Scalable Microblogs Data Management

Amr Magdy
amr@cs.umn.edu
Supervised by: Prof. Mohamed F. Mokbel
Department of Computer Science and Engineering
University of Minnesota - Twin Cities

ABSTRACT

Microblogs, e.g., tweets, reviews, or comments on news websites and social media, have become so popular among web users that many applications are exploiting them for different types of analysis. The distinguishing characteristics of microblogs have motivated a lot of research for managing such data. However, the developed technology for microblogs is still scattered efforts here and there which leads to several data management gaps that limits supporting microblogs-centric applications end-to-end. Our research aims to provide a holistic system approach to manage microblogs data, so that whoever builds new functionality on microblogs can seamlessly exploit a single data management system to power his applications. In this paper, we present a full proposal for *Kite*; the first holistic system that provides end-to-end management for microblogs data. *Kite* aims to fill the gap in existing systems to support scalable queries with selective search criteria on data that comes in high velocity and adds up to large volumes (billions of records). To this end, the system is going to exploit and extend the infrastructure of Apache Spark system. Throughout the paper, we represent a roadmap for the accomplished contributions, on-going contributions towards the first cut realization of *Kite*, and future contributions to iteratively improve the system maturity and capabilities.

1. INTRODUCTION

The striking availability and richness of microblogs, e.g., tweets, reviews, and comments on news websites and on Facebook, has motivated a lot of efforts on analyzing microblogs. Examples of such efforts include event detection and exploration [23, 29], news extraction and delivery [7, 25, 27], user analysis [16], and rescue services [12]. All these applications use a set of common queries on different microblogs attributes. The most famous examples of such queries are "*find microblogs that have certain keyword(s)*", "*find microblogs that have posted in certain location(s)*", and "*find microblogs that have posted by certain user(s)*". With the distinguishing characteristics of microblogs data, these queries were not straight forward to be managed by the existing data management technology. In particular, microblogs arrive in high rates all the time, tens of thousands every second, and hence such a large number accumulates over time to make large volumes of historical data, billions every day. Moreover, microblogs queries come on both recent data in real time, that arrive with high velocity, and old data that reside in large volume archives. Selecting microblogs with certain *keywords* out of this data was challenging enough so that new real-time indexing techniques have been introduced to manage them [9, 11]. The new indexing infrastructures have been also introduced for *location* queries [8, 21, 28], *user* queries [30], and social-aware personalized queries [17].

Despite all existing work on indexing, querying, analyzing, and visualizing microblogs, whoever develops microblogs applications still has to implement major components from scratch with all the associated complications and challenges. This is mainly due to lack of a holistic system that glues all of these components together as means of managing microblogs data. Meanwhile, relying on existing data management systems is neither efficient nor practical as they have inherent limitations to manage microblogs. In particular, Database Management Systems (DBMSs) [26] cannot support microblogs as they are not equipped to deal with high arrival rates that come with microblogs. Such major limitation in DBMSs was a main reason that systems community has introduced Data Stream Management Systems (DSMSs) that have emerged as research projects (e.g., Aurora [1] and Trill [10]) and commercial products (e.g., Apache Storm [5] and Microsoft StreamInsight [2]). Although a DSMS can efficiently digest incoming data with high arrival rates, it is mainly designed and optimized to support the concept of *continuous* queries. Continuous queries register in the system ahead of time while incoming data are processed *upon arrival*, mostly in a single pass, to provide already registered queries with incremental answers. This is fundamentally different from the needs of microblogs queries where users are mostly asking about data that has already arrived through posing snapshot queries. Hence, data needs to be digested and indexed for answering future incoming queries. Though some DSMSs support data archiving, they do not support indexing, which is a major need for microblogs queries especially in-memory indexing of recent data that receive a high fraction of queries.

A recent trend is the development of various Big Data Management Systems (BDMSs), e.g., Apache Spark [4], AsterixDB [3], and Myria [13]. Although Apache Spark can process both fast and large data, it still cannot efficiently support queries with selective search criteria like microblogs queries. The main reason is that it is geared towards analytics applications that process a large percentage of the data, rather than selective queries that find few data

---

This research is capitally supported by NSF grants IIS-0952977, IIS-1218168, IIS-1525953, CNS-1512877, and the University of Minnesota Doctoral Dissertation Fellowship.

