Demonstration of *Kite*: A Scalable System for Microblogs Data Management

Amr Magdy Mohamed F. Mokbel

Department of Computer Science and Engineering, University of Minnesota
{amr,mokbel}@cs.umn.edu

I. Introduction

Motivated by its wide availability and richness, there have been a plethora of recent work in querying, analyzing, and visualizing microblogs (see [3] for a brief survey). Examples of microblogs include tweets, online reviews, and comments on news websites. Unfortunately, existing work in microblog lacks data management tools that provide the necessary infrastructure to support efficient storage, indexing, and retrieval of microblogs. Hence, researchers, developers, and practitioners who need to process microblogs for their own purposes would need to either build their own ad-hoc techniques [5] or use any of existing general purpose big data engines, e.g., Spark, as their backbone infrastructure [4]. Relying on ad-hoc techniques does not scale neither in terms of data size nor in terms of supporting various functionality. Meanwhile, existing general purpose big data engines are built in a generic way to support various data and query workloads and are not equipped to support the specific characteristics of microblogs [2]. This results in sub par performance when supporting microblogs.

Queries on microblogs have three main distinguishing characteristics [2]: (1) All queries are temporal. For example, if a user issued a query like "*find tweets about Obama*", without explicitly specifying a temporal interval, the underlying system will add a default temporal interval, e.g., last week, otherwise we will get tweets from ten years ago, which is not practical. (2) All queries are top-k. For any issued query, even if it is not mentioned explicitly, the underlying system will add a default k to limit the size of the answer to top-k according to a specified or default ranking function, otherwise we will get an excessive number of tweets for every query. If the user needs more than k results, there is an option to retrieve the next k microblogs according to the same ranking function. (3) Keyword and spatial queries are very popular. A significantly high ratio of queries posed on microblogs is either asking for microblogs containing a certain keyword(s) or posted within a certain area. Unfortunately, these distinguishing characteristic are not well supported by *general purpose* systems.

In this demonstration, we present *Kite*; a data management system tailored to the specific needs of microblogs data and query workloads. *Kite* aims to be the standard platform for accessing microblogs, allowing other researchers, developers and practitioners to focus on their data analysis tasks and/or applications without worrying on the underlying management and retrieval of microblogs data. Distinguished from our work in [1], *Kite* is not tailored for a specific family of queries on certain attributes, instead it could query arbitrary attributes featuring a wide variety of queries and analysis capabilities.

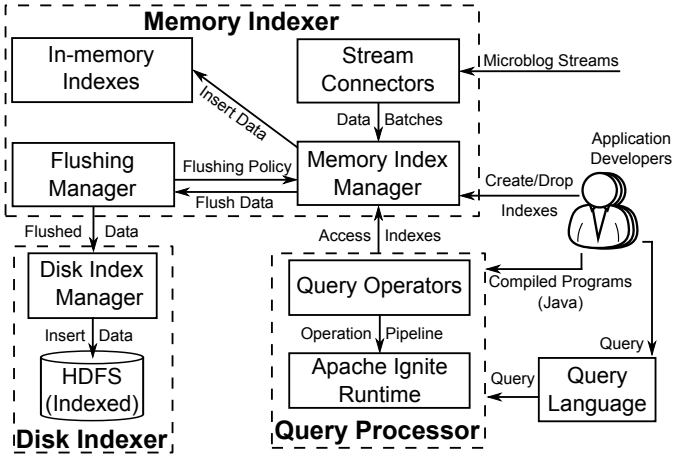

Fig. 1. *Kite* System Overview.

To this end, *Kite* is equipped with light spatial and keyword in-memory index structures that can digest the high rate of incoming microblogs. Then, the in-memory indexes are equipped with a buffer manger policy that is specifically designed to favor top-k and temporal queries. When the memory is full, a portion of the data is evicted from memory and goes to secondary storage index structures that are organized temporally and built in the Hadoop Distributed File System (HDFS) for both spatial and keyword queries. A scalable query processor is exploiting these system infrastructures to provide efficient querying on arbitrary microblogs attributes for both in-memory and in-disk data. All system facilities are accessed through a declarative SQL-like query language to ease building applications on top of microblogs data.

We demonstrate *Kite* with a system deployment on Amazon EC2 and using real tweets. The demonstration scenarios include showing the system capabilities in terms of scalability and query performance, in addition to showing the easiness of building rich applications on top of *Kite*.

II. System Overview

Figure 1 depicts *Kite* system overview. The system consists of three main components, namely, *Memory Indexer*, *Disk Indexer*, and *Query Processor*. The system components are realized exploiting the in-memory infrastructure of Apache Ignite system. *Kite* receives streams of microblogs that are digested in main-memory indexes with high arrival rates. Whenever the allocated memory budget of a certain index is filled, its data is subject to flushing to a corresponding disk index. Indexes are created and/or dropped by system users on

arbitrary attributes. Meanwhile, application developers exploit the rich features of *Kite* through either Java programming APIs, just like Hadoop or Spark, or SQL-like query language. The different system components are briefly discussed below.

