MoDisSENSE: A Distributed Spatio-Temporal and Textual Processing Platform for Social Networking Services

Ioannis Mytilinis [⋆] Ioannis Giannakopoulos [⋆] Ioannis Konstantinou [⋆]
Katerina Doka [⋆] Dimitrios Tsitsigkos [◇] Manolis Terrovitis [◇] Lampros Giampouras [‡]
Nectarios Koziris [⋆]

[⋆]CSLAB, NTUA
{gmytil,ggian,ikons,katerina, nkoziris}@cslab.ece.ntua.gr

[◇]IMIS, RC Athena
{tsitsigkosdim,mter}@imis.athena-innovation.gr

[‡]Athens Technology Center S.A., Athens, Greece
l.giampouras@atc.gr

ABSTRACT

The amount of social networking data that is being produced and consumed daily is huge and it is constantly increasing. A user's digital footprint coming from social networks or mobile devices, such as comments and check-ins contains valuable information about his preferences. The collection and analysis of such footprints using also information about the users' friends and their footprints offers many opportunities in areas such as personalized search, recommendations, etc. When the size of the collected data or the complexity of the applied methods increases, traditional storage and processing systems are not enough and distributed approaches are employed. In this work, we present MoDisSENSE, an open-source distributed platform that provides personalized search for points of interest and trending events based on the user's social graph by combining spatio-textual user generated data. The system is designed with scalability in mind, it is built using a combination of latest state-of-the art big data frameworks and its functionality is offered through easy to use mobile and web clients which support the most popular social networks. We give an overview of its architectural components and technologies and we evaluate its performance and scalability using different query types over various cluster sizes. Using the web or mobile clients, users are allowed to register themselves with their own social network credentials, perform socially enhanced queries for POIs, browse the results and explore the automatic blog creation functionality that is extracted by analyzing already collected GPS traces.

1. INTRODUCTION

The advent of Web 2.0 has brought an unprecedented data explosion on the web. The wide adoption of social networks has concluded in terabytes of produced data every day. In June 2014 for example, Facebook had on average 654 million mobile active users [6] on a daily basis. Tweets, Facebook "likes", Foursquare check-ins and Instagram pictures are just a part of the social-media data deluge.

As this data is mostly a product of human communication, it is susceptible to reveal relationships between different entities. This linkage of data provides tremendous opportunities for data analytics [15], predictions [17] and smart recommendation systems [**?**]. Taking advantage of the data access that social network APIs provide, developers launch their own third-party social applications to offer smart services and leverage user experience.

Personalization is undoubtedly the newest trend in social networks. Indeed, the most prevalent social networks launch everyday new services that exploit the social graph in order to provide personalized experience. A very recent such example is the new Foursquare and Swarm services that enable users to search for restaurants or other venues based on tastes and friend recommendations.

To this end, we offer a more detailed architectural overview and a comprehensive demonstration of MoDisSENSE [16] a social, geo-location system built on top of a distributed big-data enabled platform. MoDisSENSE combines heterogeneous data from various data sources, such as user GPS traces from cell phones, profile information and comments from existing friends in various social networks connected with the platform (MoDisSENSE currently supports Facebook, Twitter and Foursquare, but it can be extended to more platforms with the appropriate plugin implementation). Through distributed spatio-temporal and textual analysis, our system provides the following functionalities:

- Socially enhanced search of Points of Interest (*POIs*) based on criteria such as user location, automatic sentiment extraction of a user's social media friends preferences or a combination of the above.
- Automatic discovery of new POIs and trending events, i.e., spontaneous gatherings of people in a specific location, such as concerts, traffic jams etc.
- Inference of the user's semantic trajectory through the combination of her GPS traces with background information such as maps, check-ins, user comments, etc.
- Semi-Automatic extraction of a user's daily activities in the form of a blog.

A user can search for POIs on a bounding box on the map posing both simple as well as more advanced criteria. Simple criteria include commonly used features such as keywords

SIGMOD'15, May 31 - June 04, 2015, Melbourne, VIC, Australia

http://dx.doi.org/10.1145/2723372.2735375.

characterizing the POI or a time frame of interest. Advanced criteria refer to socially charged information such as the *interest* of a POI, i.e., the opinion of one's friends of it or its *hotness*, i.e., the crowd concentration over time in it. Thus, MoDisSENSE is capable of answering queries such as "Show me the top ten restaurants in Melbourne that (a specific subset, or all) of my Facebook friends prefer for dinner during summer" or "Show me the five hottest (i.e., most visited) places in town yesterday night".