SIGMOD'16 PhD Symp, June 25-July 01 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2926693.2929898

items with certain keywords or user ids. On another hand, AsterixDB, Myria, and similar systems are primarily designed and optimized for efficient processing of big volume data, thus, they still cannot support fast data which is an essential part of microblogs data management. Lately, AsterixDB has been adapted for fast data ingestion [14]. Nevertheless, the system still cannot cope up with microblogs arrival rates as it forwards ingested data directly to disk without providing any main-memory indexing structures. Generally, systems that are primarily designed for handling big volume data has shown in [24] to be limited in practice to support fast data. Thus, handling big velocity has to be inherent in system design from the early beginning which is not currently supported in big-volume systems. This leads to a gap in existing systems as they do not provide the data management infrastructure that is appropriate to support microblogs queries. This gap limits them from supporting major microblogs applications and ease building new functionality on top of microblogs data.

Our research ultimately aims to build *Kite*; the first system that provides data management infrastructure for microblogs queries. *Kite* fills the gap in existing systems to support queries with selective search criteria on both fast data and large data. *Kite* is a full-fledged open-source system that would be available for everyone to build microblogs applications, just like how database systems eases building applications on top of relational data hiding all the underlying complications of managing the data. *Kite* supports index structures, query operators, memory management techniques, and SQL-like query language that are all geared towards the distinguishing characteristics of microblogs.

Kite makes use of the existing solid data management systems and extend their infrastructure to support microblogs. In particular, we extend Apache Spark system to add various indexing structures in both main-memory and disk storage. These index structures are exploited by a query processor that converts microblogs queries into a set of Spark operations on the supported structures. Therefore, *Kite* consists of three main components: (i) *Memory Indexer*, (ii) *Disk Indexer*, and (iii) *Query Processor*. The *Memory Indexer* is optimized to digest fast streams of microblogs data in main-memory indexes and equipped with efficient memory utilization techniques. The *Disk Indexer* is optimized for managing large data volumes in disk storage with minimal cost. The *Query Processor* is geared towards processing top-*k* and temporal queries, which are the most common aspects in microblogs queries [20]. The following sections introduce each component in a bit of details.

Kite is planned to go through three main milestones. The first milestone, which has been already accomplished, is to fill existing gaps in the literature of modules that provide microblogs data management. For this, we have successfully proposed a spatial real-time index structure for microblogs [21, 22] and novel main-memory flushing polices [19] that are able to fine tune memory utilization for microblogs queries. The second milestone, which is currently on-going, is to provide the first cut realization of *Kite* inside Apache Spark system and release it to the community to build on it. The primary release is planned to have only the essential modules [20] that enable users to build scalable applications on microblogs. The third milestone, which is planned to start by the end of this year, is to improve the primary release through adding the rest of planned modules [20] and enable easiness of extending the system by the research community.

The rest of this paper is organized as follows. Section 2 gives an overview about *Kite* system requirements and architecture. Sections 3, 4, and 5 describe the details of different *Kite* components, namely, *Memory Indexer*, *Disk Indexer*, and *Query Processor*, respectively. Section 6 introduces *Kite* SQL-like query language. Section 7 highlights *Kite* accomplished, on-going, and future milestones. Finally, Section 8 concludes the paper.

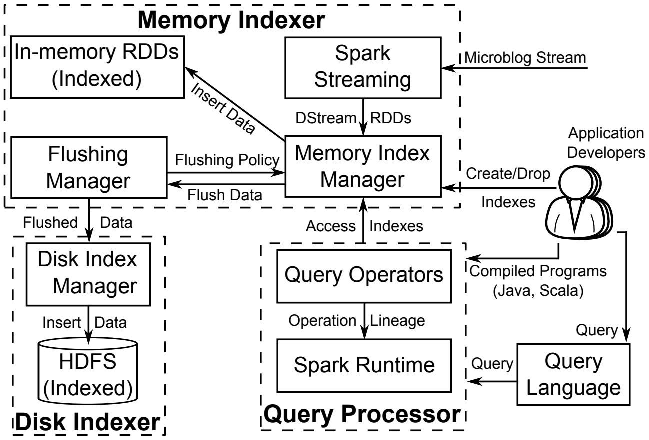

Figure 1: *Kite* System Overview.

