Towards a Microblogs Data Management System (Invited Industrial Paper)

Amr Magdy Mohamed F. Mokbel

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

***Abstract*—This paper advocates for the need to build a Microblogs Data Management System (MDMS) as an end-to-end data management system to support indexing, querying, and analyzing microblogs, e.g., tweets, comments, or check-in's. We identify a set of characteristics for microblogging environments that are distinguishing from any other data management environment. Then, we propose a system architecture for the first Microblogs Data Management System, which includes indexing, querying, and recovery components. The indexing component is responsible for indexing recent data in memory, indexing older data in disk, and synchronizing the flow of data from memory to disk without affecting the query response time. The querying component is responsible for retrieving the query answer from both memory and disk storage as well as employing online selectivity estimation techniques tuned to the behavior of microblogs data. The recovery module allows for efficiently storing and processing incoming microblogs in memory without worrying about data loss.**

I. Introduction

Microblogs, e.g., tweets, Facebook comments, or news websites comments, are witnessing unprecedented flourishing era with the widespread of mobile devices and users. Everyday, 288+ Million active Twitter users generate 500+ Million tweets [60], while 1.39+ Billion Facebook users post 3.2+ Billion comments [25]. The vast majority of microblogging activity comes from mobile users, specifically, 80+% of Twitter users and 85+% of Facebook users are mobile. Such microblogs form a stream of user-generated data with rich attributes that include keywords/hashtags, news items, location, timestamp, language, social interactions, and users information. This richness is combined with important contents that have motivated several applications. For example, Twitter users propagate news faster than conventional media, e.g., Michael Jackson death and Boston Marathon explosions [14]. In addition, rescue teams have used microblogs to locate and save people during China floods [21]. Moreover, microblogs users are actively posting about short term events, e.g., sports games, and extended events, e.g., Iranian elections [34], US elections [62], Arab Spring [9], and Ukraine conflict [61]. This rich and important content has motivated several new applications including social journalism, event tracking [58], social media analysis [57, 63], and geo-targeted advertising [49].

The richness and importance of microblogs data has triggered a plenty of research to index, query, analyze, and visualize such data. With respect to indexing, various techniques have been proposed to index incoming microblogs in memory for scalable digestion [15, 16, 40, 41, 56, 65, 66] and in disk for continuously accommodating new data in-memory [20, 40]. With respect to querying, many queries of different types have been addressed on microblogs data. This includes basic search queries that retrieve individual microblogs [16, 20, 36, 41, 42, 65] or aggregate queries that retrieve information about microblogs, e.g., frequent keywords, spatial densities, or trending topics [15, 30, 51, 53, 56]. This rich set of queries enabled different types of analysis on microblogs including news extraction [50, 54], event detection [3, 26, 53, 55], event analysis and tracking [58, 63], recommendation [17, 30, 50], sentiment and semantic analysis [13, 46, 48], user-centric analysis [30, 35, 64], topic extraction [32, 51], geo-targeted advertising [49], and generic social media analysis [57, 67]. With the plethora of microblogs data and queries, efforts have been made to provide efficient and effective visualization for microblogs applications [27, 31, 39, 43, 54, 58].

Despite all such 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) [52] cannot support microblogs as they are not equipped to deal with high arrival rates that come with microblogs. For example, Twitter arrival rate is ~6,000 tweets/second, which is beyond what any DBMS can digest.

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], Borealis [2], Nile [29], STREAM [10], TelegraphCQ [19], and Trill [18]) and commercial products (e.g., Apache Storm [8], Microsoft StreamInsight [4], IBM System S [6], and AT&T Gigascope [23]). 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

This work is partially supported by the National Science Foundation, USA, under Grants IIS-0952977 and IIS-1218168.