The latter query is a *trending events* query. Tripadvisor already offers a trending events capability. However, MoDisSENSE adds a personalized flavor, making feasible personalized trending events queries with configurable time granularity. So MoDisSENSE can resolve the query "Show me the three hottest places in Melbourne visited by my x specific Foursquare friends the last y hours".

One additional feature of MoDisSENSE is the automatic POIs detection. A distributed version [**?**] of DBSCAN, a well-known clustering algorithm, is applied to MoDisSENSE users' GPS traces. A dense concentration of traces signifies a POI existence. Furthermore, the correlation of spatio-temporal information provided by the GPS traces with POI related texts automatically produces a daily blog with the user's activity. The produced blog can be manually updated by the user and can be shared in Facebook or Twitter.

The contribution of this work is manyfold:

- We devise a highly scalable architecture that efficiently handles data from heterogeneous sources and is able to deal with big data scenarios.
- We provide an Apache License-2.0 open-source [**?**] implementation of a social network based application which leverages the capabilities of other existing applications.
- We adapt and finely-tune well-known classification and clustering algorithms in a Hadoop-based environment.
- We experiment with datasets in the order of tens of GB, from Tripadvisor, Facebook, Foursquare and Twitter.
- We validate the efficiency, accuracy and scalability of the proposed architecture and algorithms.

The remainder of this paper is organized as follows: Section 2 presents the MoDisSENSE architecture, Section 3 provides an experimental evaluation of the platform, Section 4 describes the features to be presented to the demo attendees and Section **??** presents the related work.

2. ARCHITECTURE

MoDisSENSE features an architecture that is illustrated in Figure 1. In favor of flexibility and ease of maintenance, the system follows a completely modular design. Among system modules a fundamental distinction is applied: each module is classified as a frontend or a backend module.

The frontend consists of the web and two mobile (Android and iOS) applications. The web, Android and iOS clients are developed using DotNetNuke, Java Android SDK and Objective-C technologies respectively and can be downloaded from the github repository [**?**].

The frontend applications communicate with the backend through a REST API. A specific JSON format has been defined in order to send requests to the backend and return results to the user. This feature enables the seamless integration of more client applications with the platform.

As Figure 1 shows, the backend contains a set of processing and storage modules. For the processing modules, a web server farm and a Hadoop cluster are established. The

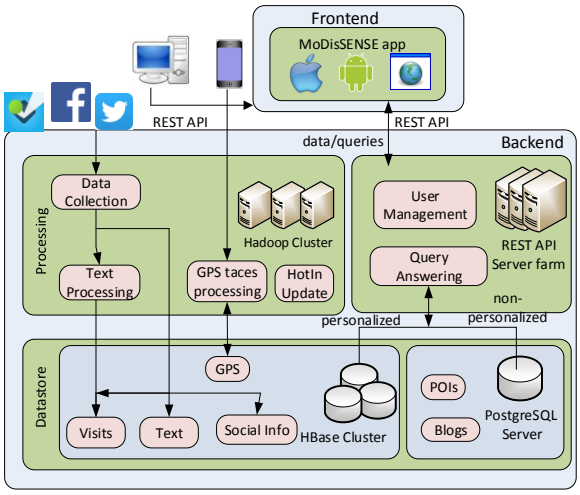

Figure 1: MoDisSENSE platform architecture.

widespread use of social networks has lead to an environment where huge amounts of data are created on high and unpredictable rates. Thus, the volume of data, needed to be processed, demands a distributed approach. Since the Hadoop framework emerges for large-scale analytics, we design and deploy the following Hadoop-based processing modules:

- Data Collection Module
- HotInt Update Module
- Text Processing Module
- Event detection module

Apart from the processing subsystem, a storage system is employed to the backend, in order to store the collected and derived data. We refer to the components of the storage subsystem as repositories. Repositories are conceptually classified to primitive and non-primitive data repositories. We consider primitive data to be the one have not been processed. Primitive data are collected from external data sources such as social networks(Facebook, Foursquare, Twitter) and GPS traces and are directly stored to the platform. Non-primitive data repositories, which are the ones serving answers to queries, hold information extracted from primitive data through spatio-textual algorithms.

When multiple concurrent users issue queries to the platform, system design should meet demand. Thus, there is a need for a scalable approach. To this end, the Apache HBase NoSQL datastore is used. However, there are queries which require either complex indexing schemes or extended random access to the underlying data. These queries cannot be efficiently executed in HBase. For this reason, we devise a hybrid architecture that uses HBase for batch queries that can be efficiently executed in parallel and PostgreSQL for online random-access queries that cannot. In addition to the already mentioned modules, we deploy the User Management and the Query Answering Module on the web server farm that acts as a gateway to the platform. Both modules are implemented as lightweight web services which put load to the datastore without stressing the web servers.