2. SYSTEM OVERVIEW

Kite system is designed to primarily address the characteristics of microblogs data and queries. Specifically, microblogs data comes with high arrival rate of tens of thousands per second. Queries that exploit such fast data ask about both recent data that is few seconds old and historical data that is several months old. This obligates to provide native data management for both fast data in main-memory (for efficient digestion and high-throughput querying) and large data in disk (for scalable querying of large volumes). In addition, all microblogs queries are mostly temporal queries, where the query limits its search space to a certain temporal period due to the timely nature of microblogs. Combined with the temporal attribute, microblogs queries are dominated by either spatial attribute, keyword attribute, or both of them. Such characteristics drive *Kite* system to feature the required data management infrastructures that are capable to handle both fast and large data. Such infrastructure are equipped to promote temporal, spatial, and keyword attributes as first class citizens.

Figure 1 depicts our proposal for *Kite* system architecture. The system components are proposed to be realized within the ecosystem of Apache Spark system, exploiting its solid infrastructure and widely-used components. The system consists of three main components, namely, *Memory Indexer*, *Disk Indexer*, and *Query Processor*. *Kite* receives a stream of microblogs that are digested in the *Memory Indexer* with high arrival rates. The incoming data are indexed in main-memory index structures so that the high fraction of incoming queries that ask about recent data are evaluated efficiently from main-memory contents. Whenever the allocated memory budget of a certain index is filled, its data is subject to flushing to a corresponding disk index, inside the *Disk Indexer* component. The *Disk Indexer* is responsible for organizing historical data with large volumes that reaches hundreds of billions of data items. Such historical data is mainly queried by analytics applications, like getting microblogs that mention a certain presidential candidate over the last three months or analyzing microblogs that are related to Ebola epidemic spread over the last year. Both memory and disk indexes are created and/or dropped by system users, either system administrators or application developers, on arbitrary attributes of microblogs data. Meanwhile, developers of microblogs applications exploit the rich features of *Kite* through its *Query Processor* component in two ways: (i) direct calls from their Java or Scala programs, just like programming on top of Apache Spark, or (ii) SQL-like declarative query language that provides a familiar and easy interface for the underlying data management infrastructure. *Kite Memory Indexer*, *Disk Indexer*, *Query Processor*, and *Query Language* are briefly discussed in Sections 3-6 followed by an emphasize on the accomplished and remaining milestones of the system in Section 7.

3. MEMORY INDEXER

The *Memory Indexer* component organizes incoming microblogs in main-memory index structures to achieve: (i) scalable digestion of incoming data with high arrival rates, and (ii) efficient in-memory query processing on recent data, which represents a high fraction of incoming queries to *Kite*. The *Memory Indexer* uses *Spark Streaming* engine to digest and pre-process the incoming data stream. Then, *Kite* modifies the way that Spark partitions its main abstraction of Resilient Distributed Datasets (RDDs). In particular, *Kite* organizes RDDs as temporal index structures, so that data is partitioned based on its temporal recency. This is mainly motivated by the dominance of *temporal* dimension in microblogs queries, so that data within certain time range need to be retrieved efficiently. Also, a high fraction of queries come on the most recent data, so it is more efficient to partition them temporally. *Kite* supports three families of temporal index structures; temporal keyword indexes for keyword attribute, spatio-temporal indexes for location attribute, and a generic temporal hash index that is used for other microblogs attributes. The first version of *Kite* will support a temporal inverted index for keywords and a temporal partial quad tree for locations. The supported indexes are decided to promote spatial and keyword attribute as first class citizens, as they are dominant in microblogs queries and applications. Thus, optimized indexes are carefully designed for efficient retrieval on these two specific attributes.

