop Yarn. Ένας κόμβος έχει αποκλειστικά το ρόλο του Master ενώ οι υπόλοιποι αυτόν του Worker. Για την

δημιουργία του cluster πρέπει να προσδιορίσουμε τον τύπο των μηχανημάτων και τους πόρους που θα διαθέτουν καθώς και τη φυσική τους τοποθεσία.

Επειδή κάναμε δοκιμαστική χρήση της υπηρεσίας είχαμε ως όριο τους 8 πυρήνες για όλη την τοπολογία μας και πίστωση 300 δολαρίων. Έτσι επιλέξαμε 4 διπύρηνα μηχανήματα υψηλής μνήμης 13GB RAM το καθένα. Από αυτά για την εκτέλεση των διεργασιών διατίθενται στο Yarn οι πυρήνες των Worker-nodes και ένα ποσοστό 0,8 της συνολικής τους μνήμης. Έτσι έχουμε 6 πυρήνες και 31.2GB μνήμης.



Εικόνα 33: Δημιουργία συστάδας στην υπηρεσία Dataproc

Χρειαζόμαστε όμως και τη MongoDB εγκατεστημένη σε κάθε μηχάνημα καθώς τα δεδομένα μας θα εδρεύουν εκεί και όχι στο HDFS το οποίο είναι ήδη διαθέσιμο.

Προκειμένου να αποφύγουμε να κάνουμε την εγκατάσταση της Mongo χειροκίνητα σε όλα τα μηχανήματα, δημιουργήσαμε κάποια script για να αυτοματοποιήσουμε τη διαδικασία, τα οποία κιόλας μπορούμε να τα συμπεριλάβουμε σαν 'Initialization action' κατά τη δημιουργία της τοπολογίας.

```
#!/bin/bash

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
echo "deb http://repo.mongodb.org/apt/debian jessie/mongodb-org/3.4
main" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
sudo apt-get update
sudo apt-get install -y mongodb-org=3.4.9 mongodb-org-server=3.4.9
mongodb-org-shell=3.4.9 mongodb-org-mongos=3.4.9 mongodb-org-
tools=3.4.9
mkdir -p /home/nickiemand16/mongodb/log
/home/nickiemand16/mongodb/data/db && touch
/home/nickiemand16/mongodb/log/mongo.log
```

Σημ. Εγκατάσταση Mongo και δημιουργία φακέλων δεδομένων και logs

Η συγκεκριμένη πρακτική μας εξοικονόμησε αρκετό χρόνο καθώς χρειάστηκε αρκετές φορές να καταστρέψουμε και να ξαναδημιουργήσουμε την τοπολογία μας προκειμένου να δοκιμάσουμε διαφορετικές παραμετροποιήσεις των μηχανημάτων αλλά και να δούμε πως λειτουργεί η εφαρμογή μας με replica sets και σε sharded cluster. Για το σκοπό αυτό τόσο τα scripts όσο και τα δεδομένα της βάσης κρατήθηκαν σε ένα bucket στο cloud storage που εδρεύει στην ίδια φυσική τοποθεσία με το cluster μας ώστε να είναι γρήγορη η ανάκτηση όλων των δεδομένων.

5.5 Δημιουργία Sharded cluster

Μετά την εγκατάσταση της MongoDB πρέπει να δημιουργήσουμε το sharded cluster. Καταρχάς στον κύριο κόμβο εγκαταστάθηκε ο config server ως replica set και αρχικοποιήθηκε με μοναδικό μέλος τον ίδιο.

```
mongod --configsvr --replSet conSer --dbpath mongodb/data/db-config

mongo --port 27019

rs.initiate({_id:"conSer", configsvr:true, members: [{_id:0, host:
"spark-jitter-m:27019" }]})
```

Έπειτα σε κάθε μηχάνημα κόμβο τοποθετήθηκε η διεργασία mongos που έχει το ρόλο του δρομολογητή ερωτημάτων. Στις παραμέτρους της εντολής δίνεται το όνομα του config server σαν replica set και προσδιορίζονται τα μέλη αυτού.

```
mongos --configdb conSer/spark-jitter-m:27019 --port 27018
```

Χρειαζόμαστε ένα δρομολογητή σε κάθε μηχάνημα προκειμένου οι διεργασίες του Spark να συνδέονται εκεί και να έχουν άμεση πρόσβαση στα δεδομένα του κόμβου που εδρεύουν και οι ίδιες ώστε να αποφευχθούν shuffle reads και writes μεταξύ των κόμβων.

Κατόπιν δημιουργούμε τα shards σε κάθε κόμβο.

```
mongod --shardsvr --directoryperdb --dbpath mongodb/data/db --port
27017
```

Έπειτα συνδεόμαστε σε ένα mongos, προσθέτουμε τα shards των Worker-nodes στη συστάδα, ενεργοποιούμε το sharding για τη βάση μας και κάνουμε shard τη συλλογή με τα μηνύματα και τους χρήστες έχοντας σαν κλειδί το username των χρηστών.

```
mongo --port 27018
sh.addShard("spark-jitter-w-0:27017")
sh.addShard("spark-jitter-w-1:27017")
sh.addShard("spark-jitter-w-2:27017")
sh.enableSharding("jitterN")
sh.shardCollection("jitterN.tweets",{ userName : 1 })
sh.shardCollection("jitterN.users", { userName : 1 })
```

Τέλος με την εντολή mongorestore συνδεόμαστε σε κάποιο mongos και επαναφέρουμε τα δεδομένα της βάσης τα οποία χωρίζονται κατευθείαν σε chunks και ισοκατανέμονται και στους τρεις κόμβους μας.

5.6 Υλοποίηση χωρίς το Spark

5.6.1 Υπολογισμός αρχικής σελίδας

Θα ελέγξουμε την απόδοση των δύο υλοποιήσεων σε τρία σενάρια. Το πρώτο αφορά την όπως διαπιστώσαμε πιο απαιτητική και συχνή λειτουργία του κοινωνικού μας δικτύου, που είναι η δημιουργία και παράδοση της αρχικής σελίδας για κάθε χρήστη. Για το σκοπό αυτό δημιουργήσαμε ένα εκτελέσιμο πρόγραμμα σε Java που δέχεται ως παράμετρο το username ενός χρήστη και επιστρέφει ταξινομημένα τα 20 πιο πρόσφατα tweets από τους χρήστες που έχει επιλέξει να ακολουθεί. Χρησιμοποιούμε ‘paging’ προκειμένου να χωρίσουμε το αποτέλεσμα σε σελίδες των 20 και ζητούμε την πρώτη από αυτές.