In the following subsections, we describe with further details the repositories and the processing modules of the MoDisSENSE platform.

2.1 Datastore Repositories

POI Repository. It contains all the information MoDisSENSE needs to know about POIs. The name of a POI, its geographical location, the keywords characterizing it and the hotness/interest metrics are all stored in this repository. A new entry to the repository can be inserted either explicitly by the user through the frontend GUI or automatically by the Event Detection Module. While POI repository has to deal with low insert/update rates, it should be able to handle heavy, random access read loads. The indexing capabilities PostgreSQL offers make it an ideal infrastructure for hosting POI repository.

Social Info Repository. This is a HBase-resident table where social graph information is held. For each MoDisSENSE user and for each connected social network, the list of friends is persisted. More specifically, we store a compressed list with the unique social network id, the name and the profile picture of each friend.

Text Repository. The textual data is the most demanding part, in terms of utilized disk space, in the MoDisSENSE ecosystem. For this reason , it is stored in the NoSQL cluster and is spread across all available cluster nodes. The Text repository holds all the collected comments and reviews about POIs. Texts are indexed by user, POI and time. For any given POI, we are able to retrieve the comments that a specified user made at any given time interval.

Visits Repository. Visits Repository is also persisted as an HBase table. In order to make POI recommendations based on social friends' preferences, we need to keep track of all visits of a user's friends. Each visit is represented by a struct with the complete POI information (name, latitude, longitude, etc). Moreover, this struct is enriched with the *interest* and *hotness* metrics. Every time a MoDisSENSE user or a user's social friend visits a POI, a visit struct indexed by user and time is added to the repository. Thus, for any given time interval, we know the places that all of a user's friends have been and a score indicating each friend's opinion.

Since the visit struct, that we persist each time someone visits a POI, contains the whole POI information, there is high replicated data. The alternative schema design strategy would be joining POI information with visit information at query time. However, our experiments suggest data replication to be more efficient. Our schema in combination with HBase coprocessors and a fully parallel query mechanism, it seems to offer more scalability and achieve lower latency numbers even when many concurrent users with many friends each stress the system.

GPS Traces Repository. The mobile devices, that have MoDisSENSE application installed, can push their GPS traces to the platform. Since the platform may continuously receive GPS traces, this repository is expected to deal with a high update rate. Furthermore, as GPS traces are not queried directly by the users but are periodically processed in bulk, there is no need to build indices on them. The volume of data, the opportunities for parallel bulk processing and the absence of indices are the main reasons why we choose to put GPS repository in HBase.

Blogs Repository. We define a semantic trajectory to be a timestamped sequence of POIs summarizing user's activity during the day. As POIs, blogs are frequently queried by users but they do not have to deal with heavy updates and thus are stored as a PostgreSQL resident table.

2.2 Processing Modules

User Management Module. The User Management module is responsible for the user authentication to the platform. The user is registered either through the mobile applications or the website. MoDisSENSE does not require a username or password. The signing-in process is carried out only with the use of the social network credentials. The registration workflow follows the OAuth protocol. The OAuth authorization framework enables a third-party application to obtain access to an HTTP service on behalf of a resource owner. When the authentication is successful, the user logs in and an access token is returned to the MoDisSENSE platform. With this token, MoDisSENSE can interact with the connected social networks on behalf of the end user. It can monitor user's activity, user's friends activity, make posts etc. Being an authorized member of the platform, the user can connect to the MoDisSENSE account more social networks. When more than one social networks are connected to the platform, MoDisSENSE joins the acquired data and enriches the information that is indexed and stored.

Data Collection Module. The functionality of this module is to collect data from external data sources. Periodically, the Data Collection Module scans in parallel all the authorized users of MoDisSENSE; each worker scans a different set of users. For each user and for all connected social networks, it downloads all the interesting updates from the user's social profile. Since MoDisSENSE provides social geo-location services, interesting updates are considered to be user check-ins and the accompanying comments as well as status updates. From this information, MoDisSENSE is able to gain knowledge about the existence of POIs and people's opinion about them. Once data is streamed to the platform, it is in-memory processed and then indexed and stored to the appropriate repositories.

