Taghreed: A System for Querying, Analyzing, and Visualizing Geotagged Microblogs*

Amr Magdy[#§], Louai Alarabi[#§], Saif Al-Harthi[§], Mashaal Musleh[§], Thanaa M. Ghanem[⋆§], Sohaib Ghani[§], Mohamed F. Mokbel[#§]

[§]KACST GIS Technology Innovation Center, Umm Al-Qura University, Makkah, KSA
[#]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN
[⋆]Department of Information and Computer Sciences, Metropolitan State University, Saint Paul, MN
{amr,louai,mokbel}@cs.umn.edu, {sharthi,mmusleh,sghani}@gistic.org,
thanaa.ghanem@metrostate.edu

ABSTRACT

This paper presents *Taghreed*; a full-fledged system for efficient and scalable querying, analyzing, and visualizing geotagged microblogs, e.g., tweets. *Taghreed* supports arbitrary queries on a large number (Billions) of microblogs that go up to several months in the past. *Taghreed* consists of four main components: (1) Indexer, (2) query engine, (3) recovery manager, and (4) visualizer. *Taghreed* indexer efficiently digests incoming microblogs with high arrival rates in light memory-resident indexes. When the memory becomes full, a flushing policy manager transfers the memory contents to disk indexes which are managing Billions of microblogs for several months. On memory failure, the recovery manager restores the system status from replicated copies for the main-memory content. *Taghreed* query engine consists of two modules: a query optimizer and a query processor. The query optimizer generates an optimal query plan to be executed by the query processor through efficient retrieval techniques to provide low query response, i.e., order of milli-seconds. *Taghreed* visualizer allows end users to issue a wide variety of spatio-temporal queries. Then, it graphically presents the answers and allows interactive exploration through them. *Taghreed* is the first system that addresses all these challenges collectively for microblogs data. In the paper, each system component is described in detail.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

Keywords

Microblogs, Spatio-temporal, Indexing, Query Processing

*This work is supported by KACST GIS Technology Innovation Center at Umm Al-Qura University, under project GISTIC-13-06, and was done while the first, second, fifth, and last authors were visiting the center.

SIGSPATIAL '14, November 04-07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666397

1. INTRODUCTION

Online social media services become very popular in the last decade which has led to explosive growth in size of microblogs data, e.g., tweets, Facebook comments, and Foursquare check in's. Everyday, 255+ Million active Twitter users generate 500+ Million tweets [40, 42], while 1.23+ Billion Facebook users post 3.2+ Billion comments [13]. As user-generated data, microblogs form a stream of rich data that carries different types of information including text, location information, and users information. Moreover, microblogs textual content is rich with user updates on real-time events, interesting keywords/hashtags, news items, opinions/reviews, hyperlinks, images, and videos. Consequently, this richness in data enables new queries and applications on microblogs that were not applicable earlier on traditional data streams. For example, performing keyword search on streaming data [7, 9, 45, 46] is now of interest for the first time. Other examples for newly emerging applications on microblogs includes event detection [1, 20, 27, 36, 44], news extraction [6, 34, 37], spatial search [24], and analysis [17, 38, 39]. Such kinds of applications are so important that major IT companies spend millions of dollars to enable them to their customers [3, 41].

Although the research community has addressed a large set of newly emerging queries and applications on microblogs [1, 7, 12, 21, 24, 26, 27, 34, 37, 45], none of those provides a full fledged system that facilitates microblogs data management to support arbitrary queries on multiple attributes. Only TweeQL [26, 27] is proposed as a general query language for Twitter data, a prime example of microblogs. However, TweeQL just provides a wrapping interface for Twitter streaming APIs without addressing the actual data management issues for microblogs big data. On the other hand, microblogs can be considered a kind of semi-structured data that could be managed with systems like AsterixDB [2]; a distributed system that supports interactive queries on big semi-structured data. However, AsterixDB is not appropriate to handle microblogs in its streaming form for two main reasons: (a) AsterixDB does not support digesting and indexing fast streaming data in real-time, and consequently (b) AsterixDB does not provide mechanisms to manage data flushing for memory-resident data to disk which is a must for managing a long history of microblogs data. Moreover, none of this work addresses interactive visualization and analysis with end users.

In this paper, we present *Taghreed*; a full-fledged system for efficient and scalable querying, analyzing, and visualizing geotagged microblogs. On the contrary to all other work on microblogs, *Taghreed* system goals is to manage, query, analyze, and visualize microblogs streaming data for long periods that go up to several months. In addition, *Taghreed* provides data management techniques that are able to support interactive query responses on different microblogs attributes, promoting spatial, temporal, and keyword as first-class attributes as they are involved in most of the important queries and application of microblogs. The main technical challenges in *Taghreed* comes from three main sources: (1) the large number of microblogs to be managed (order of Billions), (2) the continuous arrival of new microblogs in high rates (order of hundreds or thousands every second), and (3) the new types of queries on the rich microblogs data which cannot be supported by Data Stream Management Systems (DSMS). Unlike the traditional streaming data, the combination of fast-arriving large number of data items and the queries on a rich set of attributes, e.g., spatial, temporal, keyword, users, and language, requires *Taghreed* to employ both real-time and disk-based efficient and scalable data indexing to be able to digest real-time data and support such queries interactively, i.e., the query answer is provided instantly. Thus, *Taghreed* main components are: (1) Efficient and scalable data indexing, (2) efficient interactive query processing, (3) low-overhead recovery management, and (4) effective data visualization, where each of them faces different technical challenges.

Indexer. Supporting indexing on microblogs data faces three main challenges: (1) continuous digestion of real-time microblogs, with high arrival rates, in main-memory, (2) managing a large number of microblogs, order of Billions, on disk, and (3) managing data flushing from main-memory indexes to disk indexes so that the system response is not hurt and the main-memory resources are efficiently utilized. To address these challenges, *Taghreed* proposes a set of segmented hierarchical indexes on spatial, temporal, and keyword attributes of microblogs. In both main-memory and disk, *Taghreed* employs two indexes: a *keyword index* and a *spatial index*. Each index consists of temporally-partitioned index segments, where each segment organizes its data based on either spatial or keyword attributes, depending on the index type. Based on the different challenges of indexing real-time data and big historical data, the organization and the employed data structure of main-memory and disk indexes are different. *Taghreed* also adapts several flushing policies to transfer data from main-memory to disk. The flushing manager is mainly concerned with minimizing the system overhead, e.g., disk access latency and main-memory utilization, while keeping as much data as possible in main-memory to reduce the query processing time. The details of *Taghreed* indexing are presented in Section 4.

