Τμήμα ψηφιακών συστημάτων Πρόγραμμα μεταπτυχιακών σπουδών
"Συστήματα και Υπηρεσίες" Δικτυοκεντρικά Συστήματα

Διπλωματική Εργασία

Σύστημα Ανάλυσης Συναισθήματος στο Twitter σε Πραγματικό Χρόνο με το Storm

Αννέτα Αμπλά
Α.Μ.: ΜΕ13037
Υπεύθυνος Καθηγητής: Δουλκερίδης Χρ.

Ευχαριστίες

Μετά την ολοκλήρωση αυτής της διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που συνεισέφεραν με την υποστήριξη τους στην προσπάθεια μου. Αρχικά θα ήθελα να ευχαριστήσω τον κ. Χρήστο Δουλκερίδη για την βοήθεια και την υποστήριξη που μου έδωσε. Επίσης την οικογένεια μου και τον Νίκο που με στήριξαν σε όλη την διάρκεια των σπουδών μου. Στο τέλος θα ήθελα να ευχαριστήσω μέσα από την καρδιά μου την Μαρία Καρανάσου, της οποίας η στήριξη, βοήθεια και φιλία είναι ανεκτίμητη.

Πινάκας Περιεχομένων

Πρόλογος

Το Twitter λαμβάνει περίπου 190 εκατομμύρια tweets (μικρά κείμενα δημοσιευμένα στο Web) την ημέρα, στα οποία οι άνθρωποι μοιράζονται τα σχόλια και τις ιδέες τους τα για διάφορα θέματα. Αυτό το κείμενο τις περισσότερες φορές εκφράζει κάποιο συναίσθημα. Έτσι για να έχουμε μια εικόνα για αυτά τα συναισθήματα οποιαδήποτε στιγμή σε πραγματικό χρόνο και για όποιο λόγο θέλουμε, η δημιουργία ενός συστήματος που παρουσιάζει σε πραγματικό χρόνο τα συναισθήματα από το Twitter, είναι σημαντική.
Αυτό το σύστημα μπορεί να χρησιμοποιηθεί σε πολλές περιπτώσεις για την ανάλυση του συναισθήματος , όπως εκλογές, τελικός του Champions League ή Eurovision. Η ανάλυση του συναισθήματος στο σύστημα χωρίζεται σε θετικά, αρνητικά και ουδέτερα αλλά εμφανιζεται και το μέγεθος όλων των tweets σε πραγματικό χρόνο.

Εισαγωγή

Δομή Κειμένου

Αρχικά, ορίζεται το πρόβλημα όπου αυτή η διπλωματική εργασία καλείται να επιλύσει. Στην συνέχεια παρουσιάζεται μια σύντομη περιγραφή για τα θέματα που εμπλέκονται στον καθορισμό και την επίλυση του προβλήματος.
Ακολουθεί η μοντελοποίηση του προβλήματος με όλες τις λεπτομέρειες που παίζουν κάποιον ρόλο στην διαχείριση του προβλήματος, και η περιγραφή του συστήματος που αναπτύχθηκε. Επίσης αναφέρονται όλες οι τεχνολογίες που επιλέχθηκαν για την επίλυση του προβλήματος αλλά και την ανάπτυξη του συστήματος.
Στην συνέχεια παρουσιάζεται το Data visualization system με τα χαρακτηριστικά και τις δυνατότητες του.
Στο τέλος γίνεται περιγραφή των συμπερασμάτων αλλά και προτάσεις για μελλοντική εργασία και βελτιώσεις.

Ορισμός Προβλήματος

Όταν πρόκειται για συστήματα πραγματικού χρόνου ή σχεδόν σε πραγματικό χρόνο, πολλά πράγματα μπορούν να πάνε στραβά. Έτσι είναι σημαντικό να έχει την δυνατότητα το σύστημα να παρακολουθεί τόσο την δική του λειτουργία όσο και την διαδικασία του συστήματος. Για παράδειγμα, το Apache Storm διαθέτει ένα built-in έτοιμο για χρήση monitoring system με το δικό του user interface (UI), που ονομάζεται "storm-ui". Αυτό παρέχει στον χρήστη(developer) πληροφορίες σχετικά με την υγεία του συστήματος αλλά και για πιθανά bottlenecks. Για το δεύτερο, το σύστημα που έχει δημιουργηθεί με το Storm, δεν έχει το δικό του monitoring, αφού κάθε σύστημα είναι διαφορετικό και προσαρμοσμένο στις ανάγκες του. Το case study αυτής της διπλωματικής εργασίας είναι η δημιουργία ενός "Large-Scale Sentiment Analysis" συστήματος βασισμένο στο Apache Storm, και το multilang protocol που παρέχει. Ο στόχος αυτού του συστήματος είναι να προσφέρει έναν εύκολο τρόπο για την απόκτηση και παρουσίαση πληροφοριών information σε πραγματικό χρόνο.

Ανασκόπηση

Apache Storm

Το Apache Storm είναι ένα framework το οποίο στοχεύει στην real-time επεξεργασία κ ανάλυση ενός συνόλου από data streams, υποστηρίζει τους distributed real-time υπολογισμούς, με βασικά χαρακτηριστικά όπως επεκτασιμότητα(scalability), parallelism, fault-tolerance και αξιοπιστία (reliability) . Παρέχει έναν flexible μηχανισμό για τους application developers ώστε να χρειάζεται να υλοποιήσουν μόνο την λογική της προβλεπόμενης διαδικασίας, αλλά και διάφορα πιο εξειδικευμένα χαρακτηριστικά (communication, parallelism, fault-tolerance) τα οποία παρέχονται δωρεάν. [1]
Το Storm έχει πολλές περιπτώσεις χρήσης: real-time analytics, online machine learning, συνεχές computation, distributed RPC, ETL, και άλλα. Το Storm είναι γρήγορο (πάνω από εκατομμύριο tuples μπορούν να επεξεργαστούν ανά κόμβο το δευτερόλεπτο). Το Storm μπορεί να ενσωματωθεί με δίαφορες queueing και database τεχνολογίες. [2]

Χαρακτηριστικά του Storm

- ✓ Fault tolerant και distributed
- ✓ Non persistent
- ✓ Γραμμένο σε Clojure και java
- ✓ Εγγυημένη data processing
- ✓ Horizontal Scalabitity
- ✓ Higher level of abstraction καθώς το μήνυμα μεταφέρεται
- ✓ Σύστημα επεξεργασίας big data σε πραγματικό χρόνο

Ιστορία του Storm

Το Storm δημιουργήθηκε από τον Nathan Martz και την ομάδα του στο Backtype. Το αρχική release έγινε στις 17 September 2011.
Η αρχή του Storm έγινε όταν ο Nathan Martz είχε την ιδέα ενός a "stream" σαν ένα distributed abstraction. Η ιδέα ήταν η παραγωγή και η επεξεργασία των streams σε παράλληλο τρόπο (in parallel), και η παρουσίαση τους σε ένα ενιαίο πρόγραμμα σαν ένα ενιαίο abstraction . Με βάση αυτή την ιδέα παρουσιάστηκαν οι έννοιες "spout" και "bolt" .[3] Ένα spout μπορεί να παράγει νέα streams, και το bolt να δεχθεί αυτά τα streams σαν είσοδο (input) και να παράξει streams σαν output. Έτσι, τα bolt μπορούν απλά να συνεδεθούν σε οποιαδήποτε streams πρέπει να επεξεργαστεί και να υποδείξει πως το εισερχόμενο stream πρεπει να διαιρεθεί στα bolts. Σε ένα υψηλό αφαιρετικό επίπεδο, δημιούργησε τον ορισμό Τοπολογία (Topology) , ως ένα δίκτυο από spouts και bolts.
Για να την παρακολούθηση και την βελτιστοποίηση της εκτελούμενης τοπολογίας, παρουσιάστηκε το Storm metrics API, το οποίο δίνει την δυνατότητα στους χρήστες να συλλέγουν εντελώς custom, arbitrary metrics, και να στέλνουν αυτά τα metrics σε οποιοδήποτε

monitoring system. Επίσης το Storm UI (storm ‘s monitoring system) χρησιμοποιείται για την εμφάνιση των μετρήσεων (metrics) της εκτελούμενης τοπολογίας . Ένα άλλο μεγάλο τεχνικό άλμα για το Storm ήταν η ανάπτυξη του Trident, ενός micro-batching API στην κορυφή του Storm το οποίο παρέχει exactly-once processing semantics.

Εισαγωγή στο Storm

Έννοιες

Τοπολογία(Topology)
Όπως έχουμε ήδη αναφέρει, μια τοπολογία του Storm είναι ένα δίκτυο από spouts και bolts. Καταναλώνει streams από data και επεξεργάζεται αυτά τα streams με αυθαίρετα πολύπλοκους τρόπους , κατανέμοντας τα streams ανάμεσα σε κάθε στάδιο του computation,όποτε χρειάζεται. Από τη σκοπιά των database systems, μπορει κανείς να σκεφτεί μια τοπολογία σαν ένα directed graph από operators [4]

Streams
Το stream είναι μια unbouned ακολουθία από Tuples η οποία μπορεί να επεξεργαστεί και να δημιουργηθεί in parallel με distributed way.

Spouts
Τα Spouts είναι πηγή από Data Streams σε μια τοπολογία του Storm. Τα Spouts ‘διαβάζουν’ τα tuples από μία εξωτερική πηγή και τα διαθέτουν στην τοπολογία. Τα Spouts μπορούν να διαχωριστούν σε αξιόπιστα και μη (reliable/unreliable) spouts. Στην πρώτη περίπτωση το spout μπορεί να επαναλάβει το tuple αν δεν κατάφερε να επεξεργασθεί από το Storm, ενώ στην δεύτερη περίπτωση το spout ‘ξεχνάει’ το tuple μόλις γίνει emitted.

Bolts
Τα Bolts επεξεργάζονται τα input streams και παράγουν νέα streams (processes όπως running functions, filter tuples, επικοινωνία με DBs). Τα Bolts μπορούν να απλούς και πολύπλοκους μετασχηματισμούς (transformations). Στην περίπτωση πολύπλοκων μετασχηματισμών χρειάζονται περισσότερα bolts για να κατανεμηθούν πιο αποτελεσματικά οι διεργασίες που πρέπει να γίνουν. Τα Bolts κάνουν emit το stream, στο επόμενο bolt με την δυνατότητα να καλέσουν μια ack method, για κάθε tuple που επεξεργάζονται έτσι ώστε το Storm να γνωρίζει πότε έχουν ολοκληρωθεί τα tuples.