Text Processing Module. The Text Processing Module performs sentiment analysis to all textual information the platform collects through the Data Collection Module. Comments from check-ins and POI reviews are classified, real-time and in-memory, as positive or negative. The score which results from the sentiment analysis is persisted to the datastore along with the text itself.

As a classification algorithm, we choose the Naive Bayes classifier that the Apache Mahout framework provides. Naive Bayes is a supervised learning algorithm and thus it needs a pre-annotated dataset for its training. For the training, data from Tripadvisor, containing reviews for hotels, restaurants and attractions, is used. The chosen dataset offers two key advantages: First, it is semantically close to our application data and thus results in a high quality training and second, Tripadvisor comments are annotated with a rank from 1 to 5 that can be used as a classification score. After an extensive experimental study and a fine-tuning of the algorithm parameters, we managed to create a highly accurate classifier that achieves an accuracy ratio of 94% towards unseen data.

Event Detection Module. New events and POIs detection constitute a core functionality of MoDisSENSE. A distributed, Hadoop-based implementation of the DBSCAN clustering algorithm [**?**] is employed for this reason. The module is called periodically and processes in parallel the updates of GPS Traces Repository in order to find traces of high density; high density traces imply the existence of a new POI. In order to avoid detecting already known POIs

to MoDisSENSE, traces falling near to existing POIs in POI Repository are filtered out and are not taken into consideration for clustering.

HotIn Update Module. POI Repository contains all the non-personalized information about POIs. Two attributes of the information stored for each POI are: *hotness* and *interest*. Hotness and interest are inferred by an aggregation over all visits persisted in Visits Repository within a configurable time frame T. In order to aggregate hotness and interest, a MapReduce job configured with a scanner over all visits in T, is instantiated. HotIn Update Module is called periodically in order to update the hotness/interest information of the POI repository in PostgreSQL.

Query Answering Module. The Query Answering is the module used for answering search queries. A search query can take as input the following parameters:

- a bounding box on the map
- a list of keywords
- a list of social network friends
- a time window
- results sorting criteria
- the number of results to be returned

If a list of friends is provided, the query is considered to be personalized. As Figure 1 shows, non-personalized queries are answered by PostgreSQL while personalized ones by the NoSQL cluster. A non-personalized query is a *select* SQL query in PostgreSQL, since POI repository contains all the demanded information.

In the case where some selected friends' opinion should be taken into account, MoDisSENSE should know whether the selected friends have visited any place in the area of interest and what score has been extracted by their visit. In order to efficiently obtain this information, HBase coprocessors are used. Each coprocessor is responsible for a region of the Visit Repository table and performs HBase *get* requests to the users under its authority. Since different friends are located with high probability in different regions, a different coprocessor is in charge of serving their visits and multiple get requests are issued in parallel. Increasing the regions number leads to increase in coprocessors number and thus achieves higher degree of parallelism within a single query.

3. EXPERIMENTS

In this section we provide experiments and validate both our architectural design and the selected optimizations for the training of the Naive Bayes classifier.

3.1 Performance Evaluation

We first present some experiments for the scalability and performance of the query answering module. Using a synthetic dataset, we test the ability of the platform to respond to personalized queries issued by its users for various loads and different cluster sizes. Each personalized query involves a set of friends; the opinion expressed by them for POIs they have visited will determine the score of each POI for a platform user. The platform user can set a number of other parameters as well: the POI type, as expressed by the keywords that accompany it, its geographical location, its name, the time period of the visits, etc. In our experiments, we identified that the dominant factor in the execution time of the query is the number of social network friends that the platform users define.

For the synthetic dataset generation, we collected information from OpenStreetMap about 8500 POIs located in Greece. Based on those POIs, we emulated the activity of 150k different social network users, each of whom has visited a number of POIs and assigned a grade to it; this grade corresponds to the classification grade of the comment of the user for this visit. The number of visits for each social network friend follows the Normal Distribution with $\mu = 170$ and $\sigma = 10$[1]. The dataset is deployed into an HBase cluster consisting of 16 dual-core VMs with 2 GB of RAM each, running Linux (Ubuntu 14.04). The VMs are hosted in a private Openstack cluster.

At first, we are going to study the impact of the number of social network friends into the execution time of a single query. At this point we will also examine how the cluster size affects the execution time of a query. Secondarily, we are going to extend our study to multiple concurrent queries where we will also examine the behavior of the platform for different number of concurrent queries and different cluster configurations.

For the first point, we evaluate the execution time of a query for different numbers of friends. In Figure **??** we provide our findings. In this experiment, we executed one query at a time involving from 500 to 10k SN friends for three different cluster setups consisting of 4, 8 and 16 nodes. The friends for each query are picked randomly in a uniform manner. We repeated each query ten times and we provide the average of those runs.