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

```
 //.. σύνδεση με βάση

  String userName = args[0];
  List<Tweet> jits = new ArrayList<Tweet>();
  Query query = new Query(Criteria.where("userName").is(userName));
  User user = mongoOperation.findOne(query, User.class);
  List<String> followers = user.getFollowers();

  myQuery = new Query(Criteria.where("userName").in(followers));
  myQuery.with(new Sort(Sort.Direction.DESC, "date"));
  jits = tweetsPage(myQuery, 0, mongoOperation);
  myQuery = new Query(Criteria.where("userName").in(followers));
  myQuery.with(new Sort(Sort.Direction.DESC, "date"));
  jits = tweetsPage(myQuery, 0, mongoOperation);
  //εκτύπωση αποτελεσμάτων

 // κλείσιμο συνδέσεων και εκτύπωση χρόνου εκτέλεσης..
}

private static List<Tweet> tweetsPage(Query query, int i,
MongoOperations mongoOperation) {

  Pageable pageable = new PageRequest(i,20);
  query.with(pageable);
  return mongoOperation.find(query, Tweet.class);
}
```

5.6.2 Εξαγωγή σκορ δημοφιλίας

Το δεύτερο σενάριο σύγκρισης είναι η παράδοση της αρχικής σελίδας πολλών χρηστών και η υλοποίηση του είναι παρόμοια του πρώτου. Η τρίτη εργασία που υλοποιήθηκε για σύγκριση είναι η εξαγωγή ενός σκορ δημοφιλίας για κάθε χρήστη της εφαρμογής. Το σκορ αυτό βασίστηκε σε τρεις παραμέτρους, τον αριθμό των μηνυμάτων που έχει δημοσιεύσει, τις απαντήσεις που έχει λάβει σε αυτά και τα retweets που έχουν γίνει σε δημοσιεύσεις του από άλλους χρήστες. Μεγαλύτερο βάρος σε αυτή τη βαθμολογία είχαν οι αναδημοσιεύσεις και λιγότερο ο αριθμός των μηνυμάτων.

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

  //.. σύνδεση με βάση

  String userName ;
  int numOfTw = 0, numOfRepl = 0, numOfRetw = 0;

            /* find score for all users */

  HashMap<String,AuxUsr> map = new HashMap<String,AuxUsr>(100000);
```

```
  AggregateIterable<Document> output = tweets.aggregate(Arrays.asList(
      new Document("$group", new Document("_id", "$userName")
          .append("tw_count", new Document ("$sum",1)))
  ));

  for (Document dbObject : output){
      numOfTw = dbObject.getInteger("tw_count");
      map.put(dbObject.getString("_id"),new AuxUsr(numOfTw,0,0, 0.25F *
numOfTw));
  }

  output = tweets.aggregate(Arrays.asList(
      new Document("$unwind", "$replies"),
      new Document("$group", new Document("_id", "$userName")
          .append("repl_count",new Document ("$sum",1)))
  ));

  for (Document dbObject : output){
      userName = dbObject.getString("_id");
      numOfRepl = dbObject.getInteger("repl_count");
      map.put(userName, map.get(userName).repl(numOfRepl));
  }

   output = tweets.aggregate(Arrays.asList(
      new Document("$unwind", "$retweets"),
      new Document("$group", new Document("_id", "$userName")
          .append("retw_count",new Document ("$sum",1)))
   ));

  for (Document dbObject : output){
      userName = dbObject.getString("_id");
      numOfRetw = dbObject.getInteger("retw_count");
      map.put(userName, map.get(userName).retw(numOfRetw));
  }

  map.entrySet().stream()
     .sorted(Map.Entry.comparingByValue((a,b) -> a.compare(b)))
     .limit(20)
     .forEach(e ->System.out.println(e.getKey()+e.getValue().print()));
}
```

Παραπάνω βλέπουμε πως για να βρούμε καθέναν από αυτούς τους αριθμούς υποβάλλουμε ένα query στη βάση μας χρησιμοποιώντας το aggregation framework. Σε κάθε query οι εγγραφές οργανώνονται με βάση το username και στη συνέχεια προστίθενται. Για την εύρεση των αριθμών των απαντήσεων και των retweets που έχει ο κάθε χρήστης το πρώτο βήμα πριν την άθροιση στο pipeline είναι το «άνοιγμα» των πινάκων των απαντήσεων και των retweets αντίστοιχα. Αυτό γίνεται με το κατηγόρημα “unwind”, που δημιουργεί π.χ. για ένα μήνυμα με 10 απαντήσεις 10

εγγραφές με όλα τα υπόλοιπα πεδία του tweet απαράλλαχτα αλλά μία απάντηση στο πεδίο replies.

Στη συνέχεια το αποτέλεσμα που μας επιστρέφεται το κρατάμε σε μία δομή HashMap όπου κλειδί είναι το username και value ένα αντικείμενο της βοηθητικής κλάσης AuxUsr που θα δούμε παρακάτω. Αυτό γίνεται προκειμένου για κάθε χρήστη να προσθέσουμε σε αποδοτικό χρόνο και τους αριθμούς των απαντήσεων και των retweets από τα επόμενα queries.

Τέλος ταξινομούμε τη δομή HashMap με βάση το score και επιστρέφουμε τα 20 πρώτα αποτελέσματα μαζί με τον αριθμό των tweets, των απαντήσεων και των retweets. Το συγκεκριμένο βήμα απλοποιήθηκε σημαντικά κάνοντας χρήση μεθόδων συναρτησιακής φύσης που είναι πλέον διαθέσιμες με την 8^{η} έκδοση της Java.

Ακολουθεί η κλάση AuxUsr. Ενδεικτικά παραθέτουμε τα πεδία και τρεις σημαντικές μεθόδους. Οι πρώτες δύο αφορούν την προσθήκη των απαντήσεων και των retweets αντίστοιχα στο αντικείμενο και η τρίτη την ταξινόμηση με βάση το σκορ.