Stream Groupings
Στην περίπτωση της παράλληλης εκτέλεσης ενός bolt, μέσω πολλαπλών instances, μπορούν να εφαρμοσθούν διαφορετικοί κανόνες στο πως τα tuples μπορούν να κατανεμηθούν στα πολλαπλά instances ενός bolt. Αυτή η κατανομή (partitioning) είναι γνωστή ως stream grouping με αντιπροσωπευτικά ενσωματωμένα παραδείγματα:

– Shufle grouping: τυχαία κατανομή των tuples σε όλα τα task του bolt
– Fields grouping: τα tuples στέλνονται βασισμένα σε ένα ή περισσότερα καθορισμένα fields
– All grouping: τα tuples μεταδίδονται σε όλα τα task (διεργασίες) του bolt
– Global grouping: ολόκληρο το stream πηγαίνει σε ένα και μόνο ένα task του bolt

Tasks

Τα Spouts και τα Bolts εκτελούν όσο πιο πολλά tasks εγκάρσια του cluster. Ένα task (διεργασία) εκτελεί την πραγματική επεξεργασία δεδομένων και τρέχει μέσα στο thread εκτέλεσης του parent executor του. Ο ορισμός των task για ένα component είναι πάντα ο ίδιος κατά την διάρκεια της ζωής μιας τοπολογίας, αλλά ο αριθμός των executors (threads) για ένα component μπορεί να αλλάξει στην πάροδο του χρόνου. Τα stream groupings καθορίζουν πως θα κατανεμηθούν τα tuples από το ένα σετ από tasks σε ένα άλλο σετ απο tasks.

Workers (worker processes)

Μια worker διαδικασία (process) εκτελεί ένα υποσύνολο μιας τοπολογίας, και τρέχει στο δικό του JVM. Μια worker διαδικασία ανήκει σε μία συγκεκριμένη τοπολογία και μπορεί να τρέξει σε ένα ή περισσότερα executors για ένα ή περισσότερα components (spouts or bolts) αυτής της τοπολογίας . Μία εκτελούμενη τοπολογία αποτελείται από πολλές τέτοιες διαδικασίες που μπορεί να τρέχουν σε πολλά μηχανήματα σε ένα Storm cluster.[2]

Executors

Ο executor είναι ένα thread που έχει ‘γεννηθεί’ από το worker process και τρέχει στο worker’s JVM του worker . Ένας executor μπορεί να τρέχει ένα η περισσότερα tasks για το ίδιο component (spout or bolt). Ένας executor έχει πάντα ένα thread το οποίο χρησιμοποιεί για όλα του τα tasks, πράγμα που σημαίνει ότι τα tasks τρέχουν σειριακά σε έναν executor.

Parallelism(παραλληλία)

Το “Parallelism hint” χρησιμοποιείται στο Storm για να ‘παραλληλίσει’ μία εκτελούμενη storm τοπολογία. Μπορούμε να πούμε ότι ένα bolt μπορεί να έχει πολλαπλά instances. Βοηθάει στην βελτίωση της απόδοσης μιας τοπολογίας. Ο όρος Parallelism στο Storm χρησιμοποιείται για να περιγράψει το *parallelism hint* το οποίο στην πραγματικότητα σημαίνει των αρχικό αριθμό των executors στην Topology. Ο αριθμός των executor threads μπορεί να αλλάξει από την στιγμή που η τοπολογία έχει αρχίσει να εκτελείται. Το Storm επιτρέπει μόνο την εξισορρόπηση μιας τοπολογίας δημιουργώντας περισσότερα threads (executors) ή workers αλλά δεν επιτρέπει την αλλαγή του αριθμού των instances (tasks) για κάθε operator.[5]

Development Environment

Το Storm has two τρόπους λειτουργίας : τον τοπικό (local mode) και τον απομακρυσμένο(remote mode) τρόπο . Στο local mode, μπορείς να αναπτύξεις και να τεστάρεις τοπολογίες που είναι εξ ολοκλήρου σε εξέλιξη στο τοπικό μηχάνημα (local machine) . Σε remote mode, μπορείς να κάνεις submit τοπολογίες για εκτέλεση σε ένα cluster από μηχανήματα.

Αρχιτεκτονική του Apache Storm – Storm Cluster

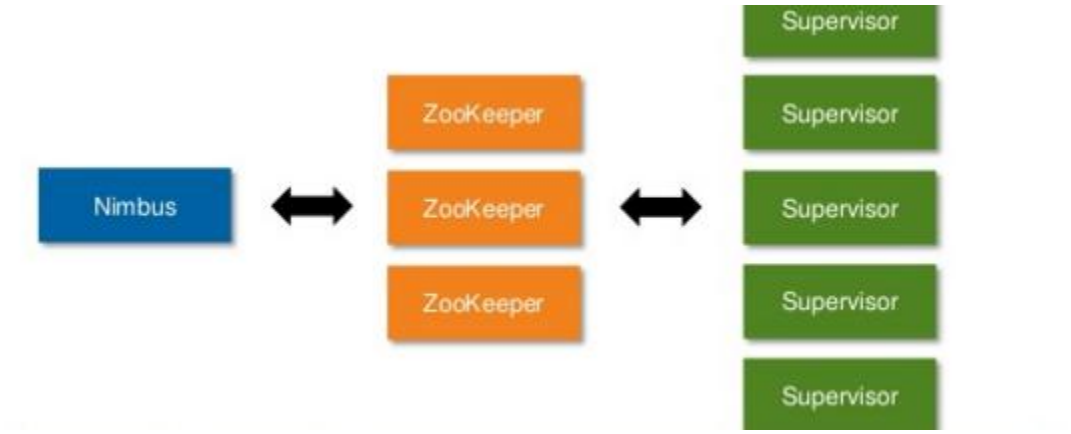

Figure 1. Apache Storm's Overview

Zookeeper

Χρησιμοποιείται για τον συντονισμό του cluster. Είναι μια εφαρμογή η οποία συντονίζει και μοιράζεται κάποιες configuration πληροφορίες μεταξύ διάφορων διαδικασιών. Αποθηκεύει όλες τις καταστάσεις που σχετίζονται με το cluster και τα δίαφορα tasks που έχουν υποβληθεί στο storm. Οι κόμβοι (nodes) Nimbus και Supervisor επικοινωνούν μόνο μέσα του ZooKeeper.

Το Storm χωρίζεται σε δύο είδη nodes σε ένα Storm cluster:

Nimbus - Master Node.

Κατανέμει τον κώδικα στο cluster και στους worker nodes, αναθέτει tasks στα μηχανήματα/supervisors, παρακολουθεί την αποτυχία των task, τα επανεκκινεί όταν χρειάζεται, είνια fail fast και stateless, αποθηκεύει όλα τα data του στο Zookeeper, μπορεί να επαννακινηθεί χωρίς να επηρεάσει τα ήδη τρέχοντα tasks στους worker nodes. Αυτός ο κόμβος τρέχει το επονομαζόμενο Nimbus daemon το οποίο είναι υπεύθυνο για να την ανάθεση tasks στα worker nodes και τέλος παρακολουθεί το cluster για τυχόν αποτυχίες.

Supervisor - Slave node.

Πρόκειται για τους worker nodes σε ένα Storm Cluster οι οποίοι τρέχουν worker processes. 'Ακούει στις εργασίες του μηχανήματος του που του έχουν ανατεθεί . Κάθε supervisor κόμβος (node) 'τρέχει' ένα supervisor daemon που είναι υπεύθυνο για την δημιουργία, εκκίνηση και την διακοπή του working process ώστε να εκτελέσει τα tasks που έχουν ανατεθεί σε αυτόν τον κόμβο από τον Nimbus. Είναι επίσης fail fast και αποθηκεύει όλα του τα δεδομένα στον Zookeeper έτσι ώστε να μπορεί να καταγράφουν χωρίς κανένα state loss. Συνήθως κανονικά ένα μόνο supervisor daemon μπορεί να χειριστεί πολλά worker processes. Κάθε worker process εκτελεί ένα υποσύνολο της τοπολογίας.

Storm UI

Το Strom UI είναι ένα built in monitoring System το οποίο χρησιμοποιείται για την παρακολούθηση και την εμφάνιση(monitoring) μετρήσεων της εκτελούμενης τοπολογίας.
Το Monitoring χρησιμοποιείται για να κάνει track την υγεία των δίαφορων components που τρέχουν στο cluster.
Τα στατιστικά (statistics) ή οι πληροφορίες που συλλέγονται κατά την διάρκεια του monitoring χρησιμοποιούνται από τον administrator - χρήστη για τον εντοπισμό κάποιου λάθους ή κάποιου bottleneck στο cluster. Το Storm UI daemon παρέχει τις εξής σημαντικές πληροφορίες: Cluster Summary, Topology Summary, Supervisor Summary, Topology Stats, στατιστικά των Spouts και των Bolts που τρέχουν , στατιστικά και μετρήσεις των executors αλλά και το Configuration της τοπολογίας. Το Storm UI ακούει στην 8080/tcp by default.

Sentiment Analysis

Η ανάλυση συναισθήματος είναι μία δύσκολη δουλειά, η οποία δυσκολεύει περισσότερο όταν η αναγνώριση του συναισθήματος αφορά ένα μικρό κείμενο όπως ένα tweet το οποίο περιορίζεται σε 140 χαρακτήρες. Για παράδειγμα, ένα tweet μπορεί να είναι πλούσιο σε ειρωνεία, το οποίο υποδεικνύεται μέσω hashtags, όπως το #irony ή έμμεσα γράφοντας "Μου αρέσει να δουλεύω όταν είμαι άρρωστος".
Η αποφασιστικότητα του πραγματικού συναισθήματος αυτού του κειμένου είναι μία πρόκληση, κυρίως λόγω του περιορισμένου μεγέθους και χαρακτηριστικών όπως οι συντομεύσεις και η slang. Επομένως το να αποκαλέσεις κάποιο κείμενο θετικό, αρνητικό ή ουδέτερο είναι αρκετά δύσκολη δουλειά.
Το πραγματικό νόημα μπορεί να είναι αρκετά διαφορετικό από αυτό που φαίνεται, όπως για παράδειγμα συμβαίνει με την ειρωνεία. Αυτό που εκφράζεται μπορεί να είναι ακριβώς το αντίθετο από αυτό που υπονοείται, όπως στην έκφραση "Δεν σου αρέσει ο σαρκασμός;".
Επίσης αυτό που δυσκολεύει περισσότερο αυτή τη δουλειά είναι η αναγνώριση της έντασης του συναισθήματος, όπως επίσης όσες περισσότερες κλάσεις τόσο περισσότερα χαρακτηριστικά θα χρησιμοποιηθούν στην ανάλυση συναισθήματος, ώστε να δώσουν το κατάλληλο βάρος στο αντίστοιχο tweet.
Για να αντιμετωπίσουμε αυτήν την πρόκληση, προτείνεται ένα σύστημα για την ανάλυση συναισθήματος σε transport ratio, βασιζόμενο σε επιλογή χαρακτηριστικών ενός tweet και η 'εκπαίδευση' ενός αλγορίθμου που θα είναι ικανός να προβλέψει το συναίσθημα ενός tweet. Δεδομένου μιας ομάδας από tweet πλούσια σε μεταφορά και ειρωνεία,

σκοπός είναι να καθορίσουμε αν ο χρήστης εξέφρασε ένα θετικό, αρνητικό ή ουδέτερο συναίσθημα, και πόσο έντονο ήταν αυτό.