Each in-memory index is allocated a maximum main-memory budget. Once the index fills the whole available memory budget, a *Flushing Manger* triggers a flushing process that selects a subset of in-memory data to spill to a corresponding disk-resident index. The first version of *Kite* is planned to implement two flushing policies: the temporal policy where the oldest microblogs are flushed [9] and the *kFlushing* policy where memory contents are smartly adjusted to support top-*k* queries [19]. To synchronize the operations between the Spark Streaming engine, the indexed RDDs, and the flushing manager, we add a new component, termed *Memory Index Manager*. The main job of this new component is to receive the pre-processed microblogs from Spark streaming engine, inserts them in the indexed RDDs based on catalog information about the existing indexes in the system, and triggers the execution of the flushing policy on certain index(es).

4. DISK INDEXER

Microblogs accumulates billions of data items every day, which forms hundreds of billions of historical data items. Such historical data is queried based on temporal, spatial, and keyword attributes for applications like social media analysis. To support these queries, *Kite* introduces the *Disk Indexer* component to Apache Spark echosystem. The main objective is to maintain a set of disk-resident index structures that correspond to the main-memory indexes. The *Disk Index Manager* receives the flushed data from the *Flushing Manager* and inserts them as one batch into corresponding disk indexes in Hadoop Distributed File System (HDFS). Each index consists of a set of HDFS blocks, where data in each block is grouped based on the index key attribute. Similar to in-memory index structures, disk-based structures are append-only temporal inverted index, temporal quad tree index, and temporal hash index.

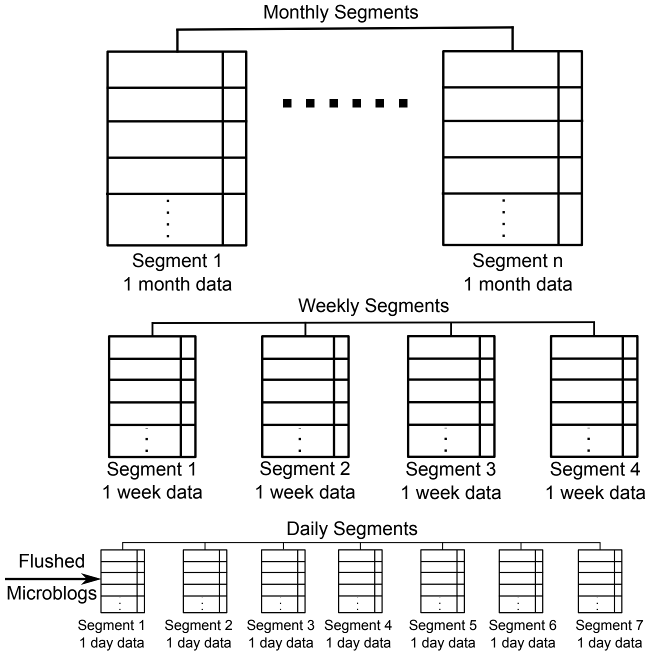

Figure 2: Example of Disk Index Temporal Hierarchy.

Unlike main-memory indexes that are primarily designed to support high digestion rates, disk indexes are designed to support queries on arbitrarily large temporal horizons (and in turn large data volumes). Thus, each disk index should be segmented and replicated in an *arbitrarily-defined* temporal hierarchy. Figure 2 shows an example for a hash disk index that is organized in a temporal hierarchy of (day, week, month). Thus, the index has three levels of segments, namely, daily segments, weekly segments, and monthly segments. Each daily segment index data of a single calendar day. For each calendar week days, daily segments are merged and replicated in one weekly segment, and so for the monthly segments. An incoming query accesses index segments within its temporal horizon so that it minimizes the response time. For example, a query that spans three weeks would access three weekly segments rather than searching twenty-one daily segments. This allows *Kite* to support relatively long-period queries with minimal querying overhead. The temporal hierarchy is arbitrary, e.g., (week, month, year) instead of (day, week, month), and can be defined by system admins based on the applications requirements.

Although *Kite* disk indexes replicate indexing overhead for same data over multiple hierarchy levels, this overhead is acceptable for two reasons: (1) Each level of replication adds approximately a storage overhead of 10% of the indexed data size which is an acceptable overhead with continuously reducing storage costs. (2) *Kite* disk index segments are read-only indexes and do not receive new data because they index historical microblogs that come in append-only fashion, hence, there is no index update overhead.