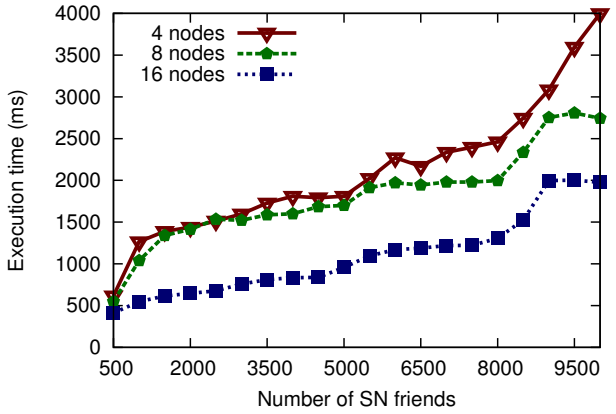

Figure 2: Query latency vs number of users

The number of friends affects the execution time in an almost linear manner. Furthermore, an increase in the cluster size leads to a latency decrease, since the execution is happening in parallel to multiple nodes. By utilizing HBase coprocessors, we managed to exploit the locality of the computations into specific portions of the data: each coprocessor operates into a specific HBase region (holding a specific portion of the data), eliminates the visits that do not satisfy the user defined criteria, aggregates multiple visits referring to the same POI and sorts the candidate POIs according to the aggregated scores. Finally, each coprocessor returns to the Web Server the sorted list of POIs which, in turn, merges the results and returns the final list of POIs to the end user.

Using the previously described technique, we achieved to get latencies lower than 1 second for more than 5000 users.

[1]The vast majority of the users has performed between 140 and 200 visits in different pois.

Bearing into consideration that social networks like Facebook, retain a limit on the maximum number of connections (5000 friends per user), we can guarantee that the latency for each query remains in an acceptable level for a real time application.

We now extend our analysis for the cases where multiple queries are issued concurrently to the platform. For our experiments we create a number of concurrent queries involving 6000 social network friends each and we measure their execution time for different cluster sizes. In Figure **??**, we provide our results. The execution time in the vertical axis represents the average execution time for each case.

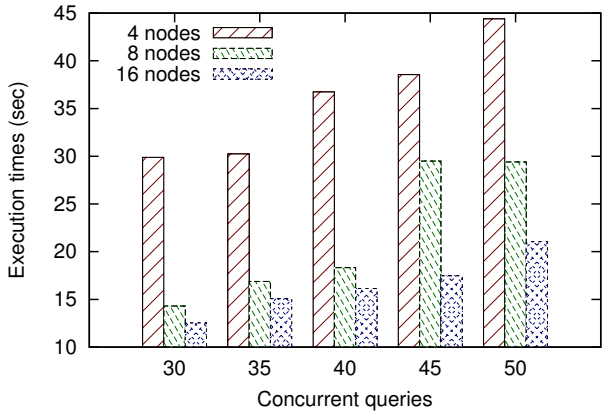

Figure 3: Average execution time for concurrent queries

As Figure **??** demonstrates, an increase in the number of concurrent queries leads to worse performance (larger execution time). However, for larger cluster sizes we can make the following observations: (a) even for the lowest number of concurrent queries the 16 cluster case is approximately 2.5 times better than the 4 cluster case, indicative of the proper utilization of more resources and (b) larger cluster sizes do not allow execution time to rise fast as the number of concurrent queries increases. Specifically, when the cluster consists of 4 nodes, the execution time is high even for the lowest number of concurrent queries and it continues to rise rapidly while the number of queries is increased. In the 8 nodes case, although at first the achieved execution times are relatively low, the increase becomes rapid for more concurrent queries, whereas in the 16 nodes case we see that the increase is held in a minimum level. This is indicative of the scalability of the platform, since more resources are properly utilized and the platform becomes resistant to concurrency.

Finally, since greater number of concurrent queries leads to more threads in the Web Server which, in turn, hits the cluster, we can avoid any potential bottlenecks by replicating the Web Servers while simultaneously, we use a load balancer to route the traffic to the web servers accordingly. In our experimental setup, we identified that two 4-cores web servers with 4 GB of RAM each are more than enough to avoid such bottlenecks.