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 [7] and AsterixDB [5]. BDMSs are primarily designed and optimized for efficient processing of big volume data, which can be supported through either in-memory lightweight distributed processing, e.g., Spark, or disk-resident index-based distributed processing, e.g., AsterixDB. Although supporting big volume data is necessary to handle the large number of microblogs, it is not sufficient as microblogs need inherent support for fast streaming data as well. Lately, BDMSs have provided facilities to support ingestion of fast data. Specifically, Apache Spark has provided Spark Streaming and AsterixDB has been adapted for fast data ingestion [28]. Nevertheless, both systems facilitate fast data ingestion without supporting any indexing on streaming data, which is essential to support scalable microblogs queries (see [15, 16, 41, 56, 65] for examples). Generally, systems that are primarily designed for handling big volume data has shown in [47] to be limited in practice to support fast data. Thus, handling big velocity has to be inherent in system design from the early beginning. Specifically, microblogs need in-memory index-based distributed processing and disk-resident index-based scalable organization which is not currently supported in any system.

In this paper, we advocate for the need to build specialized data engines to support microblogs, termed Microblogs Data Management Systems (MDMS). We believe that microblogs data and their query workloads are distinguished enough to warrant the need for building specialized management systems for indexing and querying microblogs. An MDMS would work as a backend that hides all complications of managing microblogs from end users. MDMS users would include application developers as well as researchers from many research disciplines, e.g., machine learning [37], human-computer interaction [11], social sciences [44], and/or medical sciences [22]. Users of such systems would be able to write SQL-like queries to express a myriad of useful functions posed to microblogs data and thus seamlessly support scalable applications on top of microblogs; just like DBMS for database-centric applications. In addition, an open-source MDMS would allow researchers, developers, and practitioners to add their own specialized functionality, indexing, and/or query operators.

Query workloads that come on microblogs are mostly dominated by temporal, spatial, and keyword attributes. Streaming nature, highly dynamic contents, and excessive numbers of microblogs have led to dominance of temporal aspect in the queries [16, 47]. Moreover, the mobility aspect of microblogs activity has contributed significantly in availability of location information and importance of spatial querying. Obviously, keywords represent the core user-generated textual contents that carries different information. Yet, other attributes, e.g., language, are still sporadically queried. Thus, Microblogs Data Management Systems (MDMS) should support SQL-like queries that are posted on *any* microblog attribute promoting temporal, spatial, and keyword attributes as first-class citizens. MDMS queries are distinguished by two main characteristics: (1) Pervasiveness of top-k queries: All queries on microblogs are by default top-k queries, even though if the user has not specified this explicitly. This is mainly due to the massive number of microblogs that can satisfy any query. So, it becomes impractical to report everything that satisfy the query predicates. (2) Pervasive use of temporal dimension: All queries on microblogs must have by default a temporal dimension, even though if the user is not asking for it explicitly. Without a temporal dimension, a query will be asking for all microblogs that were posted even from years ago. This is both impractical to search billions of microblogs for each incoming query and does not make any sense in the query answer.

To support such queries on data that is fast and large, Microblogs Data Management Systems (MDMS) should support indexing that is distinguished from traditional index structures as it needs to be optimized for three important features: (1) *High digestion rate*. It is important for the underlying index structure to be able to digest incoming microblogs as they arrive without losing any data. (2) *Low searchability latency*. The searchability latency refers to the time between the arrival of a microblog and the time that this microblog becomes searchable, i.e., can appear in a query result. Index structures in MDMSs should be optimized to decrease the searchability latency as a high fraction of incoming queries is interested in getting its answer from very recently posted microblogs within the last few seconds or minutes. (3) *High memory hit*. Recently incoming microblogs have to be stored and indexed in memory as it includes highly accessible data. However, memory fills out frequently and we have to flush parts to disk to free a portion of memory and accommodate more incoming microblogs. Index structures in MDMSs should deploy efficient flushing techniques that smartly decide on which portion of memory to flush to disk in a way that increases the memory hit of incoming queries so that most of them are answered entirely from in-memory contents.

To accommodate for the needs of Microblogs Data Management Systems (MDMSs), we present our proposal for the first architecture skeleton of what an MDMS should look like. We identify three core components, namely, *indexer*, *query engine*, and *recovery manager*. The *indexer* is responsible for handling microblogs data end-to-end. It digests new incoming data in real time with high arrival rates in main-memory. Once memory becomes full, a flushing policy is imposed to move portion of memory contents to disk. Disk contents are indexed in a scalable organization of historical data for long time periods. The *query engine* employs new *selectivity estimation* techniques that accommodate for: (a) the nature of microblog queries that include top-k and temporal search, and (b) the high rate of incoming microblogs, and hence the highly dynamic environment that makes selectivity estimation is highly variable over time. Meanwhile, the *recovery manager* is preventing data loss for the hundreds of millions microblogs that are managed in main-memory. Our proposal also includes a SQL-like query language that is equipped with additional constructs to support microblogs.