```
public class AuxUsr {

	private int numOfTw;
	private int numOfRepl;
	private int numOfRetw;
	private float score;

	public AuxUsr repl(int repl){
		this.setRepl(repl);
		this.setScore(0.35F * repl);
		return this;
	}

	public AuxUsr retw(int retw){
		this.setRetw(retw);
		this.setScore(0.4F * retw);
		return this;
	}

	public int compare( AuxUsr b) {
	  if (this.getScore() > b.getScore())
		return -1;
	  else if (b.getScore() > this.getScore())
		return 1;
	  else
		return 0;
    }
}
```

5.7 Υλοποίηση με χρήση Apache Spark

5.7.1 Υπολογισμός αρχικής σελίδας

Πρώτο βήμα εδώ ήταν η σύνδεση της MongoDB με το Spark. Αυτό έγινε με χρήση του connector που έχει δημιουργήσει η ίδια η Mongo. Στη συνέχεια παραθέτουμε την παραμετροποίηση του SparkSession το οποίο και αποτελεί το εισαγωγικό σημείο για τον προγραμματισμού του Spark με το Dataset και Dataframe API.

```
SparkSession spark = SparkSession.builder()
       .master("yarn")
       .appName("helloSpark")
       .config("spark.mongodb.input.uri", "mongodb://spark-jitter-w-
0:27018,spark-jitter-w-1:27018,spark-jitter-w-2:27018/
jitterN.tweets?readPreference=nearest")

.config("spark.mongodb.input.partitioner","MongoShardedPartitioner")

.config("spark.mongodb.input.partitionerOptions.shardkey","userName")

       .getOrCreate();

System.setProperty("spark.mongodb.keep_alive_ms", "6000000");
SparkContext conf = spark.sparkContext();
JavaSparkContext ctx = new JavaSparkContext(conf);
```

Configuration του SparkSession

Στις παραπάνω γραμμές ορίζουμε το όνομα της εφαρμογής μας, καθορίζουμε από που θα αντλήσει το Spark τα δεδομένα και ορίζουμε τον τρόπο που θα γίνει ο διαχωρισμός τους σε τμήματα. Συγκεκριμένα δίνουμε στο Spark τους hosts και τις θύρες όπου βρίσκονται τα mongos και καθορίζουμε ως προεπιλογή ανάγνωσης το κοντινότερο. Για τον partitioner που θα χρησιμοποιηθεί επιλέγουμε τον MongoShardedPartitioner ο οποίος συνίσταται για τις περιπτώσεις που έχουμε sharded cluster καθώς χρησιμοποιεί τα ήδη υπάρχοντα chunks σαν partitions και παρέχει στο Spark μεταδεδομένα για τη θέση όπου βρίσκονται αυτά. Τέλος προσδιορίζουμε ποιο είναι το κλειδί, βάση του οποίου έγινε το sharding.

Υλοποιήσαμε στο Spark για την μέτρηση της απόδοσης τα ίδια σενάρια χρήσης όπως και με την απλή έκδοση. Αρχικά έχουμε την παράδοση της αρχικής σελίδας ενός χρήστη. Σημειώνεται ότι στις υλοποιήσεις των δύο εκδόσεων έγινε προσπάθεια να ακολουθηθεί όσο το δυνατόν η ίδια λογική προκειμένου η σύγκριση να είναι αντιπροσωπευτική.

```
String userName = args[0];
Dataset<User> users = MongoSpark.load(spark, ReadConfig
     .create(spark).withOption("collection", "users"), User.class);
Dataset<Tweet> tweets = MongoSpark.load(spark, ReadConfig
     .create(spark).withOption("collection", "tweets"), Tweet.class);

String[] array=users.filter(col("username").$eq$eq$eq(userName))
   .select("followers").first().getList(0)
   .stream().toArray(String[]::new);
tweets.filter(col("userName").isin(array))
  .select("userName","text","date")
  .sort(col("date").desc())
  .show(false);
```

Σχηματισμός αρχικής σελίδας χρήστη

Στο παραπάνω κομμάτι κώδικα αρχικά φορτώνουμε τις συλλογές των tweets και των users ως Datasets με προσδιορισμένο τύπο. Ουσιαστικά πρόκειται για αναφορές στις συλλογές καθώς δεν πρόκειται να φορτωθούν όλα τα δεδομένα από αυτές αν δεν απαιτηθούν. Στη συνέχεια φιλτράρουμε τη συλλογή των χρηστών για να βρούμε το user που μας δόθηκε ως παράμετρος και από το έγγραφο αυτό κρατάμε μόνο τη λίστα των ατόμων που ακολουθεί σαν πίνακα. Έπειτα από τη συλλογή των tweets κρατάμε μόνο αυτά που το username τους βρίσκεται στον προηγούμενο πίνακα, τα ταξινομούμε χρονικά και προβάλλουμε τα 20 πρώτα. Είναι σημαντικό να αναφέρουμε πως και στις δύο αναζητήσεις που γίνονται στις συλλογές (με τα κατηγορήματα 'eqeq$eq' και 'isin') χρησιμοποιούνται τα δευτερεύοντα ευρετήρια της Mongo για τη γρήγορη εύρεση των αποτελεσμάτων και δε χρειάζεται να περαστεί στο Spark το σύνολο των δεδομένων.

Με βάση το πρότυπο της λάμδα-αρχιτεκτονικής που έχουμε ήδη αναλύσει το Spark μπορεί να χρησιμοποιηθεί για τον προϋπολογισμό των αρχικών σελίδων των χρηστών. Έτσι τροποποιήσαμε το παραπάνω πρόγραμμα για να υπολογίζει τις σελίδες πολλών χρηστών. Με αυτό το σενάριο ακόμα φιλοδοξούμε να καταδείξουμε τις δυνατότητες του Spark για υπολογισμούς στη μνήμη. Η κύρια διαφορά με προηγουμένως είναι πως πλέον επιλέγουμε να κρατήσουμε τη συλλογή των tweets στη μνήμη και στη συνέχεια υποβάλλουμε εκεί τα ερωτήματα μας. Αναμένουμε κάτι τέτοιο να είναι σημαντικά πιο αποδοτικό σε σχέση με την ανάκτηση των δεδομένων από το δίσκο.

```
Dataset<Row> tweets = MongoSpark.load(spark, ReadConfig.create(spark)
                .withOption("collection","tweets"),Tweet.class)
                .select("userName","text","date")
                .persist(StorageLevel.MEMORY_ONLY());
```

Διατήρηση της συλλογής των tweets στη μνήμη

Το δεύτερο σκέλος της λάμδα-αρχιτεκτονικής έχει να κάνει με την ενσωμάτωση των tweets, που παράγονται σε πραγματικό χρόνο, στις προϋπολογισμένες όψεις των αρχικών σελίδων. Για να αναπαραστήσουμε αυτή τη λειτουργία αρχικά δημιουργήσαμε ένα πρόγραμμα Java που λειτουργεί σαν TCP Server και παράγει 10 μηνύματα ανά δευτερόλεπτο, τα οποία στέλνονται και στη βάση μας. Ο αριθμός αυτός των tweets/sec ακολουθεί την αναλογία μεταξύ των χρηστών του Twitter και της δικιάς μας εφαρμογής. Η λειτουργία του είναι πολύ απλή, αναμένει για μία σύνδεση και στη συνέχεια παράγει συνεχώς μηνύματα μέχρι να δεχτεί σήμα τερματισμού.