3.2 Accuracy Evaluation

In this section we evaluate the tuning of the Naive Bayes classifier we use for sentiment analysis. As we have already mentioned, for the training of the classifier, we crawl and use data from Tripadvisor. We consider a Tripadvisor review about a place to be a classification document. We divide training documents into two sets: positive and negative opinion documents. Both sets should have almost the same cardinality. Before feed the training set to the classifier, a preprocessing step is applied which involves stemming, turning all letters to lowercase and removing all words belonging to a list of stopwords. When the preprocessing step is finished, Naive Bayes is applied to the data. Let this procedure be the baseline training process. As a next step, we experiment with the following optimizations: use of the *tf* metric, *2-grams*, *Bi-Normal Separation* and deletion of words with less than x occurrences. These optimizations can be given as parameters to the classification algorithm. Experiments with different combinations for the algorithm parameters were also conducted but are omitted due to space constraints.

Figure **??** shows the classification accuracy for various training set sizes, when the baseline and the optimized classification are used. We observe that when optimizations are applied, classification results are more accurate for any training size. Especially, for a training set of 500k documents, we achieve an accuracy of 93.8%. As we can see, the 500k documents form a threshold for the classifier. For both versions of the algorithm after this point accuracy degrades. This is because there is an overfit of data, which is a classic problem in Machine Learning approaches.

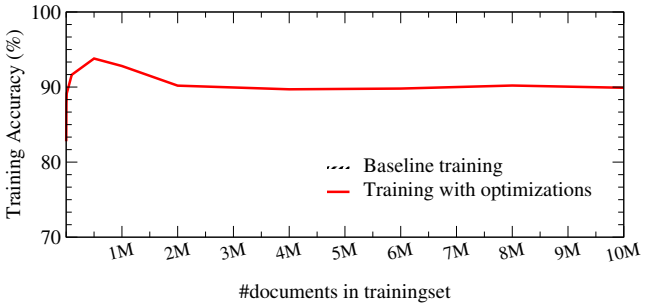

Figure 4: Classification accuracy for different training sizes

4. DEMO DESCRIPTION

The demonstration will allow attendees to interact with MoDisSENSE web and mobile clients and explore their functionality. In Figure **??** we present screenshots from the iOS client, nevertheless the Android and web clients share the same functionality and will also be available for demonstration. Attendees will be able to register themselves using their social network credentials and browse the respective clients. Due to lack of historical data and social context for the attendee accounts, we will utilize a set of demo accounts to showcase the personalized search functionality. In a nutshell, users will be able to perform the following: (a) Login and link their social network accounts with MoDisSENSE (Figure 2), (b) Perform personalized or not keyword search for POIs between a map bounding box for a time window and sorted by hotness or interest and (c) Explore the semi-automated blog creation, editing and publishing functionality.

In the first case the users will be able to join the MoDisSENSE platform by linking one or more of their Facebook, Twitter or Foursquare accounts. When the linkage is done, the social networking context (i.e., user and friend lists avatar

icons, etc) will be available at the respective client.

In the second scenario, we will showcase the personalized search functionality. Two MoDisSENSE users with completely different social profiles are going to perform the same search query on the same geographical area. The returned POIs are expected to be different depending on the user executing the query. For example, we assume that the first user's friends love fast food while the other's prefer luxurious restaurants. A personalized query with the keyword "restaurant" is expected to return fast food places in the case of the first user whereas luxurious expensive restaurants in the case of the second user.

For the third scenario, users will be able to create, edit and post their own blog entries, or examine the extracted blogs of the test accounts. The attendees will have the chance to see the inferred semantic trajectories and also edit the respective blog entries. Screenshots from the blog GUI are presented in Figure 3. In the first two screenshots the semantic trajectory of the user is presented. The leftmost screenshot permits the user to edit the order the POIs were visited while the one in the middle presents the trajectory on the map. Finally the rightmost screenshot allows the editing of visit information such as arrival and departure time.

5. RELATED WORK

This work is not the first that proposes a social network based geo-location service. In [**?**, **?**] two POI recommendation systems are presented. However, recommendations are based only on historical location data from GPS traces and check-ins, whereas MoDisSENSE combines spatio-textual data in order to produce more accurate recommendations. Furthermore, MoDisSENSE proves to be scalable while no implementation and performance information is provided for the other systems. In [**?**] a collaborative filtering based system is presented. The presented system collects check-in data from Gowalla, a location-based social network, and makes a POI recommendation. And in this system, recommendation is solely based on check-ins and not on the combination of heterogeneous data. The importance of the textual content of social networks as a mean of expressing opinion is already recognized. Many works such as [**?**, **?**, **?**] perform sentiment analysis on Twitter data in order to extract user sentiment. Nevertheless, none of these works combines the inferred knowledge with geo-location information for POI recommendation.