Real-Time Ανάλυση Συναισθήματος και Visualization

Το Twitter λαμβάνει περίπου 190 εκατομμύρια tweets (μικρά κείμενα δημοσιευμένα στο Web) την ημέρα, στα οποία οι άνθρωποι μοιράζονται τα σχόλια και τις ιδέες τους τα για διάφορα θέματα. Αυτό το κείμενο τις περισσότερες φορές εκφράζει κάποιο συναίσθημα. Έτσι για να έχουμε μια εικόνα για αυτά τα συναισθήματα οποιαδήποτε στιγμή σε πραγματικό χρόνο και για όποιο λόγο θέλουμε, η δημιουργία ενός συστήματος που παρουσιάζει σε πραγματικό χρόνο τα συναισθήματα από το Twitter, είναι σημαντική.
Αυτό το σύστημα μπορεί να χρησιμοποιηθεί σε πολλές περιπτώσεις για την ανάλυση του συναισθήματος , όπως εκλογές, τελικός του Champions League ή Eurovision. Η ανάλυση του συναισθήματος στο σύστημα χωρίζεται σε θετικά, αρνητικά και ουδέτερα αλλά εμφανιζεται και το μέγεθος όλων των tweets σε πραγματικό χρόνο.

Περιγραφή συστήματος

Στόχος

Ο στόχος του συστήματος είναι να παρέχει στον χρήστη ένα real time data visualization dashboard. Αυτό το dashboard θα εμφανίζει πληροφορίες σχετικά με τον όγκο(volume), το συναίσθημα και τον τόπο που δημιουργήθηκαν τα tweets.

Ορισμοί

Publish/ Subscribe

Πρόκειται για ένα messaging πρότυπο όπου οι αποστολείς των μηνυμάτων ονομάζονται publishers και και οι αποδέκτες subscribers. Ο subscriber εγγράφεται σε μια ουρά δείχνοντας ενδιαφέρον για τα μηνύματα που υπάρχουν σε αυτή την ουρά χρησιμοποιώντας ένα subject ή content-based rule ως filter. Αυτό έχει ως αποτέλεσμα ένα σετ από rule-based subscriptions που σχετίζονται με την συγκεκριμένη ουρά. Στο run time οι publishers κάνουν post μηνύματα σε διάφορες ουρές. Η ουρά (με άλλα λόγια, οι μηχανισμοί παράδοσης), στη συνέχεια παραδίδει τα μηνύματα που ταιριάζουν με τα διάφορες subscriptions στους κατάλληλους subscribers.

Πλεονεκτήματα

Loose coupling

Οι Publishers είναι χαλαρά συνδεδεμένοι με τους subscribers, και δεν χρειάζεται να γνωρίζουν καν την ύπαρξή τους. Με το θέμα να είναι η εστίαση, οι publishers και οι subscribers επιτρέπεται να παραμείνουν σε άγνοια για την τοπολογία του συστήματος.
Ο κάθε ένας μπορεί να συνεχίσει να λειτουργεί κανονικά, ανεξάρτητα από τον άλλο.
Στα παραδοσιακά στενά συνδεδεμένα μεταξύ τους client–server πρότυπα, ο client δεν μπορεί να κάνει post μηνύματα στον server ενώ το server process δεν τρέχει, ούτε ο server μπορεί να λάβει

μηνύματα , εκτός αν ο client είναι σε λειτουργία. Πολλά pub/sub συστήματα αποσυνδέουν όχι μόνο τα locations των publishers και των subscribers, αλλά τα αποσυνδέουν προσωρινά.
Μια κοινή στρατηγική που χρησιμοποιείται από τους middleware αναλυτές με τέτοια pub/sub συστήματα είναι να 'κατεβάσει' έναν publisher ώστε να επιτρέψει στον subscriber να εργασθεί μέσω της συσσώρευσης(backlog) .

Scalability

Το Pub/sub παρέχει την δυνατότητα για καλύτερο scalability απο τον παραδοσιακό client–server, μέσω παράλληλης λειτουργίας, message caching, tree-based ή network-based routing, κλπ.
Ωστόσο, σε ορισμένους στενά συνδεδεμένους τύπους, τα high-volume enterprise περιβάλλοντα, ως συστήματα κάνουν scale up ώστε να γίνουν data centers με χιλιάδες servers να μοιράζονται το pub/sub infrastructure, τα τρέχοντα vendor συστήματα συχνά χάνουν αυτό το πλεονέκτημα; το scalability για τα pub/sub products με υψηλό φορτίο σε αυτά τα contexts αποτελεί μια πρόκληση για έρευνα.
Έξω από το enterprise environment, απο την άλλη πλευρά, το pub/sub πρότυπο έχει αποδείξει οτι το scalability του μεγαλώνει πολύ πιο πέρα από εκείνες ενός single data center, παρέχοντας Internet-wide distributed messaging μέσω web syndication protocols όπως το RSS και το Atom (standard).
Αυτά τα syndication πρωτόκολλα δέχονται υψηλότερο latency και έλλειψης της εγγύησης παράδοσης σε αντάλλαγμα με την δυνατότητα για low-end web server που συνδυάζει μηνύματα για (μελλοντικά) εκατομμύρια διαφορετικούς subscriber nodes.

Μειονεκτήματα

Τα πιο σημαντικά προβλήματα με τα pub/sub systems είναι οι παρενέργειες του κύριου πλεονεκτήματος : η αποσύνδεση του publisher από τον subscriber.

Περιγραφή υπάρχοντος συστήματος

Η web εφαρμογή για το Data visualization βασίστηκε σε ένα ήδη υπάρχον σύστημα για scalable και real time ανάλυση συναισθήματος (sentiment analysis) των Twitter data. Αυτό το σύστημα βασίζεται σε feature extraction των Tweets, χρησιμοποιώντας και μορφολογικά(morphological) features και semantic πληροφορίες . Μία supervised learning προσέγγιση έχει υιοθετηθεί, όπου διάφοροι classifiers (SVM, N. Bayes) εκπαιδεύονται βασισμένοι στα extracted features. Το Storm ως ένα real time computation σύστημα έχει επιλεχθεί με σκοπό την υλοποίηση των feature extraction και των classification tasks. Η Python (python 2.7)χρησιμοποιήθηκε για την ανάπτυξη το sentiment analysis κομματιού (preprocessing, classification). Το μέρος την τοπολογίας και των components (spout/bolts) έχει υλοποιηθεί σε Java. Τα Python scripts καλούνται και τρέχουν από τα Java bolts χρησιμοποιώντας το Storm multilang protocol (υποστηρίζει την ενοποίηση με πολλές γλώσσες προγραμματισμού, καθώς όλα τα δεδομένα που ανταλλάσσονται με τη διαδικασία κωδικοποιούνται σε JSON).

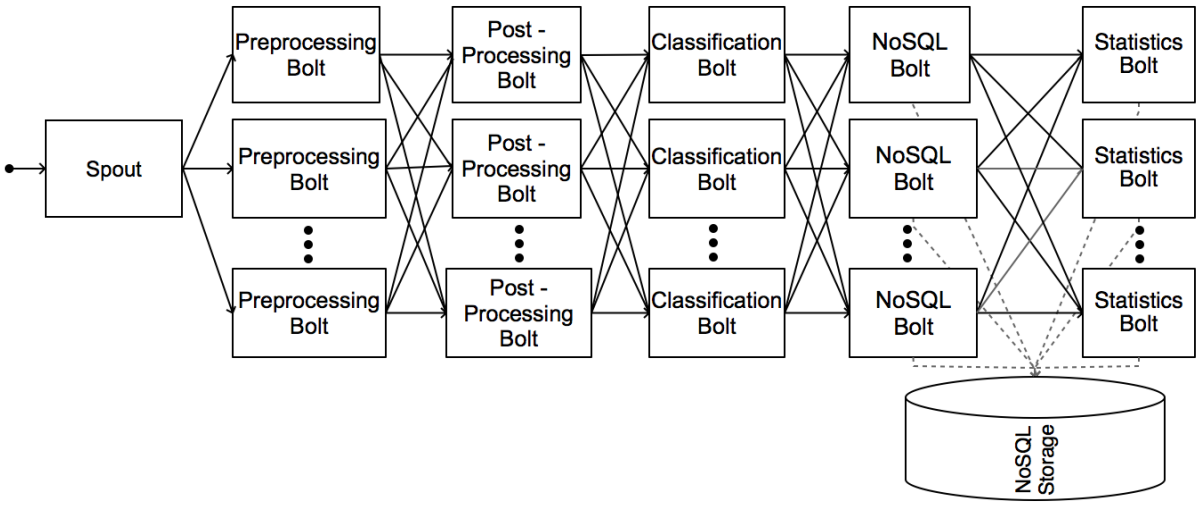

Figure 2. Existing System's Storm Topology

Η τοπολογία όπως έχουμε ήδη αναφέρει είναι ένα δίκτυο από spouts και bolts. Κάθε κόμβος σε μία τοπολογία περιέχεται το processing logic, και οι συνδέσεις μεταξύ των κόμβων υποδεικνύουν πως τα data θα πρέπει να διανεμηθούν στα spouts και τα bolts. Το stream grouping που επιλέχθηκε είναι το shuffle grouping. Σαν αποτέλεσμα, τα emitted tuples στέλνονται σε τυχαία επιλεγμένα tasks των bolts.
Μια τοπολογία πρέπει να σχεδιαστεί με τέτοιο τρόπο ώστε να επιτύχουμε το scalability και το reliability που το Storm προσφέρει. Διαφορετικά μέρη της τοπολογίας μπορούν να γίνουν scaled μεμονωμένα κάνοντας μικρές αλλαγές στο parallelism τους(πολλαπλά instances του bolt).