5. QUERY PROCESSOR

Kite query processor provides a set of generic operators that can be combined to support arbitrary queries on arbitrary microblogs attributes. Specifically, it provides the following operations: selection, aggregate count, projection, and join. All operations, except projection, mostly require top-k results. Thus, *Kite* is supporting ranking-aware query processing natively to evaluate top-k queries efficiently. The importance of top-k queries comes from the excessive numbers of microblogs data. Consequently, most of existing work on microblogs agree to put a limit k on the answer size [8, 9, 20, 30], so that the results are meaningful to end users.

All operators are expressed as Spark lineage, i.e., sequence or graph of basic Spark operations, on both in-memory RDDs and in-disk HDFS blocks. *Query Operators* module expands the incoming query into its corresponding operators and Spark lineage. Then, it forwards the lineage to *Spark Runtime* that executes the query on the underlying Spark cluster and returns the answer. In this section, we briefly sketch the distinguishing characteristics of processing the different operations in *Kite*.

Selection. With dominance of top-k queries, selection in *Kite* is top-k ranking-aware selection [15]. Incorporating top-k semantic inside query processor speeds up query latency significantly. *Kite* could use top-k selection on hash indexes as proposed in [30] and on spatial indexes as proposed in [21]. The basic idea is similar to ones presented in DBMS literature [15]. Each index entry stores multiple data lists that ordered on different partial ranking scores. Then, the lists are traversed in order to aggregate the final ranking score which is usually a monotone function of the partial scores.

Aggregate count. *Kite* does not support separate indexes for count aggregation like the proposed ones in [8, 28]. Instead, *Kite* exploit the indexes that are presented in Sections 3 and 4. Each index entry stores the count of individual microblogs in the entry. These counts are combined on the fly to get the final count for the query parameters. Due to the discrete nature of microblogs attributes, e.g., keyword or language, *Kite* uses a hash-based technique to perform efficient counting.

Join. In practice, join operations on multiple microblogs streams are currently rare and mostly involve equality comparisons, i.e., equi-join queries. Thus, a suitable technique for such operation is hash join. If hash indexes exist on join attribute, they are directly used for a classical hash join implementation. Otherwise, concise hash structures should be built and used for efficient implementation as described in [6].

Projection. In *Kite*, projection is useful to reduce the size of intermediate *disk-resident* data during query processing. This is mainly because of the relatively large number of attributes that come with microblogs, e.g., 63 attributes per tweet. This is not applicable to main-memory data as microblogs are stored as objects with random access to all attributes, unlike disk data that are stored in records with attributes stored sequentially. On another hand, projection is traditionally, e.g., in DBMSs, challenging for removing duplicates from final answer. However, removing duplicates is not important in *Kite* because most of search queries ask about microblogs text which is rarely in full duplicated.

6. QUERY LANGUAGE

Kite query language consists of three main statements: (1) **`CREATE (STREAM|INDEX)`**, (2) **`SELECT`**, and (3) **`DROP (STREAM|INDEX)`** statements in addition to auxiliary statements and commands like **`SHOW`**, **`DESC`**, and **`ALTER`**. For presentation simplicity, we introduce only **`SELECT`** statement that is used to pose queries on microblogs. Both **`CREATE`** and **`DROP`** are similar to the typical statements in the standard SQL.

```
⋆ SELECT attr_list FROM stream_name
[WHERE condition]
TOP-K k ORDER BY F(arg_list)
TEMPORAL (T_start,T_end)
```

```
⋆ SELECT grouping_attr_list,
COUNT(attr_list)
FROM stream_name
[WHERE condition]
GROUP BY grouping_attr_list TOP-K k
TEMPORAL (T_start,T_end)
```

`SELECT` statement supports basic search queries that retrieve individual microblogs (the first variation) and aggregate queries that retrieve aggregate counts on microblogs (the second variation). Both types of queries are top-k queries and include temporal aspect due to their exceptional importance in microblogs. If a query needs to omit declaring a specific time range or k, it should use special values ∞ and $-\infty$ to intentionally show the need to process all stored data or return all matching items. This prevents users from mistakenly submit poorly performing queries.

