ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατανεμημένη Αποθήκευση και Δεικτοδότηση Πληροφοριών Κοινωνικών Δικτύων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Α. Μυτιλήνης

Επιβλέπων: Νεκτάριος Κοζύρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2012

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατανεμημένη Αποθήκευση και Δεικτοδότηση Πληροφοριών Κοινωνικών Δικτύων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Α. Μυτιλήνης

Επιβλέπων: Νεκτάριος Κοζύρης

Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Μαρτίου 2012.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης Αναπληρωτής Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Επίκουρος Καθηγητής Ε.Μ.Π.	Δημήτριος Τσουμάκος Επίκουρος Καθηγητής ΙΟΝΙΟΥ ΠΑΝΕΠΙΣΤΗΜΙΟΥ

Αθήνα, Μάρτιος 2012

(Υπογραφή)

.....

ΜΥΤΙΛΗΝΗΣ ΙΩΑΝΝΗΣ

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

Περίληψη

Τα τελευταία χρόνια έχει σημειωθεί μια έκρηξη δεδομένων στο Internet. Στην έκρηξη αυτή σημαντικό ρόλο παίζουν τα κοινωνικά δίκτυα, τα οποία κερδίζουν συνεχώς όλο και περισσότερους χρήστες. Ένα από τα πλέον γνωστά κοινωνικά δίκτυα είναι το Twitter. Τα δεδομένα, που υπάρχουν στο Twitter, αποτελούν σημαντική πηγή πληροφοριών, που για να αξιοποιηθεί όμως, θα πρέπει πρώτα να οργανωθεί με κάποιο αποτελεσματικό τρόπο.

Ο όγκος των δεδομένων που υπάρχουν στο Twitter, αλλά και ο ρυθμός με τον οποίον παράγονται καθιστά τη διαχείρισή τους ιδιαίτερα προκλητική κι ενδιαφέρουσα. Σκοπός της παρούσας διπλωματικής είναι η ανάπτυξη εφαρμογής που θα διαχειρίζεται τα δεδομένα του Twitter σε πραγματικό χρόνο με τη χρήση κατανεμημένων τεχνικών.

Η λογική που ακολουθείται στην εφαρμογή μας είναι η εξής. Αρχικά γίνεται συλλογή δεδομένων από το Twitter σε πραγματικό χρόνο. Στη συνέχεια τα δεδομένα αυτά αναλύονται με NLP εργαλεία για την εξαγωγή πληροφοριών σχετικών με τις γραμματικές και συντακτικές ιδιότητες των tweets. Οι πληροφορίες που εξάγονται από την ανάλυση χρησιμοποιούνται μαζί με αυτές που προσφέρει το Twitter API, για την δεικτοδότηση των δεδομένων. Αφού δεικτοδοτηθούν τα δεδομένα αποθηκεύονται σε μια κατανεμημένη βάση δεδομένων. Με τον τρόπο αυτό δημιουργείται μια υπηρεσία έξυπνης αναζήτησης.

Η εφαρμογή αναπτύχθηκε σε περιβάλλον Cloud και συγκεκριμένα στο Google App Engine, την κατανεμημένη πλατφόρμα της Google. Με τη χρήση του App Engine, η εφαρμογή εκμεταλλεύεται όλα τα πλεονεκτήματα του Cloud Computing όπως η ανοχή σε σφάλματα, η διαθεσιμότητα και κυρίως η κλιμακωσιμότητα του συστήματος, στοιχείο ιδιαίτερα σημαντικό για μια εφαρμογή διαχείρισης δεδομένων, αφού απαιτείται να ανταποκρίνεται σε αυξανόμενο όγκο δεδομένων. Επιπλέον, το Google App Engine επιτρέπει τη διάθεση της αναπτυχθείσας εφαρμογής ως υπηρεσίας ιστού.

Μελετώντας χαρακτηριστικά σενάρια χρήσης αποδεικνύεται ότι η υπηρεσία αυτή που δημιουργήσαμε προσφέρει όντως μια πιο έξυπνη αναζήτηση από αυτήν του Advanced Twitter Search, καθώς τα αποτελέσματα που επιστρέφει βρίσκονται εννοιολογικά πιο κοντά σε αυτό που αναζητούν κάθε φορά οι χρήστες της εφαρμογής μας.

Λέξεις Κλειδιά

Twitter, Twitter API, NLP, Cloud Computing, Google App Engine, Advanced Twitter Search

Abstract

In the last few years, a data explosion has been noticed on the Internet. Social networks, as they gain more and more users, have contributed a lot in this overload of data. One of the most popular social networks is Twitter. The data that exist in Twitter can be considered as a valuable source of information. In order to exploit this information though, it must firstly be organised in an efficient way.

The amount of Twitter's data and the fact that these data are produced in real-time makes their management really challenging. Thus, there is a need for the development of data management applications. In the present thesis, we present such an application. The aim of our application is to manage real-time data that come from Twitter.

The sequence of steps that is followed in our application is the following. In the beginning, data are collected from Twitter. After that, the collected data are processed with NLP tools, as we want to retrieve information relative to the grammatical and syntactic properties of the tweets. The information, that is retrieved from this analysis, as well as the one that Twitter API offers, are used for the data indexing. Once the data are indexed, they are stored in a distributed database. In this way, we have created a clever search service.

Our application is developed in Cloud environment and specifically in Google AppEngine which is Google's distributed platform for the development of web applications. By using AppEngine, our application takes advantage of all Cloud Computing benefits, such as fault tolerance, availability and scalability. Especially scalability is very important for data management applications, since they are supposed to respond to huge amount of data. We also mention that our application belongs to the Software as a Service category and is provided as a service through the web.

As a result, we showed that the application we created, offers a more intelligent search than the one that Advanced Twitter Search offers.

Keywords

Twitter, Twitter API, NLP, Cloud Computing, Google App Engine, Advanced Twitter Search

Στην οικογένειά μου και στους φίλους μου Άγγελο, Άρτεμη και Πάνο, που είναι πάντα δίπλα μου...

Πίνακας Περιεχομένων

Πίνακας Σχημάτων

Πίνακας Πινάκων

1) Εισαγωγή

1.1) Κίνητρο

Την τελευταία δεκαετία έχει σημειωθεί παγκοσμίως μια τεράστια έκρηξη στην παραγωγή δεδομένων. Το ποσοστό της πληροφορίας που ανταλλάσσεται μέσω του Internet αυξάνεται όλο και περισσότερο. Ενδεικτικά, το 1993, το ποσοστό της πληροφορίας που ανταλλασσόταν μέσω Internet ήταν το 1% της συνολικής πληροφορίας, που ανταλλασσόταν μέσω τηλεπικοινωνιών. Το 2000, το ποσοστό αυτό αυξήθηκε στο 51%, ενώ το 2007 άγγιξε το 97% [1].

Η αύξηση αυτή των δεδομένων στο Internet, οφείλεται στην αλλαγή του τρόπου παραγωγής και διαχείρισής τους. Με την εμφάνιση του Web 2.0, ο χρήστης, εκτός από καταναλωτής, γίνεται και παραγωγός πληροφορίας. Το Internet δεν χρησιμεύει απλά για πρόσβαση σε δεδομένα. Οι web εφαρμογές δίνουν τη δυνατότητα στους χρήστες, να αλληλεπιδρούν μεταξύ τους και να μοιράζονται πληροφορίες. Οι πληροφορίες αυτές δεν είναι κατ' ανάγκη αρχεία κειμένου. Κείμενο, φωτογραφίες, video κι άλλα είδη αρχείων μεταφορτώνονται διαρκώς στο Διαδίκτυο.

Στην έκρηξη αυτή των δεδομένων έχει συμβάλει κι άλλος ένας παράγοντας: το hardware. Η πτώση των τιμών του υλικού(υπολογιστές, συσκευές αποθήκευσης κλπ.) δίνει τη δυνατότητα σε όλο και περισσότερους χρήστες, να έχουν πρόσβαση στα μηχανήματα αυτά και κατά συνέπεια, στην ανάκτηση, αλλά και παραγωγή πληροφοριών. Επίσης, η αισθητή βελτίωση της ταχύτητας των οικιακών διαδικτυακών συνδέσεων τα τελευταία χρόνια, έχει κάνει την πλοήγηση στο Internet αρκετά πιο βολική κι ευχάριστη ως διαδικασία κι έχει δώσει τη δυνατότητα στους χρήστες, να μεταφορτώνουν όλο και πιο ογκώδη δεδομένα.

Για τους παραπάνω λόγους, το Internet πλέον κατακλύζεται από πληθώρα δεδομένων. Τα δεδομένα αυτά είναι διαφόρων ειδών, όπως επιστημονικά, χρηματιστηριακά κλπ. Ο όγκος όμως αυτών των δεδομένων είναι τέτοιος που επιβάλλει την ανάπτυξη ειδικών συστημάτων για τη διαχείρισή τους. Χωρίς τη χρήση των συστημάτων αυτών, τα οποία ανήκουν στην περιοχή της Ανάκτησης Πληροφοριών (Information Retrieval-IR), είναι πλέον αδύνατο να αξιοποιήσουμε τις πληροφορίες που περιέχονται στα δεδομένα αυτά.

Στην παρούσα διπλωματική εργασία, μας ενδιαφέρει η διαχείριση των δεδομένων, που έχουν ανεβάσει στο Internet, οι χρήστες των κοινωνικών δικτύων. Τα κοινωνικά δίκτυα αποτελούν κατ'

εξοχήν παράδειγμα εφαρμογής του Web 2.0. Σε αυτές τις ιστοσελίδες, οι χρήστες μπορούν να ανεβάσουν ό,τι είδος δεδομένων θέλουν(αρχεία εικόνων, ήχου, βίντεο, κειμένου). Επίσης, οι χρήστες των κοινωνικών δικτύων συνδέονται μεταξύ τους, εξού και ο όρος “κοινωνικά δίκτυα”. Η διασύνδεση αυτή των χρηστών αυξάνει το πλήθος των δεδομένων, που μεταφορτώνονται στο Internet, καθώς πολλοί χρήστες επιθυμούν να μοιραστούν πληροφορίες με άλλους συγκεκριμένους χρήστες, με τους οποίους είναι συνδεδεμένοι. Επίσης, αξίζει να σημειώσουμε, ότι η διασύνδεση αυτή επηρεάζει τη συμπεριφορά των χρηστών και η επιρροή αυτή, που ασκείται, υπάρχει κρυμμένη στα δεδομένα, που ανταλλάσσουν μεταξύ τους.

Χαρακτηριστικά παραδείγματα κοινωνικών δικτύων είναι το Facebook και το Twitter. Με βάση τα επίσημα στατιστικά, σήμερα, το Facebook αριθμεί πάνω από 800 εκατομμύρια ενεργούς χρήστες [2], ενώ το Twitter έχει πάνω από 200 εκατομμύρια. Επίσης, για να έχουμε μια ιδέα του πλήθους των δεδομένων, που μπορούν να παράξουν αυτοί οι χρήστες, αναφέρουμε, ότι στο Facebook ανεβαίνουν κατά μέσο όρο πάνω από 250 εκατομμύρια φωτογραφίες τη μέρα [2] και στο Twitter δημοσιεύονται κατά μέσο όρο περίπου 140 εκατομμύρια tweets την μέρα [3].

Καταλαβαίνουμε επομένως, ότι ανά πάσα στιγμή, στα κοινωνικά δίκτυα παράγεται πλήθος νέας πληροφορίας. Αναλύοντας επομένως το περιεχόμενο αυτών των ιστοσελίδων, ή όπως λέμε διαφορετικά, αναλύοντας τον “ιστό πραγματικού χρόνου” (real-time Web), μπορούν να εξαχθούν ιδιαίτερα χρήσιμες πληροφορίες σχετικά με τις συμπεριφορές και τα συναισθήματα των χρηστών. Οι πληροφορίες, που μπορούμε να εξάγουμε μπορεί να είναι σχετικές με τη δημοτικότητα ενός ατόμου [4] ή ενός προιόντος [6], αλλά μπορεί να είναι και αρκετά πιο σύνθετες, όπως η πρόβλεψη ενός εκλογικού αποτελέσματος [7]. Κατανοώντας την αξία και τη δύναμη των πληροφοριών, που κρύβουν τα tweets, η ομάδα του Twitter, παρουσίασε το 2011 το Rainbird(ένα κλιμακώσιμο σύστημα αναλυτικής επεξεργασίας δεδομένων πραγματικού χρόνου).

Το Twitter όμως δεν είναι η μόνη εταιρία, η οποία συνειδητοποίησε την αξία των δεδομένων των κοινωνικών δικτύων. Πολλές εταιρίες ενδιαφέρονται για την ανάλυση των δεδομένων αυτών και για το λόγο αυτό έχουν αναπτύξει πληροφοριακά συστήματα, που συλλέγουν κι αναλύουν τα δεδομένα αυτά. Λόγω της φυσικής δομής των ίδιων των εταιριών, τα συστήματα αυτά είναι συνήθως κατανεμημένα. Ιδιαίτερα σήμερα, με την ευρεία διάδοση του Cloud Computing στον ακαδημαικό αλλά και στον επιχειρηματικό κόσμο, πολλές εταιρίες καταφεύγουν εκεί για την ανάπτυξη των εφαρμογών τους. Στην περίπτωση αυτή του Cloud Computing, οι εφαρμογές παρέχονται ως διαδικτυακές υπηρεσίες(SaaS), ενώ οι πόροι του συστήματος είναι κατανεμημένοι.

Δεδομένης αυτής της γενικής στροφής προς το Cloud Computing, μεγάλες εταιρίες πληροφορικής, όπως η Amazon, η Google και η Microsoft, αξιοποιώντας την υποδομή που έχουν, έχουν αναπτύξει πλατφόρμες(EC2, Google AppEngine, Microsoft Azure αντίστοιχα) για την ανάπτυξη εφαρμογών στο Cloud. Οι πλατφόρμες αυτές επιτρέπουν στους χρήστες του Cloud, να αναπτύξουν την εφαρμογή τους, χωρίς να χρειάζεται να απασχοληθούν εκ των προτέρων με τους πόρους(υπολογιστικούς, αποθηκευτικούς, δικτυακούς κλπ.), που θα χρειαστούν ή με την κλιμακωσιμότητα του συστήματός τους. Για την ανάπτυξη της εφαρμογής, που θα παρουσιάσουμε στην εργασία αυτή, στραφήκαμε κι εμείς προς μια κατανεμημένη λύση, χρησιμοποιώντας το App Engine της Google.

1.2) Σκοπός και σύντομη περιγραφή της εργασίας

Έχοντας ως κίνητρο τα παραπάνω, κατανοώντας την ανάγκη για διαχείριση του τεράστιου όγκου των δεδομένων που υπάρχει σήμερα, αλλά και αναγνωρίζοντας τη σημασία των πληροφοριών, που ρέουν στα κοινωνικά δίκτυα, στην εργασία αυτή, αναπτύξαμε μια εφαρμογή, η οποία χρησιμοποιεί δεδομένα του Twitter για την εξαγωγή πληροφοριών από αυτά .

Σκοπός της εργασίας είναι η ανάπτυξη ενός συστήματος, το οποίο επιτρέπει ένα προχωρημένο είδος αναζήτησης, μέσω του οποίου μπορούμε να παίρνουμε ταυτόχρονα πληροφορίες για το εννοιολογικό περιεχόμενο των tweets, που επιστρέφονται. Παράλληλα, η εφαρμογή μας, δεδομένου ότι έχει αποθηκευμένο ένα σύνολο από tweets και το έχει επεξεργαστεί, μέσω εργαλείων επεξεργασίας φυσικής γλώσσας(Natural Language Processing-NLP), μπορεί να αποτελέσει τη βάση και για άλλου είδους εφαρμογές, που να επιτελούν διαφορετικές εργασίες, όπως παραδείγματος χάρη sentiment analysis.

Η εφαρμογή μας συλλέγει δεδομένα από το Twitter και τα αποθηκεύει σε μια κατανεμημένη βάση δεδομένων, αφότου πρώτα τα δεικτοδοτήσει με βάση διάφορα χαρακτηριστικά τους. Τα χαρακτηριστικά, με βάση τα οποία δεικτοδοτούνται τα tweets, προκύπτουν είτε άμεσα, μέσω του Twitter API, είτε έπειτα από την επεξεργασία των tweets, μέσω εργαλείων επεξεργασίας φυσικής γλώσσας(Natural Language Processing-NLP).

Με τη δεικτοδότηση, με βάση τα χαρακτηριστικά, που παρέχει το Twitter API, πετυχαίνουμε κάτι ανάλογο του advanced search του Twitter. Έτσι, μπορούμε να αναζητήσουμε, αποδοτικά,tweets στη βάση μας με βάση το όνομα του χρήστη που τα δημοσίευσε, με βάση κάποια λέξη κλειδί, με βάση την ημερομηνία δημοσίευσης κλπ.

Με τα NLP εργαλεία μπορούμε να εξάγουμε πιο σύνθετες πληροφορίες από ένα tweet. Εφαρμόζοντας συντακτική ανάλυση στο κείμενο του tweet, προκύπτουν πληροφορίες, όπως ποιο είναι το ρήμα, ποιο το υποκείμενο, το αντικείμενο ή και ορισμένοι προσδιορισμοί και πώς αυτά συσχετίζονται μεταξύ τους. Γνωρίζοντας τα παραπάνω, έχουμε πλέον πληροφορίες για τη σημασιολογία του tweet που πρότερα δεν ήταν γνωστές. Δεικτοδοτώντας τα tweets με βάση αυτές τις πληροφορίες, έχουμε πλέον τη δυνατότητα να κάνουμε μια πιο εξεζητημένη αναζήτηση στη βάση μας, η οποία θα επιστρέφει αποτελέσματα πιο κοντά σε αυτό που πραγματικά ψάχνουμε. Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να βρούμε τα tweets που σχετίζονται με δηλώσεις του Barack Obama. Αναζητώντας απλά με βάση λέξεις κλειδιά θα μπορούσαμε να υποβάλουμε στη βάση ένα ερώτημα, το οποίο θα αναζητούσε tweets, που περιέχουν τις λέξεις “Barack”, “Obama” και “stated” συγχρόνως. Ένα τέτοιο ερώτημα όμως, θα μπορούσε και να επιστρέψει ένα tweet όπως αυτό: “Well, as I stated on Facebook earlier..2012 Obama get my vote again. Yes, that is my final answer. GOP candidates are less impressive.#YEP”, το οποίο προφανώς δεν είναι αυτό που ζητάμε. Αντίθετα, αν μπορούσαμε να ρωτήσουμε τη βάση μας ποια tweets έχουν σαν ρήμα το “state”(ή όποια άλλη μορφή του, πχ “stated”) και υποκείμενό του το “Obama” ή “Barack Obama”, τότε, σίγουρα, θα πετυχαίναμε καλύτερα αποτελέσματα, καθώς το ερώτημά μας, θα στηριζόταν και στη σημασία του νοήματος του tweet εκτός από τις λέξεις, που περιέχει.

Την εφαρμογή αυτή την αναπτύξαμε στο Google AppEngine, μια κατανεμημένη πλατφόρμα, που παρέχει η Google για Cloud Computing. Όπως αναφέραμε, υπάρχει μια γενική τάση προς την ανάπτυξη εφαρμογών στο “Υπολογιστικό Νέφος”. Ακολουθώντας τις τάσεις της εποχής επομένως, αλλά και συνειδητοποιώντας τα πλεονεκτήματα που προσφέρει το Cloud Computing, επιλέξαμε μια από τις μεγάλες, γνωστές πλατφόρμες κι αποφασίσαμε να αναπτύξουμε εκεί την εφαρμογή μας. Αν και τα πλεονεκτήματα αυτά του Cloud Computing θα παρουσιαστούν αναλυτικά στη συνέχεια, σε αυτό το σημείο θα αναφέρουμε ότι διευκολύνει την ανάπτυξη και τη συντήρηση της εφαρμογής, ενώ ταυτόχρονα μειώνει και το ρίσκο της επένδυσης σε μια συγκεκριμένη εφαρμογή.

Το AppEngine έχει τη δική του βάση δεδομένων, το Datastore. Το Datastore είναι μια NO-SQL βάση, στην οποία και αποθηκεύσαμε το σύνολο των tweets που συλλέξαμε. Το γεγονός ότι είναι NO-SQL καθώς και ορισμένες επιπλέον ιδιαιτερότητες, που έχει το Datastore, επέφεραν ορισμένα ζητήματα, τα οποία κληθήκαμε να αντιμετωπίσουμε. Τέτοια ζητήματα ήταν η επιλογή του σχήματος για την καλύτερη αναπαράσταση των δεδομένων μας, όπως και η διασφάλιση της συνέπειας(consistency) του συστήματος.

Επίσης, αναφέρουμε ότι το AppEngine δεν προσφέρεται για υπολογισμούς γενικού σκοπού, αλλά είναι μια πλατφόρμα ειδικά σχεδιασμένη για την ανάπτυξη διαδικτυακών εφαρμογών(web applications). Η λογική με βάση την οποία λειτουργεί η εφαρμογή μας είναι η εξής. Στο background τρέχει μια διαδικασία,που συνδέεται στο Twitter, λαμβάνει δεδομένα από το Twitter, αναλύει τα tweets, εξάγει τα χαρακτηριστικά με βάση τα οποία θα τα δεικτοδοτήσει και τέλος, αφού δεικτοδοτήσει τα tweets, τα αποθηκεύει στο Datastore. Δεδομένου ότι πρόκειται για διαδικτυακή εφαρμογή, οι παραπάνω εργασίες μπορούν να γίνουν μόνο με την εκτέλεση κάποιου servlet. Έχουμε ρυθμίσει επομένως το σύστημά μας κατά τέτοιο τρόπο, ώστε η διεύθυνση του servlet αυτού να καλείται περιοδικά και η βάση με τα tweets μας να ανανεώνεται.

Η εφαρμογή μας προσφέρεται ως διαδικτυακή υπηρεσία. Το μόνο που χρειάζεται κάποιος, για να μπορέσει να την χρησιμοποιήσει, είναι σύνδεση στο Internet. Οποιοσδήποτε χρήστης τώρα, μπορεί να χρησιμοποιήσει την εφαρμογή μας, απλά με τη χρήση ενός φυλλομετρητή ιστού(web browser). Στη σελίδα της εφαρμογής μας, θα έχει τη δυνατότητα να υποβάλει ερωτήματα στη βάση, που έχουμε στο Datastore, και να πάρει ως αποτέλεσμα τα αντίστοιχα tweets.

1.3) Δομή της εργασίας

Στο Κεφάλαιο 2, θα περιγράψουμε το Twitter. Θα παρουσιάσουμε τον τρόπο με τον οποίο συνδέονται μεταξύ τους οι χρήστες σε αυτό καθώς και το είδος πληροφοριών που ρέουν στο εσωτερικό του. Θα δούμε ποια είναι η χρησιμότητα των πληροφοριών αυτών και για ποιο λόγο επιβάλλεται η ανάπτυξη εφαρμογών, που αξιοποιούν τις πληροφορίες αυτές. Τέλος, θα παρουσιάσουμε το Twitter API, το οποίο και χρησιμοποιήσαμε στην εφαρμογή μας για τη συλλογή δεδομένων από το Twitter.

Στο Κεφάλαιο 3, θα συζητήσουμε για την ανάγκη ανάπτυξης εφαρμογών ανάκτησης πληροφοριών (information retrieval). Επίσης, θα παρουσιάσουμε τα εργαλεία επεξεργασίας φυσικής γλώσσας, που χρησιμοποιήσαμε, έτσι ώστε να εξάγουμε από τα tweets περισσότερες πληροφορίες από αυτές που παρέχει το Twitter API.

Παραπάνω αναφέραμε ότι για την ανάπτυξη της εφαρμογής μας στραφήκαμε σε μια κατανεμημένη λύση και συγκεκριμένα στο Google App Engine. Προτού περιγράψουμε το App Engine, κρίνουμε σκόπιμο να περιγράψουμε στο Κεφάλαιο 4 τι είναι το Cloud Computing. Θα δώσουμε ορισμένες βασικές αρχές και χαρακτηριστικά του και θα χωρίσουμε σε κατηγορίες τα είδη των υπηρεσιών που μπορούν να προσφερθούν μέσω αυτού.

Στο Κεφάλαιο 5, θα περιγράψουμε το Google App Engine. Το Google App Engine είναι η κατανεμημένη πλατφόρμα πάνω στην οποία αναπτύξαμε την εφαρμογή μας και η περιγραφή της είναι αναγκαία για την κατανόηση της αρχιτεκτονικής της εφαρμογής μας.

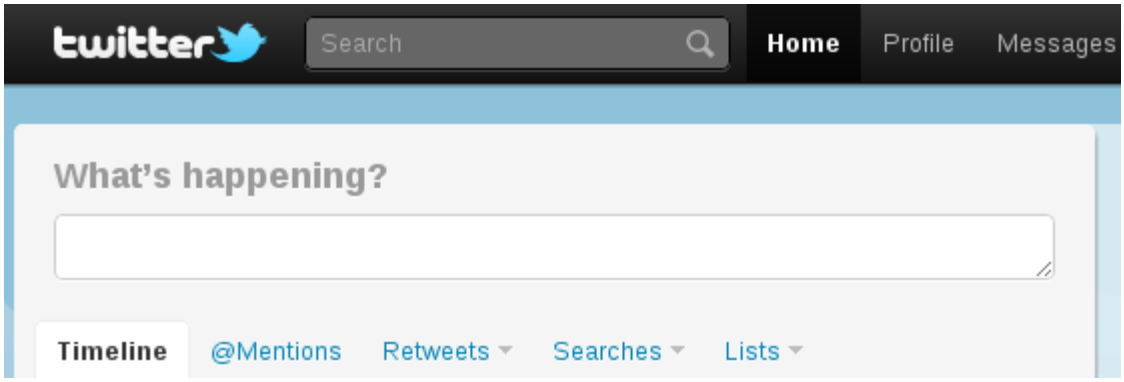
Στο Κεφάλαιο 6, θα παρουσιάσουμε την εφαρμογή μας, το TweetFinder11. Θα περιγράψουμε την αρχιτεκτονική της, τις πληροφορίες που κρίναμε άξιες προς δεικτοδότηση καθώς και το σχήμα δεδομένων που χρησιμοποιήθηκε για την αποθήκευση των πληροφοριών αυτών.

Τέλος, στο Κεφάλαιο 7, θα παρουσιάσουμε ορισμένα σενάρια χρήσης της εφαρμογής μας, έτσι ώστε να μπορέσουμε να την αξιολογήσουμε και να δούμε σε ποιο βαθμό επιτυγχάνει το στόχο της.

2) Twitter

2.1) Περιγραφή

Το Twitter είναι ένα κοινωνικό δίκτυο, που δημιουργήθηκε το 2006 από τον Jack Dorsey και σήμερα έχει πάνω από 200 εκατομμύρια χρήστες. Η λογική του είναι ότι οι χρήστες του μπορούν να δημοσιεύουν, ανά πάσα στιγμή, αυτό που σκέφτονται, μέσω ενός μηνύματος. Υπάρχει όμως ένας περιορισμός. Το μήνυμα αυτό, γνωστό και ως tweet, δεν θα πρέπει να ξεπερνά τους 140 χαρακτήρες. Από πολλούς μάλιστα έχει χαρακτηριστεί ως το SMS του Internet. Στην παρακάτω εικόνα φαίνεται η βασική του οθόνη, η οποία ζητάει από το χρήστη να δημοσιεύσει τι συμβαίνει αυτή τη στιγμή.

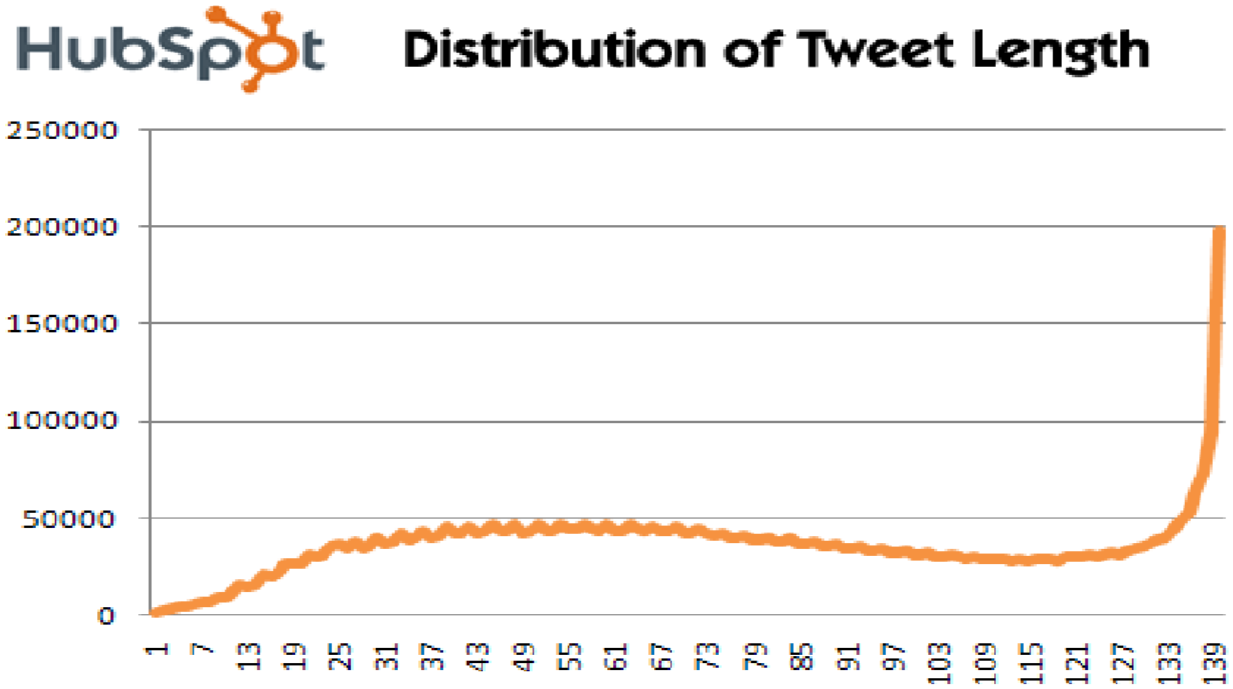


Σχήμα 1: Τμήμα της βασικής οθόνης του Twitter

Η λογική με βάση την οποία συνδέονται οι χρήστες στο Twitter είναι η εξής: κάποιος χρήστης μπορεί να κάνει “follow” έναν άλλο. Κάθε χρήστης έχει μια προσωπική σελίδα στο Twitter. Στη σελίδα αυτή, μπορεί να βλέπει και τα tweets όλων των χρηστών, που έχει κάνει “follow”, σε μια λίστα. Εξού και ο όρος microblogging. Για να έχει επομένως ένας χρήστης πρόσβαση στα tweets ενός άλλου και να ενημερώνεται διαρκώς γι' αυτά, δεν χρειάζεται αμοιβαία άδεια.

Πριν προχωρήσουμε στη χρηστική αξία του Twitter, θα περιγράψουμε ορισμένα ακόμα χαρακτηριστικά του, τα οποία θα μας φανούν χρήσιμα στη συνέχεια. Το πρώτο από αυτά είναι τα hashtags. Hashtags καλούνται οι λέξεις, οι οποίες βρίσκονται, σε ένα tweet, αμέσως μετά το ειδικό

σύμβολο “#”. Το hashtag μπορεί να δηλώσει το θέμα του tweet και χρησιμοποιείται για να αυξήσει την εκφραστικότητα του μηνύματος, καθώς και να διευκολύνει την αναζήτησή του.



