Knowledge-Based Systems 37 (2013) 105–120

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A software architecture for Twitter collection, search and geolocation services ☆

M. Oussalah [a,]*, F. Bhat [a], K. Challis [b], T. Schnier [c]

[a] University of Birmingham, School of Electronics, Electrical and Computer Engineering, Edgbaston, B15 2TT Birmingham, UK
[b] University of Birmingham, Institute of Archaeology and Antiquity, Edgbaston, B15 2TT Birmingham, UK
[c] University of Birmingham, School of Computer Science, Edgbaston, B15 2TT Birmingham, UK

ARTICLE INFO

Article history:
Received 8 December 2011
Received in revised form 15 June 2012
Accepted 23 July 2012
Available online 7 August 2012

Keywords:
Data mining
Tweet
Social network
Software architecture
Semantic analysis

ABSTRACT

The substantial increase of social networks and their combination with mobile devices make rigorous analysis of the outcomes of such system of paramount importance for intelligence gathering and decision making purposes. Since the introduction of Twitter system in 2006, tweeting emerged as an efficient open social network that attracted interest from various research/commercial and military communities. This paper investigates the current software architecture of Twitter system and put forward a new architecture dedicated for semantic and spatial analysis of Twitter data. Especially, Twitter Streaming API was used as a basis for tweet collection data stored in MySQL like database. While Lucene system together with WordNet lexical database linked to advanced natural language processing and PostGIS platform were used to ensure semantic and spatial analysis of the collected data. A functional diversity approach was implemented to enforce fault tolerance for the data collection part where its performances were evaluated through comparison with alternative approaches. The proposal enables the discovery of spatial patterns within geo-located Twitter and can provide the user or operator with useful unforeseen elements.

1. Introduction

Online Social Networks (OSNs) enable users to rapidly share information with widely dispersed networks. Users are able to post-content in a variety of formats, which can be instantly made available to their entire social network. Thus, OSNs are becoming an important platform for dissemination of information, web content discovery, opinion sharing, discussion and debate.

Among these OSNs, one shall highlight the increasing development of tweet-based social networking, which, due to absence of privacy barriers, unlike other social networks, e.g., facebook, Myspace, Badoo, becomes a subject of intensive study for decision-making, scene analysis, social behavior monitoring, trends and prediction. Besides, Twitter also conveys location information which makes it suitable for geo-location analysis.

The ability to collect and mine geo-located tweets opens new research directions by making it possible to study the spatial as well as textual characteristics of online content. Indeed, the large amounts of public data that flows through OSNs has the potential to deliver valuable new insight to the academic community, marketing agencies, NGOs and other organizations interested in understanding online behavior and monitoring social trends. It has been acknowledged that research into Twitter still is in its infancy with much studies focusing on describing properties and some statistical accounting to support some opinion or trends [30]. However the boom on smart phones with the associated available service of access to social networks, enabling anyone with access to a cell-phone to communicate rapidly with widely dispersed networks of people through SMS, boosted the tweet research well beyond its initial boundary to include more in depth studies of social interactions and message content analysis. This allowed Twitter to make “interesting inroads into novel domains, such as help during a large-scale fire emergency, updates during riots in Kenya, and live traffic updates to track commuting delays” [21], construction of earthquake map [24,29].

Its popularity increased exponentially to reach in June 2011 more than 200 millions of users generating more than 1.6 billions query per day [33,35,37], which clearly testifies of the very large database that can be collected after only few days.

As the volume of content shared publicly on OSNs continues to grow, there is a great demand for technology that can assist with the collection and mining of this content. Twitter’s potential as a tool for research and analysis is underlined by its rapid growth and emergence as a mainstream channel of communication on the web [5,12].

Designing a software system for automatic tweet collection and retrieval is quite challenging due to imperfections and open issues

☆ This paper is an extended and a refined version of a conference paper published in Proceedings of the 10th IEEE International Conference on Cybernetics and Intelligent Systems, University of East London, 2011.
* Corresponding author. Tel.: +44 1214143128.
E-mail address: M.Oussalah@bham.ac.uk (M. Oussalah).

0950-7051/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2012.07.017

pervading both the information retrieval system because of the difficulty to deal with semantic aspect, among others and the dependency on tweet APIs, which are sometimes subject to inconsistencies. Anderson and Schram [1] have suggested a software architecture for tweet collection that allows the user to search for tweets that match a keyword search and link to Twitter friends and followers. This is built using production class software frameworks (Spring, MVC, Hibernate and JPA) and infrastructure components (Tomcat, MySQL and Lucene). Perera et al. [26] have suggested an architecture for tweet collection and search functionalities that makes use of Twitter APIs (Streaming API, Search API, Rest and Twython APIs) in conjunction with Python, SQL. Especially, the Twython API was used for tweet location. The latter, however, does not support the latitude/longitude evaluation. To enable such feature the authors used separate Restful API provided by yahoo. TwitInfo has been suggested by Marcus et al. [23] as a platform to collect and explore tweets in real time and perform sentiment analysis. In term of software architecture development, TwitInfo builds on work on exploratory interfaces for temporal exploration of Statler [31,32] and Eddi et al. [4], which both provide timeline visualizations of Twitter data.

Strictly speaking, the task of collecting, searching and analyzing tweets is rather challenging for various reasons. First the sampling rate imposed by some of the Twitter APIs combined with the very large number of tweets generated by users at each instant make the task of collecting all tweets rather difficult and much dependent on the performance of the employed (server) machine. Second, the real time analysis constraint, if imposed, would definitely reduce the number of collected tweets as during the (costly) search and possibly analysis tasks, eventually, hundreds and maybe thousands of other tweets would be missed because of missing deadline. Some authors suggested the use of Search API to keep track of past tweets but the performance of such approach is known to be limited as the API only recovers a limited number of past tweets. Third, the geo-location information although encompasses the latitude/longitude values but it barely includes the address like description of the location or the position. Fourth, appropriate handling of user's text query would require accounting for textual disambiguation in order to grasp the semantic aspect of query/tweet instead of usual standard word matching currently employed in available platforms. Fifth, because of the limit of tweet message size – only 140 characters per message is allowed – Twitter messages do not necessary contain well constructed ideas, sentences or phrases, but rather incomplete, unstructured, possibly with a lot of holes and jargon words (absent in the dictionary). This clearly makes any attempt to use natural language processing tools or any computer based linguistic analysis limited. Sixth, the Twitter data is more than a collection of texts and other attributes but rather a complete interactive and dynamic network with various connections corresponding to user's followers (tweeters that follow the user), which are potentially of various level of importance, and tweeters. This would suggest that identification of the context of current tweet requires examination of the original tweet (s). Besides, the user has the possibility to twit beyond the list of followers using the *retweet* mechanism or hashtag. This clearly brings the issue of graph analysis for instance very challenging as the concepts of centrality, node weights, among others become ill-defined. Seventh, the large scale database together with data analysis timing constraints make the data type and software representation quite difficult. For instance, the structure of social network data is different from web link data as the latter consists of webpages and the connecting links, so a directed graph can easily be used for its representation because every link goes only in one direction. Bahmani noticed: "In social networks, there is no equivalent of a web host, and more generally it is not easy to find a corresponding block structure (even if such a structure actually exists)" [3]. This difference has a direct effect on the storing and retrieving of social data, because similar data cannot be stored together on one machine. This has led to emergence of distributed systems employed to data accessing and processing but at increased communication cost. Typically for the social data structure it is common to use N to N relations that exist for most nodes, which means every node connects to several nodes, and several other nodes are connected to this node. Although much of research and implementations, including Facebook, in the literature employ the relational data model MySQL like database because of its proven stability and easy installation. However, MySQL is not optimized for graph like queries. Besides the read/search operation is known to be quite expensive as compared to write operation [6,19].

This paper describes an architecture for automatic gathering of tweets within a predefined region as delimited by the upper and lower latitude–longitude coordinates of the containing rectangular box. The software implementation makes use of Python-based web framework Django together with Apache Lucene [18] linked with MySQL server database in order to benefit from Lucene's advanced indexing schemes and relational model in conjunction with the cross platform operability of MySQL. The architecture also describes basic search capabilities using text tweet, tweeter's name, location, postcode, among others. Especially, the text analysis allows us to handle the semantic aspect of the text through the use of WordNet lexical database in query expansions. On the other hand, integrating a dynamic Slang database allows the system to tackle at some extent the unstructured tweet messages due to text size constraint. The combination of PostgreSQL database server with the PostGIS spatial extension used to accommodate GeoDjango requirement allows the system to handle efficiently geo-location queries. The whole software architecture is built on server machine which allows remote access to the analysis and search applications. The developed application makes use of Streaming API, which focuses on providing access to real time posts and allows for almost unlimited number of tweets, among the available Tweet API for data collection purpose.

Relating the proposed architecture to the previously mentioned challenges reveals the following. First, since data collection and analysis tasks are handled separately so the real-time constraint is not imposed, and, on the other hand, a large scale implementation is not part of system requirement. This discards both Challenge 2 and 7 from this study. Second, the restriction to geo-tagged tweets, which has much lower sampling rate, together with the use of a set of fault tolerant machines for data collection enable us to address the first challenge. Third, the developed architecture does tackle the third, fourth, fifth and partly the sixth challenges. Indeed, the use of PostgreSQL database, WordNet lexical database and dynamic slag database contribute to location-based queries, semantic analysis and jargon words challenges, respectively. Although the analysis does not investigate the full network interaction among the Twitters but rather limit on main keywords as specified, for instance, through hashtag words, which explains why the sixth challenge is only partly tackled.

Section 2 of this paper provides an overview of key concepts of Twitter systems. Section 3 highlights the developed system architecture. This includes system requirement, architecture outline, data collection architecture, spatial storage and spatial search, full text search and slang expansion, data mapping and export, and fault tolerance. System interface and outcomes are exhibited in Section 4.

2. Overview of Twitter concepts

2.1. Key features of Twitter system

Twitter is a micro-blogging system, sometimes, referred to as the SMS of the internet, created by Dorsey in 2006 [8], that allows users to post/receive small text messages of up to140 characters called tweets. The size restriction of 140 characters was used for compatibility with SMS messages. Indeed, the original 160 character SMS limit was split up into 20 character username and 140 character post-fields. This motivates the use of several shortening forms including users' abbreviation and slang words and URL shortening services like *bit.ly*, *goo.gl*, twitpic employed by Twitter system; although, since March 2010, Twitter began using its own shortening service *twl.tl* URL. However, this adds extra difficulty to analyzers and developers if the task of recovering the original URL is part of such analysis.

Users can choose to display the messages of other users they find interesting in their line by following them. Users can tweet using either Twitter website, compatible external application on smart phone or via simple SMS messages. The communication to Twitter through SMS is enabled using short codes provided by mobile operators, although the generic platform SMSTweet [19] is emerging. Tweets are publicly available to all users by default. However, senders have the option to restrict their messages delivery to only their followers.