Query engine. *Taghreed* query engine is mainly concerned with supporting a wide set of generic interactive queries, i.e., low query response in order of milli-seconds, on the large number of managed microblogs. By generic queries we mean a framework that is not tailored to certain predefined queries. Instead, it could be easily adapted to support different types of queries with almost no loss in system performance. To this end, *Taghreed* has chosen to support efficient retrieval of individual microblogs based on the main querying attributes which are the spatial, temporal, and keyword attributes. In other words, *Taghreed* can get a set of microblogs that lie within certain spatio-temporal range and satisfy certain keyword expression very fast. Using these individual microblogs, other types of filtering, e.g., based on users, and aggregation, e.g., frequent keywords, are performed with efficient distributed data scanners. To accomplish low query responses, the query engine consists of two main modules: (1) *A query optimizer*, whose main task is to generate an optimal query plan, to hit the system indexes, based on different cost models. (2) *A query processor*, that performs effective pruning through the system indexes to efficiently retrieve the data of interest with minimal system overhead. Then, it employs efficient distributed data scanners to answer more complex or extended queries that involve attributes other than spatial, temporal, and keywords. The details of *Taghreed* query engine are presented in Section 5.

Recovery manager. With hours and even days of data managed in the main-memory, *Taghreed* provides a recovery management module that is able to restore the system status on memory failures. For social media application, the recovery manager is required to be of low-overhead so that it does not hurt the continuous real-time operations. Thus, *Taghreed* employs a memory-based triple-redundancy model that replicates the main-memory contents three time. On memory failure, the recovery manager uses the replicated copies to restore the system status without interrupting the real-time operations. The details of *Taghreed* recovery manager are presented in Section 6.

Visualizer. With all the technical details of managing and querying microblogs data in the system back end, *Taghreed* provides end-to-end solution with an interactive front end that takes users queries through web-based interfaces, dispatches them to the query engine, and receive the answers back to visualize. The visualization module comprises of an integrated interface that present answers of a rich set of queries and also provides a comprehensive applications on microblogs data. The details of *Taghreed* visualizer are presented in Section 7.

2. RELATED WORK

Taghreed related work lies mostly in three areas, namely, microblogs data processing, big data systems, and spatial-keyword search.

Microblogs data processing. Microblogs research mostly lies in two categories: (a) **Microblogs data analysis** where the main focus is to provide novel applications and exploit user activities and opinions from microblogs contents, e.g., semantic and sentiment analysis [5, 29, 31], decision making [8], news extraction [37], event and trend detection [1, 20, 27, 36], understanding the characteristics of microblog posts and search queries [21, 35], microblogs ranking [12, 43], and recommending users to follow or news to read [15, 34]. (b) **Microblogs data management** where the main focus is to provide infrastructure to handle microblogs data, e.g., logging [19], indexing [7, 9, 24, 45, 46], and query languages [26]. *Taghreed* lies in the second category. Distinguishing itself from all existing data management work on microblogs, *Taghreed* is the first system to: (1) Combine keyword search with spatio-temporal search on microblogs providing all the needed indexing, query optimization, and processing infrastructure techniques. (2) Provide flexible querying framework on other microblogs at-

tribute, e.g., users and language. (3) Handle microblogs data in both real-time and historical forms so that queries are enabled on a long history of data that goes up to several months.

Big data systems. Microblogs data is a kind of big data that is flowing in a streaming form in large numbers and with high arrival rates. Although there exists a lot of work on spatio-temporal queries over streaming data [18, 22, 30, 32, 47], the main focus of such work is on continuous queries over moving objects. In such case, a query is registered first, then its answer is composed over time from the incoming data stream. On the contrary, queries answers on microblogs are retrieved from existing stored objects that have arrived prior to issuing the query. On another hand, AsterixDB [2] and Map-D [25] are systems that support interactive queries on big data. However, none of them tackle the problem of real-time digestion and indexing of data with high arrival rates. They are mostly designed for managing big batches of stored data. Consequently, they lack the ability to handle the microblogs in its mixed form where the data comes as a fast stream, managed as hot recent data for a while, and then being flushed to a large data repository on disk.

Spatial-keyword search. Spatial keyword queries of different types are well studied on web documents and web spatial objects (see the survey [10]). However, all these techniques are not suitable for handling microblogs for two reasons: (1) none of these techniques consider the temporal dimension which is a must in all microblogs queries, and (2) these techniques mostly use offline disk-based data partitioning indexing which cannot scale to support the dynamic nature and arrival rates of microblogs [7, 10].

3. SYSTEM OVERVIEW

This section gives an overview of *Taghreed* design principles, system architecture, and supported queries.

3.1 Design Principles

Taghreed is designed based on two principles:

1. **Dominance of the temporal, spatial, and keyword attributes**: All microblogs queries have to be temporal, and then it mostly involves spatial and keyword dimensions. The real-time nature of microblogs data makes the temporal dimension a must to be included in all queries (see [7]). In addition, the richest attributes in microblogs are the spatial and keywords attributes which are involved in most of the queries and applications (see [1, 7, 12, 21, 24, 26, 27, 34, 37, 45]). Consequently, *Taghreed* promotes the three attributes, spatial, temporal, and keyword, as first-class citizens and supports native system indexes on them. In addition to their importance, indexing spatial, temporal, and keyword attributes provides effective pruning for the microblogs search space.
2. **Importance of queries on recent microblogs**: Due to the rich real-time content of microblogs, e.g., real-time updates on ongoing events [6, 11], important queries are posted on recent microblogs data, i.e., data of the last few seconds, minutes, or hours. This triggers all the work on handling queries on real-time microblogs (see [1, 7, 23, 24, 28, 31, 34, 36, 45]). Consequently, *Taghreed* is designed to support queries on real-time microblogs with all its subsequent requirements of main-memory indexing, recovery management, and flushing management.

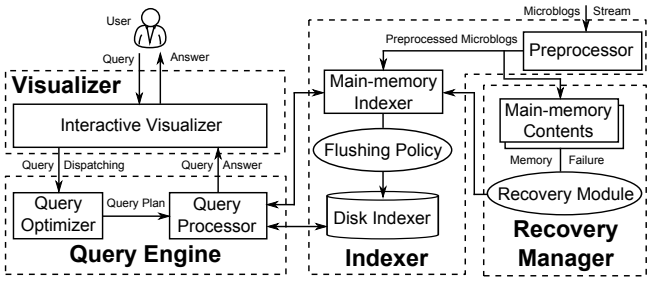

Figure 1: Taghreed Architecture.

3.2 System Architecture

Figure 1 gives *Taghreed* system architecture that consists of four main components, namely, indexer, query engine, recovery manager, and visualizer. We briefly introduce each of them below.

Indexer. *Taghreed* indexer is the main component that is responsible for handling the microblogs data. First, the real-time microblogs are going through a preprocessor that performs location and keyword extraction. Then, the real-time microblogs are continuously digested in main-memory indexes, on spatial, temporal, and keywords attributes, that provide high digestion rates and effective pruning for the search space. When the memory becomes full, a subset of the main-memory microblogs are selected, through a flushing policy module, to be consolidated into scalable disk indexes that are able to manage a large number of microblogs for long periods, go up to several months. The details of *Taghreed* indexer are presented in Section 4.