Topology Components

1. Twitter Spout
   Λαμβάνει από το stream μόνο Αγγλικά tweets
   Emits: Json string (το tweet όπως έρχεται από το stream)

2. Preprocessing Bolt:
   Το tweet που λαμβάνεται μαζί με όλη την extra πληροφορία χρησιμοποιείται για την δημιουργία του Tweet object στην preprocessing φάση
   Κάνει Extract τα Morphological Features,
   'Καθαρίζει' το κείμενο του tweet,
   Κάνει Extract άλλα features
   Emits: Json Tweet με όλη την extra information

3. Post Processing Bolt:

Επιλέγει από όλα τα features που έχουν συγκεντρωθεί στο preprocessing bolt, μόνο αυτά που χρησιμοποιήθηκαν στην 'εκπαίδευση' του classifier.
Emits: Json Tweet with με όλη την extra πληροφορία

4. Classification Bolt:

Η διαδικασία εκπαίδευσης έγινε off-line. Το μοντέλο του classifier, μαζί με τα μοντέλα των vectorizers οι οποίοι χρησιμοποιήθηκαν στην διαδικασία της εκπαίδευσης , αποθηκεύονται σε ένα κοινό φάκελο ο οποίος είναι διαθέσιμος στο Storm. Όταν το bolt τρέχει, ο classifier φορτώνει το pre-trained μοντέλο μόλις η python process ξεκινήσει. Όταν ένα νέο tuple (Tweet) φτάσει, τα features (feature_dict) αυτού του tuple μετατρέπονται σε κατάλληλο input για τον classifier (έχοντας φορτώσει τα αντίστοιχα vectorizer μοντέλα κλπ.) και γίνεται μία πρόβλεψη για το συγκεκριμένο μοντέλο. Το field "predicted_score" του Tweet ενημερώνεται με το score [-1, 1] και το Tweet γίνεται emit.
Emits: Json Tweet με όλη την extra information

5. Mongo Bolt:

Λαμβάνει το Json Tweet και το αποθηκεύει όπως είναι σε ένα mongo collection που ονομάζεται "Tweets"
Emits: Json Tweet με όλη την extra information

6. Statistics Bolt:

Υπολογίζει το σύνολο των θετικών, αρνητικών και ουδέτερων tweets που προέρχονται από το Mongo Bolt και το αποθηκεύει σε ένα mongo collection που ονομάζεται Calculation. Το bolt αυτό έπεται του Mongo Bolt ώστε να βεβαιωθεί πως τα tweets στα που χρησιμοποιούνται για τα στατιστικά θα έχουν αποθηκευτεί όλα στην βάση δεδομένων.

Περιγραφή νέου συστήματος

Αρχιτεκτονική

Η εφαρμογή όπως έχουμε ήδη αναφέρει, βασίστηκε στην παραπάνω τοπολογία. Για τις ανάγκες αυτής της εφαρμογής, η τοπολογία χρειάστηκε να αλλαχτεί. Το NoSql και το Statistics Bolt δεν χρησιμοποιούνται πια για λόγους απλούστευσης. Ένα νέο Report Bolt προστέθηκε το οποίο είναι υπεύθυνο για την επικοινωνία με την Redis και να τα κάνει publish στο "Sentiment" κανάλι. Το JSON string γίνεται publish περιέχει όλη την πληροφορία που υπολογίστηκε σε αυτό το Bolt. Ποιο συγκεκριμένα, υπολογίζεται το σύνολο των θετικών, αρνητικών και ουδέτερων tweets για εκείνη την ημέρα και από την στιγμή που ξεκίνησε η εφαρμογή να τρέχει. Επίσης περιέχει το μέρος που προήλθαν τα εισερχομένων Tweets, δηλαδή την χώρα και την περιοχή

ώστε να εμφανιστεί στον χάρτη. Η περιοχή αυτή παρουσιάζεται από ένα πολύγωνο(bounding box) το οποίο αποτελείται από γεωγραφικό πλάτος και μήκος (latitude και longitude αντίστοιχα) points pairs τα οποία όταν ενώνονται δημιουργούν μία περιοχή πάνω στον χάρτη.

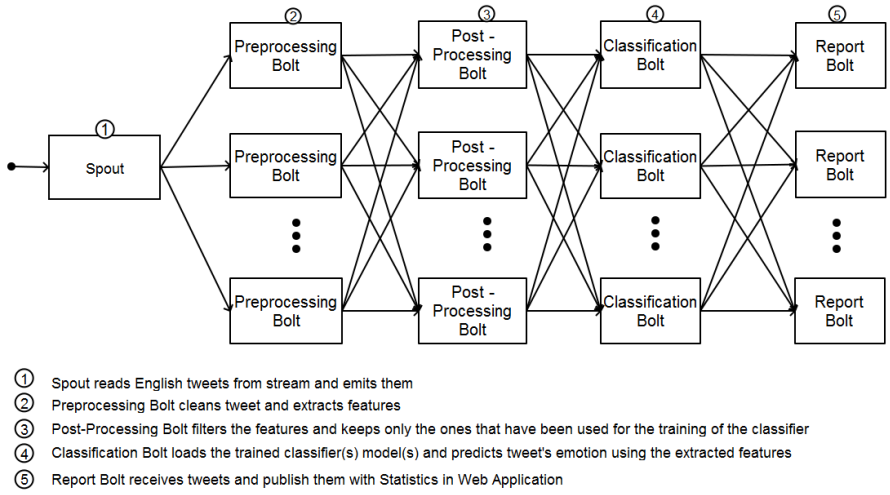

Figure 3. The modified for Web Application Topology

Web application architecture

Στην εικόνα 4 παρουσιάζεται ο σχεδιασμός της εφαρμογής.

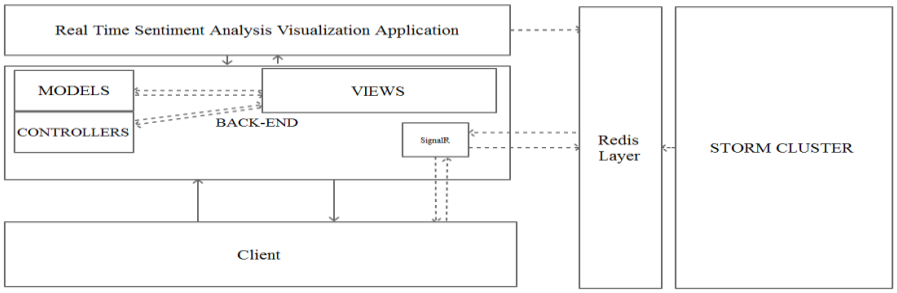

Figure 4. Overview of the System

External Libraries και Software

SignalR

Για την οπτικοποίηση δεδομένων σε πραγματικό χρόνο., το SignalR υπήρξε η κατάλληλη βιβλιοθήκη για την εν λόγω εφαρμογή. Το SignalR παρέχει τη δυνατότητα στιγμιαίας προώθησης περιεχομένου στους συνδεδεμένους client από τη στιγμή που αυτό γίνεται διαθέσιμο με χρήση λογισμικού από τη μεριά του server, αντί ο server να αναμένει τον client να ζητήσει νέα δεδομένα. Η σύνδεση μεταξύ του client και του server είναι μόνιμη, σε αντίθεση με μία κλασσική HTTP σύνδεση η οποία επανεγκαθίσταται για κάθε επικοινωνία. Το SignalR χρησιμοποιεί τη μεταφορά τύπου WebSocket, όπου αυτή είναι διαθέσιμη (αυτού του είδους οι μεταφορές εξαρτώνται από την υποστήτιξη σε HTML 5), και όποτε είναι απαραίτητο επαναχρησιμοποιεί παλαιότερους τύπους μεταφοράς (αν ο browser του χρήστη δεν υποστηρίζει HTML 5).

Η χρήση του SignalR σημαίνει πως πολλές επιπλέον λειτουργίες που υπό άλλες συνθήκες θα έπρεπε να ενσωματωθούν, είναι ήδη διαθέσιμες. Είναι σημαντικό να τονίσουμε ότι παρέχει την δυνατότητα εκμετάλλευσης του WebSocket, χωρίς να χρειάζεται να ανησυχούμε για τη δημιουργία λογισμικού ξεχωριστής διαδρομής για τους παλαιότερους clients. Το SignalR επίσης, απαλλάσσει από το άγχος των αναβαθμίσεων για το WebSocket, δεδομένου ότι το SignalR θα συνεχίσει να αναβαθμίζεται προκειμένου να υποστηρίζει αλλαγές στην υποκείμενη μεταφορά. Αυτό, σκοπό έχει να εξασφαλίσει ένα συνεκτικό interface για την εφαρμογή για όλες τις εκδόσεις του WebSocket.

Το SignalR είναι ιδανικό για διαδικτυακές εφαρμογές που απαιτούν μεγάλη συχνότητα αναβαθμίσεων από τον server, πχ. παιχνίδια σε πραγματικό χρόνο.

Το SignalR API παρέχει τη δυνατότητα δημιουργίας απομακρυσμένων κλήσεων διαδικασίας (server-to-client remote procedure calls (RPC)) μεταξύ του server και του client, οι οποίες καλούν JavaScript λειτουργίες σε client browsers (και άλλες client πλατφόρμες) από το .NET λογισμικό του server. Επιπλέον, το SignalR περιλαμβάνει API για τη διαχείριση των συνδέσεων, όπως για παράδειγμα, events σύνδεσης και αποσύνδεσης.