Σχήμα 2: Λόγω μεγέθους, τα tweets στερούνται εκφραστικότητας. Το διάγραμμα αυτό, από το [8] δείχνει ότι οι χρήστες τείνουν να εκμεταλλεύονται όσο το δυνατόν περισσότερους από τους διαθέσιμους χαρακτήρες

Εκτός από τη γενική δημοσίευση ενός μηνύματος, οι χρήστες μπορούν να απευθύνουν ένα tweet σε κάποιον άλλον χρήστη, δημιουργώντας έτσι ένα είδος συζήτησης. Για να γίνει αυτό, χρησιμοποιείται το ειδικό σύμβολο “@” και στη συνέχεια το screen name του χρήστη στον οποίο απευθύνεται το tweet. Μάλιστα σύμφωνα με το [8] οι χρήστες, χρησιμοποιούν το Twitter περισσότερο για να επικοινωνήσουν με άλλους χρήστες, παρά για να απαντήσουν στην ερώτηση “What's happening?”. Το 37.95% των tweets περιέχει αναφορά σε κάποιο χρήστη, ενώ το 33.44% είναι απαντήσεις σε προηγούμενες αναφορές.

Τέλος, ένας χρήστης μπορεί να προωθήσει ένα tweet κάποιου άλλου χρήστη σε όλους τους followers του. Τότε, το tweet αυτό λέμε ότι έγινε retweet. Σύμφωνα με το [8], μόνο το 1.44% του συνόλου των tweets είναι retweets. Επίσης, αξίζει να αναφέρουμε, ότι το πλήθος των φορών, που

γίνεται retweet ένα tweet, είναι ένας δείκτης του ενδιαφέροντος, που παρουσιάζει στους χρήστες.

2.2) Εφαρμογές

Στην προηγούμενη ενότητα περιγράψαμε τι είναι το Twitter. Στη συνέχεια, θα παρουσιάσουμε διάφορες περιπτώσεις, όπου η αξιοποίησή του μπορεί να φανεί ιδιαίτερα χρήσιμη. Στις περισσότερες από τις περιπτώσεις αυτές, χρειάζεται να εξάγουμε από ένα σύνολο από tweets τη γνώμη, σχετικά με ένα θέμα, των χρηστών που τα δημοσίευσαν. Χρειάζεται δηλαδή να γίνει sentiment analysis. Sentiment analysis καλείται η εφαρμογή natural language processing (NLP) εργαλείων για την εξαγωγή γνώμης από κάποιο κείμενο.

Ένα πρώτο παράδειγμα χρήσης του Twitter είναι η ενημέρωση. Καθώς το Twitter προσελκύει όλο και περισσότερους χρήστες, αυξάνονται οι πιθανότητες, σε περίπτωση που συμβεί κάτι σημαντικό, κάποιος χρήστης να το αναφέρει σε ένα tweet του. Έτσι, το Twitter έχει προσελκύσει την προσοχή των ειδησεογραφικών πρακτορείων.

Στο [6] φαίνεται η σημασία του microblogging και συγκεκριμένα του Twitter στο word of mouth(WOM). Με τον όρο WOM αναφερόμαστε στη διαδικασία της διάδοσης της πληροφορίας από άτομο σε άτομο κι έχει αποδειχθεί ότι παίζει σημαντικό ρόλο στις αγοραστικές συνήθειες των καταναλωτών. Τα αποτελέσματα του paper αυτού έδειξαν ότι 19% του συνόλου των tweets περιέχει την επωνυμία ενός προιόντος ή μιας εταιρίας. Από τα tweets αυτά τώρα, το 20% εκφράζει μια γνώμη για το προιόν(θετική ή αρνητική), ενώ το 80% δεν εκφράζει καμία άποψη. Αυτό δείχνει ότι οι άνθρωποι χρησιμοποιούν το Twitter περισσότερο για αναζήτηση γενικών πληροφοριών ή για την υποβολή ερωτήσεων σχετικά με ένα προιόν, παρά για να εκφέρουν άποψη για προιόντα. Από τα tweets που εξέφεραν άποψη, πάνω από το 52% εξέφεραν θετική άποψη, ενώ αρνητική άποψη εξέφερε περίπου το 33%. Αυτό με τη σειρά του δείχνει ότι πέραν της ενημέρωσης για προιόντα, το Twitter χρησιμοποιείται και για τη διαφήμιση προιόντων.

Εκτός από ενημέρωση και διαφήμιση όμως, έχει δειχθεί ότι η εκμετάλλευση της πληροφορίας, που ρέει στο Twitter, μπορεί να αξιοποιηθεί για να μας δώσει αρκετά πιο χρήσιμα και ισχυρά αποτελέσματα, όπως η πρόβλεψη εκλογικών αποτελεσμάτων ή η πρόβλεψη για την πορεία των χρηματιστηριακών δεικτών.

Στο [7] εξετάσανε κατά πόσο τα tweets με πολιτικό περιεχόμενο αντικατοπτρίζουν το πολιτικό σκηνικό, που επικρατεί στην πραγματικότητα. Η μελέτη τους έγινε με αφορμή τις εθνικές εκλογές της Γερμανίας στις 27 Σεπτεμβρίου 2009 και στηρίχθηκε σε ένα δείγμα από 104003 tweets. Αν και μέχρι τότε είχε αποδειχθεί ότι η χρήση των "παραδοσιακών" μέσων ενημέρωσης μπορούσε να χρησιμοποιηθεί για την πρόβλεψη εκλογικών αποτελεσμάτων, οι αρχές και οι μέθοδοι που χρησιμοποιούνται σε αυτά τα μέσα, δεν μπορούν να μεταφερθούν άμεσα και στην περίπτωση του Twitter. Αυτό συμβαίνει κυρίως για δυο λόγους. Πρώτον, όπως έχουμε ήδη αναφέρει, το μέγεθος ενός tweet μπορεί να είναι μέχρι 140 χαρακτήρες. Αυτό σημαίνει ότι έχει πολύ μικρό περιεχόμενο, επομένως και μικρότερη εμπεριεχόμενη πληροφορία. Επίσης, με βάση μια έρευνα [19], σχεδόν το 40% των tweets δεν περιέχουν καν κάποια χρήσιμη πληροφορία ή είναι όπως λέμε "pointless babble". Δεύτερον, το 19% των tweets [8] περιέχει link σε κάποια άλλη ιστοσελίδα και το ίδιο δεν προσφέρει κάποια συγκεκριμένη πληροφορία. Επομένως τίθεται το ερώτημα κατά πόσο μπορούμε να εξάγουμε από τα tweets αξιόπιστα αποτελέσματα για την κοινή γνώμη. Ή όπως χαρακτηριστικά έχει ειπωθεί[18], αν μπορούμε να "χρησιμοποιήσουμε τις λέξεις κοινή γνώμη και blogging στην ίδια ποταση".

Η συγκεκριμένη μελέτη που παρουσιάζεται στο [7] συνέλεξε tweets που περιείχαν τα ονόματα των 6 κομμάτων που αντιπροσωπεύονταν στο Γερμανικό Κοινοβούλιο(CDU/CSU, SPD, FDP, B90/Die Grunen, Die Linke) ή tweets που περιείχαν ονόματα πολιτικών αντιπροσώπων αυτών των κομμάτων. Η μελέτη έδειξε ότι το Twitter χρησιμοποιείται όντως σαν μέσο έκφρασης πολιτικών απόψεων και μάλιστα φάνηκε, ότι αυτή που φάνηκε να είναι η κοινή γνώμη, μέσω των tweets, ήταν όντως η άποψη του κόσμου μια βδομάδα πριν τις εκλογές.

Με ποιο τρόπο μπορούμε όμως να χρησιμοποιήσουμε το Twitter για την πρόβλεψη εκλογικών αποτελεσμάτων; Έχει δειχθεί πειραματικά, ότι στα tweets, η εμφάνιση θετικών συναισθημάτων υπερτερεί των αρνητικών. Έτσι, ένας απλός τρόπος θα ήταν να μετρήσουμε το πλήθος των tweets στα οποία αναφέρεται το όνομα οποιουδήποτε από τα κόμματα που συμμετέχουν στις εκλογές. Όπως φαίνεται και στον επόμενο πίνακα, χρησιμοποιώντας αυτήν την απλή μέθοδο, το μέσο απόλυτο σφάλμα ήταν 1.65%.

Party	All mentions		Election	
	Number of tweets	Share of Twitter traffic	Election result*	Prediction error
CDU	30,886	30.1%	29.0%	1.0%
CSU	5,748	5.6%	6.9%	1.3%
SPD	27,356	26.6%	24.5%	2.2%
FDP	17,737	17.3%	15.5%	1.7%
LINKE	12,689	12.4%	12.7%	0.3%
Grüne	8,250	8.0%	11.4%	3.3%
			MAE:	1.65%

Πίνακας 1: [7] Ακρίβεια πρόβλεψης αποτελεσμάτων για το κάθε κόμμα με βάση την προαναφερθείσα μέθοδο πρόβλεψης

Στον επόμενο πίνακα συγκρίνεται το μέσο απόλυτο σφάλμα πρόβλεψης χρησιμοποιώντας το Twitter με το αντίστοιχο σφάλμα που προκύπτει από τη χρήση άλλων παραδοσιακών poll εκλογών. Τα αποτελέσματα δεν διαφέρουν ιδιαίτερα μεταξύ τους, γεγονός που δείχνει ότι πράγματι, το Twitter μπορεί να χρησιμοποιηθεί για την πρόβλεψη εκλογικών αποτελεσμάτων.

Source	MAE (last poll)
Twitter	*1.65%*
Forsa	0.84%
Allensbach	0.80%
Emnid	1.04%
Forschungsgruppe Wahlen	1.48%
GMS	1.40%
Infratest/dimap	1.28%

Πίνακας 2: [7] Σύγκριση Twitter με άλλες μεθόδους για εκλογικά polls

Στις 9 Σεπτεμβρίου του 2011, δημοσιεύτηκε στο New Scientist ένα άρθρο με τίτλο: “Using Twitter to follow trends beats the stock market” [20]. Στο άρθρο αυτό περιγράφεται πώς η χρηματιστηριακή εταιρία Derwent Capital Markets, η οποία κάνει τις προβλέψεις της με βάση έναν αλγόριθμο, ο οποίος κάνει sentiment analysis σε σχετικά tweets, έκανε καλύτερες προβλέψεις σε

σχέση με άλλους μεγαλύτερους χρηματιστηριακούς οίκους. Τον αλγόριθμο εφηύρε ο Johan Bollen, επιστήμονας Πληροφορικής στο Πανεπιστήμιο του Bloomington της Indiana. Μια χρονιά νωρίτερα, ο Bollen, με τους αλγόριθμούς του είχε προβλέψει την πορεία του δείκτη του Dow Jones με ακρίβεια 87.6%.

Η Derwent Capital Markets δεν είναι η μόνη εταιρία, που σχετίζεται με τον οικονομικό κόσμο, που έστρεψε την προσοχή της στο Twitter. Κι άλλες μεγάλες εταιρίες, όπως η Bloomberg, έχουν συνειδητοποιήσει την αξία του Twitter και των άλλων social media κι έχουν στρέψει προς τα εκεί το ενδιαφέρον τους.

Παρ' όλα αυτά, αξίζει να σημειώσουμε ότι αναλυτές κι επενδυτές πιστεύουν ότι τα κοινωνικά δίκτυα δεν θα φέρουν την επανάσταση στην πρόβλεψη της πορείας των αγορών. Πάντα υπήρχαν τρόποι για να γίνονται γνωστές οι τάσεις και η διάθεση του κόσμου. Απλά, σήμερα, όπου έχουμε διαθέσιμα τα κοινωνικά δίκτυα κι ισχυρό software για αναλυτική επεξεργασία δεδομένων, μπορούμε να μάθουμε αρκετά πιο εύκολα και γρήγορα αυτές τις τάσεις του κόσμου και συνεπώς και των αγορών. Βέβαια, μπορεί τα κοινωνικά δίκτυα, να μην φέρουν την επανάσταση στην πρόβλεψη των αγορών, αλλά ας μην ξεχνάμε ότι ακόμη και κατά 1 ή 2% να αυξήσουν το κέρδος, μιλάμε για αρκετά δισεκατομμύρια δολλάρια.

Η αξία των παραπάνω εφαρμογών στηρίζεται στο sentiment analysis που κάνουν. Για να μπορέσει να γίνει όμως sentiment analysis προϋποτίθεται ότι υπάρχει ένα dataset από tweets. Αυτό το σύνολο δεδομένων συλλέγεται μέσω κάποιων άλλων εφαρμογών, των οποίων σκοπός είναι η αναζήτηση στο Twitter. Τα κριτήρια με βάση τα οποία γίνεται η αναζήτηση, καθώς κι ο τρόπος σύνδεσης στο Twitter και συλλογής των δεδομένων διαφέρει από εφαρμογή σε εφαρμογή.

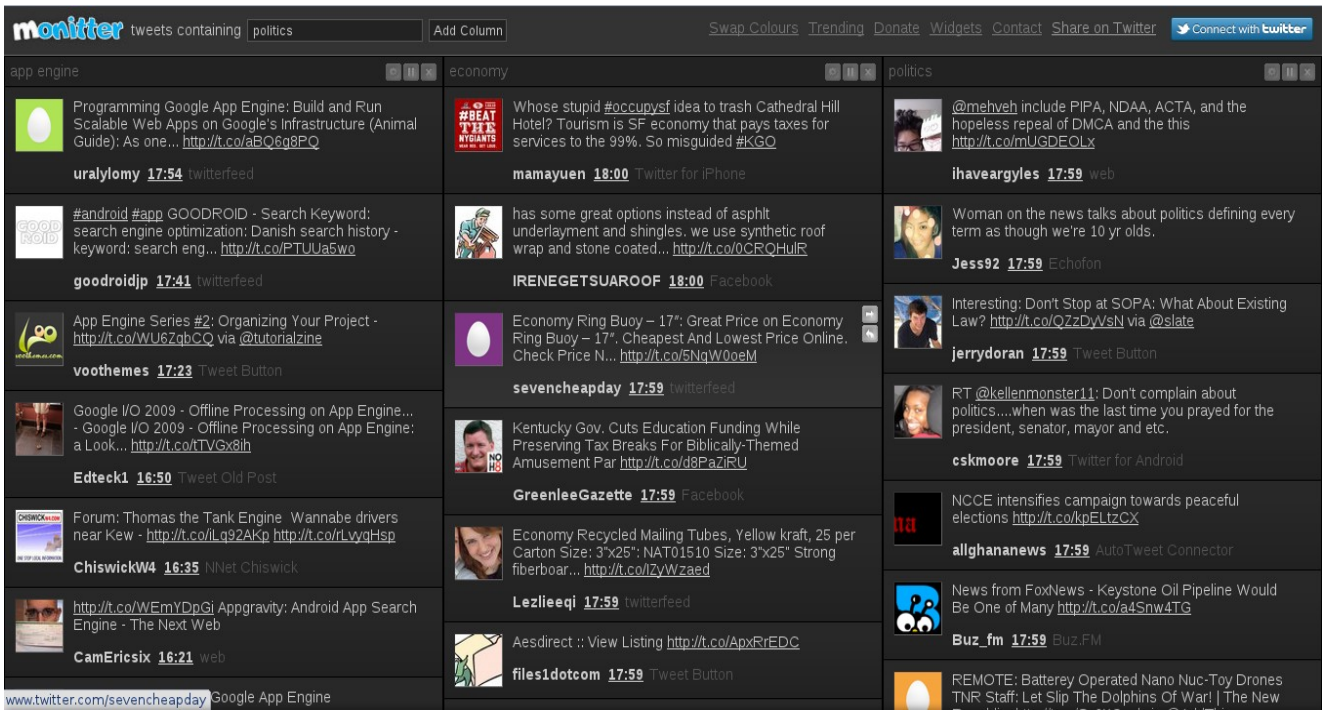
Όπως έχει ήδη αναφερθεί, σκοπός της εφαρμογής μας είναι η αναζήτηση πληροφοριών στο Twitter. Θεωρούμε σκόπιμο λοιπόν, να παρουσιάσουμε σε αυτό το σημείο μερικές αντίστοιχες εφαρμογές, που υπάρχουν ήδη. Αυτές είναι οι twendz, Twittscoop, monitter, ICEROCKET, TweetScan και πολλές άλλες. Βασικό κοινό χαρακτηριστικό των περισσοτέρων από αυτές είναι ότι είναι real-time. Το γεγονός αυτό είναι ιδιαίτερα σημαντικό, καθώς καθιστά δυνατό να εντοπίζονται τα διάφορα trends που υπάρχουν κάθε χρονική στιγμή. Ο τρόπος δηλαδή που λειτουργούν είναι ο εξής: ζητάνε ως είσοδο κάποια λέξη-κλειδί και σαν αποτέλεσμα επιστρέφουν σε πραγματικό χρόνο τα tweets του public timeline, που περιέχουν τη λέξη αυτή.

Από εκεί και πέρα, κάθε μια από αυτές, εξειδικεύεται σε κάτι συγκεκριμένο. Για παράδειγμα, το twendz εξάγει τις βασικές θεματικές ενότητες, που υπάρχουν στα tweets, που συλλέγει, κι εμφανίζει

και ποσοστά σχετικά με την γνώμη του κοινού για τα θέματα αυτά, ενώ το monitter μπορεί να παρακολουθεί σε πραγματικό χρόνο μέχρι τρία διαφορετικά streams από tweets, που βασίζονται σε διαφορετικές λέξεις κλειδιά.

Επίσης, υπάρχουν εφαρμογές, όπως το Twinitor ή το Backtweets(το οποίο πρόσφατα αγόρασε το Twitter) ή οποίες δεν είναι real-time, αλλά η λειτουργικότητά τους στηρίζεται στο γεγονός ότι μπορούν να επιστρέφουν ως αποτέλεσμα “παλιά” tweets. Η αναζήτηση στο Twitter μπορεί να επιστρέψει tweets, τα οποία δημοσιεύτηκαν έως και δυο εβδομάδες πριν. Οι εφαρμογές αυτές λοιπόν, δίνουν πρόσβαση σε ακόμα παλαιότερα δεδομένα.

Παρ'όλο όμως που υπάρχει πληθώρα εφαρμογών με αντικείμενο την αναζήτηση tweets, όλες οι εφαρμογές, που έχουμε δει, χρησιμοποιούν ως κριτήρια αναζήτησης αυτά που χρησιμοποιεί και το Twitter Advanced Search, το οποίο θα παρουσιαστεί στη συνέχεια. Έτσι, εγκυμονεί ο κίνδυνος, τα αποτελέσματα της αναζήτησης να μην σχετίζονται εννοιολογικά άμεσα με αυτό που μας ενδιαφέρει, αλλά να περιέχουν απλώς κάποια συγκεκριμένη λέξη, που δώσαμε ως κλειδί στην αναζήτησή μας. Στην προσπάθειά μας να αποφύγουμε αυτόν τον κίνδυνο, όπως θα δείξουμε αναλυτικά και στη συνέχεια, στη δική μας εφαρμογή δεικτοδοτούμε τα tweets και κάνουμε αναζήτηση σε αυτά εκτός των άλλων και με βάση τις συντακτικές τους ιδιότητες.



Σχήμα 3: Στιγμιότυπο της εφαρμογής monitter, ενώ παρακολουθεί ταυτόχρονα τρία διαφορετικά streams από tweets

2.3) Twitter API

Όπως είδαμε, η αξιοποίηση των δεδομένων, που υπάρχουν στο Twitter, μπορεί να δώσει ιδιαίτερα χρήσιμες πληροφορίες και προβλέψεις, που σε ορισμένες περιπτώσεις ενδέχεται και να αξίζουν εκατομμύρια δολλάρια. Για να μπορέσουμε όμως να εκμεταλλευτούμε τις πληροφορίες αυτές, θα πρέπει οι εφαρμογές μας να μπορούν να αποκτούν πρόσβαση στο Twitter και να λαμβάνουν από εκεί όλα τα απαραίτητα στοιχεία.

Για το σκοπό αυτό, το Twitter παρέχει ένα application interface, έτσι ώστε οι διάφοροι προγραμματιστές να μπορούν να αναπτύξουν εφαρμογές, που να συνδέονται με το Twitter . Οι εφαρμογές αυτές μπορεί να είναι με read only αλλά και με write permissions ως προς το Twitter. Δηλαδή, οι εφαρμογές μπορούν απλά να λαμβάνουν δεδομένα από το Twitter ή μπορούν ακόμα και να μορφοποιούν το προφίλ κάποιου χρήστη.

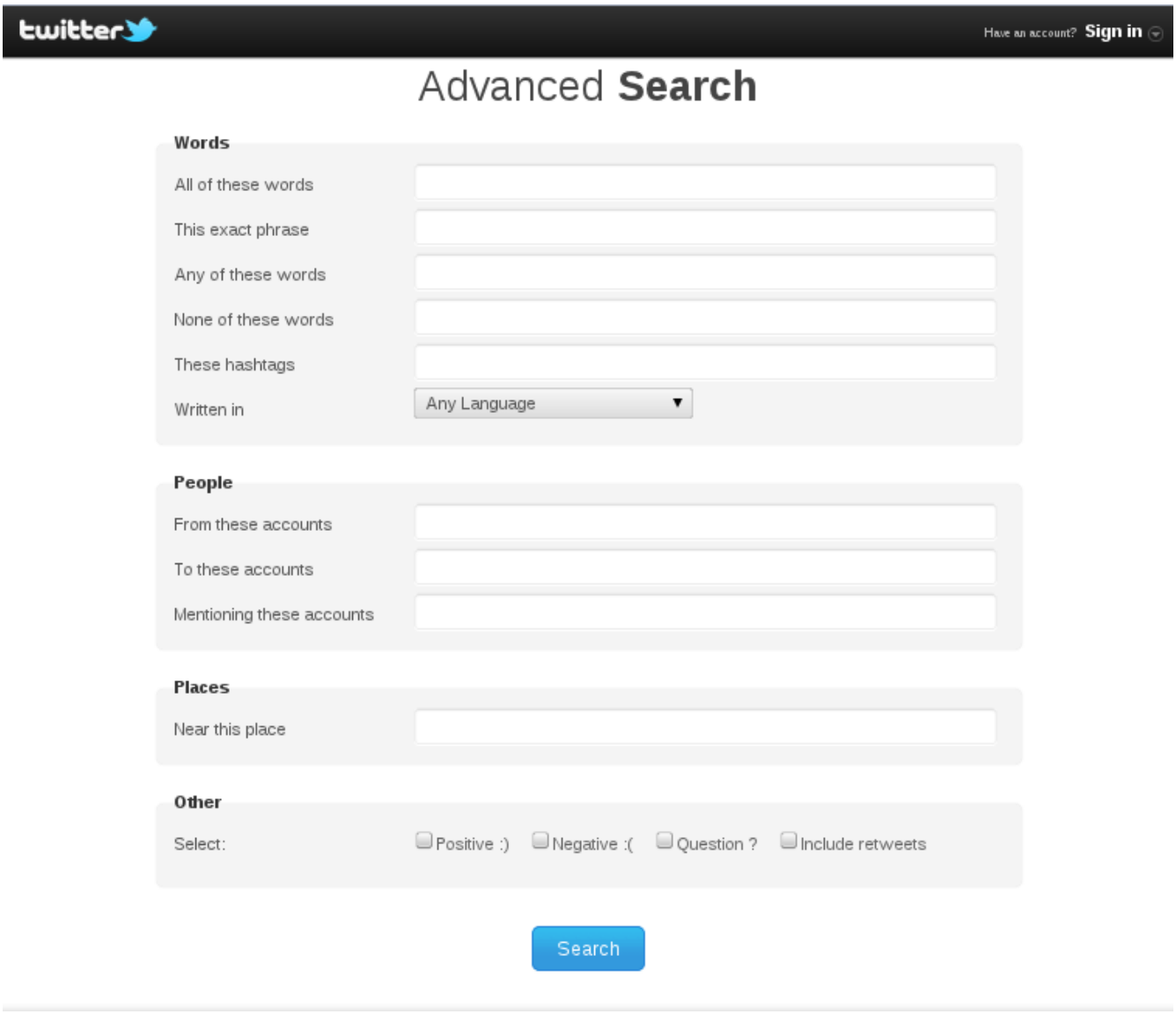
Για να δημιουργήσει κάποιος ένα twitter application πρέπει κατ' αρχάς να έχει λογαριασμό στο Twitter. Στη συνέχεια, θα πρέπει να επισκεφτεί την ιστοσελίδα: dev.twitter.com. Εκεί, δηλώνει

επίσημα στο Twitter την εφαρμογή του, ρυθμίζει τα δικαιώματα(read,write) που επιθυμεί να έχει η εφαρμογή του και λαμβάνει από το Twitter κάποια credentials με τα οποία θα πιστοποιείται η ταυτότητα της εφαρμογής του και θα συνδέεται στο Twitter. Στη σελίδα των Twitter Developers [5] μπορεί κανείς ακόμα να βρει και documentation για το Twitter API.

Στη συνέχεια, θα δούμε τι λειτουργίες επιτελεί το Twitter API . Στην πραγματικότητα, το Twitter παρέχει τρία APIs(Application Programming Interfaces). Το REpresentational State Transfer (REST) API, το Search API και το Streaming API.

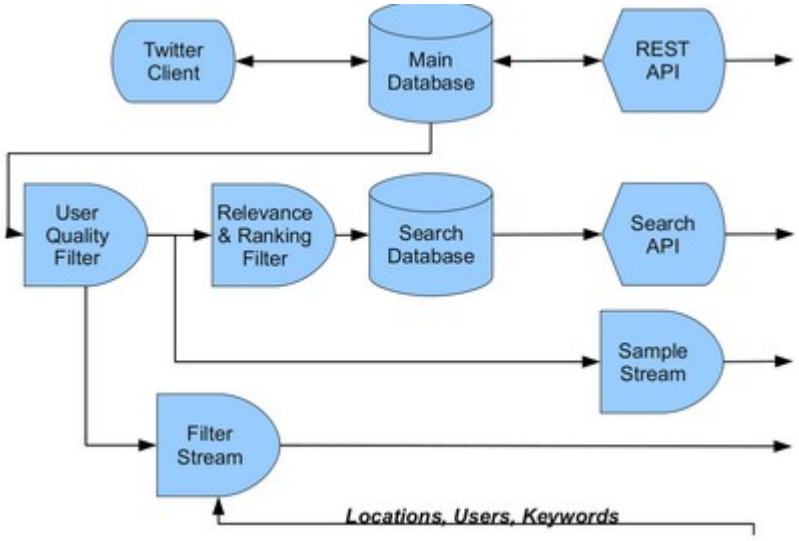
Το REST API μπορεί να υλοποιήσει όλες τις βασικές λειτουργίες που μπορεί να κάνει κανείς στο Twitter. Μέσω του REST API μπορεί δηλαδή, να δημοσιεύσει κάποιο μήνυμα στο status του, να κάνει retweet κάποιο μήνυμα, να κάνει follow και unfollow άλλα άτομα, να στείλει direct messages , ή και να οργανώσει τις λίστες του.

Το Search API δίνει τη δυνατότητα στον προγραμματιστή, να κάνει ότι μπορεί να κάνει και το Twitter Advanced Search. Μπορεί δηλαδή, να αναζητήσει tweets με βάση κάποιες λέξεις-κλειδιά ή με βάση κάποιους συνδυασμούς λέξεων. Μπορεί ακόμα να κάνει αναζήτηση με βάση το hashtag, τη γλώσσα στην οποία είναι γραμμένο το tweet, το χρήστη που το δημοσίευσε ή χρήστες που αναφέρονται στο tweet, την τοποθεσία και άλλα. Στην παρακάτω εικόνα φαίνεται η φόρμα αναζήτησης του Twitter Advanced Search.



Σχήμα 4: Φόρμα του Advanced Twitter Search

Για να κατανοήσουμε την χρησιμότητα του streaming API, θα δούμε πρώτα ποια είναι η διαδρομή ενός tweet, από τη στιγμή που το δημοσιεύει ένας χρήστης, μέχρι τη στιγμή που αυτό είναι διαθέσιμο σε ένα twitter application, μέσω κάποιου API. Η διαδρομή αυτή, την οποία θα περιγράψουμε και αναλυτικά, φαίνεται στην παρακάτω εικόνα.



Σχήμα 5: Διαδρομή ενός tweet από τη δημοσίευσή του μέχρι κάποιο twitter application

Ας υποθέσουμε ότι ένας χρήστης θέλει να δημοσιεύσει κάτι στο status του. Ο χρήστης αυτός λειτουργεί ως Twitter Client , ο οποίος κάνει μια αίτηση στο Twitter. Το tweet που θα δημοσιεύσει, αποθηκεύεται στην Main Database, αφού πρώτα επισημανθεί με ένα timestamp και με τα στοιχεία του χρήστη που το δημοσιεύει. Όπως φαίνεται και στο παραπάνω σχήμα, το REST API «βλέπει» αυτή τη βάση δεδομένων. Μπορεί να κατεβάσει και να επεξεργαστεί οποιοδήποτε tweet υπάρχει σε αυτή τη βάση.

Στη συνέχεια, τα tweets που αποθηκεύονται στην Main Database , περνάνε από ένα φίλτρο, το User Quality Filter. Το φίλτρο αυτό αξιολογεί, με τη βοήθεια διάφορων μετρικών, την ποιότητα των statuses ενός account κι ανάλογα με το αποτέλεσμα αφήνει τα tweets να περάσουν ή όχι. Τα tweets , που περνάνε από αυτό το φίλτρο, στη συνέχεια οδηγούνται σε ένα δεύτερο φίλτρο, το Relevance & Ranking Filter. Όσα tweets περνάν από το Relevance & Ranking Filter, αποθηκεύονται στην Search Database κι από εκεί είναι διαθέσιμα μέσω του Search API. Σημειώνουμε εδώ, ότι σε αντίθεση με την Main Database, στην οποία μπορούμε να τροποποιούμε τα tweets, η Search Database είναι read-only. Δηλαδή, δεν έχουμε τη δικαιοδοσία να δημιουργήσουμε, να ενημερώσουμε ή να διαγράψουμε tweets μέσω του Search API.

Τα δύο τελευταία βέλη του παραπάνω σχήματος, δείχνουν σε ποια tweets αναφέρεται το Streaming API, το οποίο χωρίζεται στο Sample stream και στο Filter stream . Το Streaming API ,

όπως και το Search API είναι read-only. Η βασική διαφορά μεταξύ των δυο APIs είναι ότι το Streaming API διαβάζει τα tweets , απευθείας από το public timeline κι όχι από κάποια βάση δεδομένων. Αναφέρουμε ακόμα, ότι για να υπάρχει ένα tweet στο public timeline θα πρέπει το προφίλ του χρήστη που το δημοσιέυει να είναι public κι όχι protected .

Τα tweets που έχουν περάσει το User Quality Filter , αλλά όχι το Relevance & Ranking Filter είναι ορατά στο Sample stream. Το Sample stream αποτελεί ένα υποσύνολο του public timeline , γνωστό και ως Firehose .

Το Filter stream είναι κι αυτό ένα υποσύνολο του Firehose. Η διαφορά του με το Sample stream είναι ότι δεν περιλαμβάνει ένα τυχαίο δείγμα του public timeline , αλλά ένα υποσύνολο που ικανοποιεί κάποια κριτήρια. Τα κριτήρια αυτά μπορεί να είναι το όνομα του χρήστη που κάνει τη δημοσίευση, κάποιο keyword, ή η τοποθεσία, που είναι δηλωμένη στο tweet.

2.4) Twitter Application

Για την εφαρμογή μας, χρειαστήκαμε ένα μεγάλο dataset από tweets . Για το λόγο αυτό, έπρεπε να αναπτύξουμε ένα πρόγραμμα, το οποίο θα επικοινωνεί με το Twitter , μέσω του Twitter API και θα συλλέγει πληροφορίες.

Η υλοποίηση του Twitter API ήταν ένα πρόβλημα που δεν μας απασχόλησε. Έχουν δημιουργηθεί ήδη βιβλιοθήκες που το υλοποιούν. Εμείς επιλέξαμε και χρησιμοποιήσαμε την twitter4j για java [21] , δεδομένου ότι είχε το καλύτερο documentation.