Query Engine. *Taghreed* query engine consists of two main sub-components: a query optimizer and a query processor. The query optimizer takes a user query from the system front end dispatcher, generates an optimized query plan to hit the system indexes efficiently, and provides the query plan to the query processor to execute. The query processor takes the query plan, executes it, and feed the query answer to the system front end to be visualize to end users. The details of *Taghreed* query engine are presented in Section 5.

Recovery Manager. With dense main-memory contents, *Taghreed* accounts for memory failures by incorporating a recovery manager component. The recovery manager employs a triple-redundancy model for backing up the main-memory contents. When the memory fails, the backup copies are used to restore the system status. The details of *Taghreed* recovery manager are presented in Section 6.

Visualizer. As a full-fledged system that provides end-to-end solution for querying microblogs, *Taghreed* provides an interactive visualizer component that interacts with end users. *Taghreed* is among few data management systems that take into account the end user perspective in its design. Such design strategy is very important for designing social media applications as the user experience plays a vital role in application usability and popularity [16]. The visualizer allows the user to issue queries, dispatches them to the query engine, and receive the answers to present them to end users in graphical form. It provides *Taghreed* users with a rich set of interactive queries which are presented in an integrated interface. The details of *Taghreed* visualizer are presented in Section 7.

3.3 Supported Queries

Taghreed supports any query on microblogs that involves the spatial, temporal, and keyword attributes. The temporal dimension is mandatory while spatial and keyword dimensions are optional. The queries are answered by filtering the search space through hitting the system indexes for the three main attributes. If the query involves other attributes, *Taghreed* employs generic distributed data scanners that refine the answer based on the other attributes. This enables *Taghreed* to support generic queries that satisfy a wide variety of applications. Section 7 presents a rich sample of queries that can be supported by *Taghreed* system along with their interactive visual interfaces to end users.

4. INDEXER

The plethora of microblogs data combined with the newly motivated queries on the rich stream, e.g., spatial-keyword search, makes data indexing an essential part to manage and query microblogs data. In other words, microblogs come in very large numbers, i.e., Millions and even Billions, so that interactive queries—where the answer is needed instantly—cannot be answered efficiently by just scanning the big data. Thus, supporting indexing on microblogs is essential for efficient retrieval of microblogs data that is needed to answer the interactive queries. Next in this section, we will discuss the selection of attributes to be indexed along with the organization and the details of the indexes.

As a user-generated data, microblogs are rich data that contains several attributes to post queries on, e.g., timestamp, location, keywords, user attributes, and language. Although *Taghreed* goal is to provide a framework to answer arbitrary queries on microblogs where more queries can be supported with minimal performance loss, it is not practical from a system point of view to support a separate index on each of microblogs attribute. Thus, *Taghreed* has chosen to provide indexes on the most effective attributes to be indexed and perform efficient distributed scanning for other attributes if involved in any of the queries. The effective attributes to be indexed are chosen based on two factors: (1) the attributes that are involved in most of the important queries on microblogs, (2) the attributes whose a wide domain of values, i.e., can take a large number of distinct values, and hence would provide the most effective pruning for the microblogs search space. An example of an attribute that has limited domain of values is the language attribute. The language attribute in Twitter data has a domain of only 56 values. If language attribute is indexed, that means that all of the incoming microblogs, that are Billions, will be divided into at most of fifty six chunks where each of them will be huge to search within. Thus, the language attribute will not provide very effective pruning for Billions of microblogs. Based on these two factors, *Taghreed* has chosen to support indexes for spatial, temporal, and keywords attributes. From one hand, most of microblogs queries and applications in the literature involve these three attributes. On the other hand, the domains of values of these attributes are wide enough to provide effective pruning for the search space so that the number of processed microblogs to answer certain query is minimized.

As a fast stream that comes with rapidly increasing high arrival rates, microblogs real-time digestion must be in light main-memory indexes that is able to cope with the increasing arrival rates (Twitter rate in October 2013 was 5,200 tweet/second compared to 4,000 tweet/second in April 2012 [42]). In addition, keeping the most recent data in main-memory speeds up the query responses as most of the real-world queries access the most recent data [7]. On the other hand, *Taghreed* would not be able to manage Billions of microblogs, for several months, in main-memory due to the scarcity of this resource. Consequently, *Taghreed* indexing would include both main-memory and disk-resident indexes. As Figure 1 shows, the streaming microblogs are handled by a real-time preprocessor for keyword and location extraction. Then, the main-memory indexes would digest the preprocessed microblogs in real-time with high arrival rates. When the memory becomes full, a flushing manager would manage to transfer main-memory contents to the disk indexes. Disk indexes are then responsible to manage a very large number of microblogs for several months.

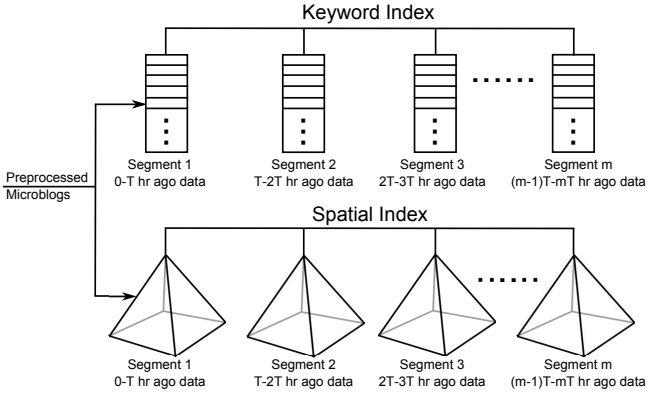

Figure 2: Taghreed In-memory Segmented Index.

In the rest of this section we describe *Taghreed* indexer. First, we describe the main-memory indexing of real-time microblogs in Section 4.1. Second, we describe the disk-based indexing in Section 4.2. Finally, the flushing management from main-memory to disk is discussed in Section 4.3.

4.1 Main-memory Indexing

Taghreed provides light-weight in-memory indexes that employ efficient index update techniques to be able to digest high arrival rates of microblogs in real-time. The indexes are supported on the spatial, temporal, and keywords attributes. In this section, we describe index structure and organization. We also discuss the real-time digestion through efficient index update operations.

Index organization. Figure 2 shows the organization of *Taghreed* in-memory indexes. *Taghreed* employs two segmented indexes in the main-memory: a keyword index and a spatial index. Both of them are temporally partitioned into successive disjoint index segments. Each segment indexes the data of T hours, where T is a parameter to be optimized by the flushing manager (see Section 4.3). The newly incoming microblogs are digested in the most recent segment (Segment 1 in Figure 2). Once the segment spans T hours of data, the segment is concluded and a new empty segment is introduced to digest the new data. Index segmentation has two main merits: (a) new microblogs are digested in a smaller index, which is the most recent segment, and hence becomes more efficient, and (b) it eases flushing data from

memory to disk under certain flushing policies.

Real-time digestion. The keyword index segment is an inverted index that organizes the data in a light hashtable. The hastable maps a single keyword (the key) to a list of microblogs that contain the keyword. The list of microblogs of each keyword is ordered reverse-chronologically so that the insertion is always in the list front item in O(1). Thus, the whole insertion in the index is O(1). With such optimization, *Taghreed* keyword segments are able to digest up to 32,000 microblog/second.