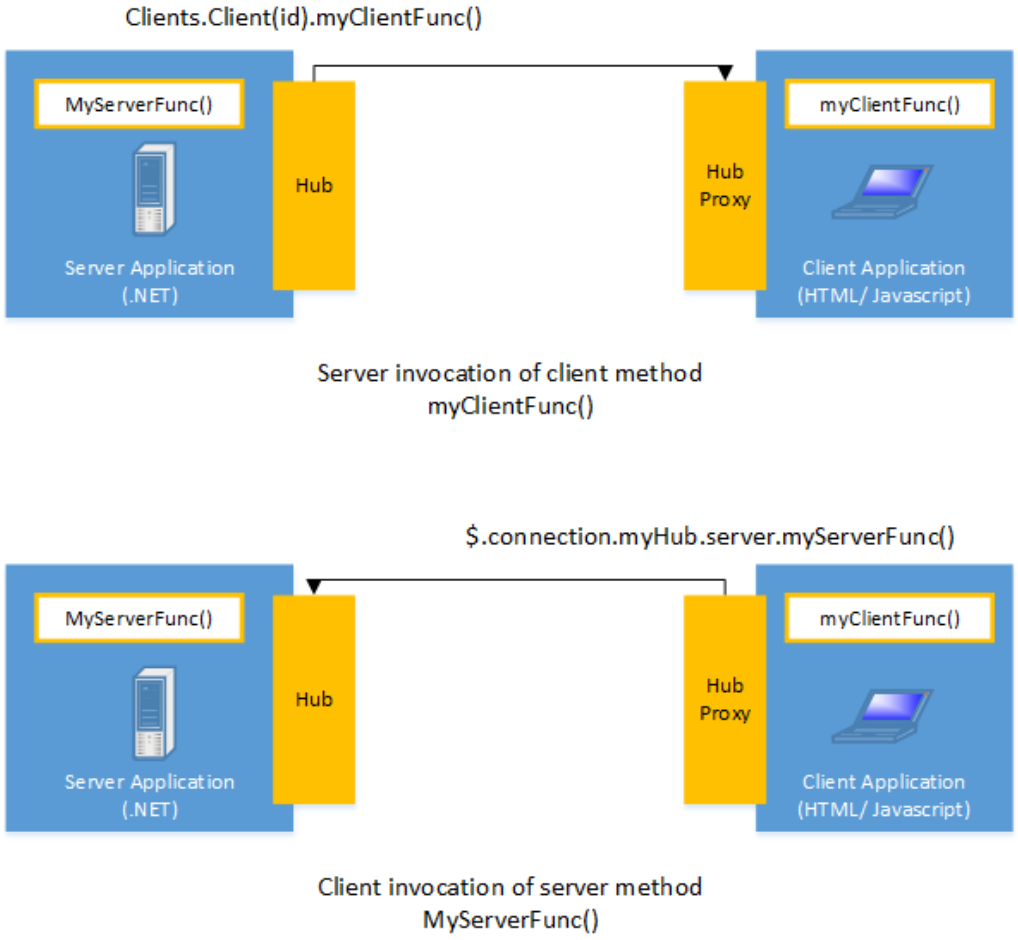

Figure 5. How signalR works

Το SignalR μπορεί να κάνει αυτόματη διαχείρηση της σύνδεσης και επιτρέπει την εκπομπή μηνυμάτων προς όλους τους συνδεδεμένους clients ταυτοχρόνως, όπως σε ένα chat room. Επιτρέπει την αποστολή μηνυμάτων σε συγκεκριμένους clients. Η σύνδεση μεταξύ client και server είναι μόνιμη, σε αντίθεση με μία κλασσική HTTP σύνδεση η οποία επανεγκαθίσταται για κάθε επικοινωνία. [7]

Το SignalR υποστηρίζει τη "server push" λειτουργία, κατά την οποία το λογισμικό του server μπορεί να καλέσει το λογισμικό του client σε ένα browser, χρησιμοποιώντας Remote Procedure Calls (RPC), αντί των καλεσμάτων που βασίζονται στο μοντέλο request-response και που είναι κυρίαρχα στο διαδίκτυο σήμερα.

Οι SignalR εφαρμογές μπορούν να κάνουν scale χιλιάδες clients χρησιμοποιώντας Service Bus, SQL Server ή Redis.

Connections και Hubs

Ένα connection αναπαριστά ένα απλό endpoint για την αποστολή single-recipient, grouped, ή broadcast μηνυμάτων. Το Persistent Connection API δίνει στον developer απευθείας πρόσβαση στο low-level communication protocol που το SignalR κάνει expose.

Ένα hub είναι ένα πιο πολύ high-level pipeline χτισμένο πάνω στο Connection API όπου επιτρέπει στον client και τον server να καλέσουν μεθόδους μεταξύ τους άμεσα. Το SignalR χειρίζεται την διεκπεραίωση στο μηχάνημα ως δια μαγείας, επιτρέποντας στους clients να καλέσουν μεθόδους στον server τόσο εύκολα όσο τις local μεθόδους, και αντίθετα.

Πως λειτουργεί το Hub

Πρέπει να σκεφτούμε το Hub ως μία κεντρικλη τοποθεσία για όλες τις συνδέσεις της εφαρμογής. Όταν ο server-side κώδικας καλεί μια μέθοδο στον client, , ένα πακέτο στέλνεται στην ενεργή μεταφορά που περιέχει το όνομα και τις παραμέτρους της μεθόδου που θέλουμε να κληθεί (όταν ένα αντικείμενο αποστέλλεται ως παράμετρος της μεθόδου, γίνεται serialized χρησιμοποιώντας JSON). Ο client ύστερα αντιστοιχεί το όνομα της μεθόδου στην μέθοδο που ορίστηκε στον client-side κώδικα. Αν υπάρχει αντιστοιχία, η client μέθοδος θα εκτελεστεί χρησιμοποιώντας τα deserialized parameter δεδομένα.

Υλοποίηση

Υπάρχουν δύο βασικά μέρη σε μια απλή Signalr εφαρμογή.
Πρώτα είναι το Server Side όπου πρέπει να χτίσουμε μια δομή που ονομάζεται *Hub*.

```
namespace TweetViz
{
    public class TweetHub : Hub
    {
        public void Send(string message, string name, string sessionId)
        {

        Clients.All.addMessageToPage(message,name, sessionId);

        }
    }
}
```

Figure 6. The function called to update signalR clients

Στον παραπάνω κώδικα το Clients Object χρησιμοποιείται για να ορίσει properties ή να καλέσει μεθόδους από εκεί. Το SignalR θα πάρει αυτές τις κλήσεις και θα τις μεταφράσει σε μηνύματα τα οποία στέλνονται σε κάθε client που είναι συνδεδεμένος στο hub.

Στην Client πλευρά χρησιμοποιούμε την SignalR jQuery βιβλιοθήκη για την επικοινωνία με ένα SignalR hub. Το βασικό καθήκον στον κώδικα είναι η δημιουργία ενός reference στο auto-generated proxy για το hub, δηλώνοντας ότι ο server μπορεί να κάνει push το content στους clients, και να ξεκινήσει την σύνδεση για την αποστολή μηνυμάτων στο hub.

Δήλωση reference στο hub proxy:

```
// Reference the auto-generated proxy for the hub.
chat = $.connection.tweetHub;
```

Figure 7. SignalR client-side connection

Το request στο SignalR/Hubs είναι ένα αρχείο το οποίο δημιουργείται δυναμικά από το SignalR και δίνει στην σελίδα όλη την απαραίτητη πληροφορία ώστε να μιλήσει πίσω σε όποιο Hub έχουμε δημιουργήσει.

Παρακάτω φαίνεται πως μπορούμε να δημιουργήσουμε callback μέθοδο στο script. Η hub class στον server καλή αυτή την μέθοδο για να κάνει push τα content updates στον κάθε client. Η παράμετρος sessionId σώζεται αρχικά με subscription και στην συνέχεια χρησιμοποιείται για να ελέγξει κατά πόσο το broadcasted μήνυμα είναι για τον συγκεκριμένο client.

```
// Create a function that the hub can call back to display messages.
chat.client.addMessageToPage = function (message, total, sessionId) {
    // to do

}
```

Figure 8.SignalR client-side addMessageToPage μέθοδος

Στο τέλος πρέπει να κάνουμε *open* και *start* την σύνδεση με το hub.

```
// Start the connection.
$.connection.hub.start();
```

Figure 9.SignalR start connection

Το MVC προσφέρει αρχιτεκτονικά πλεονεκτήματα — βοηθάει στην δημιουργία ενός καλύτερου και πιο οργανωμένου κώδικα, και ως εκ τούτου πιο maintainable. Το Model-View-Controller (MVC) αρχιτεκτονικό πρότυπο διαχωρίζει την εφαρμογή σε τρία κύρια μέρη : το μοντέλο, το view, και τον controller. Το ASP.NET MVC framework παρέχει ένα διαφορετικό πρότυπο στα ASP.NET Web Forms για την δημιουργία Web applications. Τέλος το ASP.NET MVC framework είναι 'ελαφρύ' , και highly testable presentation framework το οποίο (as with Web Forms-based applications) είναι integrated ήδη υπάρχοντα ASP.NET features, όπως master pages και membership-based authentication. [8]

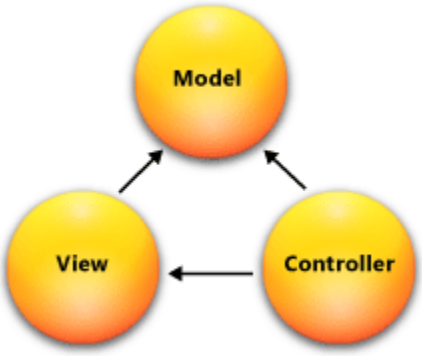

Figure 10. The MVC pattern

Models. Τα μοντέλα είναι κομμάτια της εφαρμογής που υλοποιούν την λογική για το data domain της εφαρμογής. Συχνά τα model objects ανακτούν και αποθηκεύουν το model state στην βάση δεδομένων. Το μοντέλο διαχειρίζεται την συμπεριφορά και τα data του application domain, απαντά στα requests για πληροφορίες σχετικά με την κατάσταση του (συνήθως από το view), και απαντά επίσης σε instructions για την αλλαγή της κατάστασης που βρίσκονται (συνήθως από τον controller).

Views. Τα Views είναι τα components που εμφανίζουν το user interface (UI) της εφαρμογής. Τυπικά , αυτό το UI δημιουργείται από το model data. Το view χειρίζεται την εμφάνιση των πληροφοριών.

Controllers. Οι Controllers είναι τα components που συνεργάζονται με το μοντέλο και τον χρήστη , και στο τέλος επιλέγει το view που εμφανίζει το UI. Σε μία MVC εφαρμογή, το view παρουσιάζει μόνο πληροφορίες. Ο controller χειρίζεται και απαντά σε αλληπίδραση με τον χρήστη και το input.