The rest of the paper is organized as follows: Section II outlines system characteristics. Section III outlines the system proposal. Sections IV, V, and VI detail our vision for the indexing, querying, and recovery modules. Section VII proposes the query language. Section VIII concludes the paper.

II. System Characteristics

This section characterizes Microblogs Data Management Systems (MDMS). First, we discuss characteristics of query workloads that come on microblogs. Then, we discuss distinguished characteristics of both supported queries and indexes in MDMS.

A. Query Workloads

Query workloads that come on microblogs are dominated by temporal, spatial, and keyword attributes. With respect to temporal, the streaming nature and highly dynamic contents of microblogs have led to dominance of temporal aspect in microblog queries. On one hand, submitted queries are mostly interested to retrieve their answers from recent data. This is mainly driven by high rates of users activity that lead to an overwhelming number of microblogs that match any query predicates. Thus, recency is playing a major role in relevance ranking in microblogging platforms [16]. Recency role is even boosted by time-sensitive content that are posted by social media users like real-time updates on events, e.g., sports games or celebrations, and up-to-moment news in major accidents [14] and natural disasters [21, 53]. Motivated by this real-time fashion, the White House has used a Twitter hashtag for real-time question/answer communication with the public after President Obama's State of the Union address in 2015. Even analytical queries that come on old data [57, 58, 63] are still looking for data that is bounded in certain time range, e.g., US elections tweets are posted from September to November 2012. On another hand, microblogs contents are very dynamic over time. For example, trending queries and associations among microblog hashtags/topics are shown to be varying in a time window of just few minutes [38, 47]. Such distinguished temporal nature has resulted into major revisions in traditional work on both data management and data analysis to cope up with this fundamental change (see [3, 16, 26, 36, 40, 41, 45, 48, 50, 53, 65]).

With respect to spatial, the widespread of GPS-equipped mobile devices has led to unprecedented availability of location information. Microblogs services are among affected services with the new wave of location-based services. Major microblogging service providers support geotagging their data, e.g., Facebook Places, Twitter geotagging, and Foursquare check-in's. This explosive growth in location information in microblogs has motivated many recent applications and algorithms for location-based services on microblogs. For example, spatial-aware news delivery. Los Angeles Times reported [14] how people rush to Twitter for real-time breaking news about Boston Marathon explosions in April 2013. Such users are basically interested in recently posted microblogs in a particular spatial region. Similarly, Chinese rescue teams have used microblog-propagated information to locate and save tens of people during floods in Beijing [21]. Other applications include advertising services to serve geo-targeted ads to their customers based on user activity or nearby events [49], localized event detection and tracking, and nearby friend finding. Such important applications have triggered an interest to support both fundamental spatial queries [41], e.g., range and kNN, and spatial-aware aggregate queries on microblogs, e.g., localized frequent keywords [56], local user search [35], and location-topic associations [15].

With respect to keyword, microblog text represents the main user-generated content that contains news, messages, opinions, reviews,...etc. Intuitively, any query or application would be interested in microblogs textual contents. This include both basic search queries that provide end users with individual microblogs to read and aggregate queries that analyze microblogs contents to provide higher level functionality, e.g., recommendation, news extraction, or event detection. Generally, keyword is the most rich attribute that is involved in almost all queries and applications (e.g., [3, 16, 24, 26, 36, 38, 40–43, 50, 54, 65]).

With respect to other attributes, e.g., user and language, they are still queried and used sporadically [27, 35] but with much less frequency. Therefore, Microblogs Data Management Systems (MDMS) should be generic and support queries on any microblog attribute. Yet, it should also promote temporal, spatial, and keyword attributes as first-class citizens to efficiently handle the described query workloads. Throughout the paper, we give details on native support of the three attributes in different system components.