The spatial index segment is a pyramid index structure [4] (similar to a partial quad tree [14]) that employs efficient update and structuring techniques to provide a light-weight spatial indexing. The pyramid index is a space-partitioning tree of cells, where each cell has either zero or four children cells, and sibling cells cover equal spatial areas. Unlike data-partitioning indexes, e.g., R-tree, pyramid index could support high digestion rates due to the low restructuring overhead with newly incoming data. Each cell has a capacity of certain number of microblogs, where the capacity is a system parameter. The microblogs inside each cell are stored in a reversed chronological order. When a cell encounters microblogs that exceeds its capacity, it is split into four children cells if and only if the microblogs lie in at least two different quarters of the cell. This excludes redundant split operations for highly skewed data. Similarly, to eliminate redundant merge operations, underutilized cells are not merged immediately. Instead, four siblings are merged, in lazy basis, only when three of them are completely empty. Such lazy split and merge operations saves 90% of the structuring operations in the highly dynamic microblogs environment. Also the index structure stabilizes relatively fast which decrease the structuring overhead to its minimal levels. To update the index efficiently, microblogs are inserted in batches, periodically each t seconds, instead of traversing the pyramid levels for each individual microblog. Typical values of t are 1-2 seconds so that several thousands of microblogs are inserted in each batch. Then, the pyramid levels are traversed once with minimum bounding rectangle of all the microblogs in the batch. This saves thousands of comparison operation in each insertion cycle. With these optimization, *Taghreed* spatial segments are able to digest up to 32,000 microblog/second. More details on the real-time spatial indexing in *Taghreed* can be reviewed in [24].

All the deletion operation from the in-memory indexes are handled by *Taghreed* flushing manager. Thus, the details of deletion from the main-memory indexes are explained in Section 4.3.

4.2 Disk-based Indexing

To be able to support a large number of microblogs data for long periods, that go up to several months, *Taghreed* supports disk indexes to manage the microblogs that are expelled from the main-memory. Similar to the main-memory indexes, disk indexes are supported on spatial, temporal, and keywords attributes. However, the disk indexes organization and structure are different from the main-memory ones. In this section, we describe the index organization, structure, and update operations.

Index organization. *Taghreed* employs two disk indexes: a keyword index and a spatial index. Figure 3 shows the organization of *Taghreed* disk spatial index, which is similar to the organization of the keyword index as well. Each

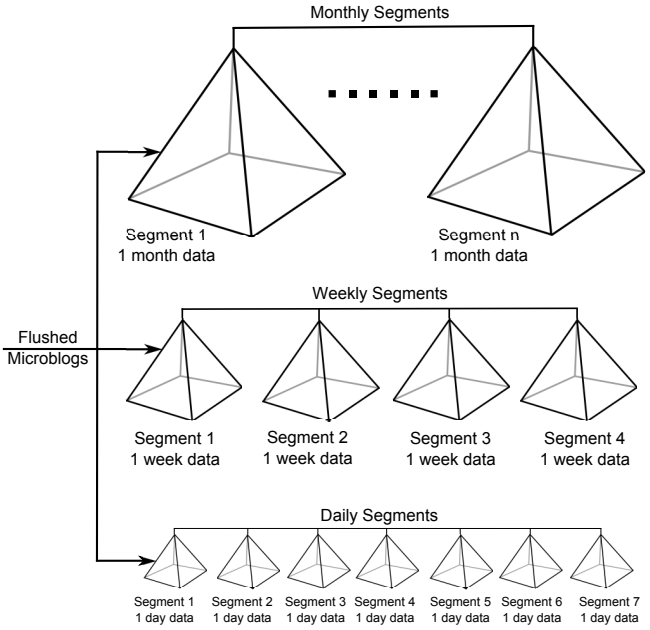

Figure 3: Taghreed Disk Spatial Index.

of both indexes is organized in temporally partitioned segments. The temporal segments are replicated in a hierarchy of three levels, namely, daily segments, weekly segments, and monthly segments. The daily segments level stores the data of each calendar day in a separate segment. The weekly segments level consolidates the data in each successive seven daily segments that forms data for one calendar week in a single weekly segment. Again, the monthly segments level consolidates data of each successive four weekly segments in a single segment that manages the data of a whole calendar month. The main reason behind replicating the indexed data on three temporal levels is to minimize the number of accessed index segments while processing queries for different temporal periods. For example, for an incoming query asking about data of two months, if only daily segments are stored, then the query processor would need to access sixty indexes to answer the query. On the contrary to the described setting, the query processor would need to access only two indexes of the two months time horizon of the query. This significantly reduces the query processing time so that *Taghreed* is able to support queries on relatively long periods.

Index structure and update. *Taghreed* disk keyword index segments structure are inverted indexes while spatial index segments are R^+-trees. Unlike pyramid structure, R^+-tree is disk-friendly where tree nodes are disk pages. At any point of time, only a single daily segment is active to absorb the expelled microblogs from the main-memory. Once a one full day passes, the current active daily segment is concluded and a new empty segment is introduced to absorb the next incoming data. Upon concluding seven successive daily segments, a weekly segment is created in the background to merged the data of the whole week in a single index. The same repeats for monthly segments upon creating four successive weekly segments.

Updating the disk indexes is performed in isolation from the employed flushing policy (see Section 4.3 for flushing

policies details). Regardless the flushing policy, disk indexes are always temporally disjoint from the main-memory indexes. In other words, whenever *Taghreed* inserts data in the disk indexes, there exists a check point timestamp t_{cp} where all the data in the main-memory indexes are more recent than t_{cp} and all the data in disk-based indexes are older than or equals t_{cp}. Guaranteeing this, consolidating data from main-memory keyword index into disk keyword index is done very efficient through bucket-to-bucket mapping without bothering with deforming the temporal organization of the data. It is worth noting that mapping memory buckets to disk buckets is inapplicable for the spatial index as the spatial structure that is employed in the main-memory (the pyramid index) is space-partitioning structure which is different than the data-partitioning structure (the R^+ tree) that is employed on disk. In the rest of the section, we explain the consolidation process from main-memory to disk for both indexes, keyword and spatial.

To consolidate data from main-memory keyword index to disk keyword index, we perform three steps. First, we check if the new data would require creating a new daily segment. For this, we check the oldest and the newest timestamps of the data to be flushed, which are provided from the flushing manager. If the two timestamps span two different days, then a new daily segment is created. The second and third steps are performed repeatedly for each keyword slot in the index. Each slot contains a list of microblogs that are stored in a reverse chronological order. The second step maps each slot from the main-memory to the corresponding slot in the active disk index segment, based on the keyword hash value. The third step then merges the main-memory data list, L, into the existing microblogs list on the disk. L data is first checked if it spans two days so that the list could be divided into two sublists and two index segments may be accessed. After that, the list (sublist) of microblogs is merged into the corresponding slot by prepending the list to the existing disk list. This is O(1) operation due to the temporal order and disjointness of the two lists. On the contrary, to consolidate data from main-memory pyramid index to disk R^+-tree index, the data are flushed in raw format, where batches of microblogs are bulk loaded to the R^+-tree without any mapping between the memory index partitions and disk index partitions. Although this builds the index partitions from scratch, it is necessary due to the difference between the employed spatial structures. As R^+-tree is disk-friendly, the bulk loading flushing is efficient enough to handle the segmented microblogs data.