```
//.. σύνδεση με βάση

ServerSocket s = new ServerSocket(9999);
Socket incoming = s.accept();

PrintWriter out = new PrintWriter (incoming.getOutputStream(),true);
while (true){
  Tweet jit = generateTweet(mongoOperation);
  out.println(jit.getUserName()+":"+jit.getText()+":"+jit.getDate());
  mongoOperation.insert(jit); //send to db also
  Thread.sleep(100);
}
//..
```

TCP Server - παραγωγή tweets

Έπειτα υλοποιήσαμε χρησιμοποιώντας το spark-streaming API το παρακάτω πρόγραμμα το οποίο συνδέεται στον TCP Server, διαβάζει τα tweets, βρίσκει όλους τους χρήστες που πρέπει να παραδοθούν αυτά και ενδεικτικά αποθηκεύει σε αρχείο για κάθε χρήστη τα tweets που του αντιστοιχούν.

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

  SparkConf conf = new SparkConf()
                  .setAppName("sparkStream")
                  .setMaster("local[*]");

  JavaStreamingContext ssc =
                new JavaStreamingContext(conf, Durations.seconds(1));
  JavaReceiverInputDStream<String> lines =
                ssc.socketTextStream("localhost", 9999);
```

```
  JavaPairDStream<String ,String> tuples = lines
          .mapToPair(x -> new Tuple2<>(x.split(":")[0],x));

  tuples.reduceByKey((i1, i2) -> i1 + "\n" + i2);
  tuples.foreachRDD(rdd -> {
    rdd.foreach(record -> {
      MongoClient mongoClient = new MongoClient( "localhost" ,27017);
      MongoDatabase db = mongoClient.getDatabase("jitterN");
      MongoCollection<Document> users = db.getCollection("users");
      Document user = users.find(eq("userName",record._1())).first();
      List<String> followers = user.get("followers", List.class);
      BufferedWriter writer;
      String tweet = record._2() + "\n";
      for(String follower : followers){
            writer = new BufferedWriter(new FileWriter("F:/stream/" +
follower + ".txt",true));
            writer.write(tweet);
            writer.close();
      }
      mongoClient.close();
     });
  });

  ssc.start();              // Start the computation
  ssc.awaitTermination();   // Wait for the computation to terminate
}
```

Υλοποίηση του Speed layer

5.7.2 Εξαγωγή σκορ Δημοφιλίας

Για το τρίτο σενάριο σύγκρισης η υλοποίηση με το Spark ήταν σημαντικά ευκολότερη και αποτελεσματικότερη καθώς το κοινό τμήμα των τριών ερωτημάτων μπορεί να κρατηθεί στη μνήμη και να μην επαναληφθεί.

```
Dataset<Row> nTweets = MongoSpark.load(spark,ReadConfig.create(spark)
    .withOption("collection", "tweets"), Tweet.class)
    .select("userName","replies","retweets")
    .persist(StorageLevel.MEMORY_ONLY());

Dataset<Row> tw = nTweets.select("userName").groupBy("userName")
    .agg(count("userName").as("numOfTweets"));

Dataset<Row> repl = nTweets.select("userName","replies")
    .withColumn("replies",explode(col("replies")))
    .groupBy("userName").agg(count("replies").as("numOfReplies"));

Dataset<Row> retw = nTweets.select("userName","retweets")
    .withColumn("retweets", explode(col("retweets")))
    .groupBy("userName").agg(count("retweets").as("numOfRetweets"));

Dataset<Row> ft = repl.join(retw,"userName").join(tw, "userName");
```

```
ft.withColumn("score",(ft.col("numOfTweets").multiply(0.25F))
   .plus(ft.col("numOfReplies").multiply(0.35F))
   .plus(ft.col("numOfReTweets").multiply(0.4F)))
   .sort(col("score").desc()).show();
```

Υπολογισμός σκορ με το Spark

Όπως βλέπουμε παραπάνω διατηρούμε τη συλλογή των tweets με τα πεδία που μας ενδιαφέρουν στη μνήμη και στη συνέχεια τα τρία query εκτελούνται σε αυτό το Dataframe. Κατά τα άλλα η λογική των ερωτημάτων είναι ακριβώς ίδια με πριν, όμως είναι πολύ πιο εύκολο να ενοποιήσουμε τα αποτελέσματα των queries με χρήση του join που διαθέτει το API των Dataframes. Τέλος δημιουργούμε τη στήλη του σκορ προσθέτοντας τις υπόλοιπες με κάποια βάρη και ταξινομούμε ως προς αυτό.

Κεφάλαιο 6^{ο} : Αποτελέσματα – Συμπεράσματα - Επεκτάσεις

6.1 Μετρήσεις απόδοσης των λειτουργιών του κοινωνικού δικτύου

Στόχος ήταν να μελετήσουμε τους χρόνους απόδοσης στις παρακάτω ενέργειες του κοινωνικού δικτύου υπό το φορτίο που δημιουργήσαμε προηγουμένως.

- Είσοδος χρήστη
- Δημοσίευση μηνύματος
- Ανάγνωση της αρχικής σελίδας
- Ανάγνωση του προφίλ κάποιου χρήστη
- Αναζήτηση με username ή hashtag

	Είσοδος	Δημοσίευση	Αρχική σελίδα	Προφίλ	Search Hashtag	Search User
Χρόνος	0,3 sec	0,4 sec	98,1 sec	0,7 sec	0,44 sec	0,39 sec

6.2 Συμπεράσματα

Όπως αναμενόταν με βάση τα όσα έχουμε πει προηγουμένως η πιο απαιτητική λειτουργία του κοινωνικού δικτύου είναι ο σχηματισμός της αρχικής σελίδας των χρηστών. Ακόμα και υπό το φορτίο που δημιουργήσαμε, όλες οι υπόλοιπες λειτουργίες μπορούν να αντιμετωπιστούν αποτελεσματικά με χρήση ευρετηρίων. Για αυτό το λόγο στη μελέτη με το Spark που ακολουθεί επικεντρωθήκαμε σε αυτή τη λειτουργία.