Πλεονεκτήματα μιας MVC-Based Web Εφαρμογής

Το ASP.NET MVC framework έχει τα εξής πλεονεκτήματα:

- Είναι πιο εύκολο να διαχειριστεί η πολυπλοκότητα χωρίζοντας την εφαρμογή σε μοντέλα model, view, και στους controller.
- ΙΔεν χρησιμοποιεί view state ή server-based forms. Αυτό κάνει το MVC framework ιδανικό για τους developers που θέλουν να έχουν πλήρη έλεγχο της συμπεριφοράς της εφαρμογής.
- Χρησιμοποιεί ένα Front Controller πρότυπο που επεξεργάζεται τα Web application requests σε έναν controller. Αυτό δίνει την δυνατότητα να σχεδιαστεί μία εφαρμογή που υποστηρίζει ένα πλούσιο routing infrastructure
- Παρέχει καλύτερη υποστήριξη για test-driven development (TDD).

Πως λειτουργεί το MVC

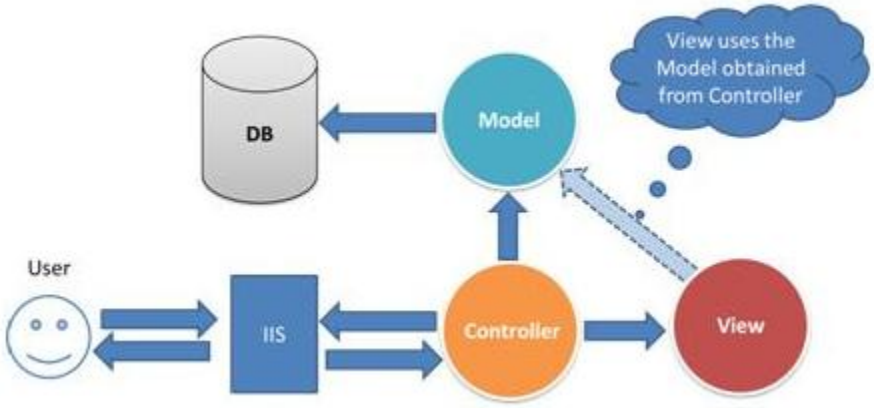

Figure 11. Abstract view of MVC functionality

- Οι χρήστες κάνουν κάποιο request για resources στον server (βάζοντας το URL στον browser).
- Το request έρχεται αρχικά στον controller (το routing engine είναι υπεύθυνο για να αποφασίσει ποιο request θα χειριστεί ο κάθε controller).
- Ο Controller αν χρειάζεται μιλάει στο μοντέλο για data.
- Το Model χρησιμοποιεί την βάση δεδομένων (ή κάποιες άλλες πηγές δεδομένων) και επιστρέφει τα δεδομένα στον controller.
- Ο Controller επιλέγει το κατάλληλο view.
- Ο Controller περνάει τα δεδομένα στο επιλεγμένο view για να εμφανιστούν τα δεδομένα
- Ο Controller στέλνει το view πίσω στον user.

Redis

- H Redis είναι ένα open source (BSD licensed), in-memory data structure store, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. [9]

Pub/Sub

Η Redis είναι ένα γρήγορο και stable Publish/Subscribe messaging σύστημα. Τα SUBSCRIBE, UNSUBSCRIBE και PUBLISH υλοποιούν το Publish/Subscribe messaging πρότυπο όπου οι senders (publishers) δεν ειναι προγραμματισμένοι να στέλνουν τα μηνυματα τους σε συγκεκριμένους receivers (subscribers). Αντίθετα , τα published μηνύματα χαρακτηρίζονται σε channels, χωρίς να γνωρίζουν τι subscribers μπορεί να υπάρχουν .Οι Subscribers εκφράζουν το ενδιαφέρουν τους για ένα ή περισσότερα channels, και να λάβουν μόνο τα μηνύματα που τους ενδιαφέρουν, χωρίς να γνωρίζουν τι publishers υπάρχουν. Αυτή η αποσύνδεση των publishers και των subscribers μπορούν να επιτρέψουν μεγαλύτερο scalability και μια πιο δυναμική τοπολογία δικτύου.

Format των pushed μηνυμάτων

Το μήνυμα είναι ένα Array reply με 3 elements.
Το πρώτο element είναι ένα είδος μηνύματος:
`subscribe`: σημαίνει ότι έχουμε κάνει επιτυχώς subscribe στο κανάλι το οποίο μας έχει δοθεί ως το δεύτερο element στην απάντηση τρίτο argument αναπαριστά τον αριθμό των καναλιών που εκείνην την στιγμή έχουμε κάνει subscribe.
`unsubscribe`: σημαίνει ότι έχουμε κάνει επιτυχώς unsubscribed από το κανάλι το οποίο μας έχει δοθεί ως το δεύτερο element στην απάντηση. Το τρίτο argument όπως είπαμε αναπαριστά τον αριθμό των καναλιών που εκείνην την στιγμή έχουμε κάνει subscribe. όταν το τελευταίο argument είναι μηδέν, δεν είμαστε πλέον subscribed σε κανένα κανάλι, και ο client μπορεί να χρησιμοποιήσει οποιοδήποτε είδους Redis command αφού είμαστε εκτός του Pub/Sub state.
`message`: είναι ένα μήνυμα που λαμβάνεται ως απάντηση απο την PUBLISH command που έχει γίνει από έναν άλλο client. Το δεύτερο element είναι το όνομα του originating καναλιού, και το τρίτο argument είναι το πραγματικό message payload.

Τύπος των μηνυμάτων που γίνονται published από το Report Bolt

Tweet message
Κάθε φορά που ένα tweet έχει επεξεργαστεί και έχει γίνει classified, ένα μήνυμα από το report bolt γίνεται publish στο Redis “Sentiment” κανάλι. Ο τύπος αυτού του μηνύματος είναι “tweet”

και περιέχει τις συντεταγμένες (polygon), το συναίσθημα του tweet που έχει προβλεθεί και το ίδιο το κείμενο. Η επόμενη εικόνα δείχνει ένα παράδειγμα ενός τέτοιου tweet μηνύματος.


```
{
  "msg_type": "tweet",
  "msg": {
    "position": {
      "polygon": [{
        "lng": 40.900789,
        "lat": -73.911271
      }, {
        "lng": 40.988346,
        "lat": -73.911271
      }, {
        "lng": 40.988346,
        "lat": -73.810443
      }, {
        "lng": 40.900789,
        "lat": -73.810443
      }],
      "country_code": "US",
      "country": "United States"
    },
    "predicted_score": -1,
    "tweet": "french fries https://t.co/M19uNwMM55"
  }
}
```


Figure 12. Example of a "tweet" type published message

Statistics Message

Κάθε τρία δευτερόλεπτα ένα μήνυμα που περιέχει στατιστικά στοιχεία γίνεται publish στο "Sentiment" κανάλι, περιέχοντας τα top 10 hashtags, των αριθμό των tweets που επεξεργάστηκαν εκείνη την μέρα, τον αριθμό των tweets που έχουν επεξεργαστεί από την στιγμή που ξεκίνησε το Strom και τον αριθμό των θετικών, αρνητικών και ουδέτερων tweets. Ένα τέτοιο μήνυμα παρουσιάζεται στην παρακάτω εικόνα.

```
{
  "msg_type": "statistics",
  "msg": {
    "top10hashtags": {
      "#YouSay": 1,
      "#ProjectMgmt": 1,
      "#McDonalds": 1,
      "#royaltyfreemusic": 1,
      "#Wien": 1,
      "#EAFW": 1,
      "#snowing": 1,
      "#Nursing": 1,
      "#inshadows": 1,
      "#futbol": 1
    },
    "total_of_the_day": 336,
    "total_since_storm_started": 336,
    "negative": 319,
    "neutral": 9,
    "positive": 8
  }
}
```


Figure 13. Example of a "statistics" type published message

GoogleCharts

Τα Google Charts είναι μία βιβλιοθληκη για την δημιουρία interactive διαγραμμάτων για browsers και mobile devices.
Υπάρχει μεγάλη ποικιλία από charts και για κάθε αναγκη. Τα Google charts είναι customizable. Είναι cross-browser συμβατά. Και τέλος διευκολύνει την σύνδεση σε δεδομένα σε πραγματικό χρόνο χρησιμοποιώντας μια ποικιλία από data connection εργαλεία και πρωτόκολλα[10].
Τα Google charts χρησιμοποιήθηκαν για την παρουσίαση των στατιστικών των εισερχόμενων tweets.
Πιο συγκεκριμένα εμφανίζει σε ένα pie chart το σύνολο των tweets, των αρνητικών των θετικών και των ουδέτερων.

GoogleMaps

Το Google Maps API χρησιμοποιήθηκε με σκοπό την εμφάνιση της τοποθεσίας που προέρχονται τα tweets. Το Google Maps API είναι μια JavaScript βιβλιοθήκη[11] . Η απεικόνιση στο google map είναι ένα πολύγωνο, το οποίο καλύπτει την περιοχή που προέρχεται το tweet [15].

ChartJs

Το ChartJS είναι μια Javascript βιβλιοθήκη για την δημιουργία interactive charts. Το Chart.js είναι modular, και κάθε ένα από τα chart types έχουν διαχωριστεί , ώστε να είναι εύκολο να φορτωθούν μόνο τα chart types που χρειάζονται για το project. Το Chart.js παρέχει default simple support για canvas tooltips στο hover/touch, αλλά μπορούμε να επεκτείνουμε τα chart interactions ώστε να κάνουμε trigger κάποια events στην εφαρμογή. Το Chart.js χρησιμοποιεί το HTML5 canvas element. Υποστηρίζεται από όλους τους συγχρόνους browsers[12]. Στην εφαρμογή χρησιμοποιήθηκε ένα line chart, ώστε να εμφανίσουμε το σύνολο των εισερχόμενων tweets (θετικά, αρνητικά και ουδέτερα) σε σύγκριση με τον χρόνο.

Bootstrap 3

With Bootstrap 3 μπορούμε να έχουμε HTML, CSS, και JS framework για την δημιουργία responsive σελίδων, για mobile συσκευές και web applications. Το Bootstrap κάνει το front-end web development γρηγορότερο και ευκολότερο. [13]

D3.js

Το D3.είναι μια JavaScript βιβλιοθήκη για την διαχείριση των documents βασισμένων σε data. Το D3 βοηθάει στην εμφάνιση των δεδομένων με την χρήση HTML, SVG, και CSS. Η έμφαση του D3 στα web standards δίνει όλες τις δυνατότητες των σύγχρονων browsers χωρίς την σύνδεση σε ένα proprietary framework, συνδυάζονται powerful visualization components και μία data-driven προσέγγιση στην διαχείριση του DOM [14]. Το D3.js χρησιμοποιήθηκε σε αυτό το project για να εμφανίσουμε με όμορφο τρόπο τα top Hashtag που έρχονται από το stream. Κάθε φορά που η τοπολογία κάνει publish τα tweets, τα top Hashtags εμφανίζονται στο dashboard.