6. ACKNOWLEDGMENTS

This work has been partially funded by the Hellenic (GSRT) "COOPERATION 2009" National Action "09SYN-72-881" MoDisSense Project and the European Commission in terms of the ASAP FP7 ICT Project under grant agreement no 619706. Dr. Terrovitis and Mr. Tsitsigkos were partly supported by the GSRT in terms of the EU/Greece funded KRIPIS Action: MEDA Project.

7. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org.
[2] Apache HBase. http://hbase.apache.org.
[3] Apache Mahout. https://mahout.apache.org/.
[4] DotNetNuke. http://www.dnnsoftware.com/.
[5] Facebook API. https://developers.facebook.com/.
[6] Facebook Stats. http://newsroom.fb.com/company-info/.
[7] Foursquare API. https://developer.foursquare.com/.
[8] HBase Coprocessors. http://hbase.apache.org/book.html#coprocessors.
[9] MoDisSENSE Web App. http://modissense.gr/.
[10] OAuth. http://oauth.net/2/.
[11] Postgresql. http://www.postgresql.org/.
[12] Swarm. https://www.swarmapp.com/.
[13] Tripadvisor. http://www.tripadvisor.com/.
[14] xCode. https://developer.apple.com/xcode/.
[15] C. C. Aggarwal. *An introduction to social network data analytics*. Springer, 2011.
[16] I. Mytilinis, I. Giannakopoulos, I. Konstantinou, K. Doka, and N. Koziris. Modissense: A distributed platform for social networking services over mobile devices. *IEEE Big Data*, 2014.
[17] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe. Predicting elections with twitter: What 140 characters reveal about political sentiment. *ICWSM*, 10:178–185, 2010.