6.3 Σύγκριση με το Apache Spark

Στην ενότητα αυτή εξετάζουμε πως επιδρά η χρήση του Spark στην απόδοση. Για το σκοπό αυτό πραγματοποιήθηκαν μετρήσεις των χρόνων τριών διαφορετικών σεναρίων:

- Υπολογισμός της αρχικής σελίδας ενός χρήστη
- Υπολογισμός αρχικής σελίδας για πολλούς χρήστες
- Εξαγωγή σκορ δημοφιλίας για όλους τους χρήστες

Οι μετρήσεις έγιναν σε ένα μόνο κόμβο, σε τέσσερις κόμβους ως sharded cluster και τέλος σε τέσσερις κόμβους πάλι με το Spark να λειτουργεί πάνω από το sharded cluster. Το πρώτο σενάριο εκτελέστηκε για τρεις χρήστες με διαφορετικό αριθμό ατόμων που ακολουθούσαν. Η εκτέλεση σε ένα μηχάνημα έγινε στο Master κόμβο της τοπολογίας μας όπου βρισκόταν και το σύνολο των δεδομένων της βάσης της εφαρμογής. Για την εκτέλεση σε πολλά μηχανήματα χωρίς το Spark δημιουργήσαμε έναν δρομολογητή-mongos μόνο στο Master μηχάνημα όπου τρέχει και το πρόγραμμά μας. Στη συνέχεια ο δρομολογητής προωθεί τα query μας στο κατάλληλο shard είτε και σε όλα μαζί ταυτόχρονα αν πρόκειται για μια εργασία στο σύνολο των δεδομένων.

Για την εκτέλεση με το Spark χρησιμοποιήσαμε την προϋπάρχουσα παραμετροποίηση του YARN, βάση της οποίας για κάθε πυρήνα του κόμβου δημιουργείτο και ένας executor δηλαδή συνολικά διαθέταμε 6. Όσο αφορά τη μνήμη ένα ποσοστό 0,8 δεσμεύεται για το YARN, έτσι ο κάθε executor είχε 4,5GB εκ των οποίο 2GB μνήμης ήταν αφιερωμένα για τη διατήρηση δεδομένων και τα υπόλοιπα για τις διάφορες εργασίες. Τέλος, όλες οι παρακάτω τιμές αποτελούν το μέσο όρο τριών μετρήσεων με ενδιάμεσο καθαρισμό της μνήμης cache.

6.4 Μετρήσεις

- Υπολογισμός της αρχικής σελίδας ενός χρήστη

usernames	Αρ. Followers	Mongo_Single	Mongo_Shard	Mongo_Spark
00mythos	9347	127,8 sec	35,3 sec	76 sec
meme0019	5238	98,1 sec	26,2 sec	41 sec
000aikia000	1100	38,3 sec	7,9 sec	19 sec

- Υπολογισμός αρχικής σελίδας για πολλούς χρήστες

	Mongo_Single	Mongo_Shard	Mongo_Spark
Συνολικός χρόνος	45,6 min	6,6 min	15 min
Χρόνος ανά χρήστη*	27,4 sec	3,9 sec	5,4 sec

*Αν εξαιρέσουμε τον πρώτο χρήστη όπου το Spark μαζεύει στη μνήμη ολόκληρη τη συλλογή.

- 6. Εξαγωγή σκορ για όλους τους χρήστες

Mongo_Single	Mongo_Shard	Mongo_Spark
8,35 min	2,95 min	6,4 min

6.5 Σχολιασμός αποτελεσμάτων

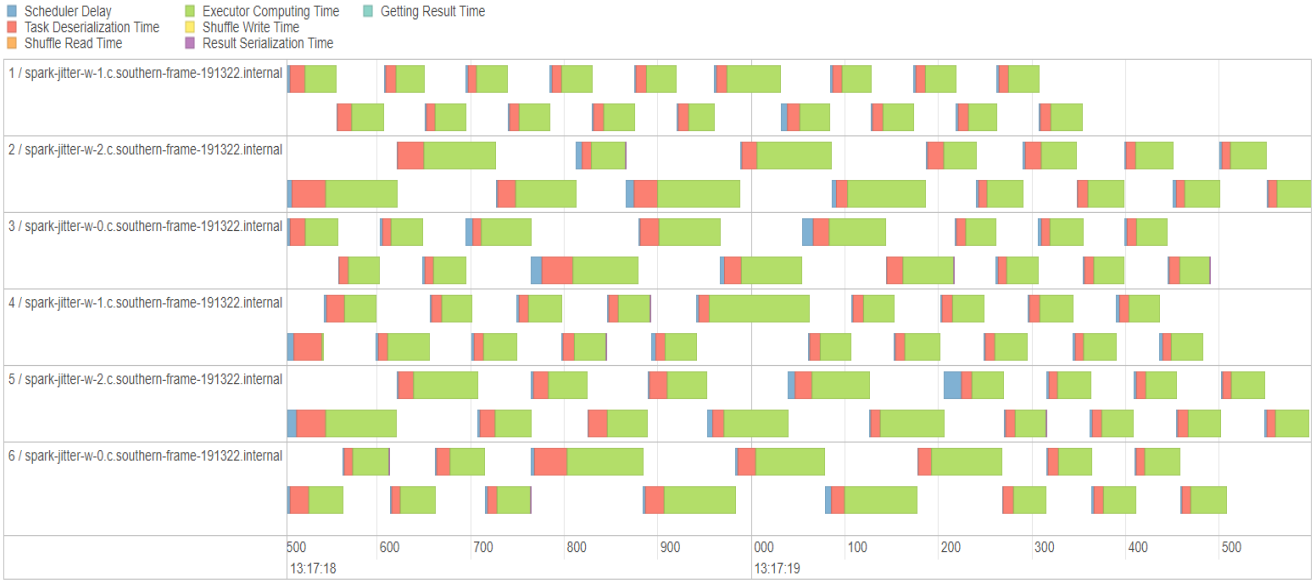
Καταρχάς όπως αναμέναμε βλέπουμε πως οι χρόνοι εκτέλεσης στο sharded cluster έναντι του ενός κόμβου είναι περίπου 3 φορές μικρότεροι στις περισσότερες των περιπτώσεων. Αυτό συμβαίνει διότι η MongoDB εκτελεί τα ερωτήματα παράλληλα και στους τρεις κόμβους αθροίζοντας τη δύναμη των υπολογιστικών πόρων. Βελτίωση πολύ μεγαλύτερη του αναμενόμενου (7 φορές ταχύτερα) παρατηρούμε στον υπολογισμό των αρχικών σελίδων πολλών χρηστών. Η εξήγηση για αυτή την επίδοση θεωρούμε πως προέρχεται λόγω της διατήρησης των αποτελεσμάτων για κάθε αρχική σελίδα στη μνήμη με αποτέλεσμα τα επόμενα ερωτήματα να απαντιούνται πολύ πιο γρήγορα. Το ίδιο πράγμα δε συμβαίνει στον έναν κόμβο διότι η διαθέσιμη μνήμη του μηχανήματος είναι μικρότερη με συνέπεια τα δεδομένα να διαγράφονται. Αυτή η διατήρηση στη μνήμη συμβαίνει κατά μη ρητό τρόπο οπότε δεν έχουμε κάποια εγγύηση ότι κάτι τέτοιο θα συμβαίνει πάντοτε.