Example 1. The following basic search query retrieves the most recent 20 tweets that mention both keywords *Obama* and *Care*:

```
SELECT *
FROM twitter_name
WHERE keyword CONTAINS ALL {Obama, Care}
TOP-K 20
ORDER BY Max(timestamp)
TEMPORAL (-∞,NOW)
```

Example 2. The following aggregate query retrieves the most frequent 10 keywords from tweets in Ukraine since February 18, 2014:

```
SELECT keyword, COUNT(*)
FROM twitter_name
WHERE location WITHIN (52,44.7,39.91,21.8)
GROUP BY keyword
TOP-K 10
TEMPORAL ("18 Feb 2014",∞)
```

7. *Kite* MILESTONES

Kite plan has three main milestones. The first milestone has proposed a full system architecture [18, 20], and hence identified gaps in the existing literature of microblogs data management. We filled the identified gaps in our work on real-time spatial querying [21, 22] and main-memory flushing polices [19]. In nutshell, we have proposed a main-memory spatial index structure [21, 22] that is optimized to support real-time indexing and scalable spatial queries on microblogs. The index uses a partial quad-tree that is equipped with batch insertion, lazy deletion, and efficient index restructuring operations. The new operations significantly reduce the overall indexing overhead and hence tens of thousands of data items can be indexed every second. In addition, we have proposed a novel main-memory flushing policy [19] that is tailored to tune memory utilization for top-k queries, which are the dominant queries on microblogs data. The policy basically identify in-memory data items that are not contributing, or less contributing, to incoming queries. Such data items become victims for the next flushing operation. By identifying and flushing the least useful data, our policy is able to significantly boost main-memory hit ratio, so that much more queries are answered entirely from main-memory contents achieving more efficient query evaluation and better memory resource utilization.

The second milestone is to realize our proposed components and system architecture inside Apache Spark system and release it to the community to build on it. This milestone is currently on-going as described throughout this paper. The first release of *Kite* is planned to have the described index structures, flushing policies, and query operators. These modules enable users to build scalable applications on microblogs. In its third milestone, which is planned to start by the end of this year, *Kite* primary release is planned to add a query optimizer and additional querying capabilities as envisioned in [20]. Also, it is important to ensure the easiness of extending the system so that it can be incubated by the research community.

8. CONCLUSION

In this paper, we have introduced *Kite*; the first microblogs data management system that is designed to address the distinguished characteristics of microblogs data. *Kite* is built within the echosystem of Apache Spark system, exploiting its solid data management infrastructure and adding a major extension to enable efficient querying of microblog data. Specifically, the system can digest fast data with high arrival rates, up to tens of thousands per second, in main-memory index structures. When the allocated memory budget is filled, a portion of in-memory contents is flushed to corresponding disk index structures. The disk indexes are partitioned in temporal slices so that it could serve hundreds of billions of data items that come in append-only fashion. Both memory and disk index structures can be built on any microblog attribute, yet, they are promoting temporal, spatial, and keyword attributes as first class citizen due to their dominance in microblogs queries. Meanwhile, *Kite* features are exploited through its query processor, either through programming language APIs or a SQL-like declarative query language. The supported query language provides a set of generic operators that can be combined to post a wide variety of queries on arbitrary microblogs attributes. Towards building the system, we have identified and filled certain gaps in the literature of microblogs data management. The system is currently being realized inside Apache Spark system and is planned to be released to the research community to build upon it.

9. REFERENCES

[1] Daniel J. Abadi and et. al. Aurora: A New Model and Architecture for Data Stream Management. *VLDB Journal*, 12(2), 2003.

[2] Mohamed H. Ali and et. al. Spatio-Temporal Stream Processing in Microsoft StreamInsight. *IEEE Data Engineering Bulletin*, 33(2), 2010.

[3] Sattam Alsubaiee and et. al. AsterixDB: A Scalable, Open Source BDMS. *PVLDB*, 7(14), 2014.

[4] Apache Spark. https://spark.apache.org/, 2014.