4.3 Flushing Management

The main task of *Taghreed* flushing manager is to determine which microblogs should be flushed from the main-memory indexes to disk indexes when the memory becomes full. Although the task looks trivial for the first glance, it highly affects the query performance [33] as it controls the main-memory contents. The incoming queries to *Taghreed* are answered from both main-memory and disk contents. The more relevant data in main-memory, the less disk access is needed to answer the queries, and then the lower query response time. Thus, *Taghreed* flushing manager enables the system administrator to employ one of multiple available flushing policies. The flushing policy tries to compromise the indexing and flushing overhead with the availability of relevant data, to incoming queries, in the main-memory. In this section, we describe three different flushing policies namely **Flush-All**, **Flush-Temporal**, and **Flush-Query-Based**. We also discuss the pros and cons of each of them.

Flush-All. The simplest flushing policy is to dump the whole memory contents to the disk. This makes the main-memory indexing very flexible as any number of segments can be used without dramatic effect on the flushing process. Also, it minimizes the disk access overhead as less number of flushing operations are performed. Obviously, *Flush-All* preserves the property of temporal disjointness between main-memory contents and disk-contents as it always dump all the old data to the disk before receiving new recent data in the main-memory. However, *Flush-All* is not a recommended flushing policy as it causes system slowdowns in terms of query responses. Once the flushing operation is performed, all the memory-indexes become empty and hence all the incoming queries are answered only from disk contents. This causes a significant sudden slowdown which is an undesirable effect from a system point of view.

Flush-Temporal. An alternative flushing policy is to expel a certain portion of the oldest microblogs to empties a room for the newly real-time incoming microblogs. To reduce the flushing overhead, this policy requires the main-memory indexing to partition the data into segments with the same flushing unit. Referring to the main-memory index organization in Figure 2, the flushing unit is defined as T hours, i.e., the oldest T hours of data are flushed periodically. In this policy, T would be a system parameter that is adjusted by the system administrator based on the available memory resources, the rate of incoming microblogs, and the desired frequency of flushing. *Flush-Temporal* also preserves the property of temporal disjointness between main-memory contents and disk-contents as it always dump data of a certain period of time. This moves the temporal check point t_{cp} by exactly T hours without causing any kind of temporal overlap. In addition, *Flush-Temporal* addresses the limitations of *Flush-All* and does not cause sudden significant system slowdowns. This comes on the cost of more frequent flushing operations which needs more frequent disk access overhead. However, as a background process, flushing disk access does not significantly affect the system performance.

Flush-Query-Based. A third policy is to expel the microblogs that are not relevant to the incoming queries. This is important when it is required to optimize the system indexes to support certain query, or set of queries, efficiently. The idea is to figure out the characteristic of data that will not satisfy the target query answer, and hence expel them. For example, if the query asks for most recent k microblogs that contains a certain keyword. Then, if the inverted index slot of any keyword contains more than k microblogs, that means all microblogs older than the k^{th} one will not make it to any query answer. Thus, those microblogs can be safely expelled to empties a space for more relevant microblogs to reside in the main-memory.

A fundamental problem in such kind of flushing policies is how to preserve the property of the temporal disjointness between main-memory contents and disk-contents. In this case, the microblogs are not expelled on either a temporal criteria or from a contiguous temporal period. On the contrary, they are expelled selectively based on the query selection criteria. Thus, by default, there is no guarantee about the temporal overlap between the memory and disk contents. The proposed solution in *Taghreed* system is to

flush the data to an intermediate disk buffer rather than flushing directly to the disk indexes. Then, the query-based flushing policy would be combined with a temporal flushing policy (with larger values of T) so that after certain point of time it is guaranteed that all main-memory data are more recent than a certain timestamp. At this point, all the data in the intermediate buffer could be merged to the disk indexes without violating the temporal disjointness. This kind of policies would help to reduce the disk access overhead during the query processing. However, the intermediate disk buffer introduce difficulties of maintaining the buffer and handle the temporal overlap gray area which is the buffer data. Currently, *Taghreed* does not implement any query-based flushing policies. However, we have rigid plans to adapt some of these policies for top-k queries to provide a system administration flexibility to tune the system performance for important queries. A detailed example for query-based flushing for top-k queries can be reviewed in our full paper on spatial search queries on microblogs [24].

It is worth noting that all the flushing operations does not affect the data availability. The main-memory data stay available to the incoming queries until the flushing operation is successfully completed. Then, the temporal check point, t_{cp}, is atomically updated to indicate the new temporal boundaries between the main-memory contents and the disk contents. If concurrent queries have already read the old t_{cp} value, the system keeps track of them using a simple pin counting technique before the flushing manager discards the flushed main-memory contents.

5. QUERY ENGINE

The second major component in *Taghreed* is the query engine. This component consists of two main modules: a query optimizer and a query processor. The two modules are discussed in Sections 5.1 and 5.2, respectively.

5.1 Query Optimizer

As Section 4 shows, *Taghreed* provides two types of indexes in both main-memory and disk: keyword index and spatial index. In addition, disk indexes data is replicated on three temporal levels, daily, weekly, and monthly index segments. Consequently, the query processor may have different ways to process the same query based on: (1) the order of performing keyword or spatial filtering based on the system indexes, and (2) the number of hit disk indexes. For example, the query that asks about only spatial data of the period from June 1 to June 9 can be answered from disk spatial indexes in two different ways: (a) either accessing nine daily index segments, or (b) accessing one weekly and two daily index segments. Each of those is called a query plan. The costs of different query plans are different. The main task of the query optimizer is to generate a plan to execute so that the estimated cost is minimal. In this section, we discuss *Taghreed* query optimization. First, we discuss the keyword/spatial index order selection of both main-memory and disk. Then, we discuss the disk index segments selection over the three levels of temporal hierarchy. Finally, we describe the whole query plan generation scenario.

Keyword/spatial index order selection. When a query involves querying both spatial and keyword dimensions, there are two ways to retrieve the microblogs within the query scope: (1) either hit the keyword index and perform spatial filtering for the retrieved microblogs, or (2) hit the spatial index and perform keyword filtering for the retrieved microblogs. To select one of the two plans, *Taghreed* query optimizer employs a cost model for each scenario. It calculates the estimated cost for each plan and select the cheap one. For a query q, the costs of both plans are calculated based on the following equations:

$$Cost(keyword|q) = A_{kw} \times query\_keyword\_count \quad (1)$$

$$Cost(spatial|q) = A_{sp} \times query\_area \quad (2)$$

Equation 1 is used to estimate the cost of hitting the keyword index given q while Equation 2 is used to estimate the cost of hitting the spatial index given q. The cost of q depends on its number of keywords and its spatial extent. *Taghreed* calculates a single number for each index, namely, A_{kw} and A_{sp}. A_{kw} is the average number of microblogs in a keyword slot. A_{sp} is the average number of processed microblogs per query area of one mile square. Using A_{kw} and A_{sp}, the query optimizer is able to estimate the number of microblogs need to be processed to provide the query answer. In main-memory, this estimates the amount of processing needed. On disk, this estimates the number of pages needed to be retrieved from disk. Next, we discuss the calculation of A_{kw} and A_{sp}.