Twitter also offers extra functionalities to a simple message posting. This includes replying to or retweet (share) any message the user wants. A user can also address a message directly to another user, without being friend of, by starting his/her message with "@destinationUser". The Twitter system offers the possibility to the user to introduce a hashtag (words of phrases prefixed with a sign "#") in his tweet message, e.g., #upraising, which makes the hashtag title (upraising) a good candidate for keyword search or so. Many tweet messages often contain a URL link which directs the reader to the news article, video or any other information the tweeter wants to convey to his followers. A key feature of Twitter is its asymmetry property with respect to social connection. Indeed, unlike Facebook or Messanger for instance, the connection does not go in both directions: a user can follow another user without the latter having to follow the former. This behavior encourages tweeter to follow their favorite stars. Especially, when a tweeter X adds a user Y as a friend, then X is considered as a follower of Y so that all tweets sent by Y will be received by X while the reverse is not necessarily true. In other words, the only relationship between users in Twitter is follow-and-be-followed where users subscribe to each other and keep up-to-date with their updates simultaneously. This enables very rapid release of news like information in almost real time, which makes Twitter gaining popularity across all news agencies. Rosenstiel and Kovach [28] pointed out that the traditional way of news propagation cycle is decentralised by the heterogeneous media in the digital world. However, with the development of blogosphere, it has been seen that many influential bloggers and news organizations tend to have Twitter profiles synchronized and updated at the same time with the hope to reaching its audience in seconds. This enables real-time reflection of the news world on a single site where Twitter acts as the main centralized point of all updates. Events like the 2009 US Airways Flight 1549 incident, the 2009 Iranian presidential election, Japan's earthquake/Tsunami and recent arab protests all demonstrate the efficiency of such blog to convey real time information about the status of the underlying scenes.

2.2. Twitter APIs and implementations

Twitter API is based on Representational State Transfer (REST) architecture introduced by Fielding in his dissertation on software architecture for distributed hypermedia systems [11] employed mainly in World Wide Web (WWW). The architecture consists of client–server like interaction where client initiates requests to server and the latter processes the request and return appropriate responses; namely, a collection of network design principles that define resources and ways to address and access the data. Especially, REST uses existing features of the HTTP protocols enabling layered proxy and gateway components to perform HTTP caching and security enforcement.

REST architecture enabled Twitter to work with web syndication formats where two of these formats are explicitly employed; namely, Really Simple Syndication (RSS) and Atom Syndication Format (Atom). Visitors can subscribe to syndication service (feeds) and receives an update every time web administrator changes the page. On the other hand, Twitter's open API enabled third party applications to read user's feeds while allowing desktop and mobile phone applications on Twitter to use this functionality as well [2].

Because of their simplicity, Twitter uses OAuth and Basic Authentication mechanisms. Indeed, the open protocol OAuth allows secure API authorization in a simple and standard method from any desktop or web applications [25]. Besides OAuth enables third party applications that are abundant on Twitter to connect to user accounts without a username and password authentication.

In order to handle the scalability issue due to exponential increase of users a cloud computing architecture and the programming platform moved recently from Ruby-On-Rails to SCALA.

The Twitter Application Programming Interface (API) [15,16] provides three main APIs: Streaming API, REST API and Search API. The Streaming API provides near real-time access to subsets of tweet public status descriptors, including replies and mentions created by public accounts, except protected accounts while requiring continuous connection to http server. For this purpose, the script that communicates with Streaming API must runs as a long-term background process or a system daemon. Especially, the Streaming API provides, through services like BirdDog and GardenHose that allow the user to have access to a much larger stream of tweets, almost zero missing real time tweet collection. This API is HTTP based with main commands GET, POST, and DELETE requests can be used to access the data. Besides, the Streaming API can filter status descriptions using quality metrics, which are influenced by frequent and repetitious status updates, etc.

The API uses basic HTTP authentication and requires a valid Twitter account while data can be retrieved in XML format.

The Search API is implemented on a separate stack from the main Twitter stack and uses a separate database, thereby, the returned data objects have different structure and Ids. Despite its simplicity as it can be called upon with a Rest URL that can be retrieved using a simple http Get request, the Search API is limited by low data rate, which restricts the number of tweets the user can get as a response to his query [36]. In theory it can retrieve up to 1500 tweets per request, but in practice it is usually much smaller. However, in contrast to other Twitter APIs, the Search API has the possibility to include past tweets (up to 1500 tweets in the past 7 days).

Rest API also provides authenticated users, through open authentication methods, e.g., OAuth, with access to their public tweets. However, likewise Search API, Rest API has also serious data rate limitation. For instance with OAuth authentication, user is allowed 350 requests per hour [36]. Consequently, when large scale tweet collection is concerned, the Streaming API had clear advantages. Especially, the Streaming API allows the system to use an expanded filter predicates, where each tweet is associated a set of attributes including the message content, the sender name, sender Id, etc. A set of attributes relevant to this work are summarized in Table 1.

2.3. System architecture of Twitter APIs

In terms of software architecture, Fig. 1 highlights a simple architecture for tweet collection using Streaming API distinguishing the

Table 1
Structure of the status object.

Element	Type	Description
Id	Long	Unique id of the tweet
Text	String	Text of the tweet 140 character max
created_at	Date	Tweet's creation date
in_reply_to_user_id	Long	The id of the user to whom this tweet is addressed
in_reply_to_screen_name	String	Screen name of the user to whom this tweet is addressed
in_reply_to_tweet_id	Long	The id of an existing tweet that this tweet is in reply to Only set if the author of the tweet is referenced
Coordinates	Long	A GeoJSON object which includes the latitude and longitude from which the tweet was posted
User dictionary (embedded within the Tweet dictionary)		
id	Long	Unique id of the user who posted the tweet
screen_name	String	Screen name of the user who posted the tweet
name	String	Full name of the user who posted the tweet. Taken from the profile information provided by the user
statuses_count	Long	Total number of tweets posted by the author
friends_count	Long	Number of users who the author is following
followers_count	Long	Number of users who are following the author
lang	Text	The user's selected language

process of gathering tweets from the presentation layer that displays the appropriate data on normalized database schema that can be optimized for user query needs. This forms the purpose of the Twitter API database cache [15].

More specifically, the first step of collecting tweets is performed by *get_tweets.php*, which is run as a continuous background process. When a new tweet is received the Twitter Streaming API, get_tweets.php uses *db_lib.php* to insert it into the *json cache* table. The connection with the Twitter Streaming API is maintained by the *Phirehose library*.

The Twitter Streaming API can return data in either XML or JSON format. To make the collection process as fast as possible, the entire JSON payload for a single tweet is saved to the database as a single string without any parsing. A separate background process is run for *parse_tweets.php*, which gets the *JSON* data for each tweet from the *json cache* table. Then, it parses it into its component parts, and inserts the outcome into separate table for use by the other modules in the *140dev framework*. Once again, *db_lib.php* is used to manage the MySQL code [16]. See Fig. 2 for a graphical illustration.

For storage purpose, Twitter uses MySQL and Cassandra that are accessed by various services through Thrift where the FlockDB storage solution built on top of MySQL was used to store relationships between users and followers/friends.

It should be noted that since its early introduction, Twitter's main functionality is to act as a messaging bridge between different formats including SMS, web, IM among others. Consequently, the design was constantly improved to keep track of the exponential increase of number of users and user queries which require almost real time analysis. Therefore, the search approach is kept very limited and only restricted to standard indexing and basic text matching. Although the relational based structure, even though widely employed in web services has shown serious limitations as tables grow exponentially or when redundancy is required to tackle data failure [16]. Search function in Streaming API uses an indexing-based-approach on a limited number of tweets (real time constraint) and allows up to 400 keyword in an AND or OR combinations, searched from up to 5000 user accounts using either basic or OAuth authentications without any inclusion of past tweets.

More specifically, the search, in advanced search option, allows the user to perform:

(i) Word level search: one can perform.
   – single word exact match,
   – a combination of keyword search using conjunctive, disjunction, exclusion like aggregation of a set of keywords,
   – Hashtag match: search for discussion topics,
   – select language.
(ii) Person-related search.
   – output tweets sent by specific user Id,
   – output tweets addressed to a specific user Id,
   – output tweets referencing a specified username,
(iii) Location specific search.
   – The location is specified by a bounding box specified as a comma separate list of longitude/latitude pairs, with the first pair denoting the southwest corner of the box. For example locations = −122.75, 36.8, −121.75, 37.8 would track tweets from the San Francisco area. Multiple bounding boxes may be specified by concatenating latitude/longitude pairs. Up to 25 bounding boxes are supported by Streaming API.
   – output tweets sent near a specified location,
   – output tweets sent within xx miles of a specific location,
(iv) Date specific search: outputs tweets sent since until the specified date.

One shall also mention the growing services provided for search purpose. This includes for instance *Twitseek* that searches for URLs Twittered along with the keywords, *Tweetscan* which updates the search result every second, *Twit-Scoop* which tracks keyword related conversations.

Nevertheless the advances in real time search cannot compensate for its deficiencies when a deep analysis of tweets over a certain period of time is required using any of Twitter API. This includes the fact that (i) very limited number of tweets is searched on; (ii) no access to past events; (iii) inability to seamlessly combine Search API and Streaming API usage to collect both real-time and past tweets; (iv) there is a limit on the maximum number of responses (e.g., tweets) that can be outputted; (v) the search mechanism is mainly indexing based, which makes any semantically

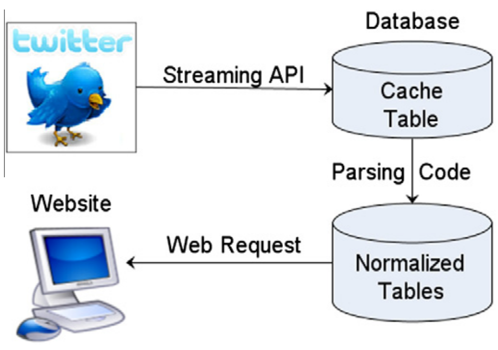


Fig. 1. Simple architecture of tweet collection [15].

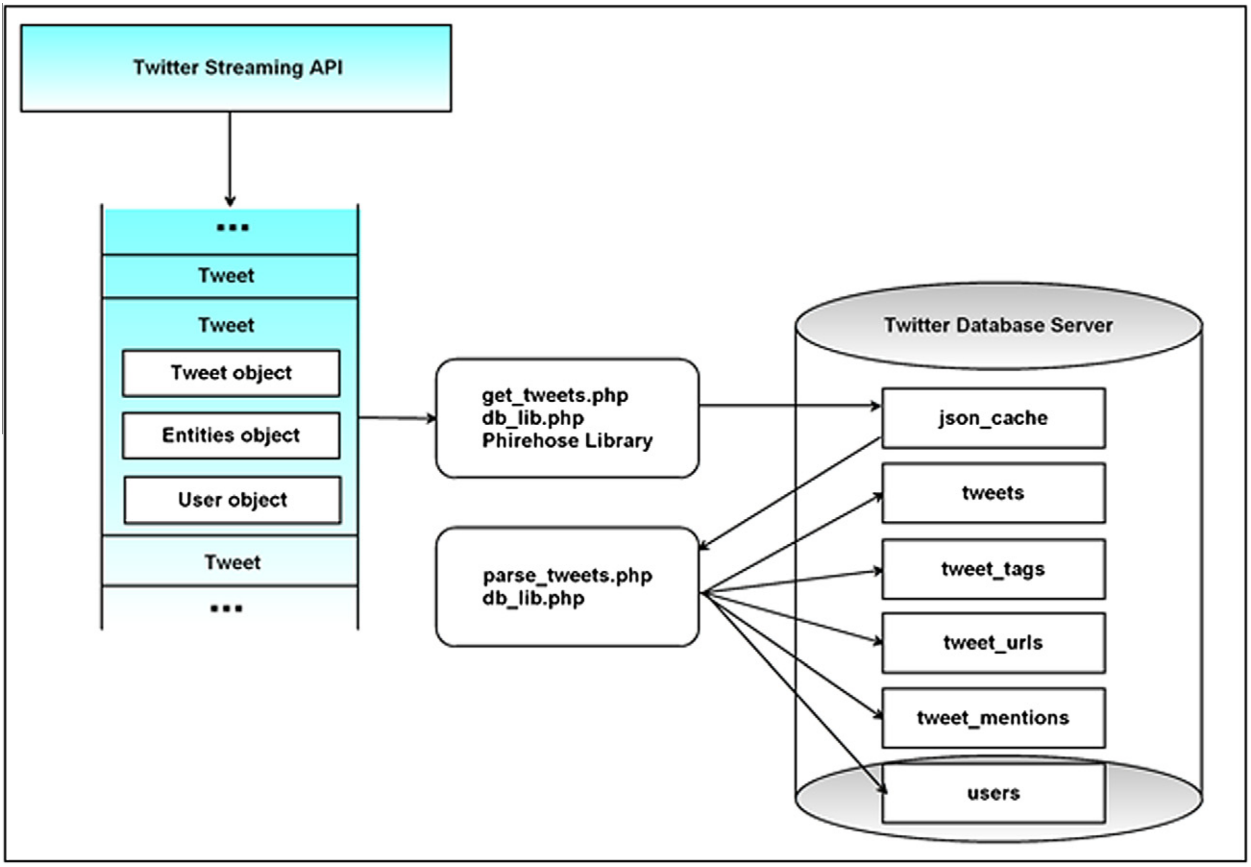


Fig. 2. Architecture of Twitter database server [16].

related analysis very limited; (vi) default outputs were provided if the indexing search does not complete within a fixed time deadline.