[5] Apache Storm. https://storm.apache.org/, 2014.

[6] R. Barber, G. Lohman, I. Pandis, V. Raman, R. Sidle, G. Attaluri, N. Chainani, S. Lightstone, and D. Sharpe. Memory-Efficient Hash Joins. In *VLDB*, 2015.

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

[8] Ceren Budak, Theodore Georgiou, Divyakant Agrawal, and Amr El Abbadi. GeoScope: Online Detection of Geo-Correlated Information Trends in Social Networks. In *VLDB*, 2014.

[9] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. Earlybird: Real-Time Search at Twitter. In *ICDE*, 2012.

[10] Badrish Chandramouli and et. al. Trill: A High-Performance Incremental Query Processor for Diverse Analytics. In *VLDB*, 2015.

[11] Chun Chen, Feng Li, Beng Chin Ooi, and Sai Wu. TI: An Efficient Indexing Mechanism for Real-Time Search on Tweets. In *SIGMOD*, 2011.

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

[13] Daniel Halperin et. al. Demonstration of the Myria big data management service. In *SIGMOD*, 2014.

[14] Raman Grover and Michael Carey. Data Ingestion in AsterixDB. In *EDBT*, 2015.

[15] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACS*, 40(4), 2008.

[16] Jinling Jiang, Hua Lu, Bin Yang, and Bin Cui. Finding Top-k Local Users in Geo-Tagged Social Media Data. In *ICDE*, 2015.

[17] Yuchen Li, Zhifeng Bao, Guoliang Li, and Kian-Lee Tan. Real Time Personalized Search on Social Networks. In *ICDE*, 2015.

[18] Amr Magdy, Louai Alarabi, Saif Al-Harthi, Mashaal Musleh, Thanaa Ghanem, Sohaib Ghani, and Mohamed Mokbel. Taghreed: A System for Querying, Analyzing, and Visualizing Geotagged Microblogs. In *SIGSPATIAL*, 2014.

[19] Amr Magdy, Rami Alghamdi, and Mohamed F. Mokbel. On Main-memory Flushing in Microblogs Data Management Systems. In *ICDE*, 2016.

[20] Amr Magdy and Mohamed Mokbel. Towards a Microblogs Data Management System. In *MDM*, 2015.

[21] Amr Magdy, Mohamed F. Mokbel, Sameh Elnikety, Suman Nath, and Yuxiong He. Mercury: A Memory-Constrained Spatio-temporal Real-time Search on Microblogs. In *ICDE*, 2014.

[22] Amr Magdy, Mohamed F. Mokbel, Sameh Elnikety, Suman Nath, and Yuxiong He. Venus: Scalable Real-time Spatial Queries on Microblogs with Adaptive Load Shedding. *TKDE*, 2016.

[23] Michael Mathioudakis and Nick Koudas. TwitterMonitor: Trend Detection over the Twitter Stream. In *SIGMOD*, 2010.

[24] Gilad Mishne, Jeff Dalton, Zhenghua Li, Aneesh Sharma, and Jimmy Lin. Fast Data in the Era of Big Data: Twitter's Real-time Related Query Suggestion Architecture. In *SIGMOD*, 2013.

[25] Owen Phelan, Kevin McCarthy, and Barry Smyth. Using Twitter to Recommend Real-Time Topical News. In *RecSys*, 2009.

[26] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems (3rd ed.)*. McGraw-Hill, 2003.

[27] Jagan Sankaranarayanan, Hanan Samet, Benjamin E. Teitler, Michael D. Lieberman, and Jon Sperling. TwitterStand: News in Tweets. In *GIS*, 2009.

[28] Anders Skovsgaard, Darius Sidlauskas, and Christian S. Jensen. Scalable Top-k Spatio-temporal Term Querying. In *ICDE*, 2014.

[29] Tracking Disease Trends. http://nowtrending.hhs.gov/, 2015.

[30] Lingkun Wu, Wenqing Lin, Xiaokui Xiao, and Yabo Xu. LSII: An Indexing Structure for Exact Real-Time Search on Microblogs. In *ICDE*, 2013.