To calculate A_{kw}, two numbers are maintained for each keyword index: the total number of microblogs inserted so far in the index $Total_M$, and the number of distinct keywords inserted in the index N_{kw}. A_{kw} can be then calculated as follows: $A_{kw} = \frac{Total_M}{N_{kw}}$. Both $Total_M$ and N_{kw} are easy to maintain during the index update operations with almost no overhead. To calculate A_{sp}, two numbers are maintained for each spatial index: the summation of average numbers of microblogs processed under each incoming query since the index is created Sum_{avg}, and the number of queries processed on the index N_q. A_{sp} can be then calculated as follows: $A_{sp} = \frac{Sum_{avg}}{N_q}$. To maintain Sum_{avg}, for each query, *Taghreed* keeps track of the total number of processed microblogs during the query. Then, this number is divided by the query area (in miles square). Finally, the division result is added to Sum_{avg} while N_q is incremented by one. It is worth noting that A_{kw} is changing over time with the data keyword distribution while A_{sp} is changing over time with the query load spatial distribution. This dynamic learning process of A_{kw} and A_{sp} continuously improves the cost estimation and hence the query performance.

Disk index segments selection. As data on disk is replicated on three levels of temporal hierarchy, there are usually different ways to access data in certain temporal range: from either a daily index, a weekly index, or a monthly index. To estimate the cost of hitting each index individually, we use Equations 1 and 2. However, there are different valid combinations of indexes. Trying to minimize the overhead of getting an optimal combination of indexes, the query optimizer starts with the combination with the minimum amount of data to be accessed. This means using weekly and monthly indexes only for whole weeks and whole months, respectively, in the query temporal horizon. For example, if the query temporal horizon spans May 29 to July 9, then the starting combination would be three daily indexes for last three days of May, one monthly index for whole June, one weekly index for first week of July, and

two daily indexes for July 8 and 9. As these indexes do not contain any data outside the query temporal boundary, so, it contains the minimum amount of data to be accessed. The next step is to figure out the best combination. The assumption made here that going up in the index temporal hierarchy would increase the cost. In our example, replacing the three daily indexes of the last three days of May with one weekly index of the last week of May would incur more cost as more disk pages would be retrieved. Thus, the employed heuristic is to go down in the index hierarchy to explore and divide weekly and monthly indexes into finer granularity indexes, i.e., days and weeks, respectively. Starting from the first generated combination, the query optimizer tries to replace weekly indexes with seven daily indexes (and monthly indexes with four weekly indexes). Checking the costs of this combinations are not costly as it is just summation of seven (four) cost parameters numbers, i.e., A_{kw} and A_{sp}. The optimizer then selects the combination with the minimum estimated cost.

Query plan generation. To generate a complete query plan, first the optimizer checks the query temporal horizon versus the memory/disk data temporal boundary, t_{cp}, to determine the temporal horizon of both memory and disk data, namely, t_m and t_d, respectively. Afterwards, a sub-plan is generated for each of them separately. In main-memory, the index segments that intersect with t_m are determined. Then, the index to be hit (either keyword or spatial) is selected based on the above selection model. On disk, an index combination is generated for both spatial and keyword index hierarchies. Then, the cost of each combination is estimated based on Equations 1 and 2 and the cheapest one is selected.

5.2 Query Processor

To provide flexible and efficient spatio-temporal querying framework, *Taghreed* has chosen to employ indexes on spatial, temporal, and keyword attributes and perform filtering on all other attributes through efficient distributed data scanners (see Section 4 for more details on indexing attributes selection). Thus, *Taghreed* query processor has two phases of answering any queries: (1) retrieving a candidate set of microblogs from a spatio-temporal or a keyword-temporal space, depending on the query plan, and (2) performing further processing through scanning on the candidate set, if needed. We discuss the two phases below.

Phase 1: Filtering on the spatio-temporal keyword attributes. In this phase, the query processor retrieves a list of candidate microblogs based on the query spatial, temporal, and keyword parameters. This is performed by executing the optimized query plan (that is generated as described in Section 5.1) through hitting the system indexes. Specifically, the query processor receives a query plan that consists of an optimized set of indexes to be accessed. Each of the indexes is queried to retrieve a list of microblogs that satisfy the user query parameters. The candidate lists are then fed to the second phase for further refinement. As the indexes provide efficient pruning on the indexed attributes, this phase prunes a huge amount of data.

Phase 2: Refinement on other attributes. The output of the phase 1 would be lists of microblogs that require further processing. This phase performs the remaining processing, through extensive distributed data scanning, to provide the final query answer. The type of processing depends on the query type and the query plan. If the spatial index is hit in phase 1, then keyword filtering would be performed in phase 2 and vice versa. Also, this scanning is piggybacked by other operations on any other attributes, e.g., counting microblogs of distinct languages or count frequent keywords, so that *Taghreed* can support a wide variety of queries on different attributes. More details and examples about the supported queries are presented in Section 7.

6. RECOVERY MANAGER

With hours and even days of data managed in main-memory, *Taghreed* system accounts for any failures that may lead to data loss. *Taghreed* employs a simple, yet effective, triple-redundancy model where the main-memory data is replicated three times over different machines. The core idea of this model is similar to Hadoop redundancy model that replicates the data three times. In this section, we briefly describe the operation of the recovery management module in *Taghreed* in both normal and failure cases.

When *Taghreed* is launched, all the main-memory modules, e.g., indexes and all data structures, are initiated on three different machines. Each machine is fed with exactly the same stream of microblogs, thus they form triple identical copies of the main-memory system status. One of the three machines is a master machine that launches all the system components, i.e., memory-resident and disk-resident components. The other two machines launch only the memory-resident components. Any flushing from memory to disk in the master machine leads to throwing the data out from the memory of the other two machines. On failure of the master machine, the other two machines continue to digest the real-time microblogs. Once the master machine is recovered, the system memory image is copied to its main-memory from one of the other machines. On the failure of one of the secondary machines, the other machine data is used to create an alternative for the failed machine. Replicating the data three times significantly reduces the probability of having the three machines down simultaneously and lose all the main-memory data.

Although *Taghreed* recovery management is not 100% strict, it is acceptable for the relaxed social media applications that favor efficiency and scalability over strict consistency and recovery management. Recovery management through main-memory replication allow these applications to scale without being limited with the overhead of disk-based recovery interactions. In addition, its cost is almost doubling or tripling the cost of the needed main-memory resources which is affordable in the age when the memory cost is becoming lower over time.