This motivates the present work for further search analysis using large scale database of selectively gathered tweets. In this respect, the search function is carried out offline, indicating the real time constraint is very much softened or partially ignored.

3. Software system architecture and semantic analysis

3.1. System requirement

From the software implementation perspective, elicitation of system requirement is a necessary stage [34]. In this respect, the purpose of the developed architecture can be summarized into the following high level system requirements:

(i) Allow users to collect a large dataset of Twitter data posted from various geographic regions within some time interval.
(ii) Allow users to retrieve relevant tweets from the collected data using spatial queries and retrieve their profiles.
(iii) Allow users to retrieve relevant tweets from the collected data using textual queries that account for both the semantic aspect and slang language.
(iv) Allow users to export tweets to a file format suitable for further research and analysis.
(v) Allow users to inspect retrieved tweets using a map-based interface.

It should be noted that the requirement of continuous tweet data collection would suggest the use of Streaming API with, possibly, further strengthening of the system reliability using like fault tolerance to prevent possible connection lost or so. The absence of real time requirement paves the way for deeper analysis of the collected tweets. The requirement of extensive search using the semantic aspect, slang words among others would exclude the use of search function of either Streaming or Search APIs. The requirement of map-based interfacing to track location of particular tweets would require use of spatial database like PostgreSQL database.

3.2. Outline of the solution

- The Python-based web framework Django [7], which is a high-level web framework based on a model-template-view (MTV) software architectural pattern, was used for our web application development. Especially, this provides a way to separate the user interface from the domain-specific representation of the data and from the domain logic that includes functional algorithms handling information exchange between database and user interface, which, in turn, makes it easier to the user to modify components independently. This also allows us to use its GeoDjango extension which is particularly useful for constructing geographic web-applications as well as linking it with related web frameworks.
- Apache Lucene system, an open source high performance information retrieval library, was used for integrating the collected tweet data into a scalable and indexed database. In other words, instead of using the standard relational model of Twitter system architecture, the highly efficient inverted index representation of Lucene will be exploited which benefit from the well developed search schemes and their optimized implementation for

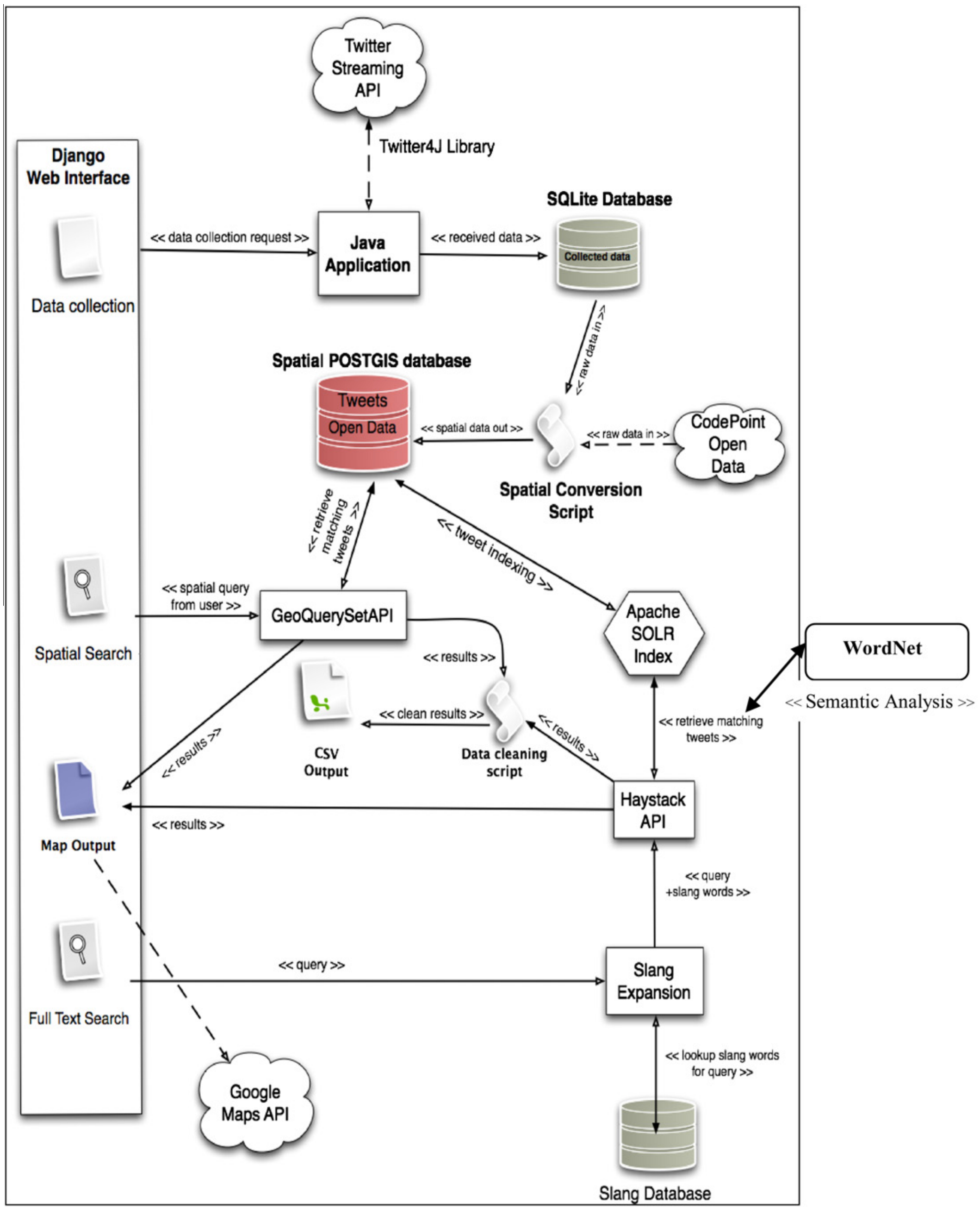


Fig. 3. Detailed system architecture.

almost real time applications. This brings more efficiency to the search function and allows us to employ the well developed professional built-in analyzers, query handlers and boosting fields.

- WordNet – a lexical database [10] where nouns, verbs, adjectives/adverbs are grouped into a set of cognitive synonyms called synsets that are interlinked through conceptual–semantic and lexical relations – is employed in order to handle the semantic aspect of the requirement. Especially, one uses the set of synonyms as well as the sufficiently semantically equivalent terms, according to some semantic similarity measure, to expand the original user query as will be detailed later on. Together with Lucene's enhanced query modules, which allows user to handle keywords, phrase, logical searches among others, this endows the system with more efficient and disambiguation search tools.
- The geo-location handling of information is enhanced by integrating two extra libraries; namely, UK Ordinance Survey Code-Point Open and 1:50 k scale gazetteer. The former provides access to precise co-ordinates for over 1.7 million UK post-codes [6]. The latter "contains over 250,000 place names and is the most detailed gazetteer available" [9]. The aim of integrating these data sets is to allow the retrieval of tweets using distance lookups from any post-code or place in the database. The data sets are made available in CSV format and the location of each post-code and place is specified using the Easting/Northing co-ordinate system. These data sets must also be represented in a spatial format before they can be used for distance lookups
- A Slang database expanding some of publicly available Slang database as well as those observed through the collection task is constructed and integrated into the system as part of prepro-

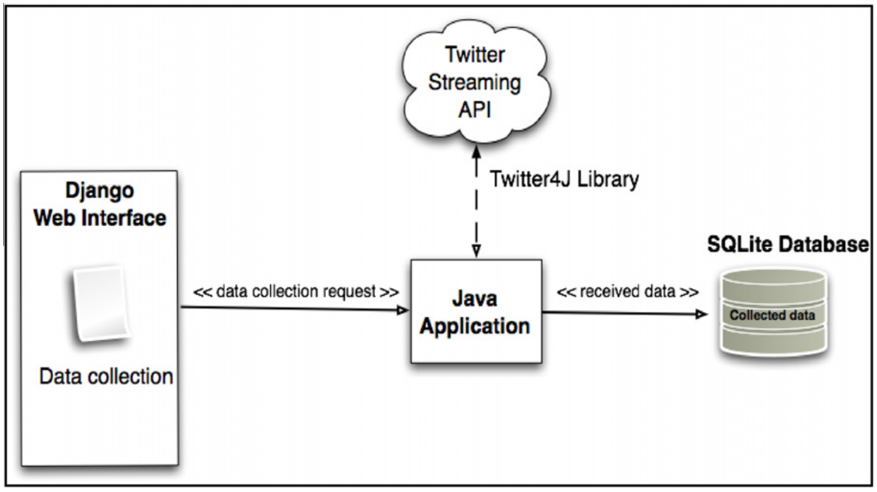


Fig. 4. Architecture of the data collection software component.

cessing stage before the search mechanism is enabled. More specifically, all words of the text field of the Twitter data in the collected are matched with WordNet lexical database, if no match was found, it is matched to named-entity field. If no correspondence is found again, then the word is matched against existing Slang database. If a match is found the underlying term is substituted by the equivalent corresponding correct English word or phrase. See Fig. 3 for overall representation.

3.3. Data collection architecture

The Django Web Framework [7] was used for web application development. This allows us to use its GeoDjango extension which is particularly useful for constructing geographic web-applications as well as linking it with related web frameworks like Google maps, Javascript API, Apache SOLR search engine, etc.

The design of the data collection component separates the web-interface logic from heavy-duty background processes (for data collection and storage), which is considered to be a good practice in web application programming. This software, on one hand, connects to the Streaming API in order to collect data and stores it in a structured database. It uses the open-source Twitter4J library to enable java applications and communicate with various Twitter APIs including the Streaming API. On the other hand, the open-source JDBC driver is used in the command-line Java application to store tweets received from the Streaming API in a structured SQLite database. The web interface built using Django allows users specify geographic locations to be monitored, manage data collection processes and monitor their progress. It automatically spawns and manages instances of the command-line Java application which perform data collection in the background. Fig. 4 highlights the main components of the Twitter collection task.

3.4. Spatial data storage architecture and spatial search

The collected data is initially stored in a non-spatial SQLite database. The geo-location information (latitude/longitude co-ordinates) contained in this data is represented in a spatial format in order to enable spatial queries on the data.