Απ’ την άλλη τώρα όσο αφορά τους χρόνους του Spark είναι καλύτεροι μεν από αυτούς του μοναδικού κόμβου όχι όμως και από αυτούς του sharded cluster. Ένας λόγος για αυτό είναι η υψηλή παραλληλοποίηση που επιχειρούσε το Spark σε συνδυασμό με τον περιορισμένο αριθμό πυρήνων που διέθετε η συστάδα. Ο MongoShardedPartitioner που χρησιμοποιείται από τον connector της Mongo, για την λειτουργία σε sharded cluster προσφέρει 420 partitions της συλλογής (όσα και τα chunks του cluster) στο Spark. Κάτι τέτοιο εισάγει μεγάλο overhead λόγω του ότι το Spark πρέπει να τρέξει 420 φορές το ίδιο ερώτημα σε κάθε partition συν το χρόνο που καταναλώνει ο scheduler για να δρομολογήσει κάθε τέτοια εργασία, συν το χρόνο για την αποσειριοποίηση των ερωτημάτων σε κάθε task και τέλος συν το χρόνο καθυστέρησης για εναλλαγή των tasks. Παρακάτω βλέπουμε αυτούς τους χρόνους για τον υπολογισμό δύο διαφορετικών αρχικών σελίδων χρηστών και στη συνέχεια την εκτέλεση των tasks στους διάφορους executors.

Summary Metrics for 420 Completed Tasks

Metric	Median	Median
Duration	34 ms	35 ms
Scheduler Delay	3 ms	2 ms
Task Deserialization Time	6 ms	10 ms
GC Time	0 ms	0 ms
Result Serialization Time	0 ms	0 ms
Getting Result Time	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B
Input Size / Records	10.9 MB / 13	10.9 MB / 13

Εικόνα 34: Μέσοι χρόνοι για δύο διαφορετικές περιπτώσεις υπολογισμού της αρχικής σελίδας



Εικόνα 35: Εκτέλεση των task στους executors και απεικόνιση των διάφορων χρόνων

Σίγουρα θα θέλαμε έναν αριθμό partitions σημαντικά μικρότερο, κάτι τέτοιο όμως δεν ήταν εφικτό λόγω του μεγάλου μεγέθους της συλλογής των tweets -13GB- και των 50 εκατομμυρίων μηνυμάτων που υπήρχαν. Αποτελεί σύμβαση για τη MongoDB ότι ένα chunk προκειμένου να μπορεί να μετακινηθεί μεταξύ των κόμβων πρέπει να περιέχει έως 250 χιλιάδες έγγραφα και έτσι εξηγείται ο μεγάλος αριθμός partitions. Απ' την άλλη ένας μικρός αριθμός partitions θα δημιουργούσε εξίσου προβλήματα καθώς το μέγεθος των tasks θα ήταν αρκετά μεγάλο και έτσι θα παρατηρούσαμε την κατάσταση όπου 5 executors έχουν τερματίσει και περιμένουν άπραγοι το τελευταίο task να ολοκληρωθεί πριν προχωρήσουν στο επόμενο stage. Τέλος στη δοκιμαστική έκδοση του Google Cloud Platform είχαμε περιορισμό στον αριθμό των πυρήνων που

μπορούσαμε να χρησιμοποιήσουμε με το μέγιστο να είναι οι 8 και τους δύο από αυτούς να διατίθενται στον master κόμβο της συστάδας.

Το γραφικό περιβάλλον του Spark μας παρέχει όλες τις πληροφορίες που μπορεί να μας ενδιαφέρουν και αφορούν την σωστή λειτουργία της συστοιχίας. Παρακάτω βλέπουμε την κατανομή των tasks στους 6 executors.

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks
1 stdout stderr	spark-jitter-w-1.c.southern-frame-191322.internal:35304	3 s	70
2 stdout stderr	spark-jitter-w-2.c.southern-frame-191322.internal:56108	3 s	67
3 stdout stderr	spark-jitter-w-0.c.southern-frame-191322.internal:49380	3 s	72
4 stdout stderr	spark-jitter-w-1.c.southern-frame-191322.internal:57222	3 s	70
5 stdout stderr	spark-jitter-w-2.c.southern-frame-191322.internal:41312	3 s	73
6 stdout stderr	spark-jitter-w-0.c.southern-frame-191322.internal:43991	3 s	68

Εικόνα 36: Χρόνοι λειτουργίας και αριθμός ολοκληρωμένων tasks ανά executor

Όσο αφορά την εύρεση σκορ για κάθε χρήστη παρατηρούμε στην παρακάτω εικόνα πως μόνο το πρώτο ερώτημα κοστίζει ακριβά όπου φορτώνεται και ολόκληρη η συλλογή στη μνήμη από εκεί και έπειτα η εκτέλεση των υπόλοιπων ερωτημάτων γίνεται στη μνήμη και κάθε stage λαμβάνει από το πρώτο ως είσοδο τη συλλογή.