A. 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* encapsulates *Stream Connectors* to digest and pre-process the incoming data streams. Then, the resulted batches of data are inserted in main-memory indexes that are primarily organized *temporally*, as most recent ones are likely to be queried more than older ones. *Kite* currently supports a temporal inverted index for the keyword attribute, a temporal partial quad tree for the spatial attribute, and a temporal hash index that is used for other microblogs attributes, all based on Apache Ignite in-memory structures. Each index is allocated a memory budget. Once the index memory is filled, a *Flushing Manger* selects a subset of in-memory data to spill to a corresponding disk-resident index. The *Memory Index Manager* synchronizes the workflow between the stream connectors, the indexes, and the flushing manager.

B. Disk Indexer

Kite adds a *Disk Indexer* component to Apache Ignite, so that it maintains a set of disk-resident index structures that digest the flushed data from the main-memory ones. 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 disk structures, disk-based structures are append-only temporal inverted index, temporal quad tree index, and temporal hash index. Each disk index is organized in temporal slices to efficiently support temporal queries that are dominant in microblogs queries.

C. Query Processor

Kite query processor provides a set of generic operators that can be combined to support arbitrary queries on arbitrary microblogs attributes. In specific, it provides **`SELECT`**, **`PROJECT`**, **`JOIN`**, **`TEMPORAL`**, (**`TOP-K`**, **`ORDER BY`**), and (**`COUNT`**, **`GROUP BY`**) operators. The first three operators are similar to the ones supported in the standard SQL. **`TEMPORAL`** determines the temporal horizon of the query, due to the importance of temporal queries in microblogs. The combination of **`TOP-K`** and **`ORDER BY`** provides a native support for top-k queries using different ranking functions. Currently, *Kite* provides a set of pre-defined widely-used ranking functions to rank the top-k results. However, the subsequent versions of the system is planned to support user-defined ranking functions for more flexibility for application developers. Finally, the combination of **`COUNT`** and **`GROUP BY`** is used to submit aggregate queries that find most frequent items for certain attribute, e.g., find the most frequent keywords in Minneapolis. Such count queries are very popular in nowadays microblogs applications to find trendy topics among microblogs users.

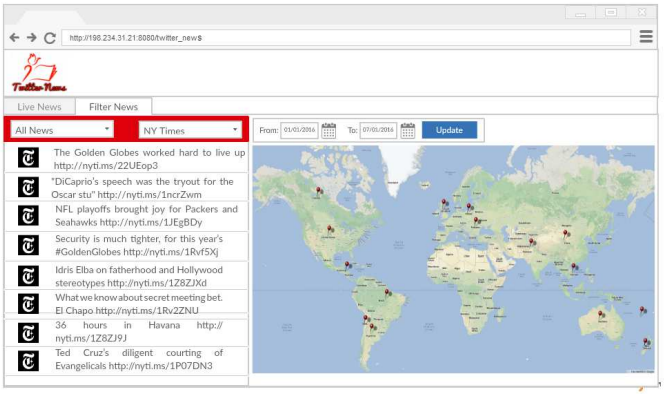

Fig. 2. Twitter News Application.

III. Demonstration Scenarios

Kite system is demonstrated using a real system deployment on an Amazon EC2 cluster of 20 nodes and using a rich dataset of real tweets. The dataset consists of ten billions tweets collected through Twitter APIs along three years. The tweets are fed to the system as a fast stream of 40,000 tweets/second. Our demonstration attendees would be able to interact with *Kite* trough one of the following scenarios.

Scenario 1: Admin Console. In this scenario, the demo attendees would be able to administrate the system through a console interface. The console facilitates connecting streams of data, creating and dropping indexes, and logging information about on-going system operations.

Scenario 2: Query Console. The console also allows the demo attendees to post queries to the system using the SQL-like query language. The console shows the results as well as monitoring query performance through logging information such as query time, touched index(es), and cluster nodes.

Scenario 3: Scalable Querying. In this scenario, we show the scalability of *Kite* compared to existing systems in searching tweets, both real-time and historical tweets. The demo attendees would post queries to find tweets that satisfy certain spatial or keyword predicates and see the query performance.

Scenario 4: Building Microblogs Applications. In this scenario, the demo attendees would experience building a quick application easily using *Kite*. The application is a *news aggregator* (Figure 2) that displays and organizes tweets that are published by news media, e.g., New York Times on Twitter. The attendees will witness creating an index on *user id* and posting a query to *Kite* to find news by source among 750 different news accounts on Twitter.

References

[1] A. Magdy et. al. Demonstration of Taghreed: A System for Querying, Analyzing, and Visualizing Geotagged Microblogs. In *ICDE*, 2015.

[2] A. Magdy et. al. Towards a Microblogs Data Management System. In *MDM*, 2015.

[3] A. Magdy et. al. Microblogs Data Management Systems: Querying, Analysis, and Visualization. In *SIGMOD*, 2016.

[4] G. Mishne et. al. Fast Data in the Era of Big Data: Twitter's Real-time Related Query Suggestion Architecture. In *SIGMOD*, 2013.

[5] W. Feng et. al. STREAMCUBE: Hierarchical Spatio-temporal Hashtag Clustering for Event Exploration Over the Twitter Stream. In *ICDE*, 2015.