7. VISUALIZER

Taghreed is a full-fledged system that provides end-to-end solution for microblogs users. Thus, *Taghreed* provides an interactive visualizer component that interacts with end users. Such component is very important for designing a rich set of interactive queries along with their friendly user interfaces. As a flexible querying framework for microblogs data, *Taghreed* core data management components are able to support a wide variety of queries efficiently which enables such richness of query design. In this section, we present an overview about the visualizer component and its integration with the back end components. Then, we present the currently supported queries along with system interfaces to visualize these queries to end users.

Visualizer overview. *Taghreed* interactive visualizer is

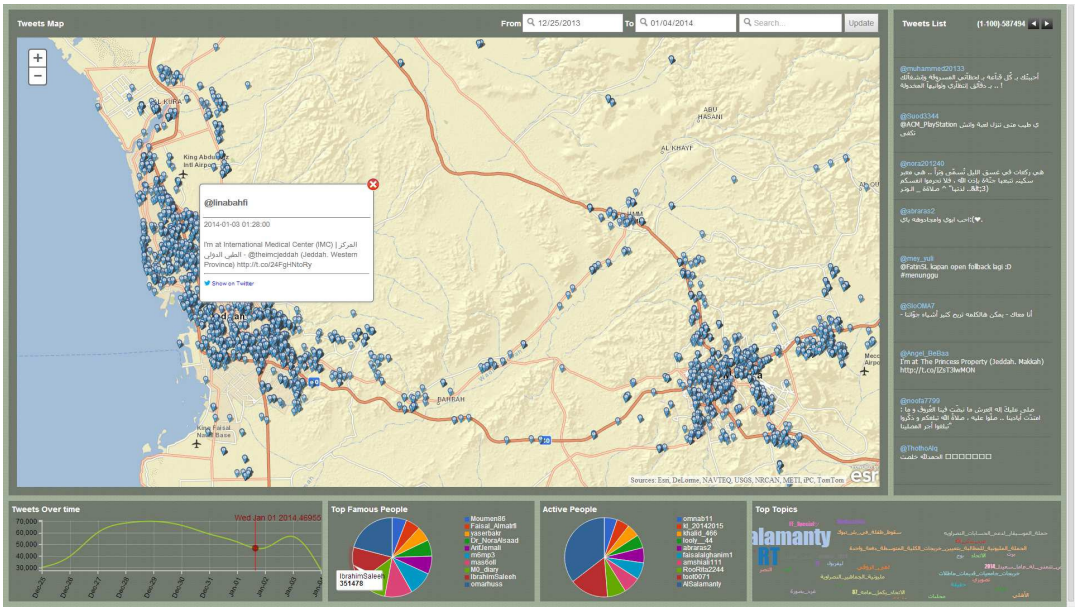

Figure 4: Taghreed Integrated Interface.

the system front end. It receives user queries through interactive web-based user interfaces. The queries are then dispatched to the query engine through Java-based function APIs that allow fast interaction, eliminating the overhead of exchanging data in standard formats, e.g., JSON or XML. The query processor then sends back the queries answers so that the visualizer present them to end users.

Supported queries. As discussed in Section 3.3, *Taghreed* system is designed to provide a flexible framework that is able to answer a wide variety of spatio-temporal queries on different microblogs attributes. Thus, in this part, we present only a sample of queries that are currently supported. However, *Taghreed* is not limited to answer the discussed queries and can be adapted to answer more spatio-temporal keyword queries. Currently, *Taghreed* supports the following spatio-temporal queries on microblogs:

1. **Keyword search**. Within given spatial and temporal ranges, find all microblogs that contain certain keywords.
2. **Top-k frequent keywords**. Within given spatial and temporal ranges, find the most frequent k keywords, for a given integer k.
3. **Top-k active users**. Within given spatial and temporal ranges, find the most k active users, for a given integer k. Active users are defined as the users who have posted the largest number of microblogs in the query spatio-temporal range.
4. **Top-k famous users**. Within given spatial and temporal ranges, find the most k famous users, for a given integer k. Famous users are defined as the users having the largest number of followers. The query answer is selected from users whose home locations lie in the query spatial range and have posted at least one microblog during the query temporal range.
5. **Daily aggregates**. Within given spatial and temporal ranges, find the number of microblogs in each day.
6. **Joint collective queries**. Within given spatial and temporal ranges, find the answer of all the previous queries collectively.
7. **Top-k used languages**. Within given spatial and temporal ranges, find the most k used languages, for a given integer k.

It is worth noting that there are a variety of queries that require scanning on attributes other than spatial, temporal, and keyword. All such scanning efforts are piggybacked on phase 2 of the query processor (see Section 5.2). Also, query 6 is an example of collective queries where multiple queries share the processing power. This significantly reduces the amount of processing consumed per microblog.

System interfaces. Currently *Taghreed* provides two user interfaces, one integrated interface to visualize queries 1 to 6 in addition to a separate interface that employs query 7 in a different way than just presenting a result of user input query. Figure 4 shows the main integrated interface. Through this interface, *Taghreed* user can input a spatial range (through the map interface), a temporal range (through the datepicker), and an optional keywords (through text box). The system then dispatches the collective query 6 with default k=10 and present to the results to the user in the five side boxes in Figure 4. Figure 5 shows another interface that employs query 7 to provide an analysis for language usage in Arab Gulf area using Twitter data. The query is issued for all the sub-regions, then the output pie charts are displayed on the map interface. The granularity of the results change at different zoom levels.

8. CONCLUSION

This paper presented *Taghreed*; a system for scalable querying and visualization of geotagged microblogs. *Taghreed* is able to manage and query Billions of microblogs through four main components. First, an indexer that handles recent microblogs in segmented main-memory indexes with their high arrival rates. When memory becomes full, certain microblogs, selected through the flushing manager,

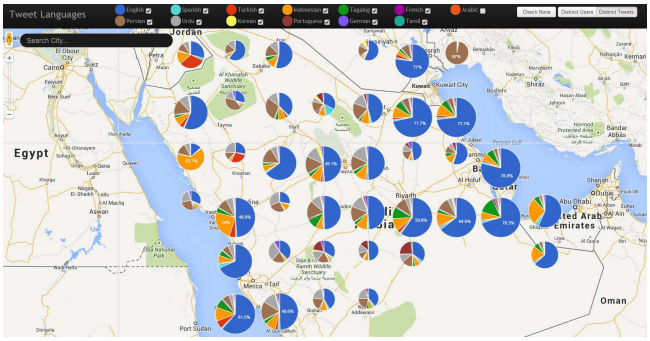

Figure 5: Language Distribution Query in Gulf Area.

are expelled to disk-resident indexes. Second, a query engine that provides query optimization and a query processing on top of system indexes. Third, a recovery manager that restores the system status in case of main-memory failure. Fourth, an interactive visualizer that interacts with system end users. *Taghreed* provides a flexible framework that can adapt several queries that involve spatial, temporal, and keyword attributes.