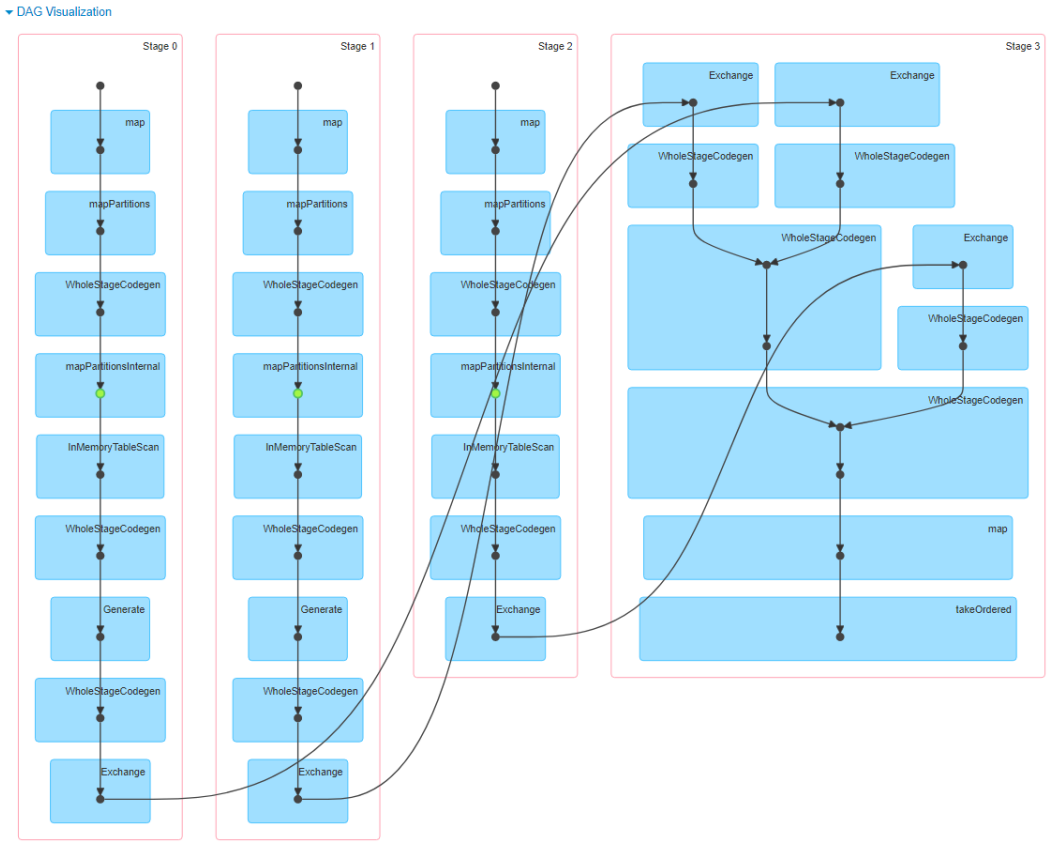
Completed Stages (4)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input
3	show at Main.java:78 +details	2018/01/23 01:45:10	16 s	200/200	
2	show at Main.java:78 +details	2018/01/23 01:39:16	18 s	420/420	3.6 GB
1	show at Main.java:78 +details	2018/01/23 01:39:16	26 s	420/420	3.6 GB
0	show at Main.java:78 +details	2018/01/23 01:39:16	5.4 min	420/420	

Εικόνα 37: Στάδια για υπολογισμό σκορ

Παρακάτω έχουμε την αναπαράσταση του ακυκλικού γραφήματος για τον υπολογισμό του σκορ. Εδώ αποτυπώνεται και η επιλογή για οκνηρή αποτίμηση των μετασχηματισμών στο Spark. Και τα τέσσερα παραπάνω stages αποτελούν μία εργασία -job-, η εκτέλεση των οποίων ξεκινά μόνο όταν γίνει η κλήση για τη δράση -action- “show”. Τα τρία πρώτα “στάδια” αφορούν την εύρεση των τριών αριθμών ενδιαφέροντος και εισάγονται μαζί. Το τέταρτο στάδιο, που αφορά την λειτουργία

join και την ταξινόμηση, απαιτεί την ολοκλήρωση των προηγούμενων προτού αρχίσει να εκτελείται.



Εικόνα 38: Αναπαράσταση του Ακυκλικού γραφήματος εκτέλεσης

Σε κάθε περίπτωση από όλα τα παραπάνω αποτυπώνεται πως η παραλληλοποίηση μιας εργασίας με την εκτέλεσή της σε μια συστάδα υπολογιστών μπορεί να πετύχει γραμμική ως προς τους κόμβους μείωση του χρόνου. Αν και τα οφέλη του Spark δεν αποτυπώνονται πλήρως, με το μέγεθος αυτού του cluster μπορέσαμε να προσδιορίσουμε τις περιπτώσεις και τους τρόπους που αυτό μπορεί να ωφελήσει στην ταχύτερη διεκπεραίωση μιας εργασίας. Καταρχάς μπορεί να εκμεταλλευτεί το σύνολο των πυρήνων των μηχανημάτων μας οδηγώντας σε μεγαλύτερη παραλληλοποίηση από εκείνη που επιτυγχάνει η MongoDB μέσω του sharded cluster. Μας παρέχει ακόμα έναν ρητό τρόπο για να επιβάλλουμε τη διατήρηση κάποιων δεδομένων στη μνήμη στοιχείο που έχει δραματική επίδραση στο χρόνο εκτέλεσης και αποτυπώθηκε πιο ξεκάθαρα στο δεύτερο παράδειγμα σύγκρισης.

Τέλος, είναι ένα πλαίσιο για εκτέλεση διεργασιών γενικού σκοπού κάτι που σημαίνει ότι μπορεί να αντιμετωπίσει αποτελεσματικότερα αυθαίρετες εργασίες σε σχέση με μία βάση δεδομένων που ο ρόλος της είναι συγκεκριμένος και περιορισμένος. Στο παράδειγμα μελέτης που αναλύσαμε, το κύριο βάρος της

εργασίας είναι τα ερωτήματα προς τη βάση μία λειτουργία που η MongoDB μπορεί να εκτελέσει αποτελεσματικά και όπως είδαμε, μέσω της λειτουργίας sharding καταφέρνει να παραλληλοποιήσει επίσης. Οπότε εδώ η χρήση του Spark δε φαίνεται να έχει τόσο σημαντικό αντίκτυπο.

6.6 Σύνοψη

Κατά τη διάρκεια αυτής της διπλωματικής εργασίας ασχοληθήκαμε με ένα ευρύ φάσμα αντικειμένων και γνωρίσαμε πολλές διαφορετικές τεχνολογίες. Εξοικειωθήκαμε με όλα τα μέρη που εμπλέκονται στην δημιουργία μιας web εφαρμογής, δουλεύοντας τόσο στο frontend όσο και στο backend. Είδαμε πως προγραμματιστικά framework όπως το Spring μπορούν να βοηθήσουν τον προγραμματιστή στην προσπάθειά του για παραγωγή οργανωμένου και αξιόπιστου λογισμικού. Ακόμα γνωρίσαμε μία νέα προσέγγιση στην αποθήκευση δεδομένων αυτή των μη-σχεσιακών βάσεων μέσω της MongoDB.