Η επιλογή μεταξύ REST, Search και Streaming API ήταν ένα ζήτημα. Ποιο από τα τρία programming interfaces καλύπτει καλύτερα τις ανάγκες της εφαρμογής μας; Τα REST και Search API , όπως είδαμε επικοινωνούν με κάποια βάση δεδομένων(Main και Search database αντίστοιχα). Επομένως, τα interfaces αυτά, για να φέρουν tweets από το Twitter στην εφαρμογή μας, θα πρέπει να υποβάλουν κάποιο ερώτημα στην αντίστοιχη βάση. Οι δοσοληψίες με μια βάση δεδομένων, καθυστερούν όμως τη διαδικασία της συλλογής των tweets . Αντίθετα, ο τρόπος με τον οποίο λειτουργεί το Streaming API είναι διαφορετικός. Εγκαθιστά μία φορά, μια σύνδεση με το Twitter και τραβάει όλα τα tweets , που υπάρχουν στο public timeline όσο είναι ενεργή αυτή η σύνδεση. Από το γεγονός αυτό φαίνεται κι ότι η διαδικασία της συλλογής tweets από μία εφαρμογή, μέσω του Streaming API συνήθως είναι ένα ανεξάρτητο thread ή daemon που τρέχει στο background .

Μια άλλη παράμετρος και ίσως η πιο σημαντική, που συνηγορεί στην επιλογή του Streaming API

για την εφαρμογή μας, είναι η εξής. Τα REST και Search API έχουν περιορισμένο rate limit , δηλαδή, περιορισμένο όριο αιτήσεων που μπορεί κανείς να υποβάλει στο Twitter . Για την ακρίβεια, το REST API δίνει τη δυνατότητα στην εφαρμογή να τραβάει το πολύ 350 tweets την ώρα. Το search API παρέχει μεγαλύτερο rate limit , το οποίο, αν και δεν είναι γνωστό δημοσίως, λέγεται ότι είναι ικανοποιητικό για τις περισσότερες εφαρμογές, αλλά όχι απεριόριστο. Όσον αφορά στο Streaming API , όπως αναφέρθηκε και παραπάνω, μπορούμε να τραβάμε δεδομένα, όσο είναι ενεργή η σύνδεση. Οπότε, κατά αυτή την έννοια δεν υπάρχει ratelimit . Παρ' όλα αυτά, στο documentation του API αναφέρεται ότι υπάρχει ratelimit , χωρίς όμως να είναι σαφές πώς το εννοεί και πώς το μετράει. Μάλιστα, αναφέρεται ακόμα, ότι η IP διεύθυνση όσων clients ξεπερνάν το ratelimit αυτό μπαίνει σε blacklist κι έκτοτε τους απαγορεύεται η πρόσβαση στους servers του Twitter .

Ένα άλλο βασικό χαρακτηριστικό του Streaming API, είναι ότι είναι real-time και τροφοδοτεί την εφαρμογή με tweets, τα οποία δημοσιεύονται στο public timeline αυτή τη στιγμή. Το γεγονός αυτό είναι ιδιαίτερα χρήσιμο για εφαρμογές, όπως η δική μας, οι οποίες έχουν σαν σκοπό την εξαγωγή πληροφοριών (information retrieval). Από τη στιγμή που το stream είναι real-time, τα περιεχόμενά του είναι όσο πιο σύγχρονα κι ενημερωμένα γίνεται. Επομένως, εξετάζοντας το περιεχόμενο των tweets αυτών, έχουμε τη δυνατότητα να ενημερωνόμαστε για γεγονότα, που συμβαίνουν την παρούσα χρονική στιγμή, καθώς και να εντοπίζουμε αλλαγές στις τάσεις, που υπάρχουν στην κοινωνία, στην οικονομία κλπ.

Επίσης, το γεγονός ότι το Streaming API δίνει πρόσβαση και στο FilterStream και στο SampleStream δίνει άλλο ένα πλεονέκτημα. Προσφέρεται και για εφαρμογές, που στοχεύουν σε tweets με συγκεκριμένο περιεχόμενο(όπου και θα χρησιμοποιηθεί το FilterStream) και για εφαρμογές που χρειάζονται ένα τυχαίο, unbiased δείγμα από tweets(τυχαία δειγματοληψία μέσω SampleStream).

Η δυνατότητα του Streaming API να μας παρέχει πρόσβαση σε μεγάλο πλήθος real-time δεδομένων(απουσία rate limit), καθώς και ο τρόπος με τον οποίο τρέχουν οι εφαρμογές που το χρησιμοποιούν, δηλαδή ότι εγκαθιστούν μια long lived σύνδεση και τρέχουν στο background , είναι δύο χαρακτηριστικά τα οποία είναι ιδιαίτερα χρήσιμα για το είδος της εφαρμογής που αναπτύξαμε. Για το λόγο αυτό, όπως είναι πλέον προφανές, για την επικοινωνία μας με το Twitter , χρησιμοποιήθηκε το Streaming API .

3) Information Retrieval

Η Ανάκτηση Πληροφοριών (Information Retrieval-IR) είναι μια επιστημονική περιοχή, που ασχολείται με την αναζήτηση εγγράφων, με την αναζήτηση πληροφοριών σε έγγραφα και μεταδεδομένων σχετικών με τα έγγραφα, καθώς και με την αναζήτηση σε δομημένα σχήματα αποθήκευσης, σε βάσεις δεδομένων και στον Παγκόσμιο Ιστό.

Το IR βασίζεται συγχρόνως σε πολλές επιστήμες, όπως είναι η Πληροφορική, τα Μαθηματικά, η Γλωσσολογία και η Ψυχολογία. Αξιοποιώντας τις γνώσεις που παρέχουν οι παραπάνω επιστήμες, έχουν αναπτυχθεί αρκετά αυτόματα συστήματα, τα οποία δίνουν τη δυνατότητα ανάκτησης πληροφοριών. Τα πιο ευρέως γνωστά είναι οι διαδικτυακές μηχανές αναζήτησης, όπως αυτή της Google.

Τα συστήματα αυτά στις μέρες μας αποκτούν ιδιαίτερη αξία. Ως γνωστόν, με την εμφάνιση του Web 2.0, οποιοσδήποτε χρήστης του Internet μπορεί να είναι εκτός από καταναλωτής και παραγωγός πληροφοριών. Όπως είδαμε και σε προηγούμενο κεφάλαιο, τα κοινωνικά δίκτυα αριθμούν πλέον εκατομμύρια χρήστες. Επομένως, η εποχή μας χαρακτηρίζεται από έναν κατακλυσμό δεδομένων, ή όπως αλλιώς είναι γνωστό, από υπερφόρτωση πληροφοριών(information overload). Στο σημείο αυτό είναι που αναδεικνύεται η αξία των IR συστημάτων. Ένα σύστημα που μπορεί να ξεχωρίζει γρήγορα την ζητούμενη πληροφορία από την περιττή, μειώνει το πρόβλημα του information overload και καθιστά τον όγκο της πληροφορίας διαχειρίσιμο.

Όπως έχουμε προαναφέρει, σκοπός της παρούσας διπλωματικής είναι η δημιουργία ενός συστήματος, το οποίο κάνει αναζήτηση σε μια βάση δεδομένων, η οποία περιέχει πληροφορίες, που έχουμε πάρει από το Twitter. Είναι πλέον εμφανές ότι πρόκειται για ένα IR σύστημα. Σκοπός μας είναι η αναζήτηση να είναι όσο το δυνατόν πιο ακριβής και στοχευμένη, έτσι ώστε τα αποτελέσματα που λαμβάνει ο χρήστης της εφαρμογής μας να είναι όσο το δυνατόν πιο κοντά σε αυτό που πραγματικά αναζητάει.

Για να το πετύχουμε αυτό, πριν αποθηκεύσουμε τα tweets στη βάση μας, τα υποβάλλουμε σε γραμματική και συντακτική ανάλυση και κρατάμε και αυτές τις επιπλέον πληροφορίες. Για την επεξεργασία αυτή, έγινε χρήση εργαλείων Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing-NLP). Το NLP είναι ένας κλάδος της Τεχνητής Νοημοσύνης, ο οποίος ασχολείται με την διάδραση μεταξύ υπολογιστών και ανθρώπινων γλωσσών.

Τα NLP εργαλεία που χρησιμοποιήσαμε για την δική μας εφαρμογή είναι ένας Tokenizer, ένας Part-Of-Speech (POS) Tagger, ένας Parser κι ένας Lemmatizer. Τα εργαλεία αυτά αναπτύχθηκαν στο Πανεπιστήμιο του Stanford και περιγράφονται αναλυτικά στα [9],[10]. Για την ακρίβεια, στην εφαρμογή μας χρησιμοποιήσαμε το Stanford CoreNLP 1.3.0, το οποίο είναι μια βιβλιοθήκη, που παρέχει την υλοποίηση όλων των παραπάνω εργαλείων. Στη συνέχεια θα τα περιγράψουμε το καθένα ξεχωριστά.

Ο Tokenizer είναι ένα εργαλείο, το οποίο λαμβάνει ως είσοδο μια συμβολοακολουθία (string) κι επιστρέφει το σύνολο των λεκτικών μονάδων που εμπεριέχονται στην ακολουθία αυτή. Για παράδειγμα, αν δεχτεί ως είσοδο το string: “ I watched a great movie!”, θα επιστρέψει το σύνολο: {“I”, “watched”,”a”,”great”,”movie”,”!”}.

Ο POS Tagger είναι ένα εργαλείο το οποίο δέχεται ως είσοδο μια πρόταση και στη συνέχεια, με τη χρήση ενός Tokenizer την διασπά σε ξεχωριστές λεκτικές μονάδες, αναθέτοντας παράλληλα ένα tag(ετικέτα) σε κάθε λεκτική μονάδα. Το tag αυτό, προσδιορίζει ως τι μέρος του λόγου εμφανίζεται η λέξη αυτή στην συγκεκριμένη πρόταση. Τα tags που χρησιμοποιεί ο POS Tagger, που χρησιμοποιήσαμε, είναι αυτά που χρησιμοποιούνται στο Penn Treebank Project [11] και φαίνονται στον παρακάτω πίνακα.

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Πίνακας 3: Tags του Penn Treebank Project

Για την καλύτερη κατανόηση του Tagger παρουσιάζουμε και το αποτέλεσμα που δίνει ο Tagger, που χρησιμοποιήσαμε, για την πρόταση: “ I watched a great movie!”. Το αποτέλεσμα είναι: I/PRP watched/VBD a/DT great/JJ movie/NN !/. Έτσι, βλέπουμε ότι το “I” είναι προσωπική αντωνυμία, το “watched” είναι ρήμα σε παρελθοντικό χρόνο, το “a” άρθρο, το “great” επίθετο και το “movie” ουσιαστικό.

Ο τρόπος με τον οποίο χρησιμοποιήσαμε τον POS Tagger είναι ο εξής: Ας υποθέσουμε ότι θέλουμε να βρούμε τα tweets που αναφέρονται σε ρολόγια χειρός. Ως λέξη κλειδί για αυτήν την αναζήτηση, σίγουρα θα δίναμε το “watch”. Παρ' όλα αυτά, το “watch” υπάρχει και ως ρήμα και μάλιστα ως ρήμα αναμένεται να εμφανίζεται πολύ πιο συχνά. Χωρίς τη χρήση του POS Tagger επομένως, το σύνολο των αποτελεσμάτων μας θα περιείχε αρκετή πληροφορία, που θα ήταν άσχετη με αυτό που πραγματικά μας ενδιαφέρει. Αν όμως χρησιμοποιήσουμε τον Tagger, κάνοντας κατάλληλη δεικτοδότηση στα δεδομένα μας, την οποία θα περιγράψουμε σε επόμενο κεφάλαιο, θα μπορούμε να αναζητήσουμε τη λέξη “watch”, αλλά μόνο όπου αυτή εμφανίζεται ως ουσιαστικό.

Το επόμενο εργαλείο που μας χρειάστηκε είναι ο Lemmatizer. Ας σκεφτούμε το εξής παράδειγμα: Θέλουμε να κάνουμε αναζήτηση στη βάση μας για να δούμε ποιοι χρήστες του Twitter πήγαν για τρέξιμο την βδομάδα που πέρασε. Μια εύλογη επιλογή των λέξεων που θα έπρεπε να χρησιμοποιηθούν ως κριτήριο στην αναζήτηση αυτή θα ήταν η επιλογή των “go” και “ running” συγχρόνως. Αν και η αναζήτηση με βάση αυτές τις λέξεις, θα μπορούσε να εντοπίσει ένα tweet όπως το “I go running everyday!”, δεν θα ηταν δυνατό να εντοπίσει ένα tweet όπως το “ I went running last weekend!”. Αυτό συμβαίνει γιατί το “go” δεν μπορεί να αντιστοιχιστεί με το “went”. Ο Lemmatizer είναι ένα εργαλείο, το οποίο παίρνει ως είσοδο μια λέξη και κάνοντας χρήση ενός λεξικού, αντιστοιχίζει τη λέξη αυτή σε μια άλλη λέξη, έτσι ώστε να ισχύει: *λέξη εισόδου= παράγωγο λέξης εξόδου*
Για παράδειγμα, αν πάρει ως είσοδο τη λέξη “went” θα επιστρέψει “go”.

Το τελευταίο εργαλείο που χρησιμοποιήσαμε είναι ο Parser. Ο Parser παίρνει ως είσοδο μια πρόταση κι επιστρέφει ένα σύνολο συντακτικών εξαρτήσεων που υπάρχουν στην πρόταση αυτή. Σε κάθε εξάρτηση εμπλέκονται δυο λέξεις. Η μια είναι η βασική λέξη που συμμετέχει στην εξάρτηση (governor) κι η άλλη είναι η εξαρτώμενη λέξη (dependent). Για παράδειγμα, αν δώσουμε στον Parser ως είσοδο την πρόταση: “He gave me a raise”, θα επιστρέψει μια εξάρτηση του τύπου dobj(gave,raise), που σημαίνει ότι το “raise” είναι συντακτικά άμεσο αντικείμενο (direct object) του “gave”.

Για να κατανοήσουμε την αξία αυτού του εργαλείου για την εφαρμογή μας θα χρησιμοποιήσουμε το παράδειγμα που αναφέραμε και στην εισαγωγή της παρούσας εργασίας. Ας υποθέσουμε ότι μας

ενδιαφέρουν τα tweets, που περιέχουν κάποιες δηλώσεις του Obama. Οπότε, θα μπορούσαμε να θέσουμε ως ερώτημα στη βάση, ότι θέλουμε τα tweets, που περιέχουν τις λέξεις “Obama” και “state” συγχρόνως. Η αναζήτηση αυτή όμως, θα μπορούσε να επιστρέψει και το ακόλουθο tweet, του οποίου το περιεχόμενο, δεν μας ενδιαφέρει στην πραγματικότητα.“Well, as I stated on Facebook earlier..2012 Obama get my vote again. Yes, that is my final answer. GOP candidates are less impressive.#YEP”. Στην επίλυση του προβλήματος που παρουσιάζεται στην προκειμένη περίπτωση, ο POS Tagger δεν μπορεί να προσφέρει κάτι παραπάνω. Με τη χρήση του Parser όμως, θα μπορούσαμε να ζητήσουμε από τη βάση τα tweets εκείνα που έχουν το ρήμα “state” κι ως υποκείμενο αυτού το “Obama”. Δηλαδή, τα tweets εκείνα που ικανοποιούν τη σχέση: nsubj(state,Obama).

Αξίζει σε αυτό το σημείο να αναφέρουμε ότι για να μπορούν να λειτουργούν και να αναγνωρίζουν φυσική γλώσσα, τα παραπάνω εργαλεία έχουν “εκπαιδευτεί” μέσω διαφόρων αλγορίθμων μηχανικής μάθησης και κυρίως στατιστικής μηχανικής μάθησης κι αναγνώρισης προτύπων. Τα πρότυπα αυτά και τα δεδομένα εκπαίδευσης διαφέρουν από γλώσσα σε γλώσσα. Επίσης, ορισμένα εργαλεία, όπως ο Lemmatizer, απαιτούν τη χρήση κάποιου λεξικού. Άρα η γλώσσα παίζει ρόλο. Με βάση αυτήν την παρατήρηση, σημειώνουμε ότι η εφαρμογή μας δεν δουλεύει για οποιαδήποτε tweets αλλά μόνο για αυτά που είναι γραμμένα στην αγγλική γλώσσα.

Η αξία της εφαρμογής μας είναι πλέον εμφανής. Με την ανάπτυξη ενός IR συστήματος που πέραν από τις δυνατότητες του Twitter Advanced Search, έχει και τη δυνατότητα να αντιμετωπίζει προβλήματα όπως αυτά που αναφέραμε στις προηγούμενες παραγράφους, θέλουμε να δείξουμε πώς η αξιοποίηση των NLP εργαλείων μπορεί να βελτιώσει την ποιότητα των αποτελεσμάτων της αναζήτησης.

Σε επόμενα κεφάλαια, θα περιγράψουμε αναλυτικά την εφαρμογή μας, καθώς και το σχήμα των δεδομένων που χρησιμοποιούμε για φυλάξουμε όλες αυτές τις επιπλέον πληροφορίες. Επίσης, θα παρουσιάσουμε ορισμένα σενάρια χρήσης (use cases), για να δούμε σε ποιο βαθμό επιτυγχάνεται η αξιοποίηση των NLP εργαλείων από την εφαρμογή μας.

4) Cloud Computing

4.1) Επισκόπηση

Με τον όρο Υπολογιστικό Νέφος (Cloud Computing) αναφερόμαστε στην παράδοση του υπολογισμού ως μια υπηρεσία (κι όχι ως προϊόν), όπου διαμοιραζόμενοι πόροι, λογισμικό(software) και πληροφορία παρέχονται σε υπολογιστές και άλλες συσκευές μέσω ενός δικτύου(συνήθως μέσω Internet).

Το Cloud Computing είναι ένας όρος, που χρησιμοποιείται για τεχνολογίες που παρέχουν υπολογισμό, software, πρόσβαση σε δεδομένα και υπηρεσίες αποθήκευσης, χωρίς να απαιτείται ο τελικός χρήστης να γνωρίζει τη φυσική τοποθεσία και το configuration του συστήματος, που προσφέρει αυτές τις υπηρεσίες. Σε αυτό το σημείο μπορούμε να κάνουμε μια παρομοίωση με το δίκτυο του ηλεκτρισμού, όπου οι τελικοί χρήστες καταναλώνουν ισχύ, χωρίς να γνωρίζουν ή να έχουν κατανοήσει την υποδομή που την παρέχει.

Επίσης, είναι ένα μοντέλο που χρησιμοποιείται στην Τεχνολογία της Πληροφορίας (Information Technology-IT) και για υπηρεσίες βασισμένες σε πρωτόκολλα Internet, το οποίο συνήθως περιλαμβάνει την παροχή δυναμικά κλιμακώσιμων(dynamically scalable) και εικονοποιημένων (virtualized) πόρων. Είναι ένα υποπροϊόν και συνέπεια της ευκολίας πρόσβασης σε απομακρυσμένες τοποθεσίες υπολογισμού, που παρέχει το Internet. Επομένως, μπορεί να πάρει τη μορφή διαδικτυακών εργαλείων και εφαρμογών, τα οποία μπορούν να χρησιμοποιήσουν οι χρήστες μέσω ενός φυλλομετρητή ιστού (web browser), σαν τα προγράμματα να ήταν τοπικά εγκαταστημένα στους υπολογιστές τους.

Στα θεμέλια του Cloud Computing υπάρχουν οι έννοιες της Συγκλίνουσας Υποδομής (Converged Infrastracture) και των διαμοιραζόμενων υπηρεσιών. Με τον όρο Converged Infrastracture αναφερόμαστε στην ένωση πολλών διαφορετικών IT συνιστωσών σε μια μοναδική, βελτιστοποιημένη υπολογιστική λύση (computing solution). Αυτό το είδος περιβάλλοντος, επιτρέπει στις επιχειρήσεις, να “ανεβάζουν” τις εφαρμογές τους και να τις τρέχουν πιο γρήγορα, ενώ παράλληλα καθιστά πιο εύκολη τη διαχείριση και συντήρησή τους. Επίσης, επιτρέπει την πιο γρήγορη προσαρμογή των IT πόρων(servers, αποθήκευση, δίκτυα) στις διακυμαινόμενες κι απρόβλεπτες ανάγκες των επιχειρήσεων.

4.2) Χαρακτηριστικά του Cloud Computing

Το Cloud Computing έχει ορισμένα βασικά χαρακτηριστικά που το προσδιορίζουν. Στη συνέχεια θα παρουσιάσουμε τα χαρακτηριστικά αυτά.

- Ανεξαρτησία από την τοποθεσία στην οποία βρίσκονται τα μηχανήματα. Οι χρήστες αποκτούν πρόσβαση στο σύστημα μέσω ενός web browser, ανεξάρτητα από την τοποθεσία τους ή τη συσκευή που χρησιμοποιούν (PC, κινητό τηλέφωνο κλπ). Από τη στιγμή που η πρόσβαση γίνεται μέσω Internet, η πρόσβαση μπορεί να είναι από οπουδήποτε.
- Τεχνολογίες εικονικοποίησης (virtualization) επιτρέπουν σε servers και συσκευές αποθήκευσης να διαμοιράζονται κι έτσι να αυξάνεται η χρησιμοποίησή τους. Επίσης, χάρη στις τεχνολογίες αυτές, οι εφαρμογές μπορούν να μεταφέρονται εύκολα από έναν φυσικό server σε κάποιον άλλο.
- Το οικονομικό μοντέλο του cloud είναι ένα μοντέλο που μετατρέπει τα έξοδα κεφαλαίου σε λειτουργικά έξοδα. Για την προσφορά κάποιας εφαρμογής στο κοινό, δεν απαιτείται η αγορά υποδομής. Οι χρήστες του Cloud πληρώνουν μόνο για όσο χρησιμοποιούν την υπάρχουσα υποδομή του Cloud.
- Το Cloud Computing χαρακτηρίζεται από κλιμακωσιμότητα (scalability) κι ελαστικότητα μέσω της δυναμικής (on-demand), σχεδόν real time παροχής fine-grained πόρων. Έτσι, οι χρήστες του Cloud δεν χρειάζεται να φροντίσουν από πριν για την περίπτωση πολύ υψηλού φόρτου στην εφαρμογή τους.
- Η συντήρηση των Cloud Computing εφαρμογών είναι ευκολότερη, επειδή οι εφαρμογές δεν χρειάζεται να εγκατασταθούν στον υπολογιστή του κάθε τελικού χρήστη. Από αυτό επίσης συνεπάγεται ότι οποιαδήποτε αλλαγή στην εφαρμογή, γίνετα αυτόματα αντιληπτή από όλους τους χρήστες.
- Το Cloud Computing προσφέρει αρκετά υψηλή αξιοπιστία και διαθεσιμότητα δεδομένων. Μια μονάδα δεδομένων δεν είναι αποθηκευμένη μόνο σε μια φυσική τοποθεσία. Έτσι, αν για οποιοδήποτε λόγο ένα data center δεν είναι διαθέσιμο κάποια χρονική στιγμή, η εφαρμογή συνεχίζει να μπορεί να έχει πρόσβαση στα δεδομένα της.

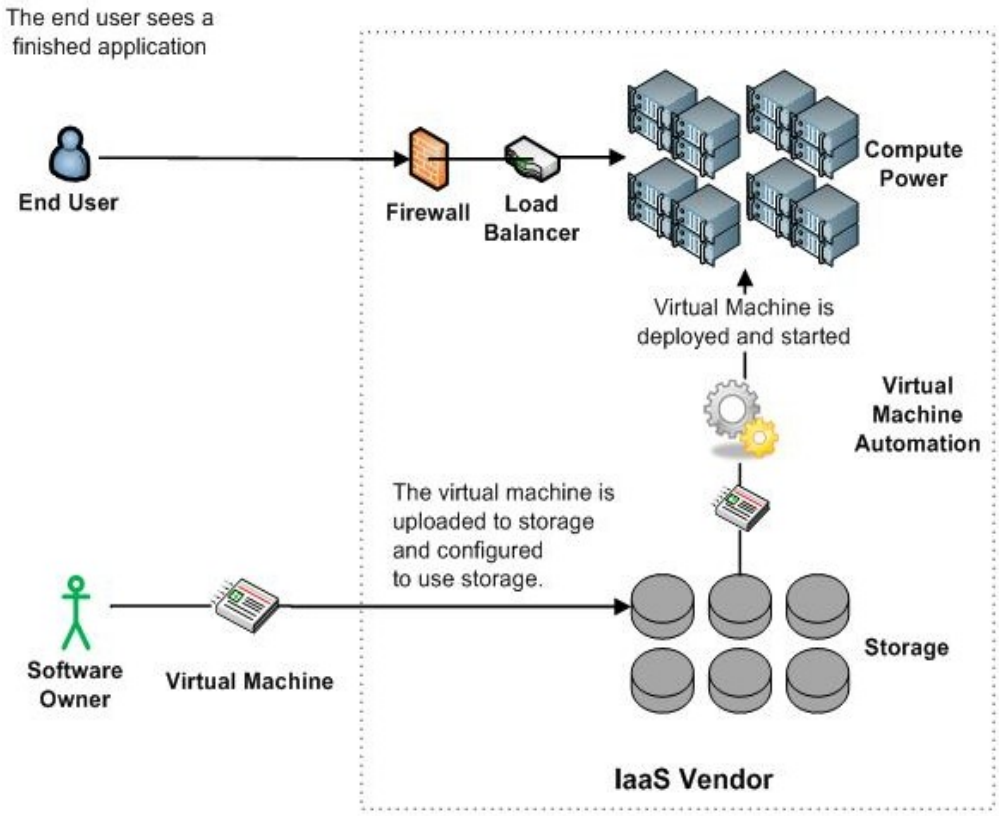
- Άλλο ένα βασικό χαρακτηριστικό του Cloud Computing είναι το multi-tenancy. Οι πόροι που διαθέτει το cloud, καθώς και τα έξοδα που έχει, διαμοιράζονται σε πολλούς χρήστες.

4.3) Προσφερόμενες υπηρεσίες

Οι υπηρεσίες που προσφέρονται μέσω Cloud Computing χωρίζονται σε τρεις βασικές κατηγορίες. Αυτές είναι η Υποδομή ως Υπηρεσία (Infrastructure as a Service-IaaS), η Πλατφόρμα ως Υπηρεσία (Platform as a Service-PaaS) και το Λογισμικό ως Υπηρεια (Software as a Service-SaaS). Στη συνέχεια θα δώσουμε τον ορισμό για καθεμιά από αυτές τις κατηγορίες.

Το Infrastructure as a Service (IaaS) προσφέρει υπολογιστική ισχύ, αποθήκευση και δικτυακή υποδομή(όπως firewalls και εξισορροπητές φόρτου (load balancers)), ως υπηρεσία μέσω του Internet. Ένας IaaS πελάτης είναι ένας ιδιοκτήτης software, ο οποίος χρειάζεται ένα περιβάλλον φιλοξενίας, στο οποίο θα τρέξει την εφαρμογή του. Αρχικά, ο όρος για τέτοιου είδους παροχές ήταν Hardware as a Service. Παρ'όλα αυτά, το IaaS κέρδισε έδαφος και είναι τώρα ο πιο ευρέως χρησιμοποιούμενος όρος για το αυτό το είδος παροχής Cloud.

Οι πάροχοι IaaS χρησιμοποιούν τεχνολογίες virtualization για να παρέχουν υπολογιστική ισχύ. Αυτό που βλέπει και μπορεί να διαχειριστεί ο ιδιοκτήτης κάποιου software είναι ένα εικονικό μηχάνημα (virtual machine). Μια εφαρμογή, καθώς και οτιδήποτε άλλο χρειάζεται αυτή για να τρέξει, πρέπει να εγκατασταθεί σε αυτό το virtual machine. Για παράδειγμα, αν μια εφαρμογή απαιτεί μια σχεσιακή βάση δεδομένων, στο virtual machine θα πρέπει να εγκατασταθεί η εφαρμογή και η βάση δεδομένων. Το virtual machine μπορεί στη συνέχεια να ανέβει στο περιβάλλον φιλοξενίας του IaaS παρόχου, όπου μπορεί να γίνει configured να χρησιμοποιεί το σύστημα αποθήκευσης του IaaS παρόχου. Από τη στιγμή που γίνει και το configuration, το virtual machine μπορεί να γίνει deploy μέσω μιας διαδικασίας όπου βρίσκεται αυτόματα διαθέσιμο hardware για να τρέξει το μηχάνημα. Οι υπολογιστές που χρειάζονται για να τρέξει η εφαρμογή, καθώς και το σύστημα αποθήκευσης που απαιτείται, ανήκουν στον πάροχο IaaS. Στην επόμενη εικόνα βλέπουμε σχηματικά τη λειτουργία του IaaS.



Σχήμα 6: Σχηματική λειτουργία του Infrastructure as a Service

Οι IaaS πάροχοι χρεώνουν την παροχή υποδομής με βάση διάφορα κριτήρια. Στον επόμενο πίνακα παρουσιάζονται μερικά από τα κριτήρια αυτά, που είναι κοινά σε διάφορους IaaS παρόχους.

Αντικείμενο που χρεώνεται	Χρέωση με βάση...
Υπολογισμός	Τις ώρες χρήσης εντός της περιόδου χρέωσης
Μεταφορά εισερχόμενων δεδομένων	Τα GB που ελήφθησαν εντός της περιόδου χρέωσης
Μεταφορά εξερχόμενων δεδομένων	Τα GB που στάλθηκαν εντός της περιόδου χρέωσης
Αποθήκευση	Τα GB εντός της περιόδου χρέωσης
Αιτήσεις για I/O αποθήκευσης	Τις αιτήσεις (σε χιλιάδες) εντός της περιόδου χρέωσης

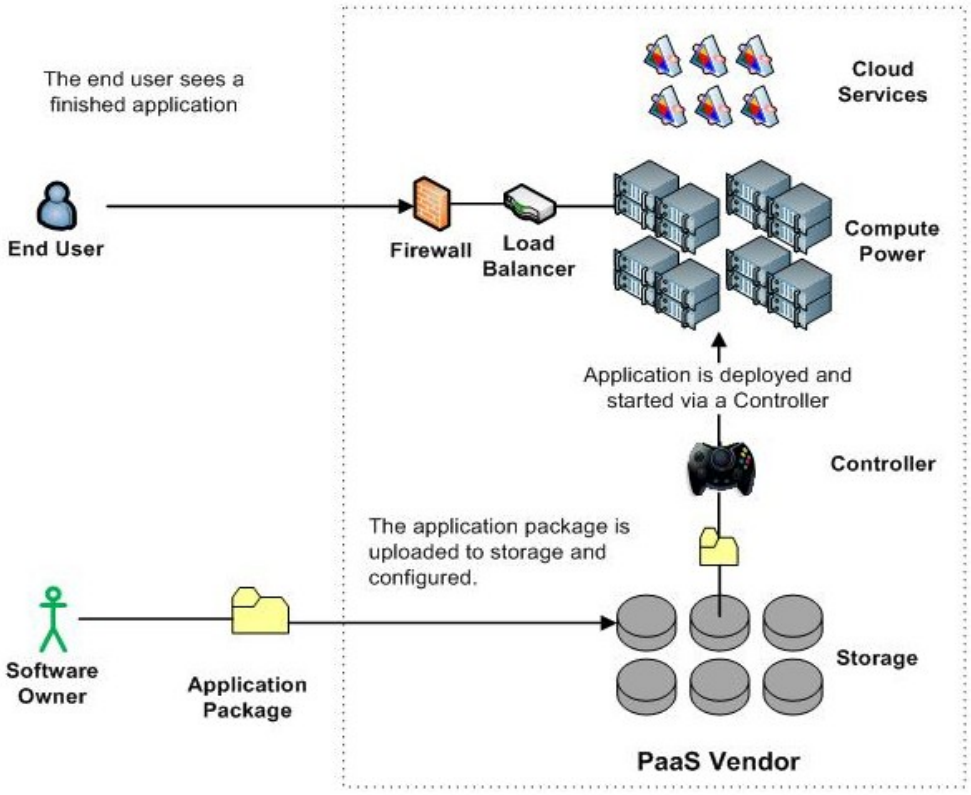
Πίνακας 4: Χρέωση πόρων σε IaaS

Τέλος, παρουσιάζουμε έναν πίνακα με ορισμένους βασικούς IaaS παρόχους:

Vendor	IaaS Offering	Hosting Environment	Storage	Cloud Services
Amazon	Amazon Web Services	Elastic Compute Cloud	Elastic Block Storage	SimpleDB Simple Storage Services (S3) CloudFront Simple Queue Services (SQS) Elastic MapReduce
ServePath	GoGrid	GoGrid Cloud Hosting	GoGrid Cloud Storage	None.
Rackspace	Mosso|The Rackspace Cloud	Cloud Servers Cloud Sites	Storage is integrated with the Cloud Servers offering.	Cloud Files

Πίνακας 5: Βασικοί πάροχοι IaaS

Όπως αναφέραμε, το δεύτερο είδος υπηρεσίας που προσφέρεται μέσω Cloud Computing είναι το Platform as a Service. Ένα PaaS προσφέρει επίσης υπολογιστική ισχύ, αποθήκευση και δικτυακή υποδομή, ως υπηρεσία μέσω Internet. Παρ'όλα αυτά, ένα PaaS προσφέρει επιπλέον ένα runtime περιβάλλον για μεταγλωττισμένο κώδικα εφαρμογών. Αυτό σημαίνει ότι δεν χρειάζεται να φτιάξουμε και να κάνουμε configure ένα ολόκληρο virtual machine, όπως χρειαζόταν στην περίπτωση του IaaS. Ο έλεγχος που έχουμε στο σύστημα δεν είναι τόσο χαμηλού επιπέδου. Το μόνο που χρειάζεται να ανεβάσουμε είναι ο κώδικας της εφαρμογής. Πελάτης ενός PaaS είναι πάλι ο ιδιοκτήτης κάποιου software , ο οποίος χρειάζεται ένα περιβάλλον φιλοξενίας για την εφαρμογή του. Στην επόμενη εικόνα βλέπουμε σχηματικά τη λειτουργία ενός PaaS.