Εφαρμογή

Εκκίνηση Storm

Το project δημιουργήθηκε στο Eclipse. Αρχικά πρέπει να έχουμε εκκινήσει την MySQL και τον Redis Server. Για να εκκινήσουμε το Storm σε local mode (local cluster) πρέπει πρώτα να εκκινήσουμε το Zookeeper deamon, ξεκινώντας το zookeeper server (zkServer.cmd). Μετά από αυτό πρέπει να εκκινήσουμε τα Nimbus, Supervisor και UI deamons σε 3 διαφορετικά command prompts, με τις εξής εντολές από το storm's bin φάκελο αντίστοιχα:

- `Storm nimbus`
- `Storm supervisor`
- `Storm ui`

Μετά την εκκίνηση όλων των deamons,μπορούμε να σιγουρευτούμε ότι το Storm τρέχει ανοίγοντας στον browser το http://localhost:8080/ .
Σε ένα άλλο command prompt πρέπει να δημιουργήσουμε ένα jar το οποίο θα γίνει deploy στο cluster με την εντολή mvn clean package, από τον φάκελο του project, όπου βρίσκεται το pom.xml.
Τώρα μπορούμε να κάνουμε deploy την τοπολογία μας στο local cluster με την jar εντολή:

- `storm jar TweetTopology -0.1.1 – SNAPSHOT .jar tweettopology`

Αν ξαναφορτώσουμε την Storm UI σελίδα, θα πρέπει τώρα να δούμε την "TweetTopology" και να μπορούμε να επιλέξουμε τα links της τοπολογίας για τα bolts τα spout αλλά και για τις άλλες πληροφορίες που το Strorm UI μας δίνει ώστε να βεβαιώσουμε ότι η τοπολογία επεξεργάζεται τα data.

Αν δεν χρειάζεται να κάνουμε deploy την τοπολογία και απλά να την τρέξουμε για developing και testing του συστήματος, μπορούμε απλά να την τρέξουμε στο Eclipse. Το output εμφανίζεται στην κονσόλα του Eclipse.

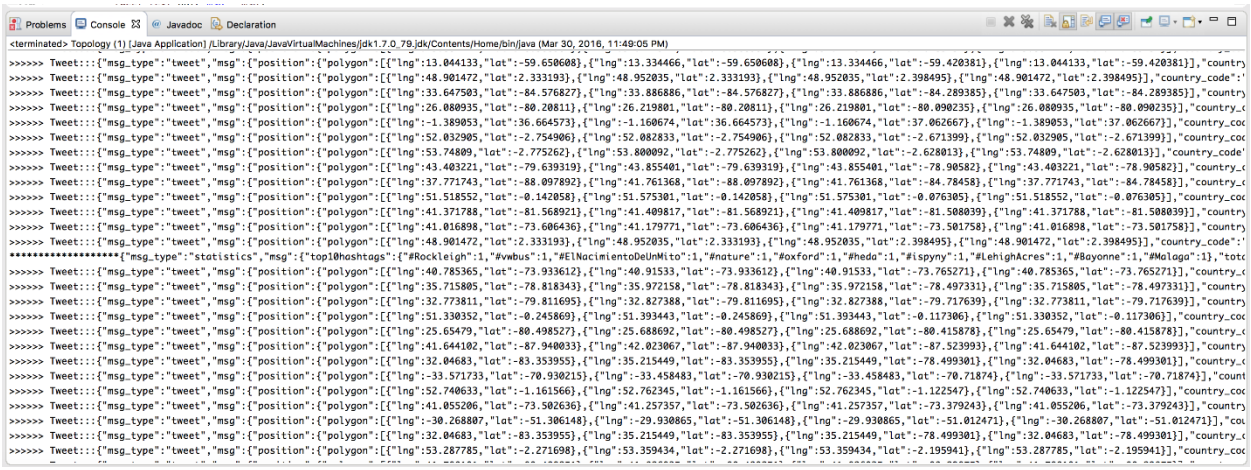

Figure 14. The debug output of Storm

Start TweetViz

Η εφαρμογή για το real time data visualization δημιουργήθηκε στο Visual Studio. Έτσι το μόνο πράγμα που πρέπει να κάνουμε αφού σιγουρευτούμε πως η Redis τρέχει, είναι να τρέξουμε το project και να το ανοίξουμε στον browser στο localhost:33195.

Σελίδες Εφαρμογής

Παρακάτω εμφανίζονται οι σελίδες της εφαρμογής.

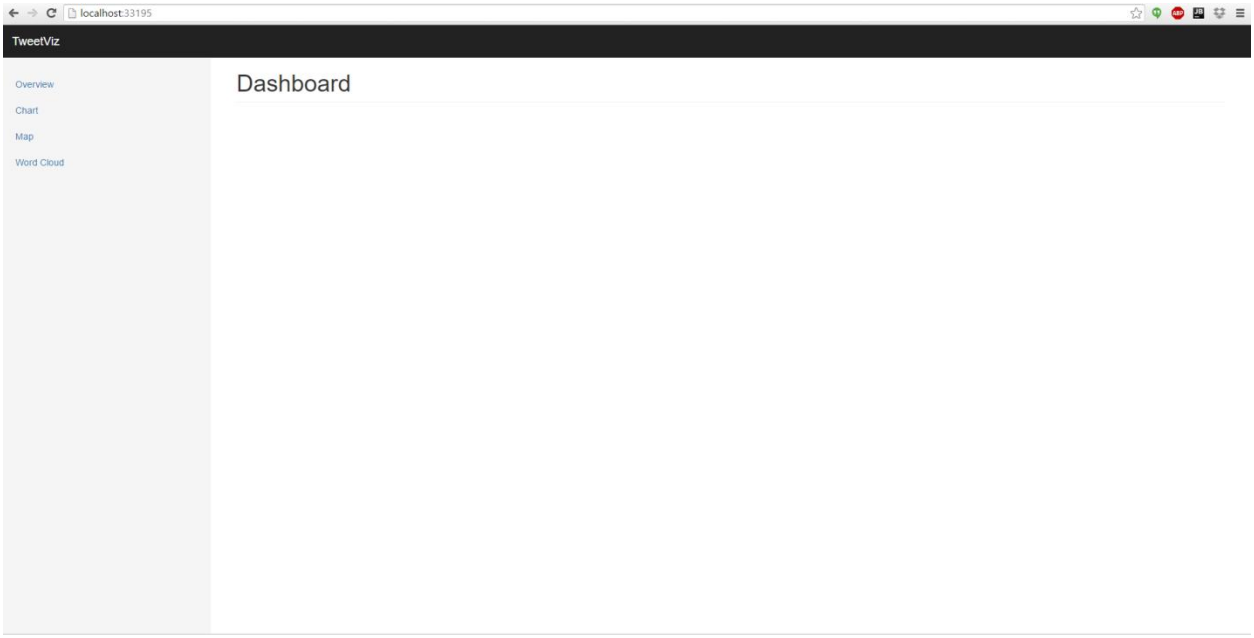

Figure 15.The main page

Στην παραπάνω εικόνα βλέπουμε την Main Page(dashboard) με τις επιλογές που μπορούμε να κάνουμε.

Στην συνέχεια όπως φαίνεται στην εικόνα 16 η εφαρμογή ξεκινάει να εμφανίζει τα δεδομένα

Overview

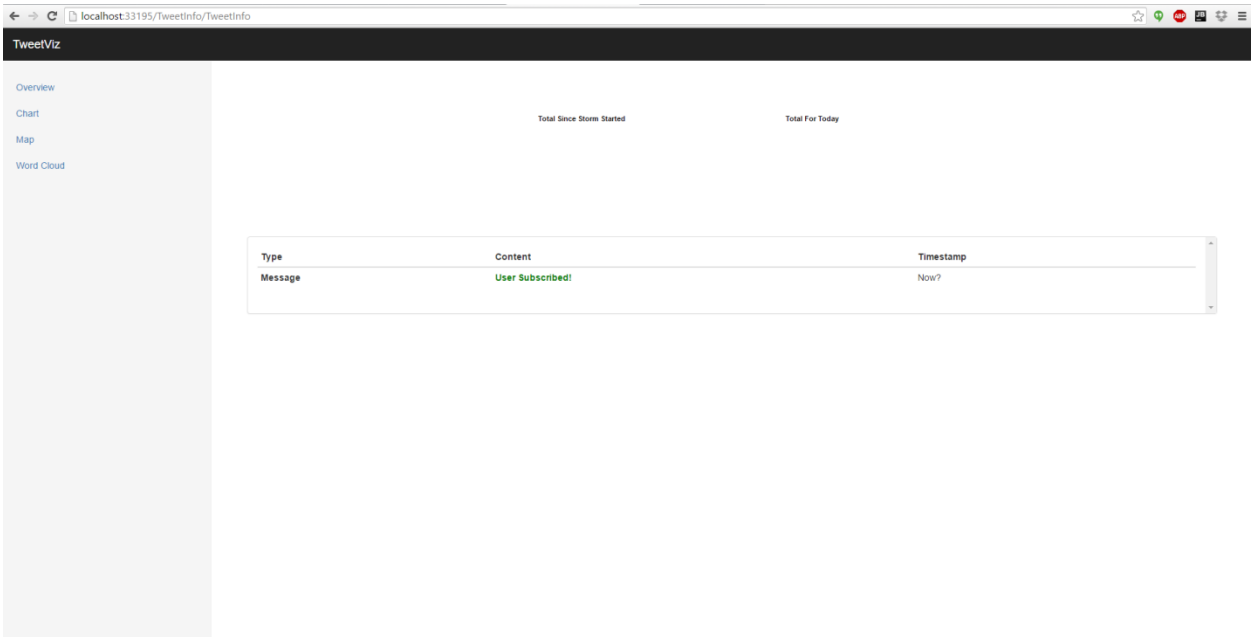

Figure 16. The Overview page without data

Στην συνέχεια εμφανίζονται όλα τα tweets που έρχονται σε πραγματικό χρόνο από το stream

Figure 17.Overview page with data

Map

Στην παρακάτω εικόνα βλέπουμε τον χάρτη που εμφανίζεται ο τόπος προέλευσης των tweets