In order to extend the spatial retrieval functionality offered to end-users, the software design integrates two geographic data sets provided by UK Ordinance Survey – Code-Point Open and 1:50 k scale gazetteer [9], which allows the retrieval of tweets using distance lookups from any post-code or place in the database. The data sets are made available in CSV format and the location of each post-code/place name is specified using the Easting/Northing co-ordinate system. The use of GeoDjango extension requires the application to store data in a spatial database. PostgreSQL database server with the PostGIS spatial extension was used to accommodate GeoDjango requirement [13,14]. The software system uses a custom python script to convert each tweet stored in the SQLite database to its equivalent Django model instance. PROJ.4 (Cartographic Projections Library) [27] is required by GeoDjango applications to convert geospatial data between different co-ordinate systems. Design and implementation of the Spatial Data Retrieval software component is straightforward after the conversion of Twitter and Ordinance Survey Data into geographic model objects, where their co-ordinates are represented by geometric PointField objects. Especially, GeoDjango's GeoQuerySet API provides a high-level, pythonic interface for querying geographic models based on their location [7]. Fig. 4 exhibits the main features of the spatial data retrieval architecture. In summary, GeoDjango provides a high level framework for retrieval and manipulation of spatial data. Fig. 5 highlights key features of the software data collection architecture.

In order to retrieve tweets originating from or within a given region, specified by the geographic co-ordinates of its bounding box, the pseudo code described in Algorithm 1 was used.

Algorithm 1. Retrieving tweets from bounding box

```
Input: bounding box parameters (latitude/longitude of South
  West and North East points of the box)
Output: All tweets originated from the delimited box
  // Create a bounding box Geometry object of type Envelope
  to store bounding box coordinates bbox = Envelope(swlng,
  swlat, nelng, nelat)
  // Retrieve all tweets from database which intersect the
  bounding box using the location__intersects operator of
  PostGIS tweets = tweets.filter(location__intersects=
  bbox.wkt)
END
```

Especially, the basis in Algorithm 1 is to use the location parameters of the tweets (latitude/longitude) and determine whether the intersection with the specified region returns a Null element or not. This can be achieved using a simple test on the latitude/longitude that

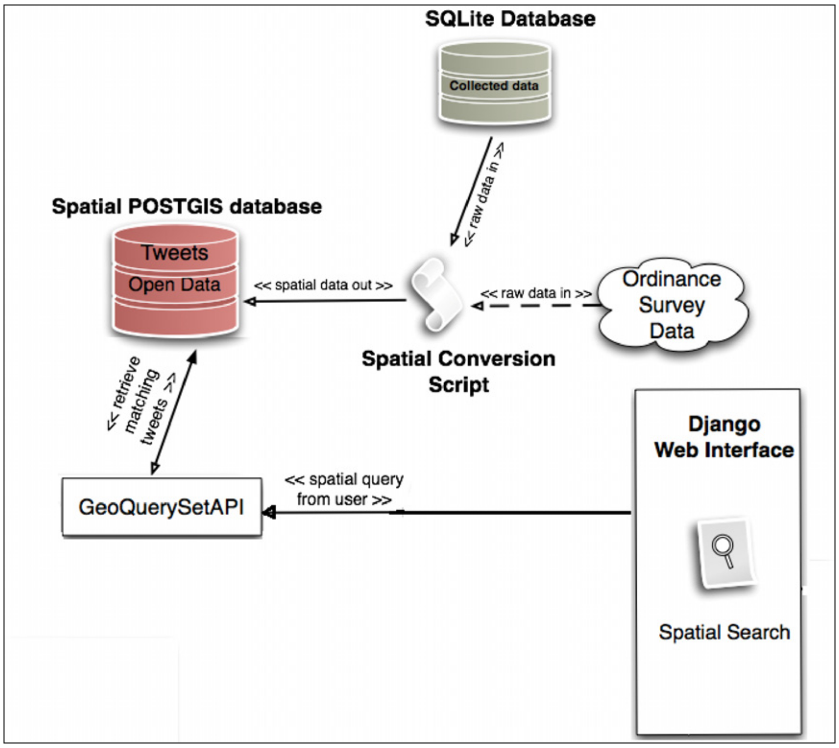


Fig. 5. Architecture of the spatial data storage and spatial search component.

determines whether its value is within the range specified by the bounding box and evaluate the formula of the great circle distance for each potential candidate. Besides, since the latitude and longitude coordinates are stored in separate columns with an index that supports range queries, the set of potential tweets falling in the bounding box are picked up straight away without the need to scan all the database. Algorithm 2 describes the retrieval of tweets originating from an arbitrary distance to a specified geographic location.

Algorithm 2. Retrieving tweets from arbitrary distance of a given location.

```
Input: location, distance from location
Output: tweets within the specified region
  // define an area object which encapsulates the location and
  Distance information area = (place.location,
  Distance(mi=distance))
  // Retrieve all tweets from database from within the
  specified distance of the location using the
// location__distance_lte operator
  tweets = Tweet.objects.filter(location__distance_lte=area)
END
```

It should be noted that the object area can be defined through its postcode, address, name of building, or any other attribute. External services provided by Ordinance Survey Data allow us to convert the place location into a single latitude/longitude point. For this purpose, it is quite common that even if the specified area is quite large, e.g., factory, the systems takes approximately its center of gravity as the representative point of interest. Therefore the use of the input distance parameter comes down to finding the set of tweets which intersect a circle whose center is the aforementioned point of interest and the radius the provided distance. So, the use of great circle allows us to determine the corresponding bounding box and thereby use the indexing to retrieve the corresponding tweets from the database.

Besides in order to guess place names from the user text input, the ‘Place’ box uses Asynchronous Javascript and XML search to lookup available ‘Places’ (from the 1:50 k gazetteer database embedded within the application) after each key stroke. The purpose of this feature is to inform users about available ‘places’ in the database as soon as they start typing, since this may not be obvious to them, see Fig. 6.

In addition to the spatial search, the system is also embedded with search capabilities to retrieve most of Twitter attributes available through Streaming API. This includes:

– Screen name search, which retrieves screen names from the database that match the query. The result is straightforward since all attributes of the database are efficiently indexed using Lucene’s indexing mechanism.
– Retrieve tweets originated from a specified screen name.
– Retrieve of replies. This function accepts a screen-name as input and returns all tweets in the database that were posted in response to the given screen-name.

3.5. Full text search and slang expansion

Although Django’s built-in QuerySet API can be used to perform textual queries on models, full-text search must be implemented to assist end users with textual analysis, speed-up queries on the data sets (which are expected to be very large), and to produce a broader set of results for each keyword specified by the user.

The proposal system uses Apache SOLR to provide full text search functionality to end-users. The search engine is integrated in the software system design using the open-source Haystack

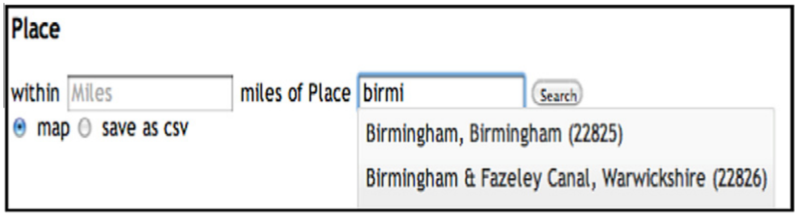

Place
within
Miles
miles of Place
birmi
Search
map
save as csv
Birmingham, Birmingham (22825)
Birmingham & Fazeley Canal, Warwickshire (22826)

Fig. 6. Place lookup using user's search.

API for Django projects [7]. The Haystack API provides a high-level, pythonic interface for indexing Django model data with Apache SOLR. On the other hand, Haystack reduces the daunting task of creating and managing data indexes to just a few lines of python code. Haystack also provides the SearchQuerySet API, which allows the application to retrieve indexed data. Consequently, the system substantially benefits from the advanced Lucene/SOLR advanced indexing and searching capabilities [18]. On the other hand, WordNet lexical database [10] was used to deal with the semantic aspect of the query.

3.5.1. Semantic analysis

The initial query is expanded to include all semantically related terms to each word of the query. This involves the use of WordNet lexical database. More formally, since WordNet itself can be described as a directed graph G(V, E), where vertices set V represents a set of synsets and edge set E represents a set of directed semantic links regardless of their type. Each node represents one synset, which consists of a set of words of semantically equivalent meaning, where each synset has links to other synsets as well. These links are of different types, e.g. hypernym, hyponym, antonym, meronym, etc. For instance, if one considers the hyponymy relationship, both *car* and *bus* are represented by nodes whose parent node is *transportation*. Similarly *tiger* is a child of a node *cat*, which itself has a parent node *animal*, etc. In this course, cat is hyponym of animal and animal is hypernym of cat, for example. A simple semantic similarity measure between two words can be formulated as the shortest distance [20], as indicated by the number of edges between the associated nodes, between these two words in the hierarchy of WordNet lexical taxonomy. Obviously, if these two words are synonyms, then they should belong to the same synset, and, therefore, the associated distance becomes null. While in the worst case scenario when the two words can only be related by the most upper root node in the taxonomy, e.g., object, fact, then the associated distance coincides with the depth of the taxonomy graph.

Now given the query words, a possible extension of the query is to add, for each word, its associated synonyms corresponding to all terms that belong to the same synset, as well as the set of direct hypernyms and hyponyms. The latter is equivalent to adding, for each word W, all words X such that

$$S(W,X) \leqslant 1 \tag{1}$$

where S stands for the semantic similarity quantified in terms of shortest distance in wordnet hierarchy when using either hyponymy or hypernymy relations.

It should be noted that the above strategy contrasts with standard approach of calculating semantic similarity between each pair of words pertaining to the given query and tweet message. The latter approach is computationally very expensive. On the other hand, the suggested approach allows us to use full Lucene's indexing capability to match the extended query with each element of database (tweet message) in efficient and effective way providing a better tradeoff between semantic analysis and computational com-

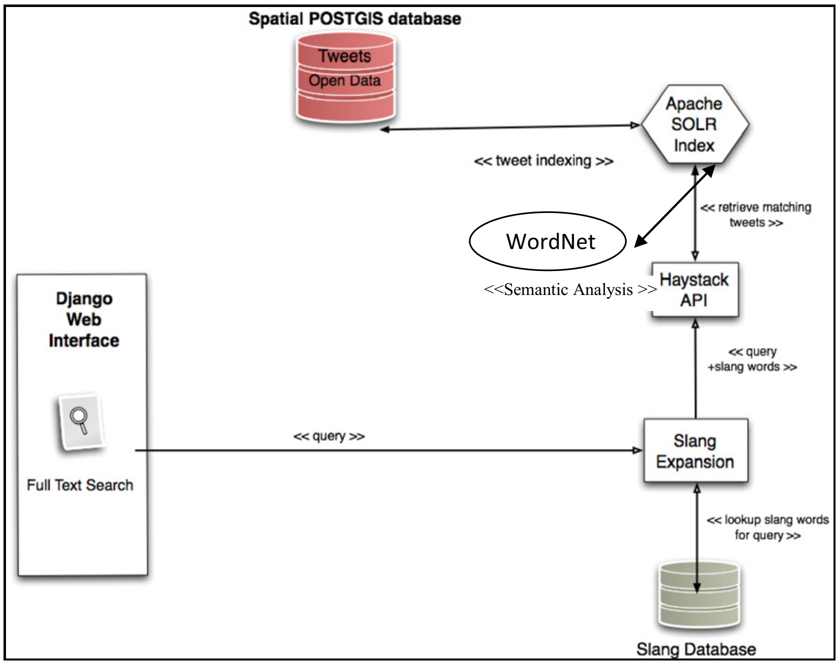


Fig. 7. Architecture of the full text search and slang expansion.

plexity. Besides, since the shortest path distance is positive natural number valued. That leaves the only two possibilities for constraint (1) as either S(W, X) = 0 which corresponds to a set of synonyms of word W or S(W, X) = 1 which corresponds to the set of hyponyms and hypernyms of word W. For instance, if the query contains the word "cat", then the query is expanded by (i) "true cat" as associated synset; (ii) animal, which is the associated hypernym, and; (iii) dog, tiger as examples of hyponyms. So the search mechanism will therefore match not only those words which are in the original word query but also those added as a result of the above query expansion through synonyms, hypernyms and hyponyms.