Σχήμα 7: Σχηματική λειτουργία PaaS

Το πακέτο εφαρμογής (application package) που φαίνεται στο παραπάνω σχήμα, περιέχει μόνο τον κώδικα της εφαρμογής, η οποία απαιτείται να έχει αναπτυχθεί σε κάποιο προγραμματιστικό περιβάλλον που υποστηρίζει ο πάροχος PaaS. Για παράδειγμα, το Google AppEngine υποστηρίζει Java, Python και Go, ενώ το Microsoft Azure υποστηρίζει το .NET Framework και PHP. Μια άλλη βασική διάκριση ανάμεσα στα PaaS και IaaS περιβάλλοντα είναι ότι τα PaaS προσφέρουν μια συλλογή από Cloud υπηρεσίες, οι οποίες προσφέρουν δυνατότητες όπως αποθήκευση δεδομένων, σύνδεση σε άλλες υπηρεσίες κι αρκετές άλλες που θα δούμε αναλυτικά σε επόμενη ενότητα. Οι υπηρεσίες αυτές είναι αναγκαίες, γιατί σε αντίθεση με το IaaS, όπου χτίζουμε όλο το virtual machine όπως επιθυμούμε, στο PaaS ανεβάζουμε μόνο τον κώδικα της εφαρμογής μας. Επομένως, ένα third party software δεν μπορεί να ανέβει και να τρέξει. Χρειαζόμαστε επομένως κάποιες υπηρεσίες που θα παρέχουν βασικές δυνατότητες, που είναι αναγκαίες για την ανάπτυξη εφαρμογών.

Ένα άλλο βασικό χαρακτηριστικό ενός PaaS περιβάλλοντος είναι ότι κλιμακώνεται (scales) αυτόματα. Έτσι, αν για παράδειγμα, χρειαζόμαστε κάποια στιγμή τρία στιγμιότυπα (instances) της εφαρμογής μας για να διαχειριστούμε τον αναμενόμενο φόρτο, το περιβάλλον του PaaS θα

δημιουργήσει αυτόματα τα τρία αυτά στιγμιότυπα.

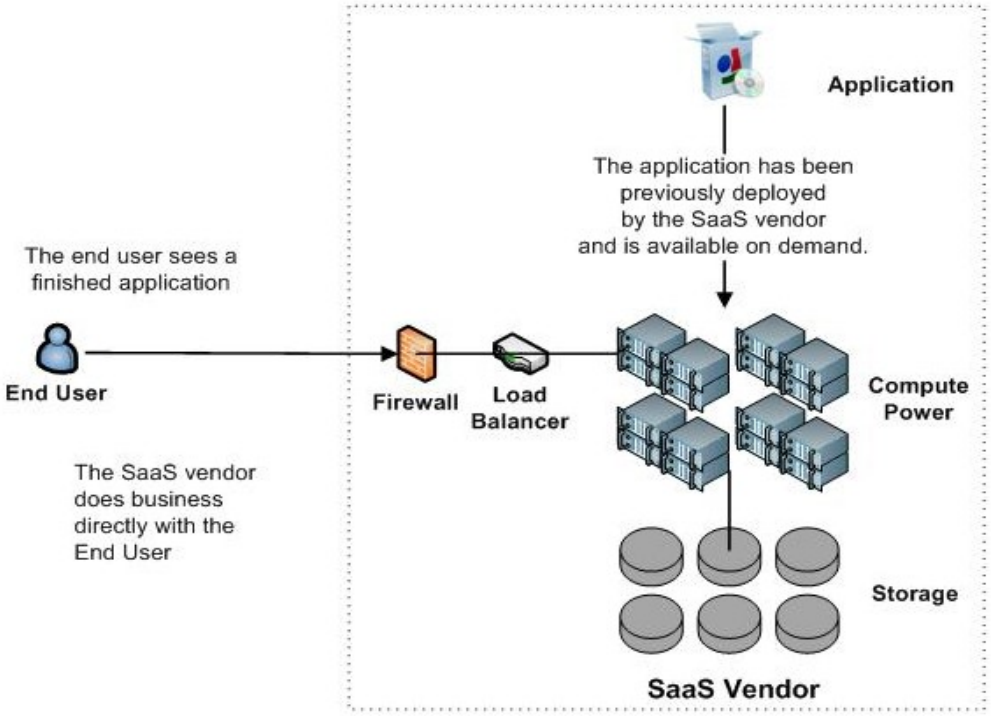
Το βασικό μειωνέκτημα του PaaS είναι οτι, για να μπορεί να κάνει scaling αυτόματα και να εγγυάται την ασφάλεια των εφαρμογών που φιλοξενεί, επιβάλλει αρκετούς περιορισμούς στις εφαρμογές. Εμείς αναπτύξαμε την εφαρμογή μας στο AppEngine της Google, το οποίο είναι ένα PaaS περιβάλλον. Οι περιορισμοί που μας επέβαλε το AppEngine καθόρισαν διάφορες σχεδιαστικές αποφάσεις μας και θα συζητηθούν σε επόμενη ενότητα. Αν κάποιος επιθυμεί μεγαλύτερη ευελιξία και έλεγχο πιο χαμηλού επιπέδου στο σύστημα, τότε το IaaS είναι ιδανικό για αυτόν.

Ο τρόπος χρέωσης για τη χρήση ενός PaaS είναι παρόμοιος με αυτόν της χρήσης ενός IaaS. Στον παρακάτω πίνακα φαίνονται μερικοί αρκετά γνωστοί PaaS πάροχοι.

Vendor	PaaS Offering	Runtime Environment	Cloud Services
Google	Google App Engine	Java Runtime Environment Python Runtime Environment	Datastore (Java, Python) Google Accounts (Java, Python) Image Manipulation (Java, Python) Mail (Java, Python) Memcache (Java, Python) URL Fetch (Java, Python)
Microsoft	Azure Services Platform	Windows Azure	Access Control Service SQL Services Workflow Services Service Bus Live Services
Salesforce.com	Force.com	Apex Code for business logic Visualforce for user interfaces	Database Services Web Service APIs

Πίνακας 6: Βασικοί πάροχοι PaaS

Η τελευταία από τις βασικές κατηγορίες προσφερόμενων υπηρεσιών μέσω Cloud Computing είναι το Software as a Service. Η ιδέα που κρύβεται πίσω από το SaaS είναι απλή. Με το SaaS, μια ολόκληρη εφαρμογή μπορεί να γίνει διαθέσιμη μέσω κάποιου SaaS παρόχου. Η εφαρμογή αυτή υπάρχει στο cloud και μπορεί να χρησιμοποιηθεί μέσω οποιουδήποτε φυλλομετρητή ιστού (web browser). Επομένως, πελάτης ενός SaaS παρόχου είναι ο τελικός χρήστης. Ο τρόπος λειτουργίας του SaaS φαίνεται στο επόμενο σχήμα.



Σχήμα 8: Σχηματική λειτουργία SaaS

Όπως φαίνεται και στο σχήμα, ο SaaS πάροχος είναι υπεύθυνος και για την παροχή της υπολογιστικής ισχύος, του συστήματος αποθήκευσης και της δικτυακής υποδομής που απαιτείται για να τρέξει η εφαρμογή. Για να τα παρέχει αυτά ο SaaS πάροχος, μπορεί να χρησιμοποιήσει αν το επιθυμεί την υποδομή που προσφέρει κάποιος IaaS ή PaaS πάροχος.

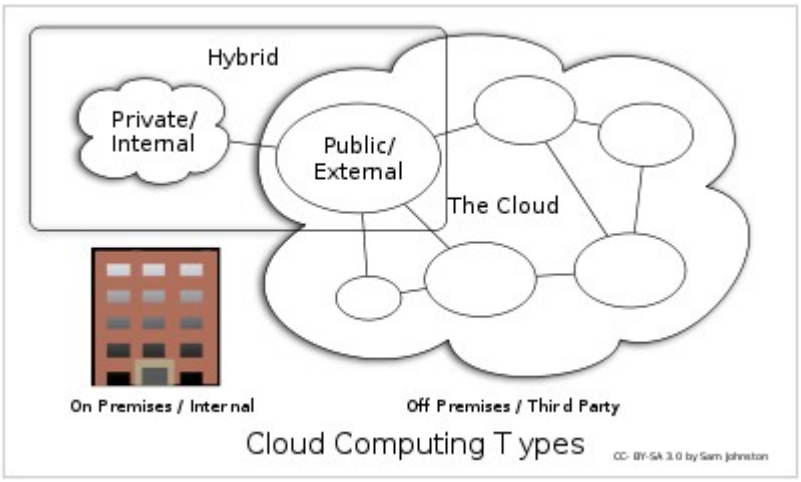
Αυτό συνέβη και στη δική μας περίπτωση. Εμείς αναπτύξαμε μια εφαρμογή, την οποία διαθέσαμε μέσω Cloud και η οποία είναι προσβάσιμη από οποιονδήποτε browser. Επομένως εμείς είμαστε ο SaaS πάροχος. Ως SaaS πάροχοι είμαστε επίσης υπεύθυνοι για την παροχή υπολογιστικής ισχύος, αποθηκευτικής και δικτυακής υποδομής για την εφαρμογή μας. Για το σκοπό αυτό, αποφασίσαμε να χρησιμοποιήσουμε ένα PaaS περιβάλλον, το οποίο στην προκειμένη περίπτωση είναι το Google AppEngine.

4.4) Deployment models

Ανάλογα με τον τρόπο που διατίθεται το Cloud στο κοινό, το διακρίνουμε σε Public Cloud, Utility Computing, Private Cloud και Hybrid Cloud. Public Cloud είναι όταν το Cloud είναι διαθέσιμο με

έναν pay-as-you-go τρόπο. Με τον όρο pay-as-you-go εννοούμε ότι ο χρήστης του Cloud δεν είναι ανάγκη να δεσμέυσει εκ των προτέρων κάποιους συγκεκριμένους πόρους για την εφαρμογή του, αλλά το σύστημα του παρέχει τους απαραίτητους πόρους για τις ανάγκες του κι ο χρήστης πληρώνει μόνο για ό,τι χρησιμοποιεί ανά πάσα στιγμή. Για να γίνει περισσότερο κατανοητό αυτό, ας σκεφτούμε μια web εφαρμογή η οποία είναι στημένη σε κάποιο Cloud. Η εφαρμογή αυτή για να λειτουργήσει, δεσμέυει κάποιους πόρους(υπολογιστικούς, δικτυακούς, αποθηκευτικούς,κλπ.). Σε κάποια χρονική στιγμή, ενδέχεται να αυξηθούν αρκετά οι αιτήσεις που θα δέχεται και στις οποίες θα πρέπει να ανταποκριθεί η εφαρμογή αυτή. Για να το πετύχει αυτό και να ικανοποιήσει τα αιτήματα όλων των χρηστών της, θα πρέπει να δεσμεύσει επιπλέον πόρους, τους οποίους ο διαχειριστής της εφαρμογής και χρήστης του Cloud θα πρέπει να πληρώσει επιπλέον. Σε κάποια άλλη χρονική στιγμή, όπου το φόρτο εργασίας της εφαρμογής θα είναι μειωμένο, οι πόροι αυτοί θα απελευθερωθούν και ο χρήστης του Cloud θα πληρώνει πάλι λιγότερα.

Με τον όρο Utility Computing αναφερόμαστε σε υπηρεσίες οι οποίες παρέχονται έναντι χρηματικού ποσού. Σε συνδυασμό με το Public Cloud που αναφέραμε, υπάρχει σήμερα η έννοια του public Utility Computing . Χαρακτηριστικά παραδείγματα από public Utility Computing είναι τα Amazon Web Services, Google AppEngine και Microsoft Azure. Δεδομένου ότι τη δική μας εφαρμογή την αναπτύξαμε πάνω στο Google AppEngine, το deployment model που χρησιμοποιήσαμε είναι το public Utility Computing. Με τον όρο Private Cloud αναφερόμαστε σε datacenters τα οποία βρίσκονται στο εσωτερικό επιχειρήσεων και δεν είναι διαθέσιμα στο ευρύ κοινό. Τέλος, με τον όρο Hybrid Cloud αναφερόμαστε στη σύνθεση δύο ή περισσότερων clouds(public ή private), τα οποία, μετά τη σύνθεση, παραμένουν ξεχωριστές οντότητες αλλά προσφέρουν πλεονεκτήματα διαφορετικών deployment models κι επιτρέπουν την εύκολη μεταφορά δεδομένων από το ένα cloud στο άλλο.



Σχήμα 9: Hybrid Cloud

4.5) Νέες εφαρμογές

Σκοπός της εφαρμογής μας είναι η διαχείριση ενός μεγάλου όγκου δεδομένων κοινωνικών δικτύων. Εφαρμογές σαν κι αυτή, οι οποίες ασχολούνται με τη διαχείριση μεγάλων datasets ή analytics εφαρμογές που κάνουν batch-processing ωφελήθηκαν αρκετά από την έλευση του Cloud Computing. Αν το είδος της επεξεργασίας, που πρέπει να γίνει είναι τέτοιο, ώστε να μπορέσει να υπάρξει παραλληλισμός δεδομένων, τότε η "συσχετιστικότητα κόστους" που υπάρχει στο οικονομικό μοντέλο του Cloud Computing, μπορεί να φανεί ιδιαίτερα χρήσιμη. Με βάση αυτή, το να χρησιμοποιηθούν εκατοντάδες υπολογιστές, για ένα σύντομο χρονικό διάστημα, κοστίζει το ίδιο με το να χρησιμοποιηθούν λιγότεροι υπολογιστές, για μεγαλύτερο χρονικό διάστημα. Αυτή η ιδιότητα, σε συνδυασμό με τον παραλληλισμό δεδομένων, οδηγεί στη γρήγορη διεκπεραίωση εργασιών, που επεξεργάζονται και αναλύουν μεγάλα σύνολα δεδομένων. Για να μπορέσουμε να κερδίσουμε όντως σε ταχύτητα, χρειαζόμαστε ένα μοντέλο υπολογισμού, το οποίο να μπορεί να εκμεταλλευτεί τον εγγενή παραλληλισμό, που ενδέχεται να υπάρχει στα δεδομένα και τη “συσχετιστικότητα” αυτή του κόστους.

Βλέπουμε λοιπόν, ότι τα εγγενή χαρακτηριστικά ορισμένων εφαρμογών ευνοούνται από το μοντέλο υπολογισμού που παρέχει το Cloud Computing. Μόλις είδαμε ότι οι analytics εφαρμογές είναι ένα είδος εφαρμογών που γνώρισε άνθιση λόγω του Cloud Computing. Αυτό όμως δεν σημαίνει ότι είναι και το μοναδικό είδος. Η αναφορά σε αυτό έγινε μόνο και μόνο λόγω της συγγένειάς του με τη

δική μας εφαρμογή.

5) Google App Engine

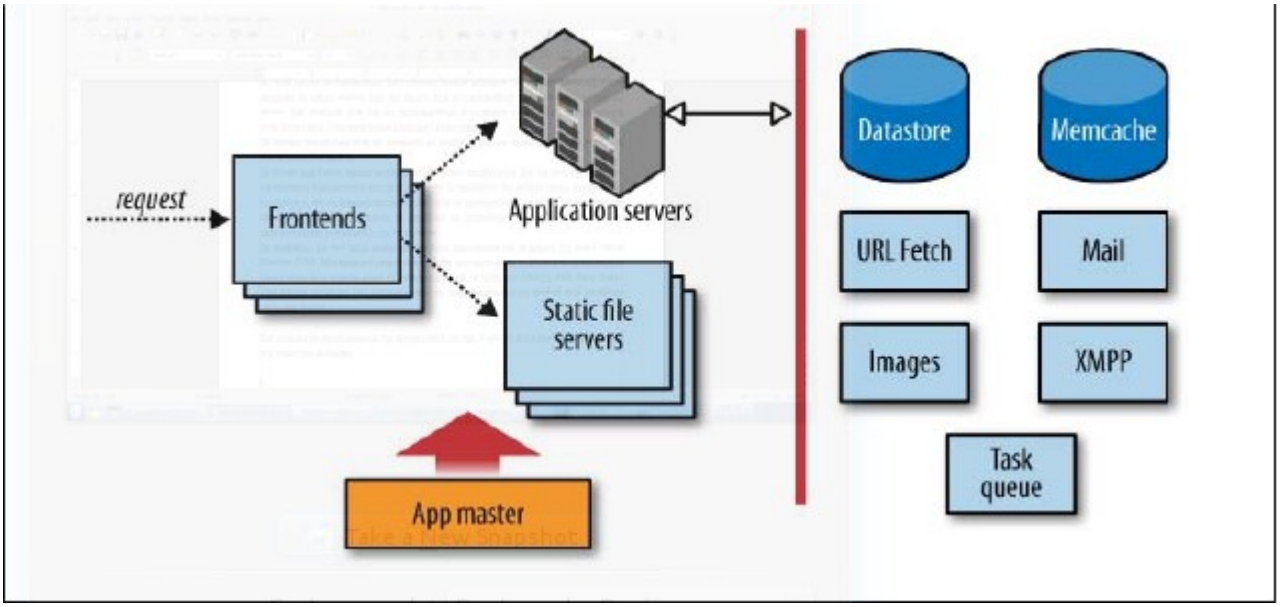
5.1) Γενική Περιγραφή

Η πλατφόρμα στην οποία επιλέξαμε να αναπτύξουμε την εφαρμογή μας είναι το AppEngine της Google. Στο κεφάλαιο αυτό, θα περιγράψουμε αναλυτικά το AppEngine και τις δυνατότητες που προσφέρει, έτσι ώστε στη συνέχεια να γίνουν κατανοητές ορισμένες σχεδιαστικές αποφάσεις που κάναμε σχετικά με την εφαρμογή μας και άπτονται άμεσα της αρχιτεκτονικής του συστήματος.

Το Google App Engine (GAE) είναι μια Cloud Computing Πλατφόρμα ως Υπηρεσία (platform as a service-PaaS) για την ανάπτυξη και φιλοξενία διαδικτυακών εφαρμογών στα κέντρα δεδομένων (datacenters) της Google. Αν και το App Engine μπορεί να φιλοξενήσει οποιουδήποτε είδους διαδικτυακές εφαρμογές, το περιβάλλον του είναι ειδικά σχεδιασμένο για την ανάπτυξη δυναμικών, πραγματικού χρόνου (real time) εφαρμογών. Πιο συγκεκριμένα, το περιβάλλον του App Engine είναι κατάλληλα σχεδιασμένο για εφαρμογές που χρειάζεται να εξυπηρετούν ταυτόχρονα πολλούς χρήστες. Όταν μια εφαρμογή εξυπηρετεί πολλούς χρήστες ταυτόχρονα, χωρίς να μειώνεται η απόδοσή της, τοτε λέμε ότι κλιμακώνεται (scales). Οι εφαρμογές που είναι γραμμένες για το App Engine κλιμακώνονται αυτόματα. Καθώς περισσότεροι χρήστες χρησιμοποιούν την εφαρμογή, το App Engine παρέχει περισσότερους πόρους στην εφαρμογή και φροντίζει και για τη διαχείριση αυτών των πόρων. Η εφαρμογή δεν χρειάζεται να γνωρίζει τίποτα σχετικά με τους πόρους που της έχουν διατεθεί ή σχετικά με τη χρήση αυτών των πόρων.

Σχετικά με το οικονομικό μοντέλο χρήσης, σε αντίθεση με τους "κλασσικούς" εξυπηρετητές (servers) για τη φιλοξενία ιστοσελίδων, με το Google App Engine, μας διατίθενται δυναμικά πόροι και πληρώνουμε μόνο για τους πόρους που χρησιμοποιούμε. Οι καταναλησκούμενοι πόροι χρεώνονται με βάση τα gigabytes χρήσης και δεν υπάρχουν πάγιες μηνιαίες χρεώσεις, όπως και δεν απαιτούνται προκαταβολές. Οι πόροι που χρεώνονται είναι η χρήση CPU για υπολογισμούς, η αποθήκευση δεδομένων, το εισερχόμενο κι εξερχόμενο εύρος ζώνης του δικτύου, καθώς και διάφορες συγκεκριμένες υπηρεσίες, που παρέχει το GAE. Επίσης, αναφέρουμε, ότι μέχρι ένα συγκεκριμένο όριο στην κατανάλωση πόρων, η χρήση του App Engine είναι χωρίς χρέωση. Έτσι, μικρές εφαρμογές, με χαμηλό φόρτο εργασίας μπορούν να φιλοξενηθούν στην υποδομή της Google χωρίς κάποιο κόστος.

Στη συνέχεια θα δώσουμε μια γενική περιγραφή της αρχιτεκτονικής του Google App Engine.



Σχήμα 10: Γενική εικόνα της αρχιτεκτονικής του Google App Engine

Μια εφαρμογή του App Engine αποκρίνεται σε αιτήματα ιστού (web requests). Ένα web request ξεκινάει όταν ένας πελάτης (client), συνήθως ο web browser κάποιου χρήστη, επικοινωνήσει με την εφαρμογή μέσω ενός HTTP request. Η πρώτη στάση για ένα εισερχόμενο request είναι το frontend του App Engine. Ένας εξισορροπητής φόρτου(load balancer) κι ένα σύστημα για την βέλτιστη κατανομή των requests σε πολλά μηχανήματα, δρομολογεί το request σε έναν από τους frontend servers. Το frontend καθορίζει, από τη διεύθυνση που υπάρχει στο request, την εφαρμογή στην οποία αναφέρεται το request. Κάθε εφαρμογή του AppEngine έχει ένα μοναδικό αναγνωριστικό (application ID) κι επίσης, με βάση το αναγνωριστικό αυτό, παίρνει ένα δικό της domain name στο appspot.com. Έτσι, η διεύθυνση προορισμού, που υπάρχει στα requests, έχει τη μορφή: http://app-id.appspot.com/url/path... Με βάση αυτό το URL επομένως, δρομολογεί το frontend το request στον κατάλληλο server. Στο σημείο αυτό σημειώνουμε ότι εκτός από το app-id.appspot.com, οι εφαρμογές του AppEngine, μπορούν να έχουν οποιοδήποτε άλλο domain name, όπως το www.example.com, αρκεί αυτό να έχει δηλωθεί στο setup της εφαρμογής, ώστε να δείχνει σε αυτήν.

Το configuration της εφαρμογής ορίζει πώς θα πρέπει να χειριστεί ένα request το frontend, ανάλογα με το URL path του request. Αν το URL path αντιστοιχεί σε ένα στατικό αρχείο, όπως μια εικόνα ή ένα αρχείο με κώδικα JavaScript, τότε αυτό θα πρέπει να σταλεί αμέσως στον client. Στην περίπτωση αυτή, το frontend δρομολογεί το request στους servers στατικών αρχείων. Οι servers αυτοί

είναι σχεδιασμένοι για την εξυπηρέτηση στατικών αρχείων. Η τοπολογία του δικτύου και η caching συμπεριφορά αυτών των server είναι τέτοια ώστε να βελτιστοποιεί την παράδοση αρχείων, που δεν τροποποιούνται συχνά.

Το URL path μπορεί επίσης να αντιστοιχεί σε έναν διαχειριστή αιτημάτων (request handler), δηλαδή σε κώδικα εφαρμογής, που καλείται για να καθορίσει την απάντηση στο request. Στην περίπτωση αυτή, το frontend στέλνει το request σε κάποιον από τους servers εφαρμογών (app servers). Τότε, το app server pool ξεκινάει ένα στιγμιότυπο (instance) της εφαρμογής σε έναν server ή χρησιμοποιεί κάποιο υπάρχον στιγμιότυπο αν υπάρχει κάποιο που τρέχει από προηγούμενο request.

Αν το URL path δεν αντιστοιχεί σε τίποτα, που να είναι δηλωμένο στο configuration της εφαρμογής, τότε το frontend επιστρέφει ως απάντηση στον client ένα: HTTP 404 “Not Found” error, όπως δηλαδή θα έκανε κι οποιαδήποτε άλλη web εφαρμογή.

Οι app servers χρησιμοποιούν διάφορες στρατηγικές για την κατανομή των requests και για την εκκίνηση των στιγμιοτύπων της εφαρμογής, ανάλογα με το φόρτο εργασίας και με τα patterns χρήσης των πόρων. Όλες όμως αυτές οι στρατηγικές είναι σχεδιασμένες, ώστε να λειτουργούν καλύτερα με request handlers, που επιστρέφουν γρήγορα. Για την ακρίβεια, Ένα request μπορεί να κάνει μέχρι 30 δευτερόλεπτα μέχρι να επιστρέψει κάποια απάντηση στον client. Σκοπός είναι να μεγιστοποιηθεί η απόδοση (throughput) των στιγμιοτύπων των εφαρμογών, ώστε να τρέχουν τόσα στιγμιότυπα, που να μπορούν να ανταποκριθούν στις τρέχουσες ανάγκες του φόρτου εργασίας. Το πλήθος των στιγμιοτύπων όμως, δεν θα πρέπει να είναι τόσο μεγάλο, ώστε να υπάρχουν στιγμιότυπα, που να είναι αδρανή, χωρίς να εκτελούν κάποια λειτουργία. Επίσης, οι app servers είναι σχεδιασμένοι έτσι ώστε, ένας request handler να μην μπορεί να επηρεάσει τη συμπεριφορά ή την επίδοση ενός άλλου handler, που τρέχει στον ίδιο server.

Τα frontends, οι app servers και οι servers στατικών αρχείων κυβερνώνται από έναν “App master”. Ανάμεσα στις άλλες αρμοδιότητές του, ο “App Master” είναι υπεύθυνος και για την ενημέρωση των εκδόσεων του software της εφαρμογής. Οι ενημερώσεις σε μια εφαρμογή διαδίδονται γρήγορα, χωρίς όμως αυτό να σημαίνει ότι δεν υπάρχει περίπτωση να τρέχει ταυτόχρονα κώδικας δυο διαφορετικών εκδόσεων της εφαρμογής. Αν η εφαρμογή λάβει κάποια requests πριν την μετάβασή της σε νέα έκδοση, τότε η επεξεργασία των request αυτών θα γίνει με τον κώδικα της προηγούμενης έκδοσης της εφαρμογής.

Όπως είδαμε, ο τρόπος με τον οποίο τρέχει μια εφαρμογή του AppEngine είναι ο εξής: το frontend δέχεται ένα request, και με βάση το URL του και το configuration της εφαρμογής, το δρομολογεί σε

έναν από πολλούς servers. Μόλις ο server λάβει το request, κάνει τους υπολογισμούς του κι επιστρέφει κάποια δεδομένα, ως απάντηση στο χρήστη.

Ο τρόπος με τον οποίο γίνεται το configuration της εφαρμογής διαφέρει ανάλογα με το runtime περιβάλλον, που χρησιμοποιούμε για την εφαρμογή. Αν και με το runtime περιβάλλον του AppEngine θα ασχοληθούμε πιο αναλυτικά στη συνέχεια, στο σημείο αυτό θα αναφέρουμε ότι το GAE υποστηρίζει τρία runtime περιβάλλοντα: ένα Java, ένα Python κι ένα Go.

Δεδομένου ότι εμείς αναπτύξαμε την εφαρμογή μας χρησιμοποιώντας το Java runtime environment, θα περιγράψουμε το configuration μιας Java AppEngine εφαρμογής. Όπως έχουμε αναφέρει, το AppEngine δεν προσφέρεται για υπολογισμούς γενικού σκοπού (general purpose computing), αλλά απευθύνεται αποκλειστικά και μόνο σε web εφαρμογές. Όλα τα αρχεία(στατικά αρχεία, μεταγλωττισμένες Java classes και configuration files) μιας Java web εφαρμογής είναι οργανωμένα σε μια δομή, που ονομάζεται Web Application Archive (WAR). Μια Java web εφαρμογή αντιστοιχίζει URL patterns σε servlets, μέσω ενός deployment descriptor, του web.xml, ο οποίος βρίσκεται στον κατάλογο WEB-INF του WAR. Έτσι, όταν μια εφαρμογή δεχτεί ένα request με κάποιο συγκεκριμένο URL, αναζητά το URL pattern αυτό στο web.xml κι ανάλογα με το servlet στο οποίο είναι αυτό αντιστοιχισμένο καλείται ο κατάλληλος request handler. Αναφέρουμε ακόμα, ότι εκτός από servlets, το AppEngine υποστηρίζει και τη χρήση JSPs. Τα JSPs είναι μια τεχνολογία για τη δημιουργία δυναμικών web σελίδων πιο υψηλού επιπέδου από αυτή των Java Servlets. Τα JSPs δεν χρειάζεται να τα δηλώσουμε στο web.xml, αλλά αντιστοιχίζονται αυτόματα με κάποιο request handler.