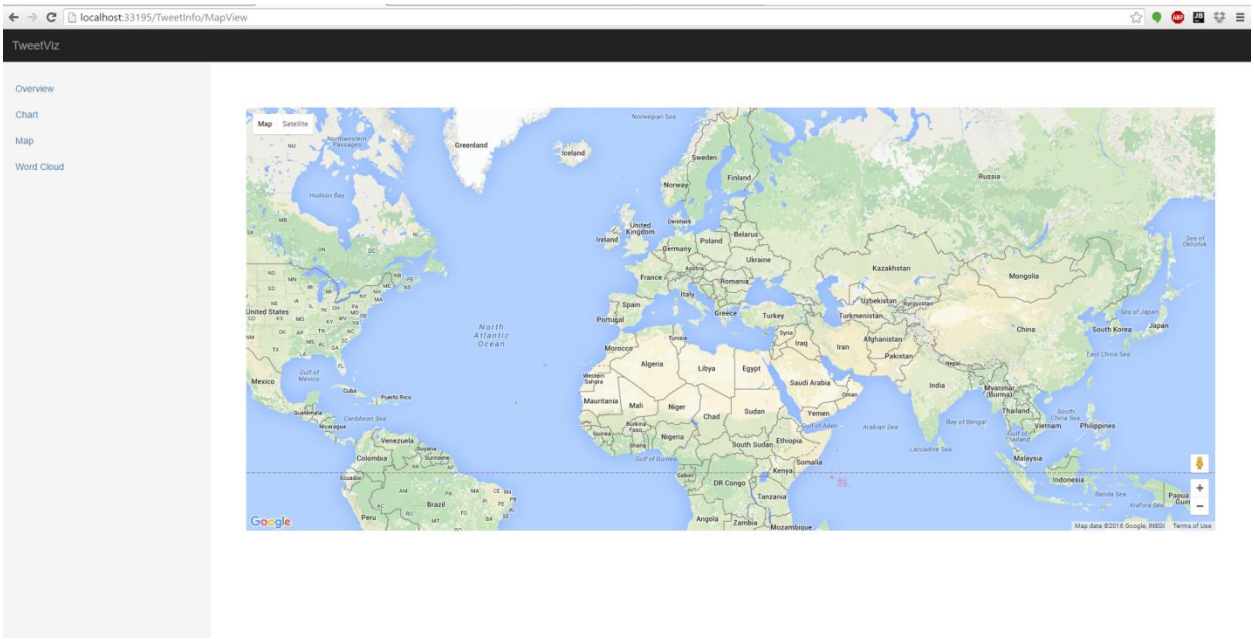

Figure 18. Map with polygons

Chart

Στην παρακάτω εικόνα εμφανίζεται ένα chart με τα tweets σε σχέση με τον χρόνο

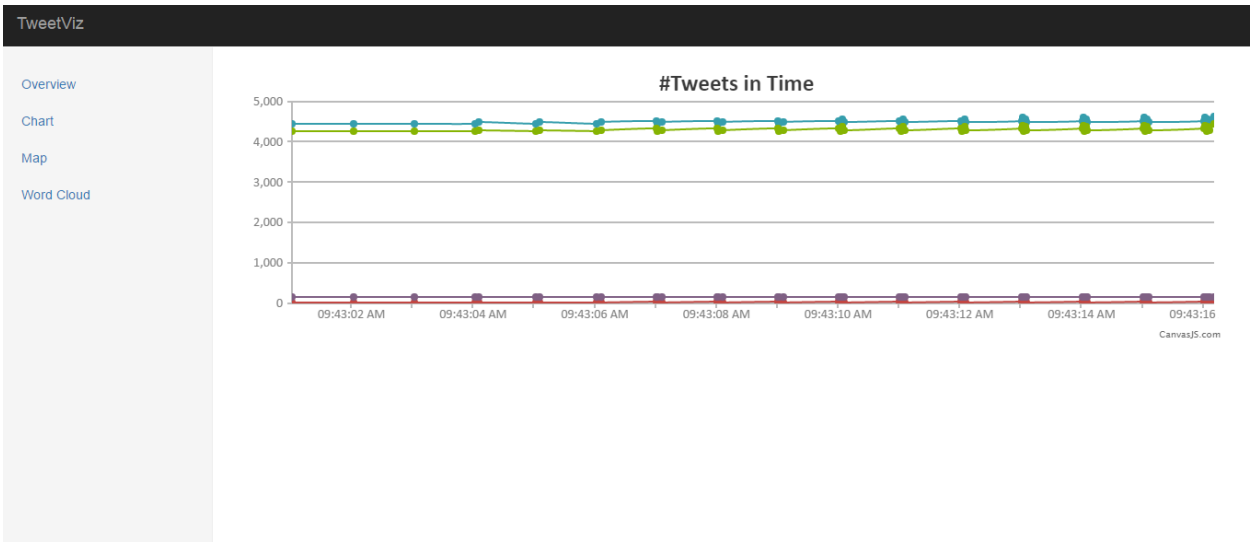

Figure 19. The chart page

WordCloud

Τέλος στην παρακάτω εικόνα εμφανίζονται τα top Tweets των τελευταίων 10 δευτερολέπτων

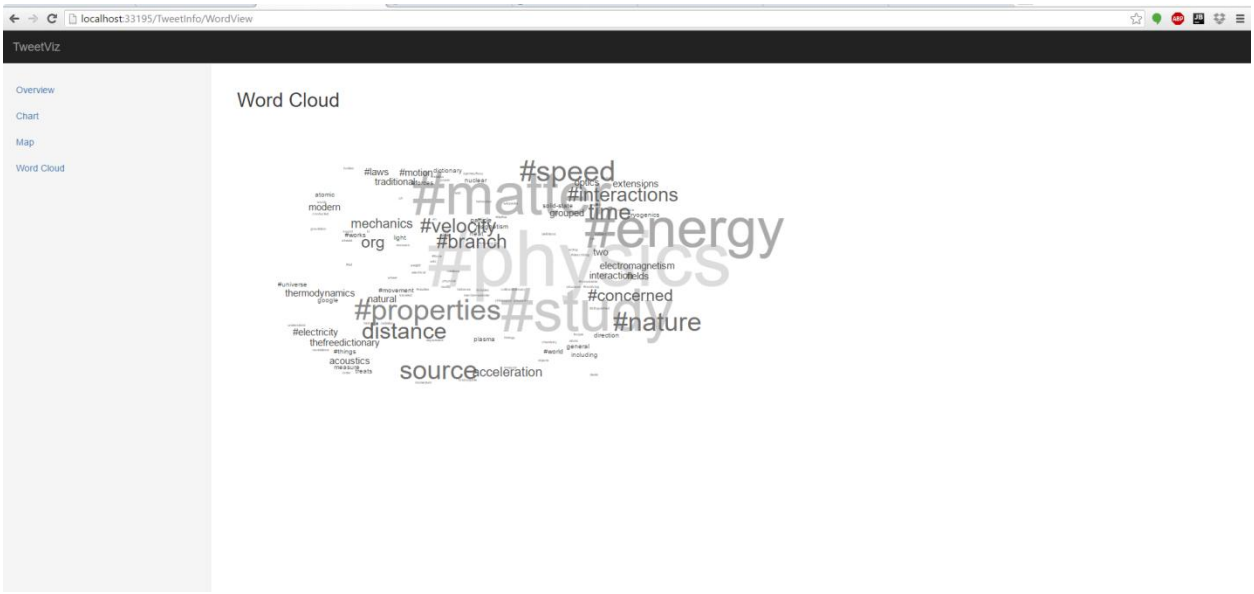

Figure 20. The word cloud page

Συμπεράσματα και μελλοντική εργασία

Η παροχή πληροφοριών σε πραγματικό χρόνο, δεν είναι ένα ασήμαντο έργο. Ωστόσο υπάρχουν πολλά εργαλεία και πολλές τεχνικές που μπορούν να χρησιμοποιηθούν για να παρέχουν το καλύτερο δυνατό αποτέλεσμα. Συνδυάζοντας τεχνολογίες που υποστηρίζουν επεξεργασία σε πραγματικό χρόνο(Storm), βιβλιοθήκες και frameworks που βοηθούν στην real time επικοινωνία client-server όπως το SignalR , βιβλιοθήκες για real time visualizations και front–end development όπως το D3 τότε μπορούμε να επιτύχουμε στην δημιουργία μιας εφαρμογής για την εμφάνιση δεδομένων σε πραγματικό χρόνο από κάθε πηγή.

Για μελλοντική εργασία, θα μπορούσαμε να βελτιώσουμε τα υπάρχοντα charts προσθέτοντας περισσότερη πληροφορία. Τέλος μια ενδιαφέρουσα προσθήκη θα ήταν η δυνατότητα για user-feedback ώστε ο χρήστης να μπορεί να κάνει evaluate το συναίσθημα του tweet και να κρατάει στατιστικά σε περίπτωση διαφορών.

Βιβλιογραφία

[1] Maria Karanasou, Anneta Ampla, Christos Doulkeridis, Maria Halkidi, Scalable and Real-time Sentiment Analysis of Twitter Data.

[2] Storm: Distributed and fault-tolerant real-time computation. Available: http://storm.apache.org/.

[3] History of Apache Storm and lessons learned , [Electronic], Available: http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html.

[4] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel*, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh, Storm @Twitter ,2014

[5] Foteini Alvanaki, Sebastian Michel, Tracking Set Correlations at Large Scale, 2014

[6] Ming Hao, Christian Rohrdant, Halldór Janetzko, Umeshwar Dayal Daniel A. Keim, Lars-Erik Haug, Mei-Chun Hsu Hewlett-Packard Labs, Visual Sentiment Analysis on Twitter Data Streams, 2011

[7] SignalR, [Electronic] , Available: https://asp.net/org

[8] MVC, [Electronic] , Available: https://asp.net/mvc

[9] Redis, [Electronic] , Available: https://redis.io

[10] Google Charts , [Electronic] , Available: https://developers.google.com/chart/

[11] Google Maps API , [Electronic] , Available: https://developers.google.com/maps/

[12] Chart.js , [Electronic] , Available: http://www.chartjs.org/

[13] Bootsrap 3, [Electronic] , Available: http://getbootstrap.com

[14] D3.js , [Electronic] , Available: https://d3js.org/

[15] Adam Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, Robert C. Miller, TwitInfo: Aggregating and Visualizing Microblogs for Event Exploration, 2011