It should be noted that, for a given word, wordnet provides various synsets, which corresponds to various interpretations of the word. This work does not involve deep disambiguation study in order to cut the number of possibilities. A simple approach which has been employed consists of using the part of speech filter to filter out those synsets which do not have the same part of speech as the original word.

An alternative to (1) is to extend the hyponym/hypernym relationship to upper level, for instance up to two levels in the lexical WordNet hierarchy. This boils down to retrieving all hyponyms of a given word and then all hyponyms of each of the retrieved hyponym. This corresponds to find words X such that S(W, X) $\leqslant$ 2.

The implementation of the above reasoning can be made easy through the use of open source python-based wordnet modules: pywordnet (http://osteele.com/projects/pywordnet/).

3.5.2. Slang word analysis

Since Twitter enforces a strict 140 character limit on the size of each tweet, abbreviations and short forms of many words and phrases are widely used to preserve message space. For example,

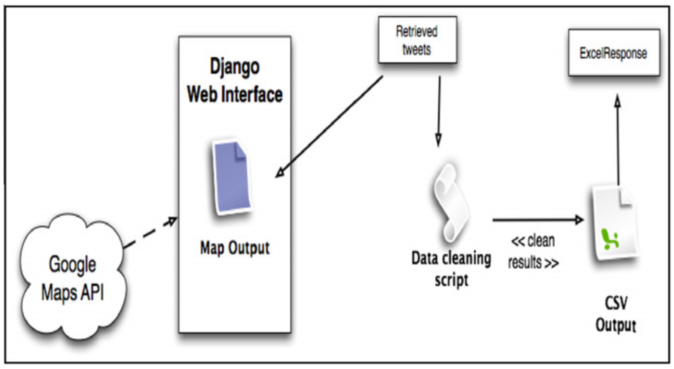


Fig. 8. Architecture of the data mapping and export.

the slang abbreviation 'IMHO' is used for shortening the phrase 'in my honest opinion'. Since most of these 'slang' abbreviations and short-forms are not present in dictionaries, they present a substantial challenge towards faithful analysis of Twitter content using natural language processing and data mining techniques.

The developed software system proposes a novel solution to this problem by incorporating a 'slang expansion' process in the textual search methodology. This process is based upon the integration of a 'slang' table into the application's database. This table contains over 5000 common slang words and abbreviations with their equivalent replacements in native English.

This stage is often computationally expensive as it requires scanning of all tweet messages and find whether there are any identifiable slang words that need substitution by the correct English word or phrase. Besides the term-slang matching also involves

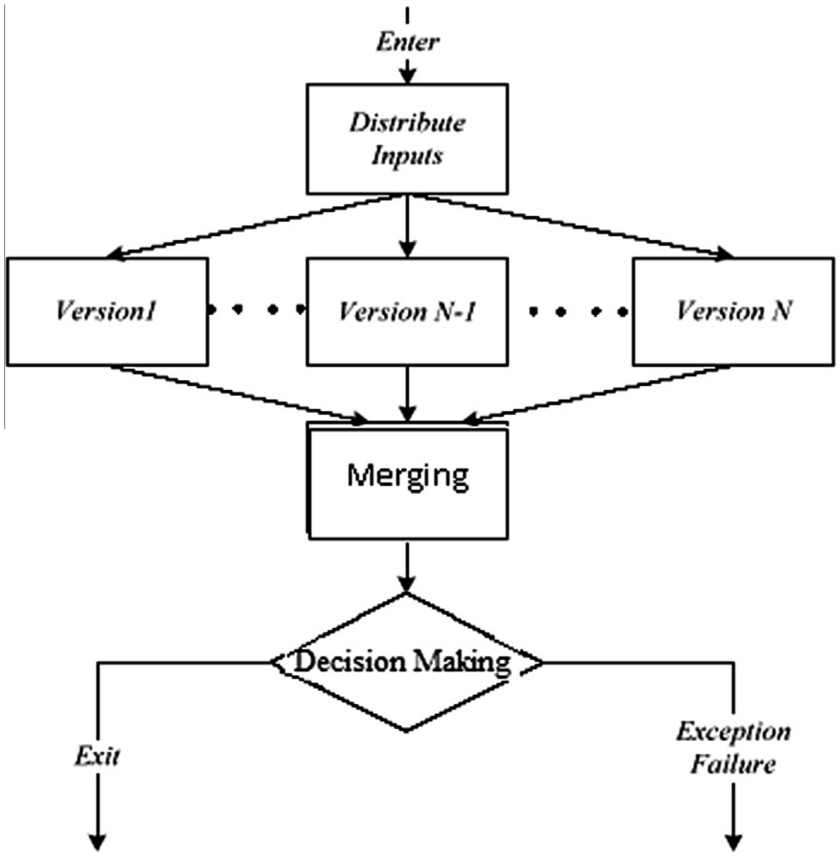


Fig. 9. Diversity-based fault tolerance.

testing whether the term is not part of some named-entity by calling upon the named-entity database; namely, ENEMAX tags developed for Message Understanding Conference [17]. A list of unmatched Slang words and unmatched named-entity words is provided to the developer to decide whether to include the new terms to the existing Slang database or not. However, once this stage is over (all tweet messages are cleaned up from Slang words), the search function becomes fast due to use of Lucene/Solr indexing mechanism. Fig. 7 provides overall architecture of text search and slang expansion.

3.6. Data mapping and export

This final component of the software system is the Data mapping and export component which is responsible for presenting results of spatial and textual queries to end-users. This architecture of this component is illustrated in Fig. 8.

Google Maps Javascript API is used within the template layer of the Django web application to display retrieved tweets to end users.

When the user chooses the export function for any set of retrieved tweets, a CSV file containing those tweets is returned to the user. This is done using the open-source ExcelResponse library for Django [7], which automatically converts any query-set object to a CSV file that is fully compatible with Microsoft Excel. CSV file was chosen as the preferred file-format for data export as it is compatible with a large number of external applications for data analysis and mining.

3.7. Fault tolerance

A key priority during the data collection period was to gather as much data as possible by avoiding outages and interruptions to the collection process. This cannot be excluded given the non-zero likelihood of internet connection or occurrence of some conflict among processes running on the server that would trigger the lost of connection. Therefore, the data collection software was designed to be robust and fault-tolerant to minimize the occurrence of such events. A common approach for this purpose is to use redundancy where several machines are used to collect information at the same time. Besides, in order to minimize the potential "data loss" due to external interruptions, in the same spirit as N-version programming fault tolerance [22], the employed machines are designed differently; namely, different operating systems: windows versus Apple for instance, different internet providers but same processor speed to guarantee the same processing time on average. This agrees with the function/design diversity principle, which stipulates that the likelihood of failure substantially if the system employs redundant modules differently designed. In this work, two machines were employed. One is based at university campus and using windows operating system while the second machine is based at home and uses Mac operating system with wireless connection. A general scheme of the concept is highlighted in Fig. 9, which is adapted from [22].

However, the use of multiple machines and redundancy arises the issue of efficient comparison of large scale databases outputted by the machines. Indeed, one expects for instance that at the end of the collection process, distinct tweets from all the SQLite databases will be gathered into a single database so that tweets which are common to both database occur only once in the new database and those which are only captured by one single database will be systematically present in the new database with corresponding chronological order. The methodology used to perform the merge has low time complexity and can be reproduced using the following algorithm.

Algorithm 3. Merging redundant databases

```
Input: Separate SQLite Databases
Output: Single SQLite database
1. Create a new, empty SQLite database file 'New.db' and
   access it using the SQL command line.
2. Create a table called "tweets" in this database using
   command line
   sqlite> CREATE TABLE tweets (text, status_id,
   in_reply_to_user_id, in_reply_to_screen_name,
   in_reply_to_status_id, user_id, screen_name, name,
   statuses_count, friends_count, followers_count, lang,
   latitude, longitude, timestamp);
3. Attach all original databases to the process.
   sqlite> attach '/path/to/first/database.db' as database1;
   sqlite> attach '/path/to/s/database.db' as database2;
...
4. Export all tweets from each attached database into New.db
   sqlite> insert into New.tweets select * from database1
   sqlite> insert into New.tweets select * from database2
...
5. Create another empty database 'Final.db'
6. Repeat step 2 for this new database.
7. Import all distinct tweets from New.db into Final.db by
   comparing the unique status_id field of each tweet. This
   will discard duplicate entries using the command line:
   sqlite> insert into tweets select * from New.tweets where
   rowid in (SELECT min(rowid) from New.tweets group by
   status_id);
```

In order to assess the performance of the developed tweet collection task a small comparative study is carried out. In this feasibility study, we performed the task of gathering geo-tagged tweets in England region during the period February–March 2012 (25 days). The region is delimited by a loose approximated bounding box (which also includes part of Wales and Scotland). The results were compared with that obtained using TweeQL,[1] which provides a SQL like query interface on top of Twitter Streaming API, and Advanced Twitter Search, accessed by any user's Twitter account, where results were saved in user Twitter account. Table 2 summarizes the results in terms of number of tweets obtained using the three above methods throughout the 25 days period.

From the above, one notices the following:

– The developed approach provides a close result in terms of total number of tweets to that of Twitter advanced search option. Although, the exact number of expected tweets is not known, it is highly guessed that the developed approach is more realistic as the use of multiple machines would reduce the risk of system interruption which cannot be fully excluded in case of Twitter advanced search option case.
– All three methods use Twitter Streaming API, however, TweeQL batches records in groups of 1000 before inserting them into the database, which may would ultimately miss data when task is stopped before this threshold is reached, which explains partly the lower performance of this approach.

[1] Available as an open source distribution at https://github.com/marcua/tweeql.

Table 2
Comparison of Twitter collection.