Στην παρακάτω εικόνα φαίνεται το web.xml της εφαρμογής μας. Παρατηρώντας το web.xml, βλέπουμε ότι η εφαρμογή μας περιλαμβάνει ένα servlet, του οποίου το url pattern είναι /datastoreinit. Έτσι, αν με τη βοήθεια ενός browser επισκεφτούμε τη σελίδα http://tweetfinder11.appspot.com/datastoreinit/, αυτόματα θα κληθεί ο κώδικας εφαρμογής που περιέχει το servlet.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">

      <servlet>
            <servlet-name>DatastoreInit</servlet-name>
            <servlet-class>gatherTweets.DatastoreInit</servlet-class>
```

```
	</servlet>
	<servlet-mapping>
		<servlet-name>DatastoreInit</servlet-name>
		<url-pattern>/datastoreinit</url-pattern>
	</servlet-mapping>

	<servlet>
	<display-name>Remote API Servlet</display-name>
	<servlet-name>RemoteApiServlet</servlet-name>
	<servlet-
class>com.google.apphosting.utils.remoteapi.RemoteApiServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
	<servlet-name>RemoteApiServlet</servlet-name>
	<url-pattern>/remote_api</url-pattern>
	</servlet-mapping>

	<servlet>
<servlet-name>WebServiceServlet</servlet-name>
<servlet-class>org.vnetcon.xml.ws.servlet.WebServiceServlet</servlet-class>
<init-param>
<param-name>WebServiceClass</param-name>
<param-value>org.vnetcon.blobdb.database.ws.WsStatement</param-value>
</init-param>
<init-param>
<param-name>SchemaFilePath</param-name>
<param-value>/org/vnetcon/blobdb/database/ws/schema1.xsd</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>WebServiceServlet</servlet-name>
<url-pattern>/blobdb/ws/*</url-pattern>
</servlet-mapping>

	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
	</welcome-file-list>
</web-app>
```

Εκτός από το web.xml, για το configuration μιας AppEngine εφαρμογής είναι απαραίτητο άλλο ένα αρχείο, το appengine-web.xml, το οποίο βρίσκεται επίσης στο WEB-INF.

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
	<application>tweetfinder11</application>
	<version>1</version>

	<sessions-enabled>true</sessions-enabled>

	<static-files>
		<include path="/*.html" />
```

```
            <include path="/*.png" />
            <include path="/*.css" />
            <include path="/*.tagger" />
      </static-files>

      <!-- Configure java.util.logging -->

      <system-properties>
            <property name="java.util.logging.config.file" value="WEB-
INF/logging.properties"/>
      </system-properties>

      <threadsafe>true</threadsafe>

</appengine-web-app>
```

Παραπάνω φαίνεται το appengine-web.xml της δικής μας εφαρμογής. Όπως βλέπουμε, στο αρχείο αυτό δηλώνεται το application id, το version της εφαρμογής, καθώς και άλλες πληροφορίες οι οποίες είναι απαραίτητες, για τον τρόπο με τον οποίο θα γίνει upload η εφαρμογή. Μια πληροφορία από αυτές, την οποία έχουμε ήδη συζητήσει, είναι ποια requests θα πρέπει να σταλούν σε static-file servers. Έτσι, το δικό μας appengine-web.xml υποδεικνύει ότι όλα τα requests, των οποίων το path τελειώνει σε “.html”, “.css”, “.png” και “.tagger”, θα πρέπει να δρομολογούνται από το frontend σε static-file servers.

Η αρχιτεκτονική του AppEngine, που περιγράψαμε, αποτελείται κατά βάση από τρία συστατικά μέρη: το runtime περιβάλλον, το datastore και τις κλιμακώσιμες υπηρεσίες που παρέχει. Θα ξεκινήσουμε με την περιγραφή του runtime environment.

5.2) Runtime environment

Το runtime περιβάλλον ενεργοποιείται μόλις ξεκινήσει η διαχείριση ενός request και εξαφανίζεται μόλις αυτή τελειώσει. Με το να μην διατηρεί στο runtime την κατάσταση ανάμεσα σε διαφορετικά requests, το App Engine μπορεί να κατανήμει το φόρτο σε όσους servers χρειάζεται, ώστε να συμπεριφέρεται σε όλα τα requests με τον ίδιο τρόπο, ανεξάρτητα με το φόρτο που διαχειρίζεται μια δεδομένη χρονική στιγμή.

Επίσης, ο κώδικας μιας εφαρμογής δεν μπορεί να έχει πρόσβαση στον server, στον οποίον τρέχει, με την κλασσική έννοια. Μια εφαρμογή μπορεί να διαβάζει τα δικά της αρχεία από το σύστημα

αρχείων του server (filesystem), αλλά δεν μπορεί να γράφει σε αρχεία, ούτε να διαβάζει αρχεία που ανήκουν σε άλλες εφαρμογές. Επίσης, μια εφαρμογή δεν μπορεί να έχει πρόσβαση στο hardware του server, που σχετίζεται με το δίκτυο. Επομένως, οι δικτυακές λειτουργίες γίνονται με τη χρήση υπηρεσιών του AppEngine, ειδικά σχεδιασμένων γι' αυτό το λόγο. Τέλος, μια εφαρμογή μπορεί να βλέπει τις μεταβλητές περιβάλλοντος, που έχει θέσει το App Engine, αλλά αλλαγές σε αυτές δεν διατηρούνται ανάμεσα σε διαφορετικά requests.

Εν ολίγοις, κάθε request ζει στο δικό του “sandbox”. Αυτό επιτρέπει στο App Engine να διαχειρίζεται ένα request με τον server, που κατά την εκτίμησή του, θα αποκρίνεται γρηγορότερα. Δεν υπάρχει τρόπος να εγγυηθεί κανείς ότι ο ίδιος server θα διαχειριστεί δυο διαφορετικά requests, ακόμα κι αν προέρχονται από τον ίδιο client και εντός σύντομου χρονικού διαστήματος.

Το “sandbox” αυτό επιτρέπει ακόμα στο App Engine, να τρέχει πολλές εφαρμογές στον ίδιο server, χωρίς η συμπεριφορά της μιας εφαρμογής να επηρεάζει την άλλη. Εκτός από την περιορισμένη πρόσβαση στο λειτουργικό σύστημα, το runtime περιβάλλον περιορίζει και τη χρήση της CPU και της μνήμης, που μπορεί να κάνει ένα request.

Οι βασικοί περιορισμοί που επιβάλλει το sandbox είναι:

- Μια εφαρμογή δεν μπορεί να δημιουργεί και να αναθέτει εργασίες (tasks) σε επιπρόσθετες διεργασίες ή νήματα (threads). Όλη η επεξεργασία ενός request πρέπει να γίνεται από τη διεργασία του request handler.
- Μια εφαρμογή δεν μπορεί να εγκαθιστά δικτυακές συνδέσεις. Ορισμένες δικτυακές δυνατότητες παρέχονται από υπηρεσίες του AppEngine, όπως είναι το URL Fetch και το Mail.
- Μια εφαρμογή μπορεί μόνο να διαβάζει από το filesystem και μπορεί να διαβάζει μόνο τον δικό της κώδικα και τα δικά της resource files. Δεν μπορεί να δημιουργεί ή να τροποποιεί αρχεία. Αντί για αρχεία, για την αποθήκευση δεδομένων, μια εφαρμογή μπορεί να χρησιμοποιεί το datastore.
- Μια εφαρμογή δεν μπορεί να δει ή να γνωρίζει οτιδήποτε για άλλες εφαρμογές ή διεργασίες που τρέχουν στο server. Το ίδιο ισχύει και για τους request handlers. Ένας request handler δεν μπορεί να δει άλλους request handlers της ίδιας εφαρμογής που μπορεί να τρέχουν παράλληλα στον ίδιο server.

Οι περιορισμοί αυτοί εφαρμόζονται σε πολλαπλά επίπεδα, έτσι ώστε να διασφαλιστεί ότι οι περιορισμοί τηρούνται, αλλά και για να διευκολυνθεί η αντιμετώπιση προβλημάτων που σχετίζεται με

το sandbox.

Το Google AppEngine παρέχει τρία διαφορετικά runtime περιβάλλοντα. Ένα για Java, ένα για Python και πρόσφατα δημιούργησε κι ένα για τη γώσσα Go. Το περιβάλλον που επιλέγει κανείς, εξαρτάται από τη γλώσσα κι από τις διάφορες τεχνολογίες που θέλει να χρησιμοποιήσει για την ανάπτυξη της εφαρμογής του. Εμείς, δεδομένου ότι επιλέξαμε να αναπτύξουμε την εφαρμογή μας σε Java, χρησιμοποιήσαμε το περιβάλλον του Java runtime.

Το περιβάλλον για Java τρέχει εφαρμογές, που έχουν δημιουργηθεί για να τρέχουν στο Java 6 Virtual Machine (JVM). Μια εφαρμογή μπορεί να αναπτυχθεί χρησιμοποιώντας τη γλώσσα προγραμματισμού Java ή όποια άλλη γλώσσα μπορεί να μεταγλωττιστεί και να τρέξει στο JVM(π.χ. PHP, Ruby, Scala). Οποιαδήποτε τεχνολογία Java λειτουργεί κάτω από τους περιορισμούς του sandbox είναι κατάλληλη για το App Engine.

Στο Java runtime περιβάλλον, οι περιορισμοί του sandbox εφαρμόζονται εντός του JVM. Εφαρμόζονται χρησιμοποιώντας ένα συνδυασμό από JVM άδειες(permissions), ένα υποσύνολο κλάσεων του Java Runtime Environment (JRE) κι από εναλλακτικές υλοποιήσεις για ορισμένες συναρτήσεις.

5.3) Datastore

Γενική Περιγραφή

Οι περισσότερες κλιμακώσιμες web εφαρμογές χρησιμοποιούν ξεχωριστό σύστημα για τη διαχείριση των web requests και για την αποθήκευση δεδομένων. Το σύστημα διαχείρισης των requests δρομολογεί κάθε request σε ένα από πολλά μηχανήματα, καθένα από τα οποία διαχειρίζεται το request χωρις να γνωρίζει τίποτα σχετικά με τα requests που έχουν δρομολογηθεί σε άλλα μηχανήματα. Κάθε request handler συμπεριφέρεται χωρίς να διατηρεί κατάσταση (stateless), δρώντας αποκλειστικά και μόνο με βάση το περιεχόμενο του request για το οποίο έχει να παράξει απάντηση. Οι περισσότερες όμως web εφαρμογές χρειάζεται να διατηρούν κατάσταση. Για παράδειγμα, η εφαρμογή μπορεί να χρειάζεται να “θυμάται” ότι ο χρήστης που έκανε ένα request είναι ο ίδιος χρήστης που έκανε ένα request νωρίτερα σε κάποιο άλλο μηχάνημα. Για το λόγο αυτό, οι request handlers πρέπει να αλληλεπιδρούν με μια κεντρική βάση δεδομένων, όπου και θα ενημερώνεται η κατάσταση της

εφαρμογής.

Όπως οι request handlers κατανέμουν τα requests σε πολλά μηχανήματα για scaling and robustness, το ίδιο κάνει κι η βάση δεδομένων. Αλλά σε αντίθεση με τους request handlers, οι βάσεις δεδομένων πρέπει εξ' ορισμού να διατηρούν κατάσταση (stateful) κι αυτό θέτει ορισμένες ερωτήσεις. Ποιο μηχάνημα θυμάται ποιο μέρος των δεδομένων; Πώς το σύστημα δρομολογεί ένα ερώτημα στο μηχάνημα (ή στα μηχανήματα) που μπορεί να απαντήσει στο ερώτημα; Όταν ο client ενημερώνει τα δεδομένα, πόσο καιρό χρειάζεται ώστε όλα τα μηχανήματα να γνωρίζουν ότι υπάρχει τελευταία έκδοση των δεδομένων και τι γυρνάει το σύστημα ως απάντηση στα ερωτήματα στο μεταξύ; Τι συμβαίνει όταν δυο clients επιχειρήσουν να ενημερώσουν τα ίδια δεδομένα ταυτόχρονα; Τι συμβαίνει όταν πέσει ένα μηχάνημα;

Όπως και με τη διαχείριση των requests, το AppEngine διαχειρίζεται την κλιμάκωση και τη διατήρηση της αποθήκευσης δεδομένων αυτόματα. Η εφαρμογή αλληλεπιδρά με ένα αφηρημένο μοντέλο, που κρύβει τις λεπτομέρειες της διαχείρισης ενός pool από data servers. Αυτό το μοντέλο κι η υπηρεσία που υπάρχει πίσω από αυτό παρέχουν απαντήσεις στα ερωτήματα που θέσαμε παραπάνω.

Πίσω από αυτό το αφηρημένο μοντέλο, το AppEngine προσφέρει δυο επιλογές για την αποθήκευση των δεδομένων, οι οποίες διαφέρουν μεταξύ τους ως προς τη διαθεσιμότητα και τη συνέπεια των δεδομένων.

Η πρώτη επιλογή είναι το High Replication Datastore (HRD). Στο HRD, τα δεδομένα αντιγράφονται σε πολλά data centers χρησιμοποιώντας ένα σύστημα που βασίζεται στον αλγόριθμο Paxos. Αυτό παρέχει το υψηλότερο επίπεδο διαθεσιμότητας για αναγνώσεις κι εγγραφές, με το κόστος υψηλότερου latency στις εγγραφές, λόγω της διάδοσης των δεδομένων. Τα περισσότερα ερωτήματα στο HRD είναι eventually consistent.

Eventual consistency είναι ένα μοντέλο συνέπειας, που χρησιμοποιείται σε παράλληλα και κατανεμημένα συστήματα. Στο μοντέλο αυτό, δεδομένου ενός αρκούντως μεγάλου χρονικού διαστήματος, στο οποίο δεν έχουν σταλεί αλλαγές στο σύστημα, όλες οι ενημερώσεις αναμένονται να διαδοθούν τελικώς στο σύστημα κι όλα τα αντίγραφα να είναι συνεπή μεταξύ τους. Σημειώνουμε ότι το High Replication Datastore είναι η default επιλογή για μια νέα εφαρμογή του AppEngine κι ότι αυτή είναι η επιλογή που χρησιμοποιήσαμε κι εμείς για την εφαρμογή μας..

Η δεύτερη επιλογή είναι το Master/Slave Datastore. Το Master/Slave Datastore προσδιορίζει ένα data center, το οποίο διατηρεί το κύριο αντίγραφο όλων των δεδομένων. Τα δεδομένα που γράφονται στο master data center διαδίδονται ασύγχρονα σε όλα τα υπόλοιπα (slave) data centers. Από τη στιγμή

που μόνο ένα data center είναι master για εγγραφές σε μια δεδομένη χρονική στιγμή, το Master/Slave προσφέρει ισχυρή συνέπεια (strong consistency) για όλες τις αναγνώσεις και τα ερωτήματα, με το κόστος προσωρινής μη διαθεσιμότητας, λόγω προβλημάτων των data centers ή προγραμματισμένου downtime. Στον παρακάτω πίνακα, φαίνεται μια σύγκριση των χαρακτηριστικών του High Replication Datastore και του Master/Slave Datastore.

	High Replication	**Master/Slave**
Performance		
Put/delete latency	1/2x–1x	1x
Get latency	1x	1x
Query latency	1x	1x
Consistency		
Put/get/delete	Strong	Strong
Most queries	Eventual	Strong
Ancestor queries	Strong	Strong
Occasional planned read-only period		
	No	Yes
Unplanned downtime		
	Extremely rare; no data loss.	Rare; possible to lose a small % of writes occurring near downtime (recoverable after event)

Πίνακας 7: Σύγκριση χαρακτηριστικών High Replication και Master-Slave datastore

Στη συνέχεια, θα παρουσιάσουμε το αφηρημένο μοντέλο του Datastore, της βάσης δεδομένων του Google AppEngine, με το οποίο αλληλεπιδρούν οι εφαρμογές του. Το Datastore του AppEngine είναι μια αντικειμενοστραφής βάση δεδομένων, η οποία στηρίζεται στο Big Table της Google, το οποίο με τη σειρά του είναι ένα σύστημα βάσεων δεδομένων που έχει αναπτυχθεί στο Google File System (GFS). Ένα αντικείμενο στο Datastore είναι γνωστό ως οντότητα (entity). Ένα entity έχει ένα κλειδί, που το αναγνωρίζει μοναδικά σε ολόκληρο το σύστημα. Τα κλειδιά μπορούν να αποθηκεύονται ως δεδομένα στα entities, ώστε να δημιουργούνται αναφορές από ένα αντικείμενο σε ένα άλλο.

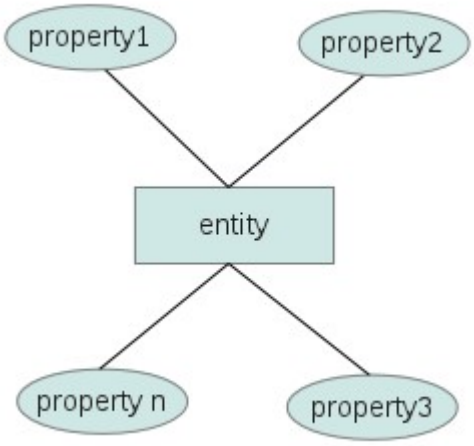
Το κλειδί ενός entity έχει διάφορα μέρη. Το πρώτο μέρος είναι το application id, το οποίο διαβεβαιώνει ότι δεν υπάρχει entity σε άλλη εφαρμογή με το ίδιο κλειδί. Επίσης διαβεβαιώνει ότι καμιά άλλη εφαρμογή δεν μπορεί να αποκτήσει πρόσβαση στα δεδομένα μιας συγκεκριμένης εφαρμογής.

Ένα άλλο σημαντικό μέρος του κλειδιού είναι το είδος (kind). Κάθε entity του Datastore ανήκει σε ένα συγκεκριμένο kind. Το kind κατηγοριοποιεί το entity για τους σκοπούς των ερωτημάτων. Για παράδειγμα, μια εφαρμογή διαχείρισης ανθρώπινου δυναμικού, ενδεχομένως να αναπαριστά κάθε πελάτη της εταιρίας ως ένα entity που ανήκει στο kind “Emploee”. Η εφαρμογή είναι υπεύθυνη για να προσδιορίσει το kind ενός entity κατά τη δημιουργία του entity.

Τέλος, το κλειδί περιέχει κι ένα entity ID. Αυτό μπορεί να είναι μια αυθαίρετη συμβολοακολουθία (string), που καθορίζεται από την εφαρμογή ή μπορεί να είναι ένα κλειδί που δημιουργείται αυτόματα από το Datastore.

Από τη στιγμή που ένα entity έχει δημιουργηθεί, το κλειδί του δεν μπορεί να αλλάξει κι αυτό ισχύει για όλα τα μέρη του κλειδιού.

Τα δεδομένα ενός entity είναι αποθηκευμένα σε μία ή περισσότερες ιδιότητες (properties). Ένα property έχει ένα όνομα και μια ή περισσότερες τιμές. Οι τιμές ενός entity για ένα δεδομένο property δεν είναι ανάγκη να είναι όλες του ίδιου τύπου. Στον πίνακα που ακολουθεί φαίνονται οι δυνατοί τύποι τιμών για ένα property.



Σχήμα 11: Σχηματική αναπαράσταση ενός entity με τα properties του

Value type	Java type(s)	Sort order	Notes
Integer	short int long java.lang.Short java.lang.Integer java.lang.Long	Numeric	Stored as long integer, then converted to the field type Out-of-range values overflow
Floating-point number	float double java.lang.Float java.lang.Double	Numeric	64-bit double precision, IEEE 754
Boolean	boolean java.lang.Boolean	false < true	
Text string (short)	java.lang.String	Unicode	Up to 500 Unicode characters Values longer than 500 characters throw IllegalArgumentException
Text string (long)	com.google.appengine.api.datastore.Text	None	Up to 1 megabyte Not indexed
Byte string (short)	com.google.appengine.api.datastore.ShortBlob	Byte order	Up to 500 bytes Values longer than 500 bytes throw IllegalArgumentException
Byte string (long)	com.google.appengine.api.datastore.Blob	None	Up to 1 megabyte Not indexed
Date and time	java.util.Date	Chronological	
Geographical point	com.google.appengine.api.datastore.GeoPt	By latitude, then longitude	
Postal address	com.google.appengine.api.datastore.PostalAddress	Unicode	
Telephone number	com.google.appengine.api.datastore.PhoneNumber	Unicode	
Email address	com.google.appengine.api.datastore.Email	Unicode	
Google Accounts user	com.google.appengine.api.users.User	Email address in Unicode order	
Instant messaging handle	com.google.appengine.api.datastore.IMHandle	Unicode	
Link	com.google.appengine.api.datastore.Link	Unicode	
Category	com.google.appengine.api.datastore.Category	Unicode	
Rating	com.google.appengine.api.datastore.Rating	Numeric	
Datastore key	com.google.appengine.api.datastore.Key or the referenced object (as a child)	By path elements (kind, identifier, kind, identifier…)	
Blobstore key	com.google.appengine.api.blobstore.BlobKey	Byte order	
Null	null	None	

Πίνακας 8: Δυνατοί τύποι δεδομένων για τις τιμές ενός property

Τα entities στο Datastore σχηματίζουν ένα ιεραρχικά δομημένο σχήμα, παρόμοιο με αυτό της δομής των καταλόγων σε ένα file system. Όταν δημιουργούμε ένα νέο entity, μπορούμε προαιρετικά, να δηλώσουμε ένα άλλο entity ως γονιό του. Η συσχέτιση αυτή, μεταξύ ενός entity και του γονιού του

είναι μόνιμη και δεν μπορεί να αλλάξει από τη στιγμή, που το entity έχει δημιουργηθεί. Ένα entity χωρίς γονιό ονομάζεται root entity. Ο γονιός ενός entity, ο γονιός του γονιού ενός entity κ.ο.κ αναδρομικά, είναι οι πρόγονοι του entity. Η ακολουθία των entities ξεκινώντας από ένα root entity και πηγαίνοντας από γονιό σε παιδί, αποτελεί το μονοπάτι των προγόνων ενός entity και συμπεριλαμβάνεται στο κλειδί του. Φαίνεται επομένως ότι τα entities συνδέονται μεταξύ τους με έναν τρόπο τέτοιο ώστε να προκύπτει μια δενδρική δομή.

Παραπάνω, είδαμε τα βασικά στοιχεία του αφηρημένου μοντέλου του Datastore. Για τη διαχείριση αυτών των στοιχείων, το AppEngine παρέχει το low-level API. Το low-level API παρέχει τις βασικές λειτουργίες που μπορούν να γίνουν με τα entities κι αυτές είναι: η δημιουργία ενός νέου entity (new), η ανάκτηση ενός entity από το Datastore (get), η αποθήκευση ενός entity στο Datastore (put), η διαγραφή ενός entity (delete) και ερωτήματα πάνω σε entities (query), που θα τα δούμε αναλυτικά στην αμέσως επόμενη ενότητα.

Το low-level API όμως δεν είναι η μόνη επιλογή για τη διαχείριση των δεδομένων σε μια εφαρμογή του AppEngine. Η ομάδα του Google AppEngine έχει φροντίσει ώστε διάφορα γνωστά frameworks του Java SDK για την μοντελοποίηση και την αποθήκευση δεδομένων να είναι συμβατά με το AppEngine. Τα frameworks αυτά παρέχουν ένα μοντέλο δεδομένων πιο υψηλού επιπέδου από αυτό του Datastore. Τέτοια frameworks είναι για παράδειγμα τα: Java Data Objects (JDO), Java Persistence API (JPA), Objectify, TwiG και Slim3. Παρ' όλων όμως των ευκολιών που παρέχουν τα frameworks αυτά, για λόγους που θα εξηγήσουμε στη συνέχεια, για την εφαρμογή μας επιλέξαμε το low-level API.

5.3.2) Ερωτήματα

Μια εφαρμογή που διαχειρίζεται δεδομένα πρέπει να μπορεί να κάνει περισσότερα από το να αποθηκεύει ή να αναζητά μια εγγραφή κάθε φορά. Θα πρέπει να μπορεί να απαντάει ερωτήσεις σχετικές με τα δεδομένα. Ειδικά οι web εφαρμογές αναμένονται όχι απλά να γνωρίζουν την απάντηση στα ερωτήματα αυτά, αλλά και να την παρέχουν γρήγορα ως απόκριση στα web requests.

Τα περισσότερα συστήματα βάσεων δεδομένων παρέχουν έναν μηχανισμό για την εκτέλεση ερωτημάτων και το ίδιο κάνει και το Datastore του AppEngine. Το Datastore, όταν μια εφαρμογή κάνει μια ερώτηση, αντί να ψάχνει όλες τις εγγραφές και να προσπαθεί να υπολογίσει την απάντηση, απλά βρίσκει την απάντηση σε μια λίστα από πιθανές απαντήσεις που έχει ετοιμάσει από πριν. Το

AppEngine μπορεί να το κάνει αυτό, επειδή μπορεί να γνωρίζει εκ των προτέρων ποιες ερωτήσεις πρόκειται να ερωτηθούν.

Αυτό το είδος λίστας, ή δείκτη (index) το App Engine το κατασκευάζει αυτόματα για ορισμένα είδη ερωτημάτων. Για ερωτήματα, που δεν ανήκουν σε αυτήν την κατηγορία, χρειάζεται να δηλώσουμε ρητά στο App Engine ότι πρέπει να κατασκευάσει κάποιο index, ενώ υπάρχουν και ερωτήματα που δεν υποστηρίζονται καθόλου από το App Engine. Στην εφαρμογή μας, όλα τα ερωτήματα που χρησιμοποιούμε είναι τέτοια ώστε το App Engine να κατασκευάζει αυτόματα τα απαραίτητα indices. Από τη στιγμή που το Datastore χρειάζεται μια απλά σάρωση ενός index για κάθε ερώτημα, η εφαρμογή επιστρέφει τα αποτελέσματα γρήγορα. Και για μεγάλες ποσότητες δεδομένων, το AppEngine μπορεί να μοιράσει τα δεδομένα και τα indices σε πολλά μηχανήματα και να πάρει πίσω τα αποτελέσματα, χωρίς να χρειάζεται να εκτελέσει κάποια ακριβή συναθροιστική λειτουργία.

Στη συνεχεια θα δουμε τι είδους ερωτήματα μπορούν να ερωτηθούν στο Datastore του AppEngine, τι ακριβώς είναι τα indices και πώς αυτά δημιουργούνται.

Αν γνωρίζουμε το κλειδί ενός entity, τότε μπορούμε να φέρουμε το entity αυτό γρήγορα από το Datastore. Το κλειδί ενός entity όμως συνήθως δεν είναι γνωστό. Ας σκεφτούμε ότι το κλειδί μπορεί να είναι ένα νούμερο που έχει ανατεθεί στο entity από το AppEngine. Έτσι, για να φέρουμε ένα entity από το Datastore, το αναζητάμε με βάση κάποια κριτήρια που θέλουμε να πληρεί. Για να βρίσκουμε entities με αυτόν τον τρόπο, η εφαρμογή εκτελεί ένα ερώτημα. Ένα ερώτημα περιλαμβάνει:

- Το kind των entities που αναζητούμε.
- Κανένα ή περισσότερα φίλτρα. Δηλαδή, κριτήρια που οι τιμές των properties πρέπει να ικανοποιούν, ώστε το ερώτημα να επιστρέψει το συγκεκριμένο entity.
- Καμία ή περισσότερες σειρές ταξινόμησης, που προσδιορίζουν τη σειρά με την οποία θα επιστραφούν τα αποτελέσματα, με βάση τις τιμές των properties τους.

Ένα ερώτημα που βασίζεται στις τιμές κάποιου property μπορεί να επιστρέψει entities μόνο ενός kind. Αυτός είναι και ο κύριος σκοπός των kinds: να προσδιορίζουν ποια entities είναι πιθανά αποτελέσματα ενός ερωτήματος. Στην πράξη, το kind δείχνει αυτό που καταλαβαίνουμε και διαισθητικά: entities του ίδιου kind αναπαριστούν δεδομένα του ίδιου είδους. Αλλά σε αντίθεση με άλλα συστήματα δεδομένων, είναι ευθύνη της εφαρμογής να είναι συνεπής ως προς αυτό κι αν κρίνεται σκόπιμο η εφαρμογή μπορεί και να μην το τηρεί.

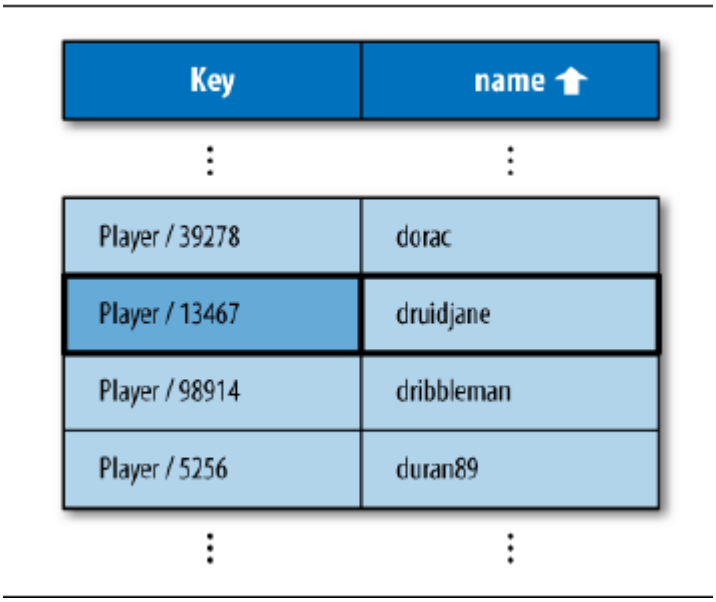
Όταν το Datastore επιστρέφει τα αποτελέσματα ενός ερωτήματος, επιστρέφει ως αποτέλεσμα στην εφαρμογή ολόκληρα τα entities που ικανοποιούν το ερώτημα. Παρ' όλα αυτά, για μεγάλα entities και για εξοικονόμηση χρόνου, μπορούμε να ζητήσουμε να επιστρέφονται μόνο τα κλειδιά των entities που ικανοποιούν το ερώτημα.

Όπως έχουμε ήδη αναφέρει, για κάθε ερώτημα που εκτελεί μια εφαρμογή, το AppEngine διατηρεί ένα index, έναν πίνακα με τις πιθανές απαντήσεις για το ερώτημα. Για την ακρίβεια, διατηρεί ένα index για ένα σύνολο ερωτημάτων που χρησιμοποιούν τα ίδια φίλτρα και τις ίδιες σειρές διάταξης, πιθανώς με διαφορετικές τιμές για τα φίλτρα.

Ας θεωρήσουμε το επόμενο απλό ερώτημα:

```
SELECT * FROM Player WHERE name = 'druidjane'
```

Για να εκτελέσει αυτό το ερώτημα, το AppEngine χρησιμοποιεί ένα index που περιέχει τα κλειδιά κάθε Player entity και τις τιμές για το name property όλων των entities, διατεταγμένες σε αύξουσα σειρά. Ένας τέτοιος πίνακας φαίνεται στην επόμενη εικόνα.



Σχήμα 12: Index για ερώτημα ισότητας πάνω σε ένα property

Για να βρει όλα τα entities που ικανοποιούν το ερώτημα, το AppEngine βρίσκει την πρώτη σειρά του index που ικανοποιεί το ερώτημα και προχωράει προς τα κάτω μέχρι να βρει την πρώτη γραμμή

που δεν το ικανοποιεί. Στο τέλος επιστρέφει τα αποτελέσματα που βρίσκονται σε αυτήν την περιοχή. Δεδομένου ότι ο πίνακας είναι ταξινομημένος, είμαστε σ'ιγουροι ότι τα αποτελέσματά μας, θα βρίσκονται σε διαδοχικές γραμμές του πίνακα.

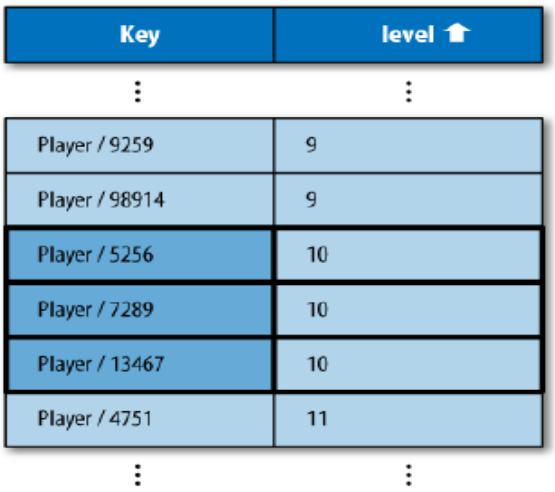
Ο μηχανισμός αυτός είναι γρήγορος και λειτουργεί καλά ακόμα και με μεγάλα σύνολα δεδομένων, αφού μπορεί να τα κατανήμει σε διαφορετικά μηχανήματα που θα εκτελούν παράλληλα το ερώτημα ανεξάρτητα το ένα από το άλλο. Ένας άλλος λόγος που κάνει αυτόν το μηχανισμό αποδοτικό είναι η ταχύτητα με την οποία βρίσκει την πρώτη γραμμή που ικανοποιεί το ερώτημα. Μάλιστα, ο αλγόριθμος που χρησιμοποιείται για να βρει την πρώτη γραμμή που ικανοποιεί το ερώτημα είναι τέτοιος που χρειάζεται περίπου τον ίδιο χρόνο πάντα, ανεξάρτητα από το μέγεθος του index, επομένως και ανεξάρτητα από το μέγεθος των δεδομένων.