Στη συνέχεια μελετήσαμε ποιες ανάγκες οδήγησαν στην υιοθέτηση των συστάδων υπολογιστών για την εκτέλεση απαιτητικών εργασιών. Αναζητήσαμε τις μεθόδους, τις αρχιτεκτονικές, τις τεχνολογίες με τις οποίες ένα κοινωνικό δίκτυο μπορεί να ανταποκριθεί στην κίνηση και τον όγκο δεδομένων που αυτό δημιουργεί. Η μελέτη αυτή αποτυπώθηκε στην υλοποίηση που κάναμε με το Apache Spark το οποίο ανήκει σε μια ευρύτερη σουίτα προγραμμάτων ανοικτού κώδικα που όλα μαζί φιλοδοξούν να δώσουν λύσεις στην ανάλυση και διαχείριση μεγάλου όγκου δεδομένων και όχι μόνο.

Τέλος με τη μεταφορά της τοπολογίας μας στο Google Cloud Platform εξερευνήσαμε και επωφεληθήκαμε από τις δυνατότητες που μας προσφέρουν οι υπηρεσίες υπολογιστικού νέφους.

6.7 Επεκτάσεις

Ιδιαίτερα στο κομμάτι της εφαρμογής υπάρχουν πολλές βελτιώσεις που μπορούν να γίνουν καθώς η έμφαση σε αυτή τη διπλωματική εργασία δόθηκε κυρίως στη μελέτη και βελτίωση της λειτουργίας του υπό φορτίο. Έτσι λοιπόν σίγουρα πολλά βήματα μπορούν να γίνουν στην βελτίωση της διεπαφής καθώς για αυτό το σκοπό έχουν αναπτυχθεί πάρα πολλές τεχνολογίες το τελευταίο διάστημα που μειώνουν κατά πολύ τον κόπο που απαιτείται από πλευράς προγραμματιστή και ταυτόχρονα προσφέρουν ένα ανώτερο αποτέλεσμα από άποψη αισθητικής.

Επιπλέον οι λειτουργίες που προσφέρει το δίκτυο μπορούν να επεκταθούν. Ένα σημαντικό κομμάτι είναι η δημιουργία του μηχανισμού των ειδοποιήσεων των χρηστών για όλες τις δραστηριότητες που τους αφορούν. Ακόμα έμφαση πρέπει να δοθεί και στο κομμάτι της ασφάλειας απ' την στιγμή που ένα κοινωνικό δίκτυο διαχειρίζεται ευαίσθητα προσωπικά δεδομένα των χρηστών.

Όσο αφορά την υλοποίηση με το Spark σημαντική κρίνεται η διαδραστική υποβολή ερωτημάτων σε αυτό. Η κλάση SparkLancher καθώς και ο Spark Rest Server που αποτελεί τμήμα της διανομής του Spark προϋποθέτουν την ύπαρξη έτοιμων jobs –jar files- προς εκτέλεση. Ουσιαστικά αναπαράγουν προγραμματιστικά τη λειτουργία του spark-submit κάτι που απαιτεί με κάθε εκτέλεση την εκκίνηση νέων executors. Ύστερα από αναζήτηση που έγινε βρέθηκε πως υπάρχουν εργαλεία εκτός του Spark project που μπορούν να του προσδώσουν διαδραστικές δυνατότητες. Τέτοια είναι τα Livy REST Server και Spark Job Server μέσω των οποίων μπορούμε να υποβάλλουμε απευθείας κώδικα προς εκτέλεση στο Spark. Χάρη σε αυτά τα εργαλεία το Spark ξεφεύγει από το ρόλο για τον οποίο αρχικά σχεδιάστηκε, αυτό του εργαλείου ανάλυσης δεδομένων, και μπορεί πλέον να λειτουργεί στο backend των εφαρμογών.

Άλλη μία επέκταση αφορά την πλήρη υλοποίηση της λάμδα-αρχιτεκτονικής. Πέρα των δύο στρωμάτων επεξεργασίας, batch και streaming, σημαντικό είναι και το στρώμα παράδοσης των δεδομένων που αναλαμβάνει την ενοποίηση των αποτελεσμάτων από τα δύο άλλα, και επιτρέπει την γρήγορη πρόσβαση σε αυτά. Το τμήμα αυτό απαιτούσε την χρήση άλλων τεχνολογιών βάσεων δεδομένων όπως είναι οι βάσεις γράφων και οι triple-store που έμειναν εκτός του πεδίου μελέτης αυτής της διπλωματικής εργασίας.

Βιβλιογραφία

[1] eMarketer – Nearly One-Third of the World Will Use Social Networks Regularly, https://www.emarketer.com/Article/Nearly-One-Third-of-World-Will-Use-Social-Networks-Regularly-This-Year/1014157

[2] The server side - About Java, http://www.theserverside.com/feature/Why-Java-is-the-most-popular-programming-language

[3] JSP - https://en.wikipedia.org/wiki/JavaServer_Pages

[4] Tiobe - https://www.tiobe.com/tiobe-index/

[5] Tutorialspoint - Spring framework, https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

[6] MDN – MVC Architecture https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture

[7] Wikipedia – JavaScript https://en.wikipedia.org/wiki/JavaScript

[8] Statista - monthly active Twitter users https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[9] Martin Kleppmann - Designing Data-Intensive Applications, O' Reilly 2017

[10] Dzone – Lambda Architecture, https://dzone.com/articles/lambda-architecture-with-apache-spark

[11] Wikipedia Scalability, https://en.wikipedia.org/wiki/Scalability

[12] Wikipedia – Cloud computing https://en.wikipedia.org/wiki/Cloud_computing

[13] Stack Overflow – Horizontal vs Vertical Scaling https://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases

[14] MongoDB https://www.mongodb.com/compare/mongodb-oracle

[15] Mongo - Spark Connector https://docs.mongodb.com/spark-connector/master/java-api/

[16] Mongo – Sharding, https://docs.mongodb.com/v3.4/sharding/

[17] Mongo – Replication, https://docs.mongodb.com/v3.4/replication/

[18] MongoDB - MongoDB Architecture Guide, 2016

[19] Apache Spark – Parallelization, https://www.percona.com/blog/2016/08/17/apache-spark-makes-slow-mysql-queries-10x-faster/

[20] Spark: Cluster Computing with working sets https://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

[21] Data-Flair – What is Spark, https://data-flair.training/blogs/what-is-spark/

[22] GitHub, https://github.com/spark-jobserver/spark-jobserver