Developed approach	TweeQL architecture	Twitter advanced search option
1980,203	1774,432	1979,489

– Because of Twitter restriction where only one single Twitter account uses Streaming API, several accounts have been used for this task. Therefore, time synchronization is an important challenge that faced this study. For this purpose, one used the reported timestamp that we convert to regular time scale (day-month-year) through online service. However, little

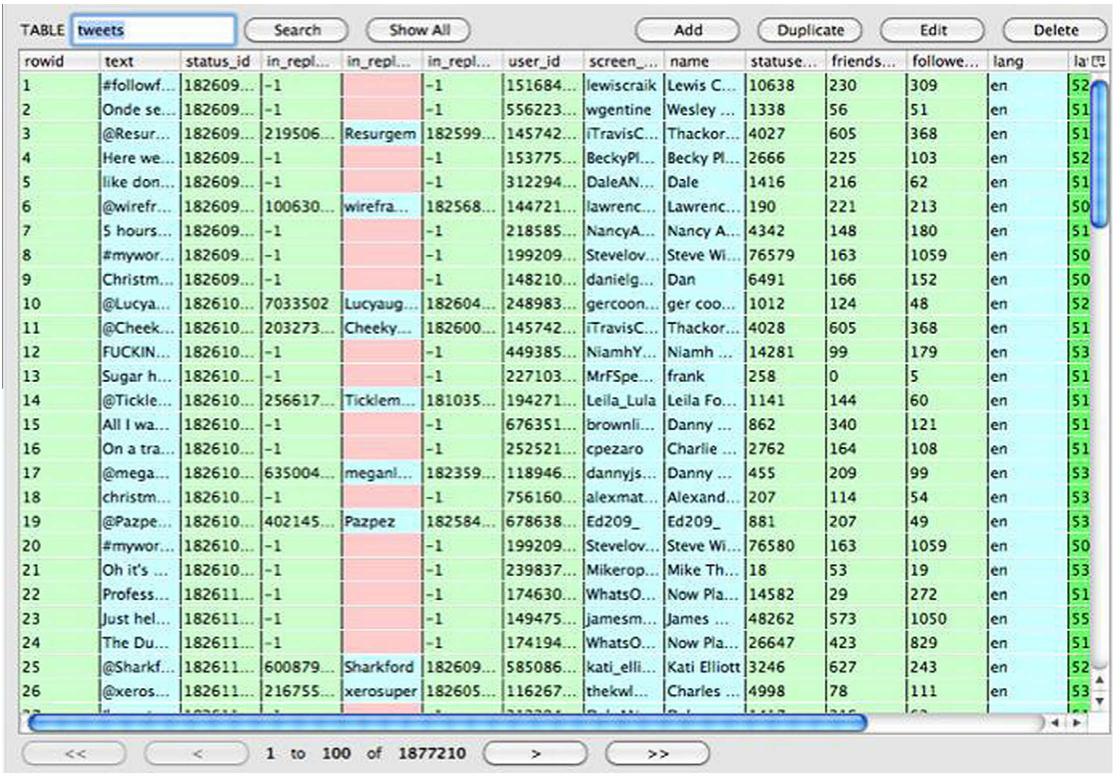
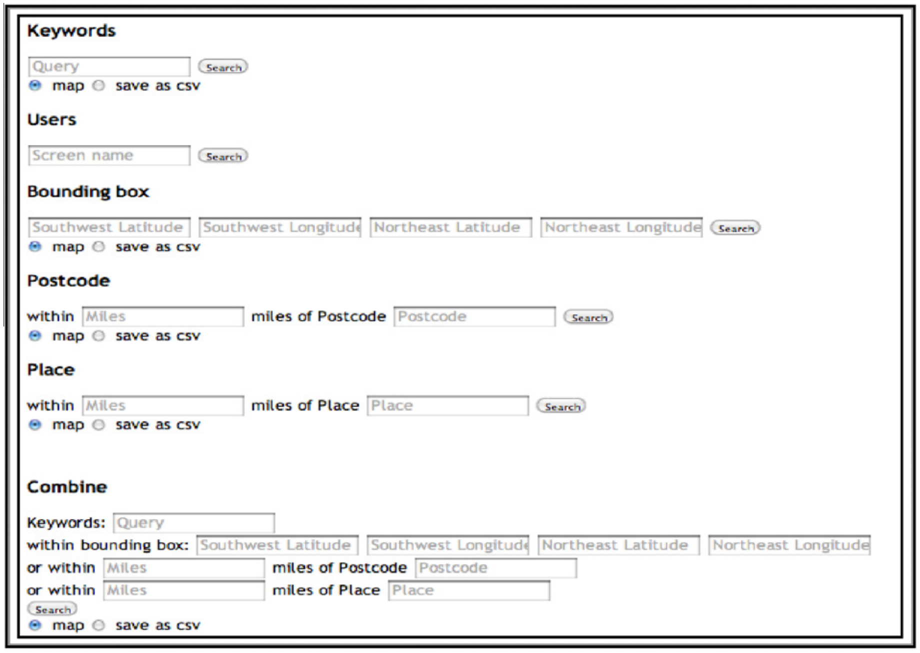


TABLE tweets

rowid	text	status_id	in_repl...	in_repl...	in_repl...	user_id	screen_...	name	statuse...	friends...	followe...	lang	la
1	#followf...	182609...	-1		-1	151684...	lewiscraik	Lewis C...	10638	230	309	en	52
2	Onde se...	182609...	-1		-1	556223...	wgentine	Wesley ...	1338	56	51	en	51
3	@Resur...	182609...	219506...	Resurgem	182599...	145742...	iTravisC...	Thackor...	4027	605	368	en	51
4	Here we...	182609...	-1		-1	153775...	BeckyPl...	Becky Pl...	2666	225	103	en	52
5	like don...	182609...	-1		-1	312294...	DaleAN...	Dale	1416	216	62	en	51
6	@wirefr...	182609...	100630...	wirefra...	182568...	144721...	lawrenc...	Lawrenc...	190	221	213	en	50
7	5 hours...	182609...	-1		-1	218585...	NancyA...	Nancy A...	4342	148	180	en	51
8	#mywor...	182609...	-1		-1	199209...	Stevelov...	Steve Wi...	76579	163	1059	en	50
9	Christm...	182609...	-1		-1	148210...	danielg...	Dan	6491	166	152	en	50
10	@Lucya...	182610...	7033502	Lucyaug...	182604...	248983...	gercoon...	ger coo...	1012	124	48	en	52
11	@Cheek...	182610...	203273...	Cheeky...	182600...	145742...	iTravisC...	Thackor...	4028	605	368	en	51
12	FUCKIN...	182610...	-1		-1	449385...	NiamhY...	Niamh ...	14281	99	179	en	53
13	Sugar h...	182610...	-1		-1	227103...	MrFSpe...	frank	258	0	5	en	51
14	@Tickle...	182610...	256617...	Ticklem...	181035...	194271...	Leila_Lula	Leila Fo...	1141	144	60	en	51
15	All I wa...	182610...	-1		-1	676351...	brownli...	Danny ...	862	340	121	en	51
16	On a tra...	182610...	-1		-1	252521...	cpezaro	Charlie ...	2762	164	108	en	51
17	@mega...	182610...	635004...	meganl...	182359...	118946...	dannyjs...	Danny ...	455	209	99	en	53
18	christm...	182610...	-1		-1	756160...	alexmat...	Alexand...	207	114	54	en	53
19	@Pazpe...	182610...	402145...	Pazpez	182584...	678638...	Ed209_	Ed209_	881	207	49	en	53
20	#mywor...	182610...	-1		-1	199209...	Stevelov...	Steve Wi...	76580	163	1059	en	50
21	Oh it's ...	182610...	-1		-1	239837...	Mikerop...	Mike Th...	18	53	19	en	53
22	Profess...	182611...	-1		-1	174630...	WhatsO...	Now Pla...	14582	29	272	en	51
23	Just hel...	182611...	-1		-1	149475...	jamesm...	James ...	48262	573	1050	en	55
24	The Du...	182611...	-1		-1	174194...	WhatsO...	Now Pla...	26647	423	829	en	51
25	@Sharkf...	182611...	600879...	Sharkford	182609...	585086...	kati_elli...	Kati Elliott	3246	627	243	en	52
26	@xeros...	182611...	216755...	xerosuper	182605...	116267...	thekwl...	Charles ...	4998	78	111	en	53

1 to 100 of 1877210

Fig. 10. SQLite database containing sample data collected using Harvest.jar.


Keywords

Query [Search]
[✓ map] save as csv

Users

Screen name [Search]

Bounding box

Southwest Latitude | Southwest Longitude | Northeast Latitude | Northeast Longitude [Search]
[✓ map] save as csv

Postcode

within Miles miles of Postcode Postcode [Search]
[✓ map] save as csv

Place

within Miles miles of Place Place [Search]
[✓ map] save as csv

Combine

Keywords: Query
within bounding box: Southwest Latitude | Southwest Longitude | Northeast Latitude | Northeast Longitude
or within Miles miles of Postcode Postcode
or within Miles miles of Place Place
[Search]
[✓ map] save as csv

Fig. 11. Screenshot of system interface for tweet collection and search.

imperfections pervading such conversion cannot be fully excluded. Therefore, results of Table should only be taken on some relative scale.

4. System interface and outcomes

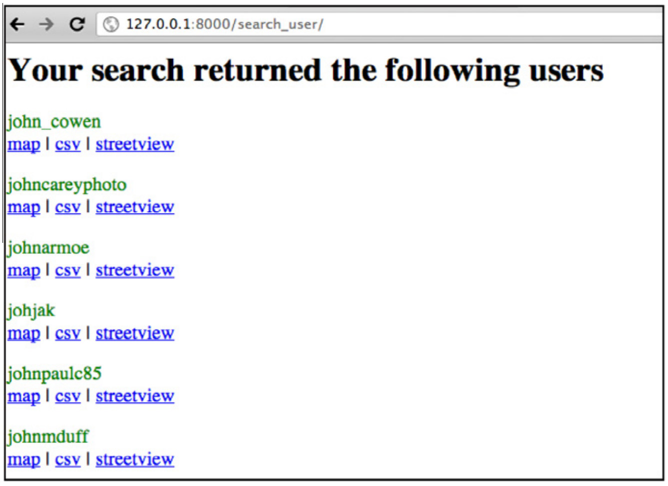
The developed Java application uses the open-source Twitter4J library to connect to the Streaming API for monitoring tweets originating from a specified region. It also utilizes the JDBC SQLite driver to create and update database files for storing received tweets. The SQLite database in which the Harvest.jar application stores collected data contains a 'tweets' table whose structure is represented by the following SQL statement:

create table tweets (text, status_id, in_reply_to_user_id, in_reply_to_screen_name, in_reply_to_status_id, user_id, screen_name, name, statuses_count, friends_count, followers_count, lang, latitude, longitude, timestamp);

Each row in this table corresponds to a data element in the status object returned from the Streaming API.

The screen-shot highlighted in Fig. 10 shows an instance of SQLite database containing some sample data collected using the Harvest.jar application. These attributes were summarized earlier in Table 1.

The interface presents drop-down lists of database files and Twitter credentials already stored within the application. Users can also create new instances of these required parameters using the + sign shown next to them. Each template file in the application that deals with data input contains error checking routines which present users with helpful error messages. The following screenshot shows how error messages are displayed by the create.html template when inputs are missing or invalid.