Επίσης, είναι βασικό να αναφέρουμε ότι κάθε φορά που ενημερώνεται η τιμή ενός property, το AppEngine ενημερώνει και όλα τα σχετικά indices.

Το AppEngine φυλάει δυο indices για κάθε όνομα property και kind: ένα με τις τιμές των properties σε αύξουσα σειρά κι ένα με τις τιμές σε φθίνουσα σειρά. Τα indices αυτά τα δημιουργεί το AppEngine αυτόματα, χωρίς να χρειάζεται να το δηλώσουμε εμείς ρητά σε κάποιο configuration file. Τα ερωτήματα που μπορούν να απαντηθούν με χρήση αυτών των indices, που δημιουργούνται αυτόματα, είναι ερωτήματα που χρησιμοποιούν διαδοχικές γραμμές του index και είναι τα εξής:

- Ερώτημα για όλα τα entities ενός δεδομένου kind, χωρίς φίλτρα ή σειρές ταξινόμησης.

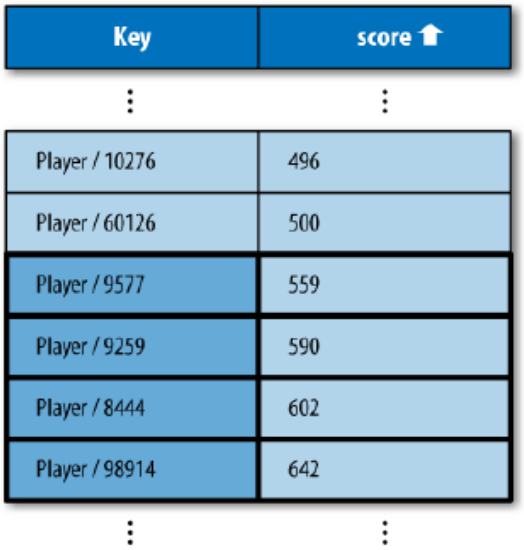
Σχήμα 13: Index για ερώτημα για όλα τα entities ενός δεδομένου kind



Σχήμα 14: Index για ερώτημα ισότητας πάνω σε ένα property

- Ερώτημα πάνω σε ένα property χρησιμοποιώντας τελεστή ισότητας (=).

- Ερώτημα με φίλτρα που χρησιμοποιούν τους τελεστές >, ≥, <, ≤ πάνω σε ένα property.



- Ερώτημα με μια σειρά ταξινόμησης, αύξουσα ή φθίνουσα, χωρίς φίλτρα ή με φίλτρα στο ίδιο property και χρησιμοποιώντας την ίδια σειρά διάταξης.
- Ερώτημα με φίλτρα ή με κάποια σειρά ταξινόμησης πανω στο κλειδί του entity.
- Ερωτήματα που δεν αφορούν σε κάποιο συγκεκριμένο kind (Kindless ερωτήματα). Εκτός από ερωτήματα σε entities ενός δεδομένου kind, το Datastore επιτρέπει κι ένα περιορισμένο σύνολο ερωτημάτων σε entities όλων των kinds. Τα kindless ερωτήματα δεν μπορούν να χρησιμοποιούν φίλτρα ή ταξινόμηση πάνω σε properties. Μπορούν όμως να χρησιμοποιούν φίλτρα ισότητας και ανισότητας σε κλειδιά.

Τα indices που δημιουργεί αυτόματα το AppEngine δεν καλύπτουν όλα τα ερωτήματα.Παρ' όλα αυτά, πρέπει να υπάρχει κάποιο index για κάθε ερώτημα. Για το λόγο αυτό, μας δίνεται η δυνατότητα να δηλώσουμε στο AppEngine τα indices που θέλουμε εμείς να δημιουργήσει για τα ερωτήματά μας. Για τις Java εφαρμογές του AppEngine, υπάρχει ένα configuration file στο οποίο δηλώνουμε τα indices που θέλουμε να ετοιμάσει το AppEngine για εμάς. Το αρχείο αυτό είναι το datastore-indexes.xml και βρίσκεται κάτω από τον κατάλογο WEB-INF.

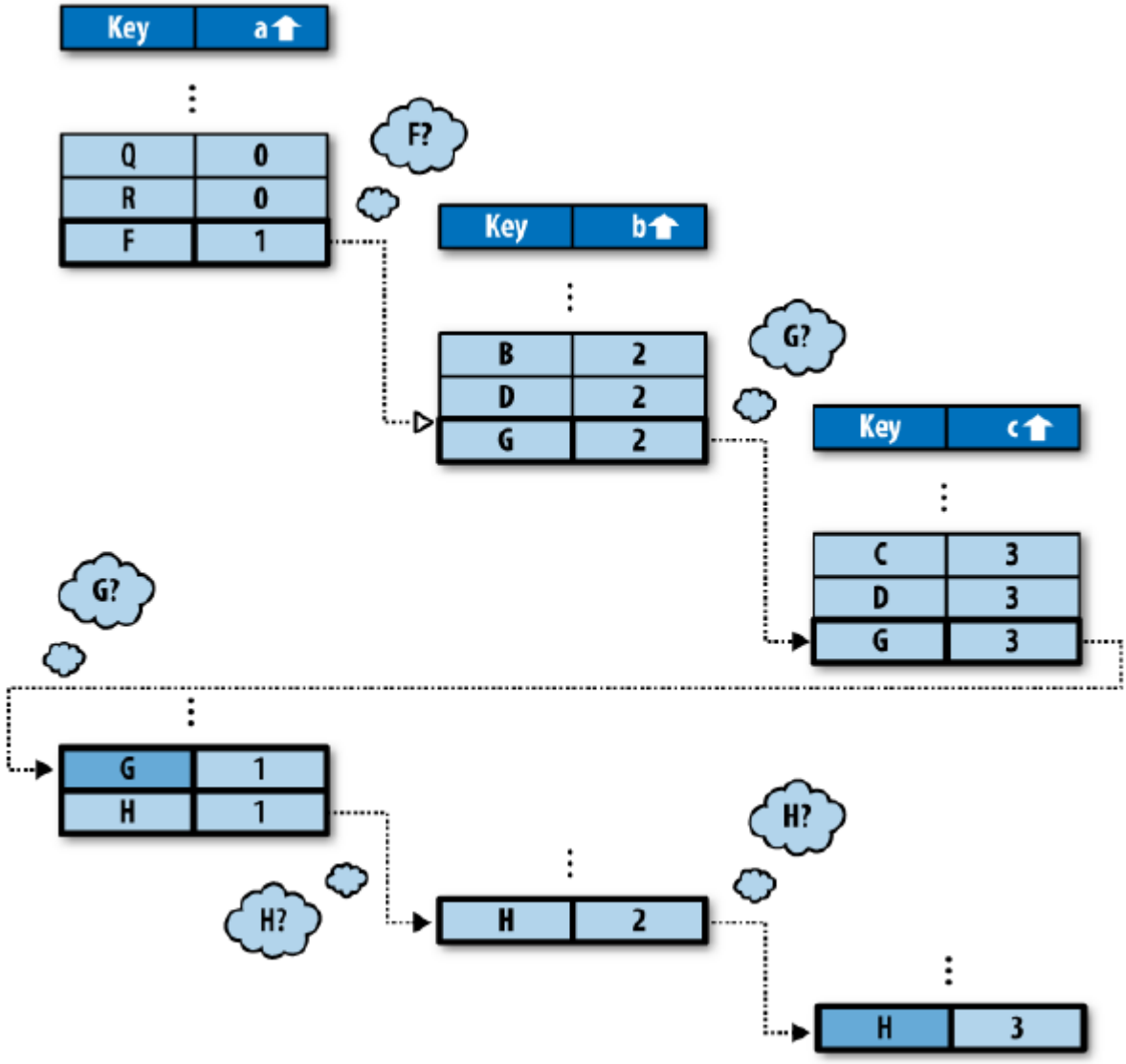
Στην δική μας εφαρμογή, δεν χρειάστηκε να κάνουμε κάποιο ερώτημα που να μην καλύπτεται από τα αυτόματα indices, γι' αυτό και δεν θα επεκταθούμε περισσότερο. Τα ερωτήματά μας ήταν μόνο ερωτήματα ισότητας πάνω σε ένα ή περισσότερα properties ενός entity.

Σχετικά με τα ερωτήματα που έχουν φίλτρα ισότητας σε πολλά properties δεν χρειάζεται κάποιο

ειδικό index, αλλά υπάρχει ένας ειδικός αλγόριθμος που χρησιμοποιεί τα αυτόματα indices για την περίπτωση αυτή. Ο αλγόριθμος αυτός κάνει ουσιαστικά ένα merge join στα αποτελέσματα που βρίσκει στο index, που έχει δημιουργηθεί αυτόματα από το AppEngine, για φίλτρο ισότητας πάνω σε κάποιο property. Τα βήματα που ακολουθεί είναι τα εξής:
Ας θεωρήσουμε το ερώτημα:

SELECT * FROM Kind WHERE a=1 AND b=2 AND c=3

i. Το Datastore θα ελέγξει το index για το property a για να βρει την πρώτη γραμμή με τιμή 1. Το entity του οποίου το κλειδί είναι στη γραμμή αυτή είναι υποψήφιο, αλλά δεν έχει επιβεβαιωθεί ακόμα ως αποτέλεσμα.
ii. Στη συνέχεια ελέγχει το index για το property b, για να βρει την πρώτη γραμμή με τιμή 2 και της οποίας το κλειδί είναι μεγαλύτερο ή ίσο με το κλειδί του υποψήφιου entity από το βήμα 1. Πριν από αυτή τη γραμμή μπορεί να εμφανιστούν κι άλλες γραμμές με τιμή 2 στο b index, αλλά το Datastore γνωρίζει ότι αυτές δεν είναι υποψήφιες γιατί στην αναζήτηση στο a index προσδιορίσαμε το υποψήφιο entity με το μικρότερο κλειδί.
iii. Αν το Datastore βρει το κλειδί του υποψηφίου από το a index, στην περιοχή του b index, που προσδιορίσαμε στο βήμα 2, αυτό το κλειδί εξακολουθεί να είναι υποψήφιο και προχωράει για έλεγχο με αντίστοιχη διαδικασία στο c index. Αν το Datastore δεν βρει το υποψήφιο κλειδί στο b index, αλλά βρει ένα άλλο μεγαλύτερο κλειδί με τιμή που ταιριάζει, αυτό το κλειδί γίνεται το νέο υποψήφιο και προχωράει για έλεγχο στο c index. Αν δεν βρει ούτε το υποψήφιο, ούτε γραμμή, με μεγαλύτερο κλειδί, που να ταιριάζει στο ερώτημα, το ερώτημα ολοκληρώνεται.
iv. Αν βρεθεί ένα υποψήφιο entity που ικανοποιεί όλα τα κριτήρια σε όλα τα indices, αυτό επιστρέφεται ως αποτέλεσμα. Το Datastore τότε ξεκινάει αναζήτηση για ένα νέο υποψήφιο entity, χρησιμοποιώντας όμως το κλειδί του προηγούμενου υποψηφίου ως το μικρότερο αποδεκτό κλειδί.



Σχήμα 15: Σχηματική παρουσίαση αλγορίθμου για την περίπτωση πολλαπλών φίλτρων ισότητας

Σχετικά με τα indices, μένει να αναφέρουμε ακόμα δυο ιδιομορφίες που παρουσιάζονται στο Datastore του AppEngine. Η πρώτη από αυτές σχετίζεται με τα properties ενός entity στα οποία δεν έχει ανατεθεί κάποια τιμή. Το App Engine δεν υποστηρίζει την έννοια του “κενού πεδίου”. Από τη στιγμή που το AppEngine δεν επιβάλλει κάποιο ενιαίο σχήμα σε όλα τα entities, αν κάποιο entity δεν χρειάζεται να έχει τιμή σε κάποιο property, το μόνο που μπορεί να κάνει είναι να μην περιλαμβάνει καν στις ιδιότητές του το property αυτό. Αν όμως ένα entity δεν έχει τιμή για κάποιο συγκεκριμένο property, δεν περιλαμβάνεται και στο αντίστοιχο index, οπότε δεν μπορεί και να επιστραφεί με

ερώτημα πάνω σε αυτό το property.

Η δεύτερη ιδιομορφία είναι ότι το App Engine δεν δημιουργεί indices για κάποιους συγκεκριμένους τύπους δεδομένων. Το Datastore υποστηρίζει διαφορετικούς τύπους για την αποθήκευση μικρών και μεγάλων strings. Λέγοντας μικρών, εννοούμε strings μικρότερα των 500 bytes, ενώ μεγάλων εννοούμε μεγαλύτερα των 500 bytes. Για τύπους που χρησιμοποιούνται για την αποθήκευση μεγάλων strings δεν δημιουργούνται indices.

5.3.3) Δοσοληψίες

Στις web εφαρμογές πολλοί χρήστες αποκτούν πρόσβαση κι ενημερώνουν δεδομένα ταυτόχρονα. Συχνά, χρειάζεται πολλοί χρήστες να διαβάσουν ή να γράψουν την ίδια μονάδα δεδομένων την ίδια χρονική στιγμή. Αυτό απαιτεί ένα σύστημα δεδομένων που να μπορεί να δίνει κάποιες βεβαιώσεις ότι ταυτόχρονες λειτουργίες δεν θα αλλοιώνουν την εικόνα που έχει οποιοσδήποτε χρήστης για τα δεδομένα. Τα περισσότερα συστήματα δεδομένων εγγυώνται ότι μια μοναδική λειτουργία σε μια μονάδα δεδομένων διατηρεί την ακεραιότητα των δεδομένων, συνήθως δρομολογώντας τις εργασίες που δρουν σε μια μονάδα δεδομένων, έτσι ώστε να εκτελούνται μια κάθε φορά.

Πολλές εφαρμογές χρειάζονται παρόμοιες εγγυήσεις ακεραιότητας δεδομένων, όταν εκτελείται ένα σύνολο πολλών λειτουργιών, πιθανώς πάνω σε πολλές μονάδες δεδομένων. Ένα τέτοιο σύνολο λειτουργιών καλείται δοσοληψία. Ένα σύστημα δεδομένων που υποστηρίζει δοσοληψίες εγγυάται ότι αν η δοσοληψία ολοκληρωθεί επιτυχημένα, όλες οι λειτουργίες της θα εκτελεστούν εξ' ολοκλήρου. Αν οποιοδήποτε βήμα της δοσοληψίας αποτύχει, τότε καμιά από τις επιδράσεις της δεν θα εφαρμοστεί στα δεδομένα. Τα δεδομένα παραμένουν σε συνεπή κατάσταση πριν και μετά τη δοσοληψία, ακόμα κι αν άλλες διεργασίες επιχειρήσουν να τροποποιήσουν τα δεδομένα την ίδια χρονική στιγμή.

Μια κλιμακώσιμη web εφαρμογή έχει αρκετές απαιτήσεις, που έρχονται σε αντίθεση με τις δοσοληψίες. Για παράδειγμα, μια εφαρμογή χρειάζεται γρήγορη πρόσβαση στα δεδομένα, ανεξάρτητα από το πόσα δεδομένα υπάρχουν στο σύστημα και πώς αυτά είναι κατανεμημένα στους διάφορους servers. Όσο περισσότερο χρόνο χρειάζεται μια δοσοληψία για να ολοκληρωθεί,τόσο χρόνο χρειάζεται να περιμένουν οι άλλες διεργασίες που θέλουν πρόσβαση στα δεδομένα που έχουν δεσμευτεί από τη δοσοληψία. Το πλήθος των δοσοληψιών, που μπορούν να ολοκληρωθούν σε μια χρονική περίοδο καλείται throughput. Για τις web εφαρμογές, το υψηλό throughput είναι σημαντικό.

Κάθε entity ανήκει σε ένα entity group. Το entity group είναι ένα σύνολο από ένα ή περισσότερα

entities, που μπορούμε να διαχειριστούμε σε μια δοσοληψία. Οι entity group σχέσεις λένε στο AppEngine να αποθηκεύσει ορισμένα entities στο ίδιο μέρος του κατανεμημένου δικτύου.

Όταν κατά τη δημιουργία ενός entity, του αναθέτουμε ένα γονιό, αυτόματα το νέο entity μπαίνει στο entity group του γονιού του. Έχουμε δει ότι από τη συσχέτιση entities, μέσω σχέσεων γονιού-παιδιού, προκύπτει μια δενδρική δομή. Κάθε τέτοιο δέντρο, με ρίζα κάποιο root entity, αποτελεί επομένως ένα entity group. Ένα root entity χωρίς παιδιά αποτελεί ένα entity group από μόνο του.

Όταν δημιουργούμε, ενημερώνουμε ή διαγράφουμε ένα entity, οι αλλαγές συμβαίνουν σε μια δοσοληψία: ή όλες οι αλλαγές θα ολοκληρωθούν επιτυχημένα ή καμία. Αν αλλάξουμε δυο properties ενός entity και το αποθηκεύσουμε, κάθε request handler, που χρησιμοποιεί το entity θα δει και τις δυο αλλαγές. Σε κανένα σημείο κατά τη διάρκεια της αποθήκευσης δεν θα είναι ορατή η καινούρια τιμή μόνο του ενός property, ενώ του άλλου όχι. Κι αν το update αποτύχει, το entity παραμένει όπως ήταν πριν την αποθήκευση. Λέμε λοιπόν ότι η ενημέρωση του entity είναι ατομική (atomic).

Συχνά είναι χρήσιμο να ενημερώνονται ατομικά πολλά entities, έτσι ώστε η όψη των δεδομένων που έχει οποιαδήποτε διεργασία να είναι συνεπής με τις τελευταίες αλλαγές. Για να το πετύχει αυτό με έναν κλιμακώσιμο τρόπο, το AppEngine πρέπει να γνωρίζει εκ των προτέρων ποια entities θα συμπεριλαμβάνονται σε μια διεργασία. Τα entities αυτά αποθηκεύονται κι ενημερώνονται μαζί, ώστε να μπορεί το Datastore να τα κρατάει συνεπή και παρ' όλα αυτά να έχει γρήγορη πρόσβαση σε αυτά. Για να υποδείξουμε στο AppEngine ποια entities συμπεριλαμβάνονται στην ίδια δοσοληψία, χρησιμοποιούμε τα entity groups.

Κάθε entity ανήκει υποχρεωτικά σε ένα και μόνο entity group. Αναθέτουμε το entity σε ένα group κατά τη δημιουργία του. Από τη στιγμή που το entity έχει δημιουργηθεί, δεν μπορεί να αλλάξει group.

Το Datastore χρησιμοποιεί entity groups για να καθορίσει τι θα συμβεί όταν δύο διεργασίες επιχειρήσουν να ενημερώσουν δεδομένα στο entity group την ίδια χρονική στιγμή. Όταν συμβαίνει αυτό, η πρώτη ενημέρωση που ολοκληρώνεται “κερδίζει” και η άλλη ακυρώνεται. Το AppEngine ενημερώνει ποιανής διεργασίας η ενημέρωση ακυρώθηκε σηκώνοντας ένα exception. Στις περισσότερες περιπτώσεις η διεργασία που απέτυχε μπορεί να προσπαθήσει ξανά την ενημέρωση και να επιτύχει. Αλλά η εφαρμογή θα πρέπει να αποφασίζει αν και πότε θα πρέπει μια διεργασία να ξαναπροσπαθήσει, αφού ανάμεσα στις προσπάθειες μπορεί να έχουν αλλάξει σημαντικά δεδομένα.

Αυτός ο τρόπος διαχείρισης της ταυτόχρονης πρόσβασης σε δεδομένα είναι γνωστός ως optimistic concurrency control. Είναι “αισιόδοξος” υπό την έννοια, ότι η βάση δεδομένων επιχειρεί να εκτελέσει τις λειτουργίες που πρέπει, χωρίς πρώτα να ελέγξει αν κάποια άλλη διεργασία χρησιμοποιεί τα ίδια

δεδομένα. Ελέγχει απλά στο τέλος για συγκρούσεις κι ευελπιστεί ότι οι λειτουργίες θα πετύχουν.

Το optimistic concurrency control είναι καλή επιλογή για web εφαρμογές, αφού η ανάγνωση δεδομένων είναι γρήγορη κι επιτυγχάνει σχεδόν πάντα. Αν μια ενημέρωση αποτύχει λόγω ταυτόχρρονης πρόσβασης, συνήθως είναι εύκολο να την επιχειρήσουμε ξανά ή να επιστρέψουμε ένα μήνυμα λάθους στο χρήστη. Επίσης, συνήθως, στις περισσότερες web εφαρμογές σπάνια χρειάζεται διαφορετικοί χρήστες να ενημερώσουν τα ίδια δεδομένα, επομένως αποτυχίες εξαιτίας ταυτόχρονης πρόσβασης είναι σπάνιες.

Εκτός από τις ενημερώσεις, όπου η αξία των δοσοληψιών είναι προφανής, πολλές φορές είναι χρήσιμο και οι αναγνώσεις να γίνονται εντός δοσοληψιών. Η ανάγνωση πολλών entities σε μια δοσοληψία διασφαλίζει ότι τα entities είναι συνεπή μεταξύ τους. Όπως και με τις ενημερώσεις, τα entities που διαβάζουμε σε μια δοσοληψία πρέπει να είναι μέλη του ίδιου entity group.

Μια δοσοληψία που μόνο διαβάζει entities δεν αποτυγχάνει ποτέ λόγω συγχρονισμού(concurrency). Όπως όταν διαβάζουμε ένα μόνο entity, μια δοσοληψία ανάγνωσης-μόνο βλέπει τα δεδομένα όπως φαινόντουσαν στην αρχή της δοσοληψίας, ακόμα κι αν άλλες δεργασίες έχουν κάνει αλλαγές στα δεδομένα μετά το ξεκίνημα της δοσοληψίας και πριν αυτή ολοκληρωθεί.

Το Datastore μπορεί να το κάνει αυτό, επειδή θυμάται προηγούμενες εκδόσεις των entities, χρησιμοποιώντας timestamps που σχετίζονται με τα entity groups. Το Datastore σημειώνει την τρέχουσα ώρα στην αρχή κάθε λειτουργίας και δοσοληψίας κι έτσι καθορίζει ποια έκδοση των δεδομένων βλέπει η δοσοληψία. Αυτή η τακτική είναι μια μορφή optimistic concurrency control και λέγεται multiversion concurrency control. Ο μηχανισμός αυτός είναι εσωτερικός στο Datastore. Η εφαρμογή δεν μπορεί να βλέπει προηγούμενες εκδόσεις των δεδομένων ή τα timestamps.

5.4) Υπηρεσίες

Είχαμε αναφέρει ότι τα βασικά συστατικά μέρη του AppEngine είναι το runtime περιβάλλον, το Datastore και οι κλιμακώσιμες υπηρεσίες που παρέχει. Έχοντας περιγράψει τα δυο πρώτα, μας μένει να περιγράψουμε τις υπηρεσίες που παρέχει το AppEngine. Αυτές είναι οι: App Identity, Blobstore, Google Cloud Storage, Capabilities, Conversion, Channel, Images, Mail, Memcache, Multitenancy, Oauth, Prospective Search, Task Queues, URL Fetch, Users και XMPP. Στη συνέχεια θα δώσουμε μια

σύντομη περιγραφή για την κάθε υπηρεσία.

App Identity

Το App Identity επιτρέπει στην εφαρμογή μας να πιστοποιεί την ταυτότητά της σε άλλα εξωτερικά συστήματα.

Blobstore

Το Blobstore API επιτρέπει στην εφαρμογή μας να χρησιμοποιεί αντικείμενα δεδομένων, τα οποία ονομάζονται blobs, και είναι πολύ μεγαλύτερα σε μέγεθος από τα αντικείμενα του Datastore. Ένα blob δημιουργείται ανεβάζοντας ένα αρχείο μέσω ενός HTTP request. Συνήθως αυτό γίνεται με τον εξής τρόπο: η εφαρμογή παρουσιάζει μια φόρμα, η οποία έχει ένα πεδίο που καλεί το χρήστη να δώσει ένα αρχείο που θέλει να ανεβάσει. Μόλις η φόρμα υποβληθεί, το Blobstore δημιουργεί το blob από τα περιεχόμενα του αρχείου κι επιστρέφει μια αναφορά στο blob, η οποία χρησιμοποιείται στη συνέχεια για τη διαχείριση του blob.

Google Cloud Storage

Το Google Cloud Storage είναι μια ακόμα υπηρεσία αποθήκευσης που παρέχει το AppEngine και προσφέρει ορισμένα χαρακτηριστικά που δεν έχει το Datastore ή το Blobstore. Τα χαρακτηριστικά αυτά είναι:

- Access Control Lists (ACLs) για τη διαχείριση των δεδομένων
- υποστήριξη για χρήση της μεθόδου POST
- Oauth 2.0 authentication
- RESTful API

Το Google Cloud Storage είναι ακόμα σε πειραματικό επίπεδο.

Capabilities

Με το Capabilities API η εφαρμογή μας μπορεί να ανιχνεύει διακοπές ή προγραμματισμένες περιόδους μη διαθεσιμότητας ορισμένων χαρακτηριστικών της εφαρμογής μας και να απενεργοποιεί τα χαρακτηριστικά αυτά πριν επηρεάσουν τους χρήστες της.

Conversion

Το Conversion κάνει μετατροπές μεταξύ διάφορων τύπων αρχείων. Οι τύποι αρχείων που υποστηρίζει είναι: HTML, PDF, text και διάφορα formats για εικόνες, ενώ οι μετατροπές μπορούν να γίνονται τόσο σύγχρονα όσο και ασύγχρονα. Ακόμη, υποστηρίζει και την επιλογή να κάνει οπτική αναγνώριση χαρακτήρων (OCR). Το Conversion είναι επίσης σε πειραματικό στάδιο.

Channel

Το Channel API δημιουργεί μια μόνιμη σύνδεση μεταξύ της εφαρμογής μας και των Google servers, επιτρέποντας στην εφαρμογή μας να στέλνει real time μηνύματα, χωρίς τη χρήση polling, σε JavaScript clients. Το χαρακτηριστικό αυτό είναι ιδιαίτερα χρήσιμο για εφαρμογές που θέλουν να ενημερώνουν άμεσα τους χρήστες τους σχετικά με νέες πληροφορίες ή για εφαρμογές όπου ένας χρήστης επιθυμεί να στέλνει real time πληροφορίες σε άλλους χρήστες. Χαρακτηριστικά παραδείγματα τέτοιων εφαρμογών είναι τα παιχνίδια που απαιτούν πολλούς παίκτες και τα chat rooms.

Images

Με το Images, το AppEngine παρέχει τη δυνατότητα διαχείρισης εικόνων. Με την υπηρεσία αυτή, μπορούμε να αλλάζουμε το μέγεθος, να περιστρέφουμε, να κόβουμε και να κάνουμε flip εικόνες.

Mail

Με την υπηρεσία αυτή, οι εφαρμογές του AppEngine μπορούν να στέλνουν emails εκ μέρους των διαχειριστών της εφαρμογής καθώς κι εκ μέρους χρηστών των Google Accounts. Το Mail δίνει ακόμα τη δυνατότητα στις AppEngine εφαρμογές να λαμβάνουν emails σε διάφορες διευθύνσεις. Οι εφαρμογές, χρησιμοποιώντας το Mail, λαμβάνουν και στέλνουν emails με τη μορφή HTTP requests.

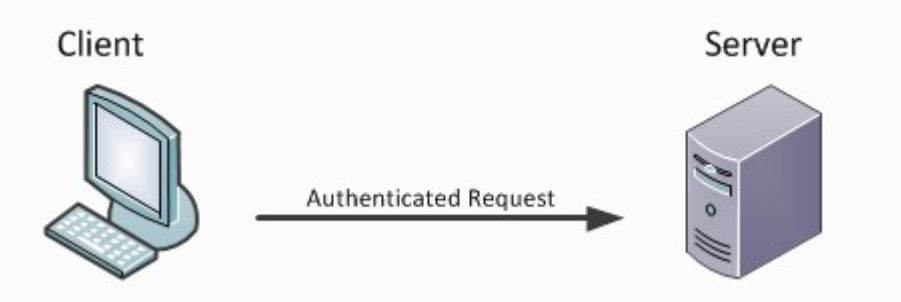
Memcache

Οι κλιμακώσιμες web εφαρμογές υψηλής επίδοσης, συχνά χρησιμοποιούν για ορισμένες εργασίες μια κατανεμημένη cache μνήμη επιπλέον ή στη θέση της μόνιμης robust αποθήκευσης. Για το σκοπό αυτό, το AppEngine έχει το Memcache.

Oauth

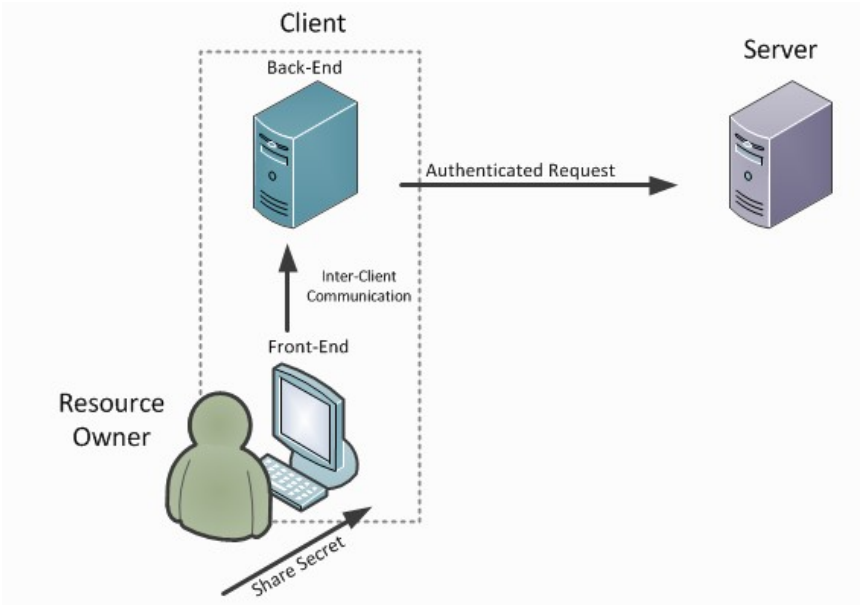
Το Oauth είναι ένα πρωτόκολλο πιστοποίησης. Για να συνδεθεί μια εφαρμογή στο Twitter, το

πρωτόκολλο πιστοποίησης που χρησιμοποιεί είναι το Oauth και για το λόγο αυτό θα το περιγράψουμε αναλυτικά. Στην κλασσική εκδοχή του Server-client μοντέλου, έχουμε έναν client ο οποίος θέλει να αποκτήσει πρόσβαση σε κάποια εφαρμογή που τρέχει στο server. Για να το κάνει αυτό στέλνει τα διαπιστευτήριά του (credentials) στο server μέσω ενός request. Το σενάριο αυτό παρουσιάζεται στην παρακάτω εικόνα.