[t]0.50

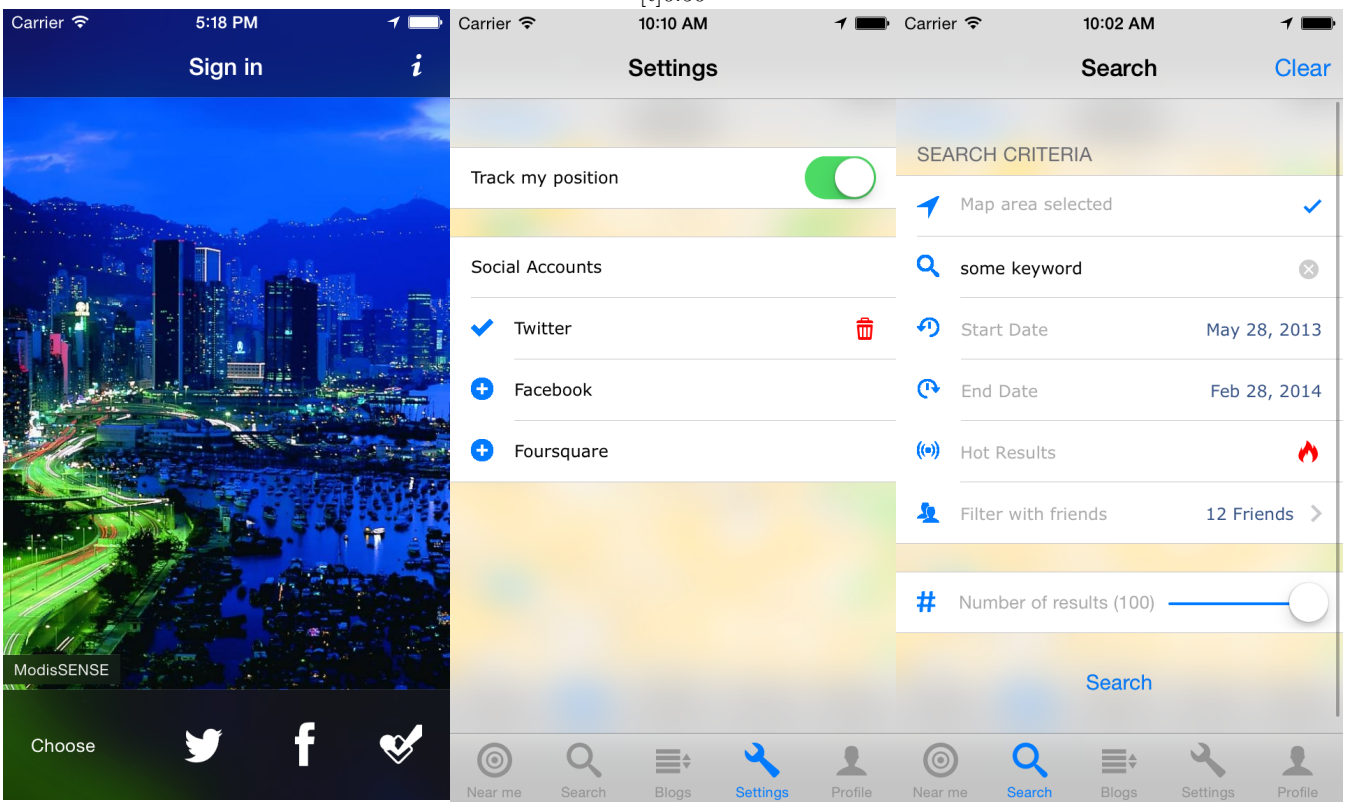

Figure 5: Login and search GUI.

[t]0.50

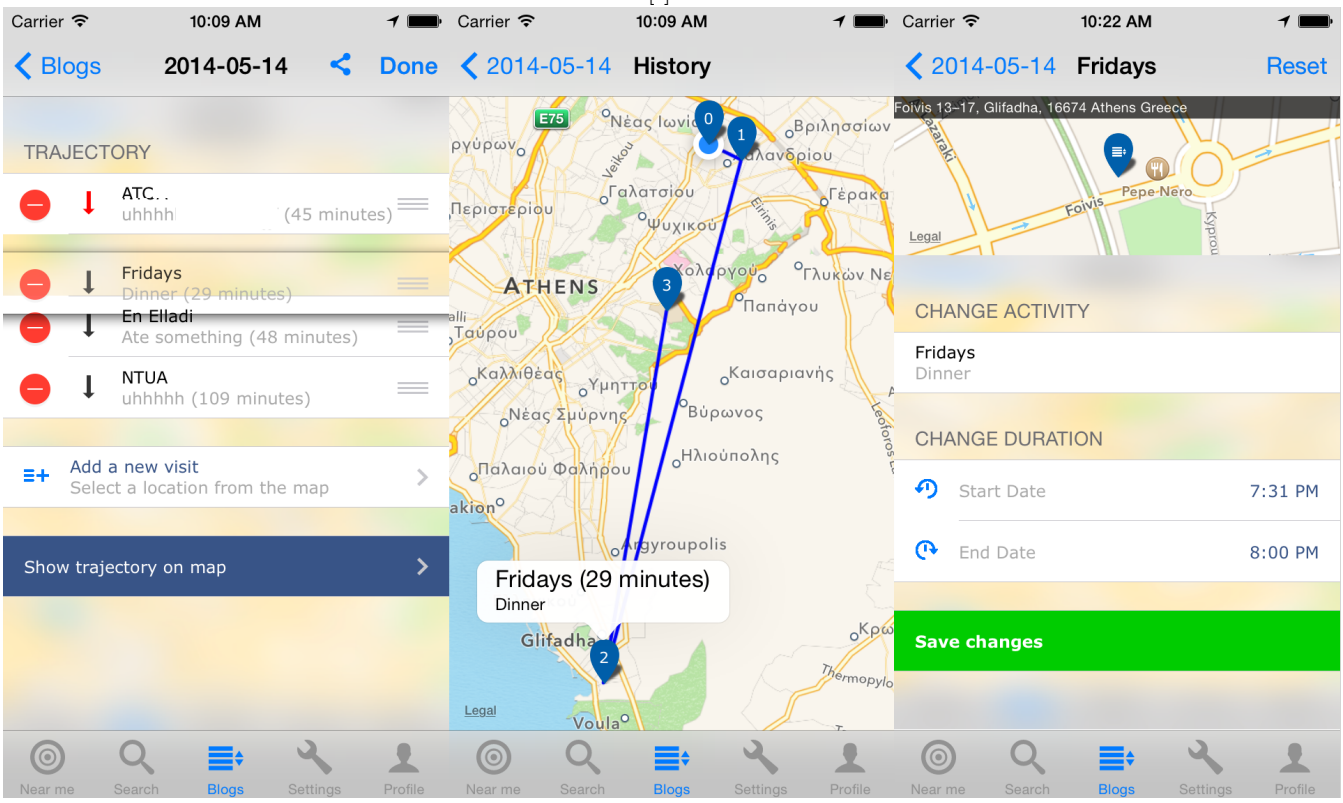

Figure 6: Blog GUI functionalities.

Figure 7: Screenshots from the iOS mobile application.