127.0.0.1:8000/search_user/

Your search returned the following users

john_cowen
map | csv | streetview

johncareyphoto
map | csv | streetview

johnarmoe
map | csv | streetview

johjak
map | csv | streetview

johnpaulc85
map | csv | streetview

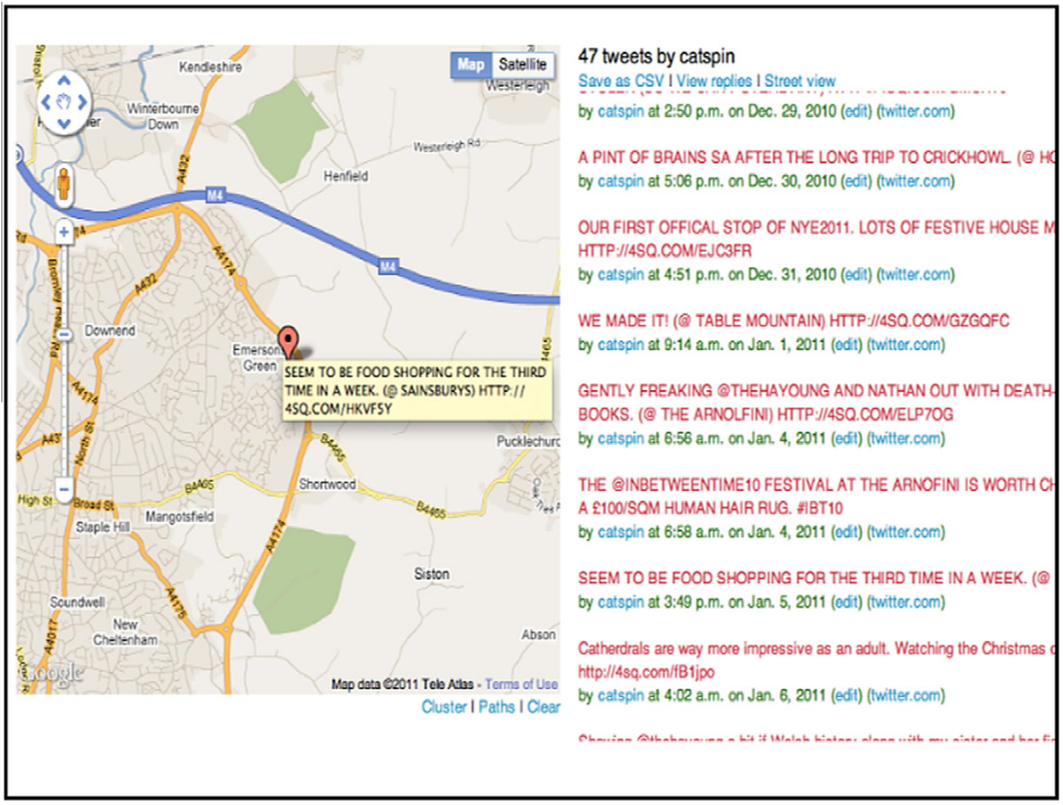
johnmduff
map | csv | streetview

Fig. 12. User search results template.

A screenshot of the interface used to collect user's tweets is provided in Fig. 11.

The template file contains error-checking routines that are triggered when invalid inputs are entered in the search forms. Helpful error messages are displayed to users whenever an exception is raised. For example, if the 'distance' parameter is not specified during 'Postcode search', the system outputs an html page ''Distance Not specified''.

Results of user search are provided in a paginated format with the user_search_results.html template. A list of Twitter users from the database whose screen name matches the search query are dis-



SEEM TO BE FOOD SHOPPING FOR THE THIRD TIME IN A WEEK. (@ SAINSBURYS) HTTP://4SQ.COM/HKVF5Y

47 tweets by catspin
Save as CSV | View replies | Street view

by catspin at 2:50 p.m. on Dec. 29, 2010 (edit) (twitter.com)

A PINT OF BRAINS SA AFTER THE LONG TRIP TO CRICKHOWL. (@ H
by catspin at 5:06 p.m. on Dec. 30, 2010 (edit) (twitter.com)

OUR FIRST OFFICAL STOP OF NYE2011. LOTS OF FESTIVE HOUSE M
HTTP://4SQ.COM/EJC3FR
by catspin at 4:51 p.m. on Dec. 31, 2010 (edit) (twitter.com)

WE MADE IT! (@ TABLE MOUNTAIN) HTTP://4SQ.COM/GZGQFC
by catspin at 9:14 a.m. on Jan. 1, 2011 (edit) (twitter.com)

GENTLY FREAKING @THEHAYOUNG AND NATHAN OUT WITH DEATH-
BOOKS. (@ THE ARNOLFINI) HTTP://4SQ.COM/ELP7OG
by catspin at 6:56 a.m. on Jan. 4, 2011 (edit) (twitter.com)

THE @INBETWEENTIME10 FESTIVAL AT THE ARNOFINI IS WORTH C
A £100/SQM HUMAN HAIR RUG. #IBT10
by catspin at 6:58 a.m. on Jan. 4, 2011 (edit) (twitter.com)

SEEM TO BE FOOD SHOPPING FOR THE THIRD TIME IN A WEEK. (@
by catspin at 3:49 p.m. on Jan. 5, 2011 (edit) (twitter.com)

Catherdrals are way more impressive as an adult. Watching the Christmas
http://4sq.com/fB1jpo
by catspin at 4:02 a.m. on Jan. 6, 2011 (edit) (twitter.com)

Fig. 13. Map.html interface.

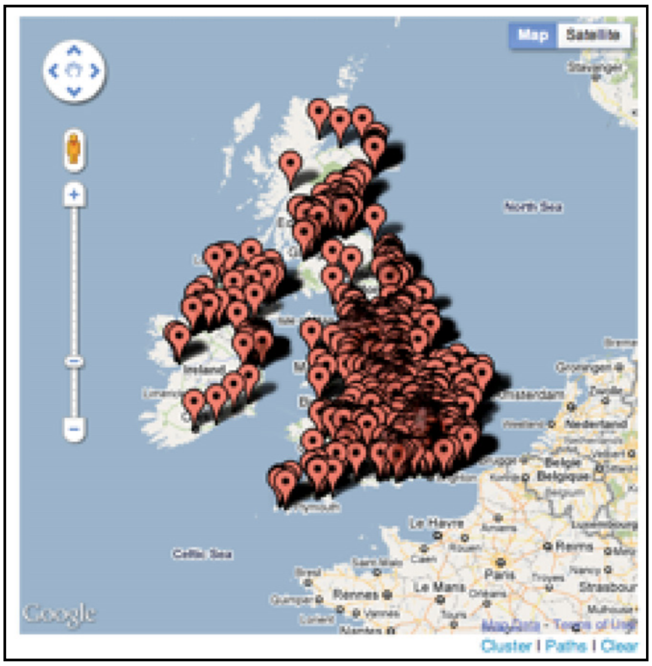


Fig. 14. Clustered search results for the keyword 'health'.

played. Each result contains hyperlinks for display the users tweets on a map or exporting them in CSV format (see Fig. 12).

The map template is used by numerous view functions to provide mapping functionality. The interface is split into a map on the right hand side and a sidebar on the left hand side, which shows the textual content of the tweets and provides a number of hyperlinks as exemplified in Fig 13.

Each time the map.html template is loaded, the map is automatically centered at the first retrieved tweet and an appropriate zoom level is calculated based upon the spatial bounds of the set of tweets supplied to the map. This is done by adjusting the "bounds" property of the Google map in the initialize() javascript function defined in maps.html.

Clicking on a tweet in the sidebar causes the map to automatically pan to the location of that tweet and displays a 'marker' object (google.maps.Marker) representing that tweet. This click-handling functionality is coded in the activateTweets() function defined in maps.html.

Clicking on the cluster hyperlink (underneath the map) calls the javascript function cluster() defined in maps.html. This function displays markers for all retrieved tweets at once, see Fig. 14.

Clicking the *paths* hyperlink (underneath the map) calls the javascript function *paths*() defined in maps.html. This function plots a 'polyline' object (google.maps.Polyline) between each retrieved tweet. The path begins at the earliest posted tweet and ends at the latest one. This function is particularly helpful when viewing tweets posted by a single user as it helps visualize the geographic path traversed by a user over time. Fig. 15 highlights for example the path followed by a specific user (@catspin) in the database when posting his 47 tweets.

The *view_replies* hyperlink shown in Fig. 13 enables the system to search the database for all tweets made in reply to the specified user and loads them on a new map.

While clicking the *street_view* hyperlink causes the map to load the streetview.html template. This is identical to the maps.html template except that it uses the google.maps.StreetViewPanorama projection instead of the roadmap projection used in the maps.html template. The street view template allows the user to inspect tweets in photographic format. An example is shown in Fig. 16.

5. Conclusion

This paper described a successful design and implementation of a software system that enabled us to collect a large amount of geolocated Twitter data on live and enabled various search functionalities on the database.

The proposal builds on the assumption that the real time constraint is softened which would allow us to employ more advanced semantic analysis. More specifically, the proposed software architecture is developed on Python-based web framework Django. The software system is integrated with Apache Lucene/SOLR system, which allows the architecture to benefit from the highly effi-

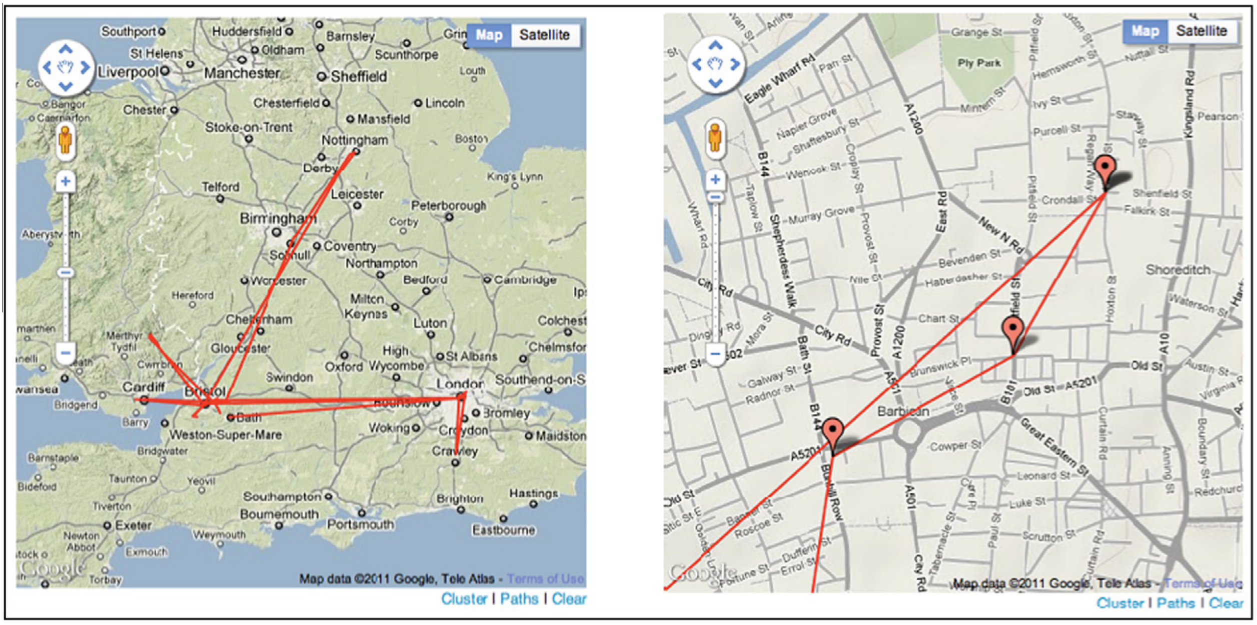
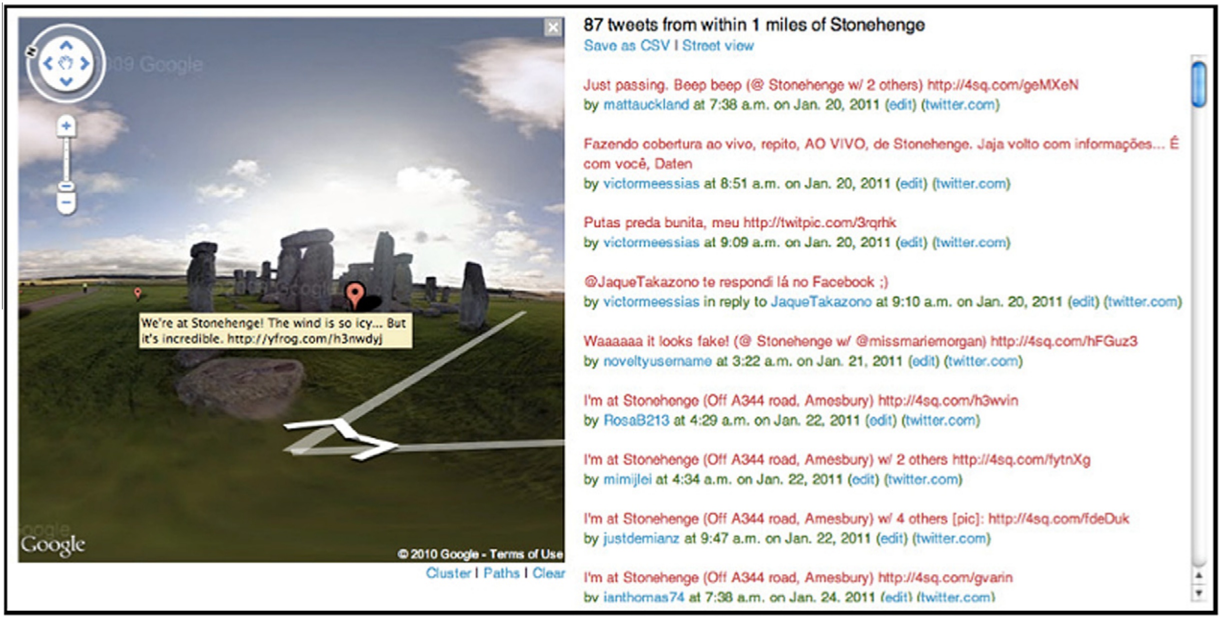


Fig. 15. Path traversed by a specific user while posting his tweets.