Σχήμα 16: Κλασσική εκδοχή πιστοποίησης ενός client στο Server

Στην περίπτωση του Oauth εκτός από τον server και τον client έχουμε και έναν χρήστη ή όπως αλλιώς αναφέρεται στη βιβλιογραφία κάτοχος πόρων (resource owner). Στο σενάριο αυτό, ο client, θέλει να συνδεθεί στο server, αλλά θέλει να συνδεθεί εκ μέρους του resource owner. Έτσι, ο client δεν θα συνδεθεί χρησιμοποιώντας τα δικά του credentials, αλλά χρησιμοποιώντας τα credentials του resource owner. Το νέο σχήμα φαίνεται αμέσως παρακάτω:



Σχήμα 17: Πιστοποίηση με χρήση OAuth

Prospective Search

Το Prospective Search επιτρέπει σε μια εφαρμογή να δηλώσει ένα μεγάλο σύνολο ερωτημάτων που αντιστοιχίζονται αυτόματα σε ένα stream από έγγραφα εισόδου. Κάθε φορά που παρουσιάζεται ένα νέο έγγραφο, το prospective search επιστρέφει τα IDs όλων των δηλωμένων ερωτημάτων που αντιστοιχούν στο έγγραφο αυτό. Το Prospective Search είναι ακόμα σε πειραματικό στάδιο και είναι ιδιαίτερα χρήσιμο για εφαρμογές που επεξεργάζονται streaming data sources.

Task Queue

Με τα Task Queues, οι AppEngine εφαρμογές μπορούν να κάνουν εργασίες που ξεκινάν με το request κάποιου χρήστη, αλλά μπορούν να ολοκληρωθούν έξω από τα πλαίσια του συγκεκριμένου request. Αν κάποια εφαρμογή χρειάζεται να εκτελέσει κάποια εργασία στο background, χρειάζεται το Task Queue API για να την οργανώσει σε μικρές διακρτιές μονάδες εργασίας που καλούνται tasks. Στη συνέχεια η εφαρμογή βάζει τα tasks σε μια ή περισσότερες ουρές προς εκτέλεση. Η οργάνωση των tasks σε ουρές εξαρτάται από το configuration της εφαρμογής.

URL Fetch

Όπως έχουμε αναφέρει, μια AppEngine εφαρμογή ζει στο δικό της sandbox. Ένας από τους

περιορισμούς που επιβάλλει το sandbox είναι ότι η εφαρμογή δεν μπορεί να δημιουργεί αυθαίρετα δικτυακές συνδέσεις με άλλες web τοποθεσίες. Οι AppEngine εφαρμογές χρησιμοποιούν το URL Fetch για να κάνουν HTTP και HTTPS requests και να συνδέονται σε άλλες τοποθεσίες.

Users

Με το Users API, οι AppEngine εφαρμογές μπορούν να πιστοποιούν χρήστες, οι οποίοι έχουν λογαριασμό στο Google Accounts, λογαριασμούς σε κάποιο Google App domain ή OpenID αναγνωριστικά. Επίσης, η εφαρμογή μπορεί να καταλαβαίνει αν κάποιος χρήστης έχει κάνει sign-in ή όχι, όπως και το αν είναι συνδεδεμένος ως administrator ή ως απλός χρήστης. Αυτό διευκολύνει αρκετά την ανάπτυξη περιοχών στις οποίες έχουν πρόσβαση μόνο οι διαχειριστές της εφαρμογής.

XMPP

Με το XMPP API, οι εφαρμογές του AppEngine μπορούν να στέλνουν και να λαμβάνουν στιγμιαία μηνύματα από και προς χρήστες XMPP συμβατών υπηρεσιών, όπως το Google Talk.

6) TweetFinder11

6.1) Επισκόπηση

Έχοντας περιγράψει όλο το θεωρητικό υπόβαθρο, αλλά και τις τεχνολογίες που χρησιμοποιήσαμε, βρισκόμαστε πλέον σε θέση να περιγράψουμε την εφαρμογή μας, το TweetFinder11. Το TweetFinder11 είναι μια web εφαρμογή, η οποία κάνει αναζήτηση σε ένα σύνολο από tweets, τα οποία έχουν πρώτα δεικτοδοτηθεί με βάση διάφορα χαρακτηριστικά τους, ώστε να επιταχύνεται η αναζήτηση σε αυτά και στη συνέχεια έχουν αποθηκευτεί σε μια κατανεμημένη βάση δεδομένων.

Ως πλατφόρμα για την ανάπτυξη του TweetFinder11 χρησιμοποιήσαμε, όπως έχουμε ήδη αναφέρει, το Google App Engine κι επομένως, ως βάση δεδομένων το Datastore. Η χρήση του App Engine για την ανάπτυξη της εφαρμογής, μας διευκόλυνε αρκετά καθώς μας προσέφερε όλα τα πλεονεκτήματα ενός PaaS.

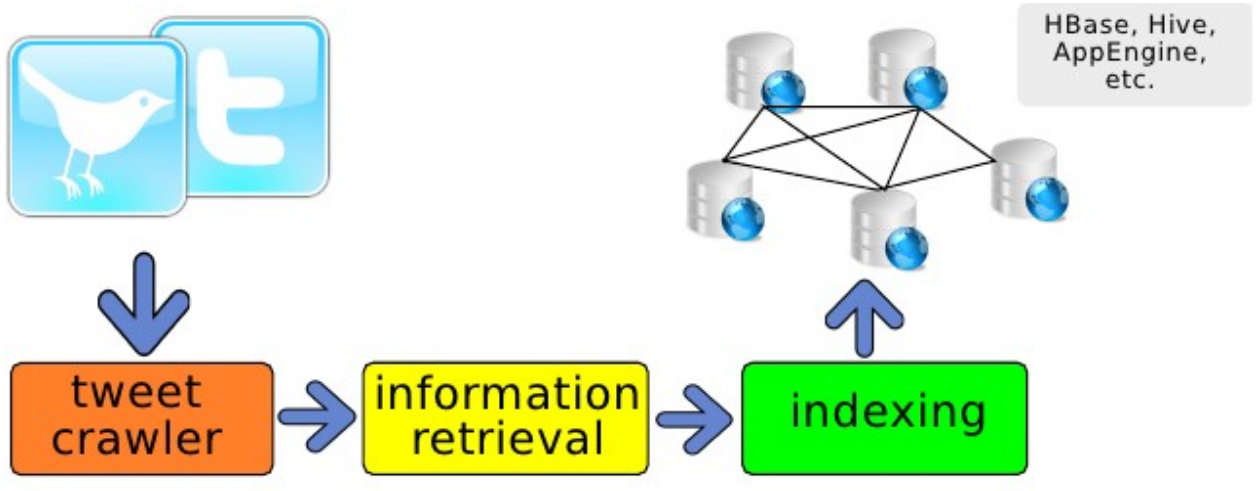
Στη συνέχεια του κεφαλαίου, θα περιγράψουμε αναλυτικά την αρχιτεκτονική της εφαρμογής μας, το σχήμα δεδομένων που χρησιμοποιήσαμε για την αποθήκευση των tweets στη βάση, καθώς και τα χαρακτηριστικά, με βάση τα οποία έγινε η δεικτοδότηση.

6.2) Αρχιτεκτονική Εφαρμογής

Όπως εξηγήσαμε στο σχετικό με το Twitter κεφάλαιο της παρούσας εργασίας, για IR εφαρμογές όπως η δική μας είναι σημαντικό να μπορούμε να λαμβάνουμε μεγάλο όγκο από real-time δεδομένα. Είχαμε επομένως καταλήξει στη χρήση του Streaming API του Twitter για τις ανάγκες της εφαρμογής μας και είχαμε δικαιολογήσει αναλυτικά την επιλογή μας αυτή.

Η αρχική ιδέα για την αρχιτεκτονική της εφαρμογής μας ήταν να δημιουργείται μια σύνδεση μεταξύ του Twitter κι ενός server του App Engine, στον οποίο τρέχει κάποιος request handler της εφαρμογής μας, και μέσω αυτής της σύνδεσης, να λαμβάνουμε το stream με τα tweets και στη συνέχεια, αφού τα επεξεργαστούμε και τα δεικτοδοτήσουμε με βάση διάφορα χαρακτηριστικά τους, να τα αποθηκεύουμε στο Datastore. Από τη στιγμή που τα δεδομένα βρίσκονται στο Datastore, οποιοσδήποτε χρήστης της εφαρμογής, θα μπορεί να υποβάλει ερωτήματα στη βάση, αρκεί να

επισκεφτεί την ιστοσελίδα της εφαρμογής, όπου βρίσκεται η φόρμα των ερωτημάτων.



Σχήμα 18: Βασική ιδέα της αρχιτεκτονικής του συστήματος

Παρ' όλα αυτά, η ιδέα αυτή δεν ήταν δυνατόν να υλοποιηθεί εξ' αιτίας των περιορισμών που επιβάλλει το App Engine. Όπως έχουμε αναφέρει, ένας request handler του App Engine, πρέπει να επιστρέφει κάποια απάντηση σε αυτόν που τον κάλεσε το πολύ εντός 30 δευτερολέπτων. Το Streaming API του Twitter όμως, λειτουργεί διαφορετικά. Εγκαθιστά μια long-lived σύνδεση για τη μεταφορά των δεδομένων. Η λογική αυτή όμως είναι εντελώς αντίθετη με αυτή των γρήγορα αποκρινόμενων request handlers του App Engine.

Εκτός από τα instances των App Engine εφαρμογών που περιγράψαμε στο κεφάλαιο για το App Engine, το App Engine παρέχει και μια άλλη κατηγορία, την οποία ονομάζει Backends. Τα backend instances δεν απαιτούν κάποιο συγκεκριμένο χρονικό διάστημα μέσα στο οποίο θα πρέπει να απαντήσουν οι request handlers, διαθέτουν μεγαλύτερη μνήμη και προσφέρουν υψηλότερα όρια σε ταχύτητες CPU και requests στο Datastore. Αν και σε πρώτη φάση μοιάζουν να είναι το κατάλληλο περιβάλλον για τη χρήση του Streaming API, σύμφωνα με το [19], το Streaming API του Twitter δεν δουλεύει ούτε στα Backends.

Έτσι, αφού συνειδητοποιήσαμε την ανάγκη χρήσης του Streaming API, αλλά και την αδυναμία ενσωμάτωσής του στο App Engine καταλήξαμε στην ακόλουθη σχεδιαστική απόφαση. Θα χρησιμοποιούμε το Streaming API για τη συλλογή δεδομένων από το Twitter, αλλά αυτό δεν θα γίνεται μέσω App Engine. Η συλλογή των δεδομένων θα γίνεται σε κάποιους τοπικούς, δικούς μας υπολογιστές και τα δεδομένα θα αποθηκεύονται σε κάποια σχεσιακή βάση δεδομένων, που θα έχουμε

στήσει στους υπολογιστές αυτούς. Στη συνέχεια, ανά τακτικά χρονικά διαστήματα, θα ανεβάζουμε τα δεδομένα που έχουμε συλλέξει στο Datastore του App Engine.

Για τη μεταφορά των δεδομένων από κάποιο τοπικό μηχάνημα στο Datastore, χρησιμοποιήσαμε το Remote API που προσφέρει το App Engine. Για τη χρήση του Remote API αρκεί να συμπεριλάβουμε τα ακόλουθα στοιχεία στο web.xml της εφαρμογής μας.

```
<servlet>
<display-name>Remote API Servlet</display-name>
<servlet-name>RemoteApiServlet</servlet-name>
<servlet-class>com.google.apphosting.utils.remoteapi.RemoteApiServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>RemoteApiServlet</servlet-name>
<url-pattern>/remote_api</url-pattern>
</servlet-mapping>
```

Ουσιαστικά δηλαδή, με τη χρήση του Remote API καλούμε κάποιον request handler, ο οποίος μαζί με το request λαμβάνει και τα δεδομένα προς αποθήκευση, καθώς και το σχήμα τους κι αναλαμβάνει να τα αποθηκεύσει το Datastore.

Το Remote API υποστηρίζει μόνο το low-level API του Datastore για την διαχείριση των δεδομένων. Αυτός ήταν κι ο ένας λόγος για τον οποίο επιλέξαμε το low-level API για την αναπαράσταση και διαχείριση των δεδομένων στην εφαρμογή μας. Ο άλλος λόγος είναι ότι με τη χρήση του low-level API μπορεί να επιτευχθεί καλύτερη απόδοση στις λειτουργίες που σχετίζονται με το Datastore.

6.3) Δεικτοδοτούμενα χαρακτηριστικά

Στην ενότητα αυτή, θα αναφέρουμε τα χαρακτηριστικά που μας ενδιέφεραν ως προς την αναζήτηση ενός tweet στη βάση μας κι επομένως τα χαρακτηριστικά με βάση τα οποία δεικτοδοτήσαμε τα tweets .

Σε πρώτη φάση θέλουμε η εφαρμογή μας να μπορεί να βρει όλα τα tweets που μπορεί να βρει και το Advanced Twitter Search. Επομένως, θέλουμε να μπορούμε να κάνουμε αναζήτηση με βάση τα ίδια κριτήρια που κάνει και το Advanced Twitter Search. Τα κριτήρια αυτά είναι λέξεις-κλειδιά, hashtags, ονόματα χρηστών και τοποθεσίες χρηστών. Εκτός από αυτά, το Twitter API μας δίνει τη δυνατότητα

ανάκτησης και άλλων πληροφοριών σχετικά με τα tweets, οι οποίες θα ήταν ενδιαφέρουσες για κριτήρια αναζήτησης κι επομένως για δεικτοδότηση. Οι πληροφορίες αυτές είναι η ώρα που δημοσιεύτηκε το tweet (timestamp), ο αριθμός των followers και ο αριθμός των friends ενός χρήστη.

Πέραν αυτών των στοιχείων όμως, στην εφαρμογή μας δεικτοδοτούμε και ορισμένα ακόμα. Όπως αναφέραμε στο κεφάλαιο για το IR, στην εφαρμογή μας, μας ενδιαφέρει να κρατάμε και μερικές ακόμα πληροφορίες, οι οποίες άπτονται άμεσα με τις γραμματικές και συντακτικές ιδιότητες των tweets. Οι πληροφορίες αυτές είναι το τι μέρος του λόγου (ουσιαστικό,ρήμα κλπ.) είναι κάθε λέξη που περιλαμβάνεται σε ένα tweet, καθώς επίσης και οι συντακτικές εξαρτήσεις που υπάρχουν σε ένα tweet.

6.4) Σχήμα δεδομένων

Παραπάνω είπαμε ότι για την αποθήκευση των δεδομένων μας χρησιμοποιήσαμε πρώτα μια σχεσιακή βάση, έτσι ώστε να καταστεί δυνατή η χρήση του Streaming API και στη συνέχεια τα δεδομένα αυτά τα ανεβάσαμε στο Datastore του App Engine. Στην ενότητα αυτή θα περιγράψουμε και τα δύο αυτά σχήματα αποθήκευσης. Θα περιγράψουμε εν συντομία το σχεσιακό σχήμα δεδομένων που χρησιμοποιήθηκε για την αποθήκευση στο τοπικό σύστημα και στη συνέχεια θα περιγράψουμε πιο αναλυτικά το σχήμα που χρησιμοποιήθηκε στο Datastore και που είναι το σχήμα με το οποίο δουλεύει η εφαρμογή μας..

Θα ξεκινήσουμε με το σχήμα της σχεσιακής βάσης. Η σχεσιακή μας βάση δεδομένων περιλάμβανε τους ακόλουθους πίνακες:

Users				
user_id	screen_name	Location	followersnm	friendsnm

Πίνακας 9: Δομή Users στη σχεσιακή βάση

Tweets					
user_id	status_id	timestamp	retweets	content	tid

Πίνακας 10: Δομή Tweets στη σχεσιακή βάση

Keywords	
status_id	word

Πίνακας 11: Δομή Keywords στη σχεσιακή βάση

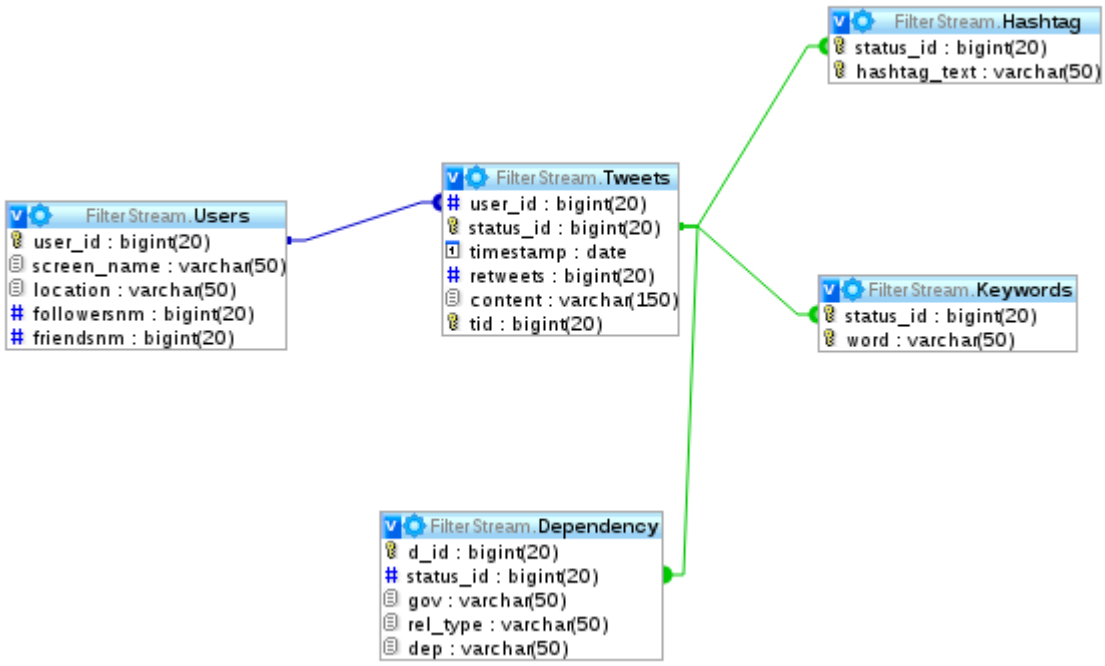
Hashtags	
status_id	hashtag_text

Πίνακας 12: Δομή Hashtags στη σχεσιακή βάση

Dependency				
d_id	status_id	gov	rel_type	dep

Πίνακας 13: Δομή Dependency στη σχεσιακή βάση

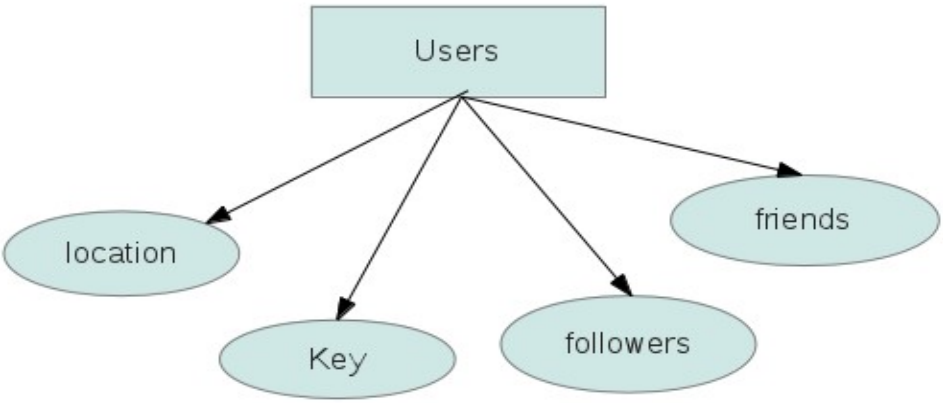
Ο τρόπος με τον οποίον συσχετίζονται οι παραπάνω πίνακες στη βάση μας, φαίνεται στην εικόνα που ακολουθεί:



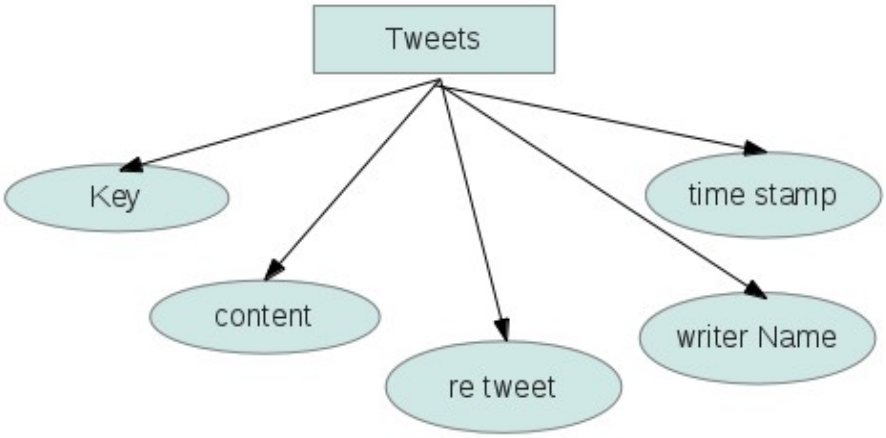
Σχήμα 19: Σχήμα δεδομένων της SQL βάσης μας

Στη συνέχεια θα περιγράψουμε το σχήμα δεδομένων, που χρησιμοποιήσαμε για την αναπαράσταση των δεδομένων μας στο Datastore του App Engine. Πρώτα θα παρουσιάσουμε τα είδη από entities που χτησιμοποιήσαμε και στη συνέχεια, τον τρόπο που αυτά συνδέονται μεταξύ τους, εξηγώντας παράλληλα την οποιαδήποτε σχεδιαστική μας απόφαση.

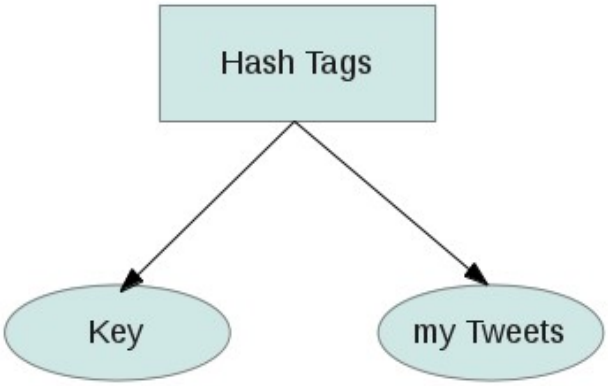
Δεδομένου ότι τα δεδομένα που θα αποθηκεύσουμε στο Datastore είναι αρχικά αποθηκευμένα σε μια σχεσιακή βάση, θεωρήσαμε ότι για την διευκόλυνσή μας, το μοντέλο δεδομένων που θα χρησιμοποιήσουμε για το Datastore θα πρέπει να είναι όσο πιο κοντά γίνεται σε αυτό της τοπικής μας βάσης. Έτσι, κατ' αντιστοιχία με τους πίνακες Users, Tweets, Keywords, Hashtags, Dependency της σχεσιακής βάσης, ορίσαμε τα αντίστοιχα ομώνυμα kinds. Εκτός από αυτά τα kinds, στο μοντέλο μας χρησιμοποιήσαμε τρία ακόμα. Αυτά είναι τα: DummyRoot,HelpHash και HelpKeywords. Κάθε entity, οποιουδήποτε εκ των παραπάνω kinds, έχει αποθηκευμένα ως properties, όλα τα στοιχεία που είναι αποθηκευμένα στα πεδία του αντίστοιχου σχεσιακού πίνακα. Ακολουθούν μερικά διαγράμματα, που αναπαριστούν τη δομή που έχει ένα entity καθενός από τα παραπάνω kinds.



Σχήμα 20: Δομή Users entity



Σχήμα 21: Δομή Tweets entity



Σχήμα 22: Αρχική ιδέα για δομή HashTags entities

Προτού συνεχίσουμε με την παρουσίαση του σχήματος και των υπόλοιπων kinds, θα θέλαμε να κάνουμε μερικά σχόλια. Παρατηρούμε κατ' αρχάς, ότι στους Users δεν περιλαμβάνεται κάποια ιδιότητα σχετική με το όνομα χρήστη, ενώ γνωρίζουμε ότι σαν πληροφορία μας είναι απαραίτητη κι ενώ υπάρχει ως πεδίο στη σχεσιακή μας βάση. Ο λόγος που γίνεται αυτό είναι ότι το όνομα χρήστη χρησιμοποιείται στο κλειδί του entity. Το Twitter μας εγγυάται ότι το screen name ενός χρήστη είναι μοναδικό, οπότε και μπορούμε να το χρησιμοποιήσουμε ως κλειδί. Έτσι, κάθε φορά που χρειάζεται να εισάγουμε κάποιο νέο χρήστη, δημιουργούμε ένα νέο entity με κλειδί το όνομά του. Επίσης, το όνομα

του χρήστη είναι κι ένα από τα κριτήρια με βάση τα οποία κάνουμε αναζήτηση. Αν γνωρίζουμε το κλειδί ενός entity, μπορούμε να το φέρουμε πολύ γρήγορα από το Datastore, χωρίς να χρειάζεται να δημιουργήσουμε κάποιο ερώτημα προς τη βάση. Με αυτό τον τρόπο επιταχύνεται αρκετά η αναζήτηση ενός χρήστη με βάση το όνομά του.

Στα Tweets τώρα, παρατηρούμε ότι υπάρχει ένα property το οποίο ονομάζεται writerName. Το property αυτό μας δίνει το όνομα του χρήστη, που έγραψε το συγκεκριμένο tweet. Αν και όπως θα δούμε στη συνέχεια, Users και Tweets συσχετίζονται και την πληροφορία αυτή θα μπορούσαμε να την ανακτήσουμε μέσω ενός ερωτήματος, χωρίς να χρειαστεί να την αποθηκεύσουμε ξανά, επειδή στην παρουσίαση των αποτελεσμάτων ενός οποιουδήποτε ερωτήματος, δίνουμε και το όνομα χρήστη εκτός από το tweet, θέλαμε να αποφύγουμε την εκτέλεση αρκετών επιπλέον ερωτημάτων. Έτσι, με την ανάκτηση ενός tweet, έχουμε αμέσως διαθέσιμη και την πληροφορία αυτή.

Στο σημείο αυτό θα αναφέρουμε και πώς συσχετίζονται τα Users entities με τα Tweets entities. Ένας χρήστης μπορεί να δημοσιεύσει πολλά tweets. Επομένως, ακολουθώντας τον τρόπο σύνδεσης των entities στο Datastore, θα αναπαραστήσουμε τη σχέση τους με ένα δέντρο όπου το Users entity θα είναι ο γονιός και τα Tweets entities τα παιδιά. Άρα, κάθε Tweets entity θα τοποθετείται κατά τη δημιουργία του κάτω από το Users entity, που αναπαριστά τον χρήστη που το δημοσίευσε.

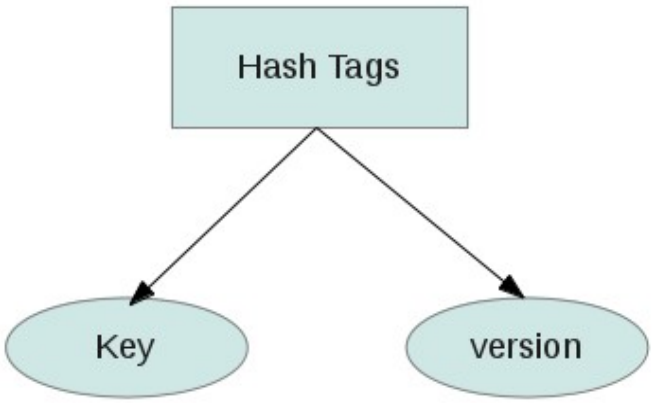
Στη συνέχεια θα ασχοληθούμε με την αναπαράσταση των hashtags. Για την γρηγορότερη ανάκτηση ενός tweet, με βάση ένα hashtag, για το κάθε διαφορετικό hashtag δημιουργείται ένα νέο entity. Το entity αυτό έχει ένα property, το οποίο έχει τη δυνατότητα να λαμβάνει πολλές τιμές (έχουμε αναφέρει ότι το App Engine υποστηρίζει multi valued properties), και στο οποίο αποθηκεύουμε τα κλειδιά όλων των tweets, στα οποία εμφανίζεται το hashtag αυτό. Αφού για κάθε διαφορετικό hashtag δημιουργείται νέο entity, μπορούμε να χρησιμοποιήσουμε ως κλειδί του entity αυτού το ίδιο το hashtag. Έτσι, η αναζήτηση με βάση το hashtag επιταχύνεται σημαντικά. Ο τρόπος επομένως με τον οποίο κάνουμε αναζήτηση με βάση ένα hashtag είναι ο εξής: Αφού το hashtag είναι και κλειδί, φέρνουμε άμεσα και γρήγορα από το Datastore το Hashtags entity που μας ενδιαφέρει. Στο entity αυτό υπάρχουν αποθηκευμένα τα κλειδιά όλων των tweets στα οποία εμφανίζεται το εν λόγω hashtag. Έτσι, με χρήση αυτών των κλειδιών, στη συνέχεια ανακτούμε τα tweets που μας ενδιαφέρουν. Αν δεν είχαμε δεικτοδοτήσει τα tweets με βάση τα hashtags με αυτόν τον τρόπο, αλλά είχαμε αποθηκεύσει το κάθε hashtag στο tweet που εμφανιζόταν, για να ανακτήσουμε κάποιο tweet με βάση ένα hashtag, θα έπρεπε να διατρέξουμε σειριακά όλα τα tweets.

Ακόμη και αυτό το σχήμα για τα hashtags όμως έχει ένα πρόβλημα. Όπως είδαμε, ένα Hashtags

entity έχει ένα multi valued property, το myTweets. Σε όσα πιο πολλά tweets εμφανίζεται το συγκεκριμένο hashtag, τόσες πιο πολλές τιμές θα έχει το myTweets property κι επομένως τόσο πιο μεγάλο θα είναι το μέγεθος του entity σε bytes. Αυτό έρχεται σε σύγκρουση με δυο περιορισμούς που θέτει το App Engine. Ο πρώτος περιορισμός υπαγορεύει ότι ένα entity δεν μπορεί να είναι μεγαλύτερο του 1MB. Έτσι, δεν μπορούμε να έχουμε πολύ μεγάλα entities.

Στο κεφάλαιο για το App Engine, είχαμε πει ότι το Datastore δημιουργεί ένα index για κάθε property ενός entity. Στην περίπτωση των multi valued properties, το index που δημιουργείται έχει μια γραμμή για κάθε τιμή του property. Στην περίπτωση αυτή επομένως, μπορεί να έχουμε το πρόβλημα των exploding indexes και το index να γίνει πολύ μεγάλο. Για την αποφυγή αυτού του προβλήματος το App Engine επιβάλλει τον περιορισμό ότι κάθε index θα πρέπει να έχει το πολύ 5000 εγγραφές.

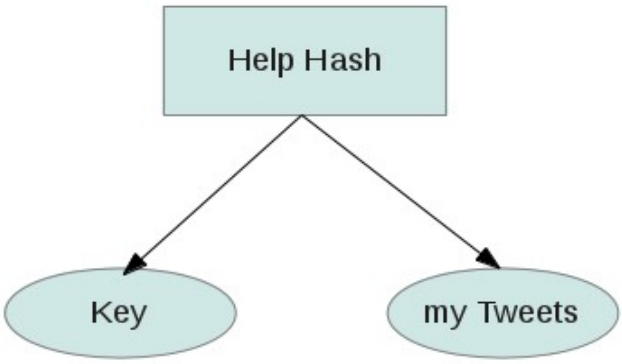
Για τους λόγους αυτούς λοιπόν, αναγκαστήκαμε να οργανώσουμε διαφορετικά τα Hashtags entities, διατηρώντας όμως τα πλεονεκτήματα του παραπάνω σχήματος. Αυτό που κάναμε είναι το εξής. Κατ' αρχάς αλλάξαμε το σχήμα των Hashtags entities και το κάναμε όπως φαίνεται στην εικόνα που ακολουθεί.