9. REFERENCES

[1] H. Abdelhaq, C. Sengstock, and M. Gertz. EvenTweet: Online Localized Event Detection from Twitter. In *VLDB*, 2013.

[2] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, R. Grover, Z. Heilbron, Y.-S. Kim, C. Li, N. Onose, P. Pirzadeh, R. Vernica, and J. Wen. ASTERIX: An Open Source System for "Big Data" Management and Analysis. *PVLDB*, 5(12):1898–1901, 2012.

[3] Apple buys social media analytics firm Topsy Labs. http://www.bbc.co.uk/news/business-25195534, 2013.

[4] W. G. Aref and H. Samet. Efficient Processing of Window Queries in the Pyramid Data Structure. In *PODS*, 1990.

[5] A. Bermingham and A. F. Smeaton. Classifying Sentiment in Microblogs: Is Brevity an Advantage? In *CIKM*, 2010.

[6] After Boston Explosions, People Rush to Twitter for Breaking News. 2013. http://www.latimes.com/business/technology/la-fi-tn-after-boston-explosions-people-rush-to-twitter-for-breaking-news-20130415,0,3729783.story.

[7] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-Time Search at Twitter. In *ICDE*, 2012.

[8] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to Ask? Jury Selection for Decision Making Tasks on Micro-blog Services. *PVLDB*, 5(11), 2012.

[9] C. Chen, F. Li, B. C. Ooi, and S. Wu. TI:An Efficient Indexing Mechanism for Real-Time Search on Tweets. In *SIGMOD*, 2011.

[10] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial Keyword Query Processing: An Experimental Evaluation. In *VLDB*, 2013.

[11] Sina Weibo, ChinaŠs Twitter, comes to rescue amid flooding in Beijing. 2012. http://thenextweb.com/asia/2012/07/23/sina-weibo-chinas-twitter-comes-to-rescue-amid-flooding-in-beijing/.

[12] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: Improving recency ranking using twitter data. In *WWW*, 2010.

[13] Facebook Statistics. http://newsroom.fb.com/Key-Facts/Statistics-8b.aspx, 2012.

[14] R. A. Finkel and J. L. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *ACTA*, 4(1), 1974.

[15] J. Hannon, M. Bennett, and B. Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *RecSys*, 2010.

[16] J. Hart, C. Ridley, F. Taher, C. Sas, and A. Dix. Exploring the Facebook Experience: A New Approach to Usability. In *Proceedings of the 5th Nordic Conference on Human-computer Interaction, NordiCHI*, pages 471–474, 2008.

[17] Harvard Tweet Map. http://worldmap.harvard.edu/tweetmap/, 2013.

[18] S. J. Kazemitabar, U. Demiryurek, M. H. Ali, A. Akdogan, and C. Shahabi. Geospatial Stream Query Processing using Microsoft SQL Server StreamInsight. *PVLDB*, 3(2), 2010.

[19] G. Lee, J. Lin, C. Liu, A. Lorek, and D. V. Ryaboy. The Unified Logging Infrastructure for Data Analytics at Twitter. *PVLDB*, 5(12), 2012.

[20] R. Li, K. H. Lei, R. Khadiwala, and K. C.-C. Chang. TEDAS: A Twitter-based Event Detection and Analysis System. In *ICDE*, 2012.

[21] J. Lin and G. Mishne. A Study of "Churn" in Tweets and Real-Time Search Queries. In *ICWSM*, 2012.

[22] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing. Discovering Spatio-temporal Causal Interactions in Traffic Data Streams. In *KDD*, 2011.

[23] A. Magdy, A. M. Aly, M. F. Mokbel, S. Elnikety, Y. He, and S. Nath. Mars: Real-time Spatio-temporal Queries on Microblogs. In *ICDE*, pages 1238–1241, 2014.

[24] A. Magdy, M. F. Mokbel, S. Elnikety, S. Nath, and Y. He. Mercury: A Memory-Constrained Spatio-temporal Real-time Search on Microblogs. In *ICDE*, pages 172–183, 2014.

[25] map-D: Massively Parallel Database. http://mapd.csail.mit.edu/, 2014.

[26] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. Tweets as Data: Demonstration of TweeQL and TwitInfo. In *SIGMOD*, 2011.

[27] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. Twitinfo: Aggregating and Visualizing Microblogs for Event Exploration. In *CHI*, 2011.

[28] D. McCullough, J. Lin, C. Macdonald, I. Ounis, and R. M. C. McCreadie. Evaluating Real-Time Search over Tweets. In *ICWSM*, 2012.

[29] E. Meij, W. Weerkamp, and M. de Rijke. Adding semantics to microblog posts. In *WSDM*, 2012.

[30] E. Meskovic, Z. Galic, and M. Baranovic. Managing Moving Objects in Spatio-temporal Data Streams. In *MDM*, 2011.

[31] G. Mishne and J. Lin. Twanchor Text: A Preliminary Study of the Value of Tweets as Anchor Text. In *SIGIR*, 2012.

[32] M. F. Mokbel and W. G. Aref. SOLE: Scalable On-Line Execution of Continuous Queries on Spatio-temporal Data Streams. *VLDB Journal*, 17(5), 2008.

[33] M. F. Mokbel, M. Lu, and W. G. Aref. Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results. In *ICDE*, pages 251–262, 2004.

[34] O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *RecSys*, 2009.

[35] D. Ramage, S. T. Dumais, and D. J. Liebling. Characterizing Microblogs with Topic Models. In *ICWSM*, 2010.

[36] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: Real-time event detection by social sensors. In *WWW*, 2010.

[37] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. TwitterStand: News in Tweets. In *GIS*, 2009.

[38] Topsy Pro Analytics: Find the insights that matter. http://topsy.com/, 2013.

[39] TweetTracker: track, analyze, and understand activity on Twitter. http://tweettracker.fulton.asu.edu/, 2013.

[40] Twitter Data Grants, 2014. https://blog.twitter.com/2014/introducing-twitter-data-grants.

[41] New features on Twitter for Windows Phone 3.0, 2013. https://blog.twitter.com/2013/new-features-on-twitter-for-windows-phone-30.

[42] Twitter Statistics. http://business.twitter.com/en/basics/what-is-twitter/, 2013.

[43] I. Uysal and W. B. Croft. User Oriented Tweet Ranking: A Filtering Approach to Microblogs. In *CIKM*, 2011.

[44] K. Watanabe, M. Ochi, M. Okabe, and R. Onai. Jasmine: A Real-time Local-event Detection System based on Geolocation Information Propagated to Microblogs. In *CIKM*, 2011.

[45] L. Wu, W. Lin, X. Xiao, and Y. Xu. LSII: An Indexing Structure for Exact Real-Time Search on Microblogs. In *ICDE*, 2013.

[46] J. Yao, B. Cui, Z. Xue, and Q. Liu. Provenance-based Indexing Support in Micro-blog Platforms. In *ICDE*, 2012.

[47] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal and Spatio-temporal Aggregations over Data Streams using Multiple Time Granularities. *Information Systems*, 28(1-2), 2003.