We're at Stonehenge! The wind is so icy... But it's incredible. http://yfrog.com/h3nwdyj
Google
© 2010 Google - Terms of Use
Cluster | Paths | Clear

87 tweets from within 1 miles of Stonehenge
Save as CSV | Street view

Just passing. Beep beep (@ Stonehenge w/ 2 others) http://4sq.com/geMXeN
by mattauckland at 7:38 a.m. on Jan. 20, 2011 (edit) (twitter.com)

Fazendo cobertura ao vivo, repito, AO VIVO, de Stonehenge. Jaja volto com informações... É com você, Daten
by victormeessias at 8:51 a.m. on Jan. 20, 2011 (edit) (twitter.com)

Putas preda bunita, meu http://twitpic.com/3rqrhk
by victormeessias at 9:09 a.m. on Jan. 20, 2011 (edit) (twitter.com)

@JaqueTakazono te respondi lá no Facebook ;)
by victormeessias in reply to JaqueTakazono at 9:10 a.m. on Jan. 20, 2011 (edit) (twitter.com)

Waaaaaa it looks fake! (@ Stonehenge w/ @missmariemorgan) http://4sq.com/hFGuz3
by noveltyusername at 3:22 a.m. on Jan. 21, 2011 (edit) (twitter.com)

I'm at Stonehenge (Off A344 road, Amesbury) http://4sq.com/h3wvin
by RosaB213 at 4:29 a.m. on Jan. 22, 2011 (edit) (twitter.com)

I'm at Stonehenge (Off A344 road, Amesbury) w/ 2 others http://4sq.com/fytnXg
by mimijlei at 4:34 a.m. on Jan. 22, 2011 (edit) (twitter.com)

I'm at Stonehenge (Off A344 road, Amesbury) w/ 4 others [pic]: http://4sq.com/fdeDuk
by justdemianz at 9:47 a.m. on Jan. 22, 2011 (edit) (twitter.com)

I'm at Stonehenge (Off A344 road, Amesbury) http://4sq.com/gvarin
by ianthomas74 at 7:38 a.m. on Jan. 24, 2011 (edit) (twitter.com)

Fig. 16. Tweets within 1 mile of the Stonehenge displayed using street view projection.

cient inverted index representation of Lucene instead of the relational representation of the original Twitter architecture.

One notices that there is a stream in Twitter development community that attempts since 2010 to migrate Twitter architecture to Lucene while modifying some of the core functions of Lucene in order to optimize its real time requirement (http://engineering.twitter.com/2010/10/twitters-new-search-architecture.html). In order to handle the semantic aspect of the search, we first integrated a flexible Slang database into the original system which allows the system to substitute correct English phrases to the identified slangs in tweets, and second, we used wordnet lexical database and short path based semantic similarity to expand the original query to synonyms, hyponyms and hypernyms of each query word. Finally, by applying advanced Lucene's query handling, word and phrase based query with various logical constraints are enabled efficiently. While geo-location queries are handled by integrating two extra libraries; namely, UK Ordinance Survey Code-Point Open and 1:50 k scale gazetteer, which allow the system to integrate the latitude/longitude coordinates of UK postcodes and place names. Therefore, the combination of PostgreSQL database server with the PostGIS spatial extension used to accommodate GeoDjango requirement allows the system to handle efficiently geo-location queries. The developed interface also allows the user to retrieve user profiles, street view using Google APIs and track the location of the different tweets of the same user. A fault tolerant system has been enforced for the task of Twitter collection by using separate machines at the same time for collection purpose, and then merging the two databases into a single one.

The sophisticated mapping and retrieval features built into the system make it a versatile tool for cross-disciplinary research involving Twitter. In the preliminary analysis that we carried out, we were able to identify applications of this system in areas of National Health, Heritage, Transport and Tourism, which are actually under investigations.

Acknowledgment

This work is partly supported by EPSRC Bridging the Gap Project 2009–2011.

References

[1] K. Anderson, A. Schram, Design and implementation of a data analytics infrastructure in support of crisis informatics research, in: Proceedings of the 33rd International Conference on Software Engineering, Hawaii, 2011, pp. 844–847.
[2] D.A. Bader, K. Madduri, Parallel algorithms for evaluating centrality indices in real-world networks, in: Proceedings of the 35th Proceeding on Parallel Processing (ICPP'2006), 2006, pp. 497–504.
[3] B. Bahmani, A. Chowdhury, A. Goel, Fast incremental and personalized page rank, in: Proceedings of the VLDB Endowment, Vol. 4, No. 3, 2010, pp. 173–184.
[4] M.S. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, E.H. Chi. Eddi, Interactive topic-based browsing of social status streams, in: Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology, 2010, pp. 303–312.
[5] A. Bifet, Richard Kirkby, Data Stream Mining. A Practical Approach, The University of Waikato Press, MOA, 2009.
[6] Code-Point® Open data.gov.uk, 2011. <http://data.gov.uk/dataset/os-code-point-open> (May 2011).
[7] Django Documentation. <http://docs.djangoproject.com/en/dev/ref/contrib/gis/tutorial/> (accessed May 2011).
[8] J. Dorsey, Just Setting up my twttr, 2006. <http://twitter.com/#!/jack/status/20> (accessed May 2011).
[9] Dracos Open Data Gazetteer – GitHub. <https://github.com/dracos/opendata-gazetteer/> (accessed May 2011).
[10] C. Falbium, WordNet, An Electronic Lexical Database (Language Speech and Communication), MIT Press, 1998.
[11] R.T. Fielding, Architectural Styles and the Design of Network Based Software Architecture, Ph.D. Thesis, University of California, 2000.
[12] L. Freeman, Visualizing social networks, Journal of Social Structure 1 (1) (2000). <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.6279&rep=rep1&type=pdf>.
[13] GEOS API, Django Documentation. http://docs.djangoproject.com/en/dev/ref/contrib/gis/geos/#django.contrib.gis.geos.GEOSGeometry (accesses May 2011).
[14] Geometry Engine Open Source. <http://trac.osgeo.org/geos/> (accessed May 2011).
[15] A. Green, Twitter API Programming Tips, Tutorials, Source Code Libraries and Consulting. <http://140dev.com/twitter-api-programming-tutorials/twitter-api-database-cache/> (accessed May 2011).
[16] Twitter API Programming Tips, Tutorials, Source Code Libraries and Consulting, <http://140dev.com/free-twitter-api-source-code-library/twitter-database-server/code-architecture/> (accessed May 2011).
[17] R. Grishman, Beth Sundheim, Message understanding conference: a brief history, in: Proceedings of the 16th International Conference on Computational Linguistics (COLING), Copenhagen, 1996, pp. 466–471.
[18] E. Hatcher, O. Gospodnetic, M. McCandless, Lucene in Action, second ed., Manning Publication, Stamford, USA, 2010.
[19] A. Java, X. Song, T. Finin, B. Tseng, Why we twitter: understanding microblogging usage and communities, in: Proceedings of the Joint 9th WebKDD and 1st SNA-KDD Workshop on Web Mining and Social Network, Analysis, 2007, pp. 56–65.

[20] J. Jiang, D. Conrath, Semantic similarity based on corpus statistics and lexical taxonomy, in: Proceedings of International Conference on Research in Computational Linguistics, 1997, pp. 19–33.

[21] B. Krishnamurthy, Phillipa Gill, Martin Arlitt, A few chirps about Twitter, in: Proceedings of the 1st ACM Workshop on Online Social Networks, 2008, pp. 19–24.

[22] B. Littlewood, P. Popov, L. Strigini, Modelling software design diversity: a review, ACM Computing Surveys 33 (2) (2001) 177–208.

[23] A. Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, Robert C. Miller, TwitInfo: aggregating and visualizing microblogs for event exploration, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, Vancouver, 2011, pp. 227–236.

[24] M. Milian, Twitter Sees Earth-shaking Activity During SoCal Quake. <http://latimesblogs.latimes.com/technology/2008/07/twitter-earthqu.html> (accessed 2008).

[25] L. Page, S. Brin, R. Motwani, T. Winograd, The Pagerank Citation Ranking: Bringing Order to the Web, in.scientificcommons.org, January 1998. <http://en.scientificcommons.org/42893894>.

[26] R.D.W. Perera, S. Anand, K.P. Subbalakshmi, R. Chandramouli, Twitter analytics: architecture, tools and analysis, in: Proceedings of Military Communication Conference, USA, 2010, pp. 2186–2191.

[27] PROJ.4, Cartographic Projections Library, <http://trac.osgeo.org/proj/> (accessed May 2011).

[28] T. Rosenstiel, B. Kovach, Warp Speed: America in the Age of Mixed Media, Twentieth Century Fund Publisher, 1999.

[29] T. Sakaki, M. Okazaki, Y. Matsuo, Earthquake shakes Twitter users: real-time event detection by social sensors, in: WWW '10: Proceedings of the 19th ACM International Conference on World Wide Web, New York, NY, USA, 2010, pp. 851–860.

[30] J. Sankaranarayanan, H. Samet, B. Teitler, M. Lieberman, J. Sperling, TwitterStand: news in tweets, in: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic, Information Systems, 2009, pp. 42–51.

[31] D. Shamma, L. Kennedy, E. Churchill, Tweetgeist: can the twitter timeline reveal the structure of broadcast events? in: Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2010), 2010, pp. 589–593.

[32] D.A. Shamma, L. Kennedy, E.F. Churchill, Statler: summarizing media through short-messaging services, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2010), 2010, pp. 551–552.

[33] SMS Traffic, Financial Express. <http://www.financialexpress.com/news/almost-200-000-smses-sent-every-second-itu/699606/> (December 2010).

[34] I. Sommerville, Software Engineering, seventh ed., Pearson Education Limited, Harlow, 2004.

[35] Twitter Blog. <http://blog.twitter.com/2011/03/numbers.html> (accessed May 2011).

[36] Twitter Rate Limiting, Twitter API. <https://dev.twitter.com/docs/rate-limiting> (accessed May 2011).

[37] Twitter Traffic, The Guardian. <http://www.guardian.co.uk/technology/blog/2009/jun/29/twitter-users-average-api-traffic> (accessed December 2010).