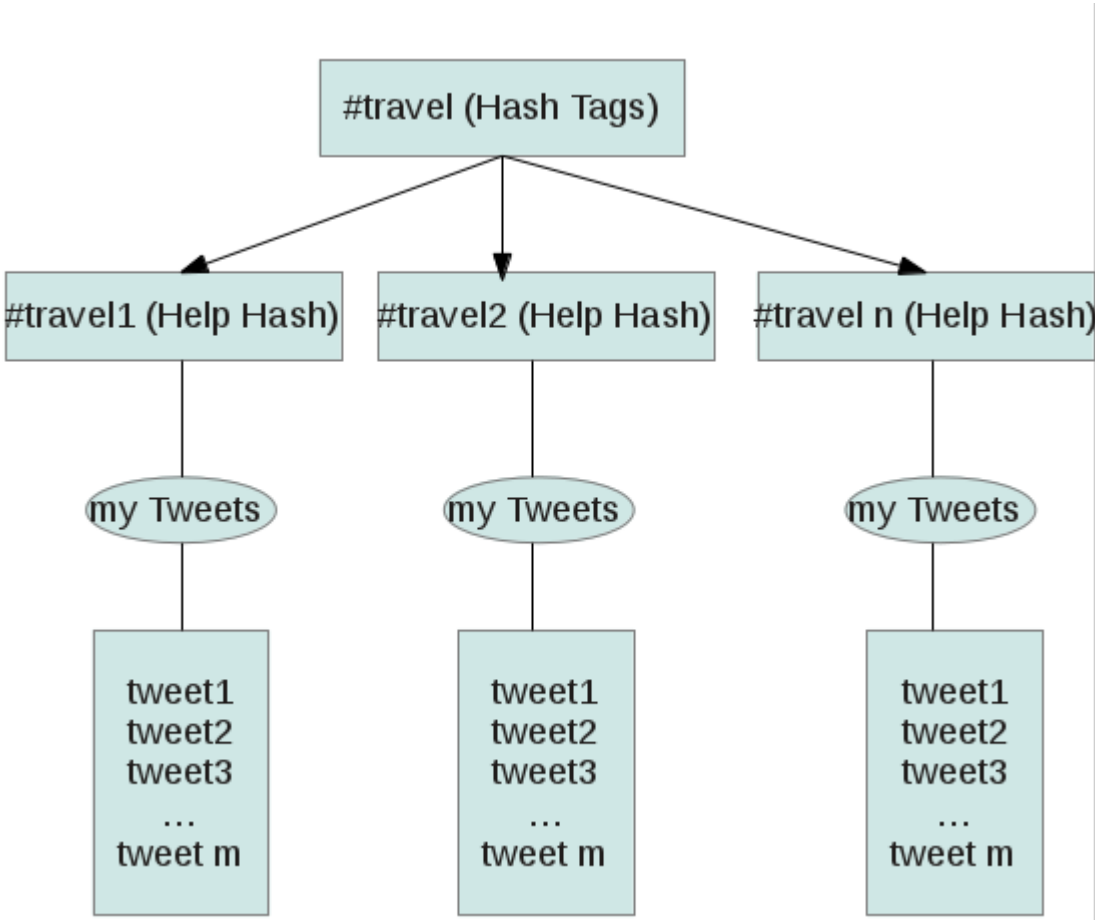
Σχήμα 23: Δομή HashTags entities

Τα HashTags εξακολουθούν να έχουν ως κλειδί το hashtag, έτσι ώστε η εύρεσή τους να γίνεται το ίδιο γρήγορα, αλλά έχουν κι ένα επιπλέον property, το version. Επίσης, δημιουργήσαμε ένα ακόμα είδος entity, το HelpHash. Κάθε HelpHash entity τοποθετείται ως παιδί ενός HashTags entity. Ένα HelpHash entity διατηρεί τη λίστα με τα tweets που δεικτοδοτεί το hashtag. Μόλις η λίστα αυτή μεγαλώσει αρκετά, δημιουργείται ακόμα ένα HelpHash entity που τοποθετείται ως παιδί του ίδιου HashTags entity κοκ. Το version property του HashTags entity χρησιμοποιείται στον σχηματισμό του

κλειδιού ενός νέου HelpHash entity που τοποθετείται κάτω από αυτό. Στις επόμενες εικόνες βλέπουμε το HelpHash entity καθώς κι ένα παράδειγμα που δείχνει τη σχέση HashTags-HelpHash entity.

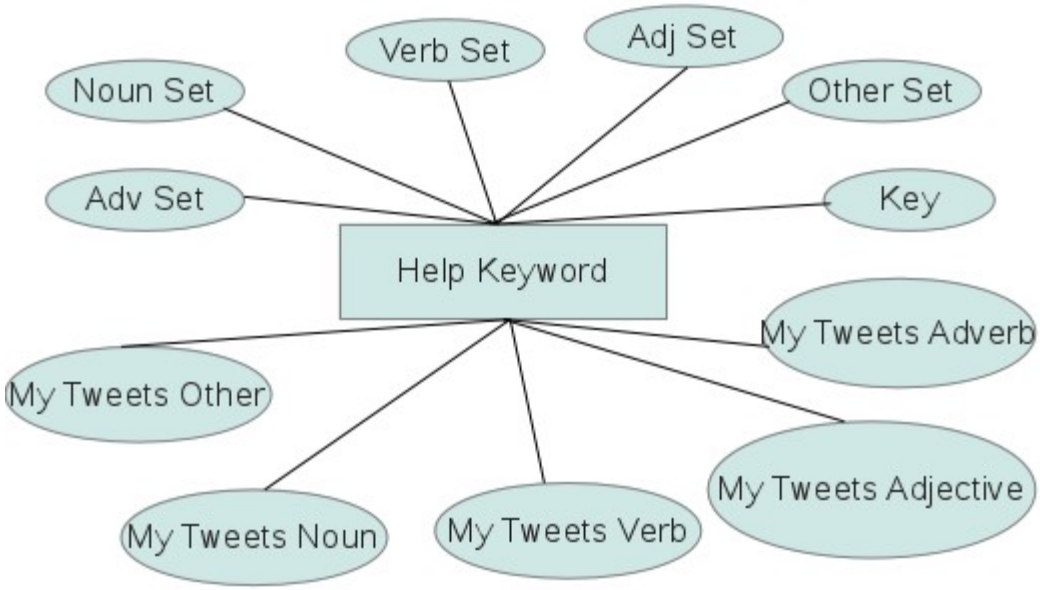


Σχήμα 24: Δομή HelpHash entities



Σχήμα 25: Παράδειγμα συσχέτισης HashTags και HelpHash entities

Την ίδια λογική με τα hashtags χρησιμοποιήσαμε και για τις λέξεις-κλειδιά των tweets, με βάση τις οποίες τα δεικτοδοτήσαμε. Για κάθε λέξη, δημιουργήσαμε ένα Keyword entity, του οποίου το κλειδί ήταν η ίδια η λέξη. Η αναζήτηση με βάση κάποιο keyword είναι αρκετά συχνή κι έτσι η εφαρμογή μας κερδίζει σε απόδοση. Και εδώ, στην περίπτωση των keywords δημιουργήσαμε ένα βοηθητικό είδος entity, το HelpKeyword, το οποίο διατηρεί τα κλειδιά των tweets, στα οποία εμφανίζεται η λέξη με τον ίδιο ακριβώς τρόπο, που γινόταν και στην περίπτωση των hashtags. Παρ' όλα αυτά, στο σημείο αυτό υπάρχει μια διαφοροποίηση με τα hashtags. Ήρθε η ώρα να αξιοποιήσουμε την πληροφορία που μας παρέχει ο POS Tagger και να δεικτοδοτήσουμε τα tweets με βάση αυτή. Η ίδια λέξη μπορεί να εμφανίζεται σε ένα tweet ως ρήμα και σε ένα άλλο ως ουσιαστικό. Εμείς επιθυμούμε να έχουμε τη δυνατότητα, να ανακτήσουμε μέσω ενός ερωτήματος μόνο τα tweets, στα οποία εμφανίζεται ως ουσιαστικό. Για το λόγο αυτό, χρησιμοποιούμε για τα HelpKeyword entities το σχήμα που φαίνεται αμέσως παρακάτω.

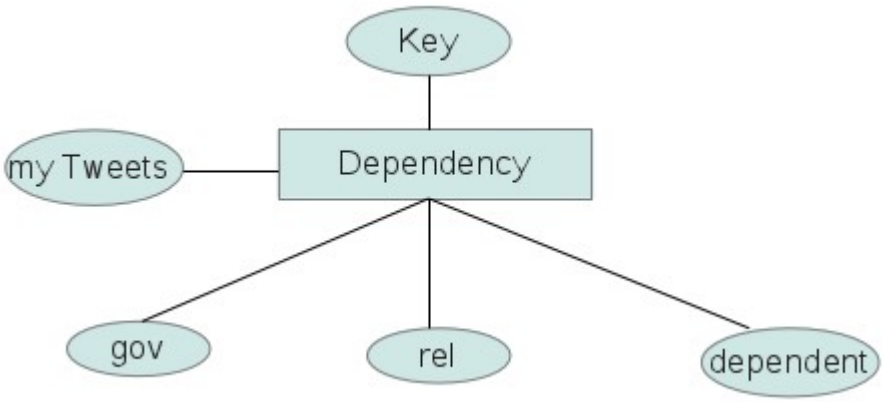


Σχήμα 26: Δομή HelpKeyword entities

Τα MyTweetsNoun, MyTweetsVerb, MyTweetsAdjective, MyTweetsAdverb και MyTweetsOther είναι multi valued properties και κρατάνε τα κλειδιά των tweets, στα οποία εμφανίζεται η λέξη ως ουσιαστικό, ρήμα, επίθετο, επίρρημα ή καποιο άλλο μέρος του λόγου αντίστοιχα. Ουσιαστικά δηλαδή, βλέπουμε ότι “σπάμε” το αντίστοιχο myTweets properties που υπήρχε για τα hashtags σε επιμέρους κατηγορίες.

Τα properties NounSet, VerbSet, AdvSet, AdjSet και OtherSet χρησιμοποιούνται βοηθητικά για να γνωρίζουμε ποια από τα multi valued properties περιέχουν τιμές και ποια είναι κενά. Για την ακρίβεια, όπως έχουμε πει, το App Engine δεν επιτρέπει την ύπαρξη κάποιου property στο οποίο να μην έχει ανατεθεί τιμή. Επίσης, δεν επιβάλλει κάποιο ενιαίο σχήμα για τα entities του ίδιου kind. Αυτό σημαίνει ότι αν μια λέξη δεν εμφανίζεται πουθενά ως επίρρημα, το HelpKeyword entity δεν θα έχει το MyTweetsAdverb property. Γι' αυτό και χρησιμοποιούμε τα NounSet, VerbSet, AdvSet, AdjSet και OtherSet. Για να γνωρίζουμε ποια properties υπάρχουν στο συγκεκριμένο entity και ποιες λειτουργίες είναι αποδεκτές.

Τέλος, χρειαζόμαστε να εκμεταλλευτούμε και τις πληροφορίες που μας προσφέρει ο Parser και να αποθηκεύσουμε τις συντακτικές εξαρτήσεις που υπάρχουν στα tweets. Για κάθε συντακτική εξάρτηση, δημιουργούμε ένα Dependency entity. Η δομή του φαίνεται στην παρακάτω εικόνα.



Σχήμα 27: Δομή Dependency entities

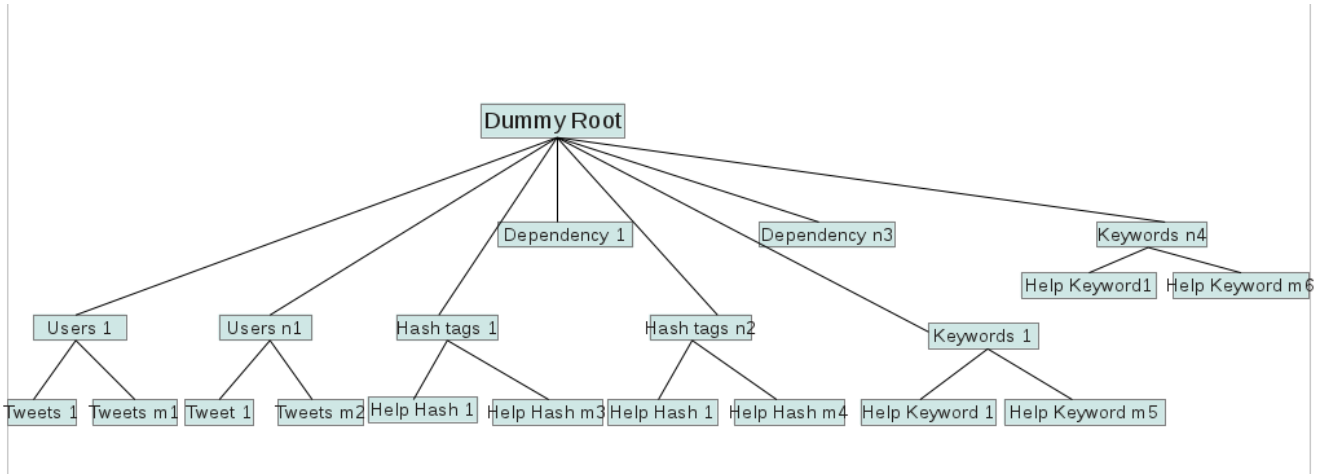
Στα properties gov,rel και dependent αποθηκεύουμε τον governor, το είδος της σχέσης και τον dependent της σχέσης αντίστοιχα, όπως τα ορίσαμε αυτά στο κεφάλαιο για το IR. Για να επιταχύνουμε την αναζήτηση με βάση τις συντακτικές εξαρτήσεις, ως κλειδί ενός Dependency θέσαμε το string: τιμή-governor/είδος σχέσης/τιμή-dependent.

Το myTweets property χρησιμοποιείται για την δεικτοδότηση των tweets. Ένα Dependency entity δεν εμφανίζεται σε tweets τόσο συχνά όσο ένα keyword ή ένα hashtag. Στο δικό μας dataset, η συχνότητα εμφάνισης των Dependency entities δεν δημιούργησε κάποιο πρόβλημα και για το λόγο αυτό δεν το χωρίσαμε σε δυο διαφορετικά entities όπως έγινε με τα keywords και τα hashtags. Αν σε κάποιο μεγαλύτερο dataset δημιουργείτο πρόβλημα εξαιτίας αυτού του λόγου, θα ακολουθούσαμε απλά την

ίδια λογική με τα keywords και τα hashtags.

Έχοντας περιγράψει τα entities που χρησιμοποιήσαμε για την αναπαράσταση των δεδομένων, για να έχουμε μια πλήρη εικόνα του σχήματος, μένει να εξετάσουμε τις σχέσεις με τις οποίες αυτά συνδέονται.

Για την διασφάλιση της συνέπειας των δεδομένων, είναι σημαντικό, όπως εξηγήθηκε και στο κεφάλαιο για το App Engine, οι εγγραφές στο Datastore να πραγματοποιούνται εντός δοσοληψιών. Γνωρίζουμε ακόμη ότι σε μια δοσοληψία μπορούν να εμπλέκονται μόνο entities του ίδιου entity group. Όμως, κατά την αποθήκευση ενός tweet στο Datastore, χρειάζεται εκτός από τον user και το tweet να αποθηκεύσουμε αρκετά hashtags, πολλές λέξεις κλειδιά κι ακόμα πιο πολλές συντακτικές εξαρτήσεις. Καθένα από τα παραπάνω στοιχεία αναπαριστάται με ένα ξεχωριστό entity. Εύκολα μπορούμε να καταλάβουμε ότι αυτό που χρειαζόμαστε είναι όλη μας η βάση να αποτελείται από ένα και μόνο entity group. Παρ' όλα αυτά δεν υπάρχει κάποια λογική συσχέτιση, όπως υπάρχει μεταξύ Users-Tweets ή HashTags-HelpHash, που να οργανώνει όλα τα entities σε μια μοναδική δενδρική δομή. Για το λόγο αυτό, δημιουργούμε ένα ακόμα entity, το DummyRoot entity, το οποίο δεν έχει κανένα property και καμιά φυσική σημασία. Το χρησιμοποιούμε απλά, για να τοποθετήσουμε κάτω από αυτό όλα τα επιμέρους δέντρα που δημιουργούνται και να τα ενώσουμε σε ένα ενιαίο δέντρο, οργανώνοντας έτσι όλα τα δεδομένα της βάσης μας σε ένα μοναδικό entity group. Στην εικόνα που ακολουθεί φαίνεται το σχήμα της βάσης μας.



Σχήμα 28: Συνολική εικόνα του σχήματος δεδομένων που χρησιμοποιήσαμε για το Datastore

7) Σενάρια Χρήσης

Στο προηγούμενο κεφάλαιο, περιγράψαμε το σχήμα δεδομένων που χρησιμοποιήσαμε στην εφαρμογή μας. Στο κεφάλαιο αυτό, θα παρουσιάσουμε ορισμένα σενάρια χρήσης και θα σχολιάσουμε τα αποτελέσματα της εφαρμογής μας στις περιπτώσεις αυτές.

Για τα πειράματά μας, συλλέξαμε ένα σύνολο δεδομένων, το οποίο αποτελείτο από περίπου 25000 tweets. Όπως έχει αναφερθεί ήδη, για την συλλογή των tweets, χρησιμοποιήθηκε το Streaming API του Twitter. Είχαμε δει ακόμα ότι το Streaming API παρέχει πρόσβαση και στο Sample Stream και στο Filter Stream του Twitter. Αν κι η αρχική μας πρόθεση ήταν να κάνουμε χρήση του Sample Stream, έτσι ώστε στη βάση μας να περιέχεται ένα τυχαίο δείγμα της πληροφορίας που υπάρχει στο Twitter, αυτό παρουσίαζε την εξής δυσκολία: κάνοντας χρήση του Sample Stream δεν μπορούσαμε να καθορίσουμε τη γλώσσα στην οποία είναι γραμμένα τα tweets. Έτσι, κάνοντας sample, μας επιστρέφονταν πολλά tweets τα οποία δεν είχαν κάποιο νόημα ή ήταν γραμμένα σε άλλες γλώσσες κι όχι στα αγγλικά. Να θυμίσουμε σε αυτό το σημείο ότι τα IR εργαλεία που χρησιμοποιήσαμε, έχουν εκπαιδευτεί έτσι ώστε να λειτουργούν με την αγγλική γλώσσα και μόνο. Για το λόγο αυτό, χρησιμοποιήσαμε το Filter Stream, το οποίο μας δίνει τη δυνατότητα να συλλέγουμε από το public timeline μόνο tweets που περιέχουν συγκεκριμένες λέξεις, τις οποίες καθορίζουμε εμείς. Αυτό που σκεφτήκαμε είναι ότι αν οι λέξεις αυτές είναι αγγλικές λέξεις, τότε έχουμε αρκετά καλή πιθανότητα, ολόκληρο το tweet να είναι γραμμένο στα αγγλικά. Αν κι αυτό δεν είναι απόλυτο, και υπήρξαν και αστοχίες, τα tweets που συλλέξαμε με αυτόν τον τρόπο ήταν όντως επί των πλείστων γραμμένα στα αγγλικά. Βέβαια, η μέθοδος αυτή, περιορίζει τη θεματολογία που υπάρχει στα tweets. Οι λέξεις, που χρησιμοποιήσαμε εμείς για τη συλλογή των tweets είναι οι: music, movie, economy, greek, travel, science, crisis, song, sport.

Στα σενάρια χρήσης, που θα παρουσιάσουμε, θα κάνουμε ορισμένα ερωτήματα στη βάση δεδομένων και θα δούμε κατά πόσο αυτά επιτυγχάνουν. Δηλαδή, θα δούμε κατά πόσο τα αποτελέσματα των ερωτημάτων δίνουν πληροφορίες σχετικά με αυτό που πραγματικά θέλουμε να μάθουμε.

Στο πρώτο σενάριο χρήσης, επιθυμούμε να βρούμε τα tweets που αναφέρονται σε ρολόγια χειρός. Αν αξιοποιήσουμε μόνο εκείνες τις δυνατότητες της εφαρμογής μας που παρέχει και το Advanced Twitter Search, τότε ο μόνος τρόπος για να ανακτήσουμε αυτά τα tweets είναι να κάνουμε αναζήτηση με βάση τη λέξη “watch”. Στα αγγλικά, το watch εκτός από ρολόι χειρός, σημαίνει και παρακολουθώ.

Η αναζήτηση, με βάση τη λέξη “watch”, στην βάση δεδομένων της εφαρμογής μας, δίνει 516 αποτελέσματα. Παρατηρούμε όμως ότι σχεδόν παντού το watch εμφανίζεται ως ρήμα. Κατ' αυτόν τον τρόπο, το αποτέλεσμα της αναζήτησης δεν επιστρέφει κάτι σχετικό με αυτό που ψάχνουμε.

Στην εφαρμογή μας όμως, έχουμε τη δυνατότητα να αναζητήσουμε τα tweets, στα οποία το “watch” εμφανίζεται ως ουσιαστικό. Η αναζήτηση αυτή επιστρέφει 12 αποτελέσματα. Βλέπουμε επομένως ότι το πλήθος των tweets που πραγματικά μας ενδιέφεραν ήταν μόνο το $\frac{12}{516}=\frac{1}{43}$ των αποτελεσμάτων που θα επέστρεφε το Advanced Twitter Search. Στον παρακάτω πίνακα, φαίνονται τα αποτελέσματα της αναζήτησης του “watch” ως ουσιαστικό.

Search Results: There are 12 results

User	Tweet
sweetas_CUPCAKE	@b__dope i yu watch da movie not da cartoon one wen dey was wlkin n da musuem dey fans was dressd like dem smokin weed!
_hafis	[watch] jane s addiction - new music video “underground” worldwide http://t.co/hlybvebt @janesaddiction
Osibelking	lmfao! as in..the muvi is sick!rt @funkeoba: na u know nah no be only u dey watch am??rt @osibelking: this movie is freakn sick!!! lmfao!
vicentagwhorton	> invicta men s 1138 corduba black. dial black. leather watch save money http://t.co/kn4xqpym
Cyclefilm	@cottydale @looneysonya unlike the global economy, cyclefilm biz is booming. no one s getting fired on my watch!!!
PermaGoddess	rt @permculturenow: trailer for mike ruppert s collapse movie, well worth a watch http://t.co/fvodxoum http://t.co/hemd8nvh
DENNISBITCHMOB	rt @lilbthebasedgod: lil b - februarys confessions *music video*one of the best songs 2012 an...: http://t.co/rwsylbjd watch #new
NikaStEtIc	fina lay dwn and watch tv and listen ta music...
vicentagwhorton	> fossil stella boyfriend aluminum watch - berry save money http://t.co/4mzrxfaw
Ltd_Charlie	@mmsherbert tall uggs are 160 whenever you have money we can definitely hook you up.the more we sell the better. and we sell watches too :-)
IdeasWeekly	rt @randyice: i need a movie watch app on my ipad on these lonely nites. ????
kenhess	do you think erin brockovich will get me some money for my tic? oh wait, nevermind, it s my watch. dammit. erin, call me anyway. ;-)

Πίνακας 14: Αποτελέσματα για το "watch" ως ουσιαστικό

Παρατηρώντας τα αποτελέσματα, μια άλλη επισήμανση που κάνουμε είναι ότι ακόμα κι ανάμεσα στα 12 αυτά tweets, υπάρχουν tweets τα οποία δεν μας ενδιαφέρουν. Βασικά βλέπουμε, ότι το “watch” εμφανίζεται ως ουσιαστικό στα 7 από τα 12 tweets. Αμέσως παρακάτω βλέπουμε δυο από τα tweets στα οποία είχαμε αστοχία.

sweetas_CUPCAKE	@b__dope i yu watch da movie not da cartoon one wen dey was wlkin n da musuem dey fans was dressd like dem smokin weed!
_hafis	[watch] jane s addiction - new music video "underground" worldwide http://t.co/hlybvebt @janesaddiction

Πίνακας 15: Αστοχίες ερωτήματος

Στο πρώτο tweet βλέπουμε ότι το “watch” χρησιμοποιείται ως ρήμα. Όμως, η φράση “i yu watch da movie” δεν είναι σωστά συντεταγμένη με βάση τους κανόνες της αγγλικής γλώσσας και για τον λόγο αυτό ο Tagger αποτυγχάνει να την αναγνωρίσει ως ρήμα, όπως και θα έπρεπε.

Στο δεύτερο tweet βλέπουμε ότι ενώ το “watch” υπάρχει σε αυτό, δεν συνδέεται συντακτικά με την υπόλοιπη πρόταση. Παρατηρήσαμε γενικά, ότι όποτε είχαμε κάποια λέξη συντακτικά ασύνδετη σε ένα tweet, ο Tagger την αναγνώριζε ως ουσιαστικό.

Το πρόβλημα αυτό δεν έτυχε να παρατηρηθεί σε αυτές μόνο τις περιπτώσεις, αλλά εμφανίζεται γενικά λόγω της φύσης των tweets. Οι δημοσιεύσεις των χρηστών του Twitter τείνουν να μην χρησιμοποιούν σωστό συντακτικό και να χρησιμοποιούν συντμήσεις λέξεων, γεγονός που κάνει διάφορα NLP εργαλεία να μην δουλεύουν σωστά. Το πρόβλημα αυτό έχει παρατηρηθεί γενικά κι έχουν αναπτυχθεί NLP εργαλεία που απευθύνονται αποκλειστικά σε tweets, όπως ο tagger που παρουσιάζεται στο [12]. Παρ'όλα αυτά, εμείς επιλέξαμε για την εφαρμογή μας το Stanford CoreNLP διότι παρείχε μια ολοκληρωμένη λύση. Δηλαδή, το API του προσέφερε και tagger και lemmatizer και parser και δεν βρήκαμε κάποιο Tweet NLP εργαλείο που να τα παρέχει όλα αυτά. Άλλωστε, σκοπός της παρούσας εργασίας δεν είναι η ανάπτυξη NLP εργαλείων, αλλά η υλοποίηση μιας ιδέας, όσο το δυνατόν πιο ολοκληρωμένης για τη διαχείριση των δεδομένων που προέρχονται από το Twitter.

Στο δεύτερο σενάριο χρήσης, μας ενδιαφέρει να βρούμε τα tweets, στα οποία κάποιος χρήστης αναφέρει ότι ακούει μουσική. Χρησιμοποιώντας το Advanced Twitter Search, για να βρούμε αυτά τα tweets, ενδεχομένως θα κάναμε αναζήτηση με βάση τις λέξεις “listen” και “music”. Θα αναζητούσαμε δηλαδή τα tweets που περιέχουν τις δυο λέξεις αυτές συγχρόνως. Η αναζήτηση αυτή δίνει ως αποτέλεσμα 170 tweets. Παρατηρούμε ότι στο σύνολο αυτών των tweets υπάρχουν και tweets όπως το “whenever i listen to nirvana i think of second year music, hahaha.”, όπου οι λέξεις “listen” και “music” δεν συνδέονται άμεσα. Με χρήση όμως της εφαρμογής μας, μπορούμε να ζητήσουμε τα tweets αυτά στα οποία το ρήμα είναι το “listen” και άμεσο αντικείμενο είναι το “music”. Στην περίπτωση αυτή επιστρέφονται 20 αποτελέσματα, δηλαδή περίπου το 1/9 των αποτελεσμάτων της αναζήτησης του Advanced Twitter Search. Για άλλη μια φορά επομένως, παρατηρούμε ότι η εφαρμογή μας δίνει τη δυνατότητα για πιο στοχευμένα ερωτήματα.

8) Επίλογος

Αντιλαμβανόμενοι την έκρηξη δεδομένων που παρατηρείται στις μέρες μας και την αξία των analytics εφαρμογών, στην εργασία αυτή παρουσιάσαμε ένα σύστημα για την διαχείριση των πληροφοριών που προέρχονται από το κοινωνικό δίκτυο του Twitter. Το σύστημά μας αποθηκεύει tweets στο Datastore, την κατανεμημένη βάση του Google App Engine, και δίνει τη δυνατότητα αναζήτησης σε αυτά τα δεδομένα.

Η εφαρμογή μας, το TweetFinder11, υπερτερεί αρκετά σε σχέση με το Advanced Twitter Search. Αυτό συμβαίνει, διότι προτού αποθηκεύσουμε τα tweets στη βάση που έχουμε στο Datastore, τα περνάμε από γραμματική και συντακτική ανάλυση που γίνεται με χρήση NLP εργαλείων και τα δεικτοδοτούμε με βάση τις πληροφορίες που προκύπτουν από την ανάλυση αυτή. Αυτό έχει ως αποτέλεσμα να μπορούμε να κάνουμε πιο “έξυπνα” και στοχευμένα ερωτήματα στη βάση δεδομένων μας κι επομένως να ανακτούμε πληροφορίες που σχετίζονται πιο άμεσα με τα ενδιαφέροντά μας.

Επίσης, τονίζουμε για ακόμα μια φορά, ότι αν και η εφαρμογή μας δεν είναι παρά μια εξελιγμένη μηχανή αναζήτησης, ο συνδυασμός της με τη χρήση άλλων εφαρμογών, θα μπορούσε να πετύχει την εξόρυξη κι εκμετάλλευση των πληροφοριών που εμπεριέχονται στα tweets σε ακόμα μεγαλύτερο βαθμό. Για παράδειγμα, μια εφαρμογή που κάνει sentiment analysis σε tweets, για να εξάγει την κοινή γνώμη πάνω σε ένα θέμα, θα μπορούσε να χρησιμοποιεί τη δική μας εφαρμογή σε ένα πρώτο στάδιο, έτσι ώστε το σύνολο δεδομένων που θα αναλύσει στη συνέχεια να είναι πιο σχετικό με το θέμα της έρευνας.

Τέλος, οφείλουμε να πούμε ότι η εφαρμογή μας έχει ακόμα αρκετά περιθώρια βελτίωσης. Εξετάζοντας την αποτελεσματικότητά της, παρατηρήσαμε ότι υπάρχουν περιπτώσεις που η λειτουργία της δεν είναι αυτή που θα έπρεπε κι αυτό οφείλεται στον τρόπο που συντάσσουν τα tweets οι χρήστες του Twitter. Η ανάπτυξη και η επιλογή NLP εργαλείων που έχουν σχεδιαστεί αποκλειστικά για την γλωσσική ανάλυση των tweets, ενδεχομένως να έδινε καλύτερα αποτελέσματα, οδηγώντας έτσι την εφαρμογή μας να επιστρέφει ακόμα πιο στοχευμένα αποτελέσματα στα ερωτήματα των χρηστών.

9) Αναφορές

[1] Martin Hilbert, Priscila Lopez The World's Technological Capacity to Store, Communicate, and Compute Information.

[2] http://www.facebook.com/press/info.php?statistics

[3] http://www.marketinggum.com/twitter-statistics-2011-updated-stats/

[4] twittercounter.com

[5] dev.twitter.com

[6] Bernard J. Jansen, Mimi Zhang, Kate Sobel, Abdur Chowdury Twitter Power: Tweets as Electronic Word of Mouth

[7] Andranik Tumasjan, Timm O. Spregner, Philipp G. Sandner, Isabelle M. Welpe Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment

[8] Zarrella D. 2009a. State of the TwitterSphere

[9] Kristina Toutanova, Christopher D. Manning, Enriching the Knowledge Sources Used in a Maximum Entropy Part-Of-Speech Tagger

[10] Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning 2006. Generating Typed Dependency Parses from Phrase Structure Parses.

[11] http://www.cis.upenn.edu/~treebank/

[12] Kevin Gimpel, Nathan Schneider, Brendan O' Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan and Noah A. Smith, School of Computer Science, Carnegie Mellon University, Pittsburgh: Part-Of-Speech Tagging for Twitter: Annotation, Features and Experiments

[13] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia “Above the Clouds: A Berkleley View of Cloud Computing”, February 2009

[14] Dan Sanderson, Programming Google AppEngine, O'Reilly|Google Press

[15] http://code.google.com/appengine/docs/java/

[16] http://www.keithpij.com/Home/tabid/36/EntryID/27/Default.aspx

[17] Perlmutter, D. D. 2008. Political Blogging and Campaign

[18] pearanalytics 2009. Twitter Study

[19] http://groups.google.com/group/google-appengine-java/browse_thread/thread/066e09a7d6848a8a?

pli=1

[20] http://www.newscientist.com/article/mg21128295.900-using-twitter-to-follow-trends-beats-the-stock-market.html

[21] http://twitter4j.org/en/index.html

[22] http://hueniverse.com/oauth/guide/terminology/