B. Queries Distinguished Characteristics

Microblogs Data Management System (MDMS) should support arbitrary SQL-like queries that can be posed on any microblog attribute. MDMS queries are distinguished by the following characteristics:

1. **Pervasiveness of top-k queries**: All queries on microblogs are by default top-k queries, even though if the user has not specified this explicitly, where k is a reasonable number for human users to navigate. This basically comes from the excessive numbers of microblogs data so that it is impractical for user to report all microblogs/keywords that satisfy query predicates. This approach is adopted by all major microblog service providers. For example, Twitter and Facebook keyword search provides, by default, only top-20 microblogs, nevertheless, users can request more results on demand. Moreover, aggregate queries, e.g., Twitter trending hashtags/topics, report only top-10 results due to multitude of posted keywords. Consequently, MDMS should be designed to primarily handle top-k queries. This would affect different components of the system. For instance, this triggers an importance for adopting ranking-aware query processing techniques that natively support top-k queries inside query engine. In addition, this may affect different aspects of index design so that it is optimized for the context of supporting top-k queries. For example, index may allow cells to keep track of top-k items or order items based on arbitrary ranking functions. As a result, index cost estimation models (that are used by the query optimizer) would be affected. Ultimately, as a generic system for handling microblogs, MDMS design should provide generic support of top-k queries, i.e., arbitrary ranking functions on arbitrary attributes, and should allow different optimizations for this.

2. **Pervasive use of temporal dimension**: All queries on microblogs must have, by default, a temporal dimension, even though if the user is not asking for it explicitly. This is because excessive number of incoming microblogs makes it impractical to search billions of microblogs for everyday queries. Thus, number of accessed microblogs per query need to be generally bonded by certain attribute range. With the streaming nature

and highly dynamic contents of microblogs data, time has a lead in incoming queries as explained in Section II-A. Hence, temporal dimension is the most powerful candidate to limit number of processed microblogs per query and should be included in every query. In agreement with this view, Twitter engineering team design their real-time index [16] and historical index [59] based on time shards. Consequently, MDMS should promote timestamp as the pivotal attribute that drive optimizations in different system components. In particular, it is pivotal to efficiently locate and retrieve microblogs that lie within certain temporal range, either recent or old range and either short or term range. To this end, all system indexes should be primarily organized based on timestamp regardless the indexed attribute. For example, for a keyword index, the whole index can be segmented into time shards where each shard is a keyword index for certain time period. Collectively, all shards compose a large index for all data while microblogs in certain time ranges are efficiently accessible. Such pivotal promotion for temporal dimension would heavily affect the system query optimizer. A single index would have many shards which gives a big overhead to maintain a cost estimator separately for each shard. Thus, cost estimation models (which are the key players in query optimizer) are expected to exploit temporal-aware index organization to invent selectivity estimators that give high estimation accuracy while keep an overall low-overhead storage and maintenance.

C. Index Distinguished Characteristics

Microblogs Data Management System (MDMS) indexes are mainly distinguished from traditional indexing by supporting real-time indexes that digest and organize recent data in main-memory. Thus, MDMS indexes should be optimized for the following performance measures:

i. **Digestion rate**. As microblogs are arriving in high rates, MDMS must digest them in comparable rates so that it does not miss any data. The higher the digestion rate, the more scalable MDMS for faster microblogs streams. Acceptable digestion rates for microblogs real-time applications are thousands microblogs per second [16, 41]. This means that we need to digest each incoming microblog in a fraction of a milli-second.

ii. **Searchability latency**. A new measure becomes of interest to gauge how fast incoming data would be available in search results [16]. We define this measure as the average latency between *any* microblog being available in the system till it becomes available in search results. We call this the searchability latency. Most of real-time applications that are built on top of microblogs would accept searchability no more than few seconds [16, 41, 65].

iii. **Memory hit**. MDMS manages hundreds of millions of microblogs that have recently arrived in main-memory and receives many queries on this recent data. Thus, it becomes important to measure memory hit performance. This can be seen from two aspects. First, we need to measure the number of queries that can retrieve their answers entirely from memory. The larger this number, the better overall query performance. Second, we need to measure percentage of in-memory data that is actually used to answer incoming queries. This makes sure that memory resources are efficiently utilized. The optimal is to have 100% of memory contents are actually used.

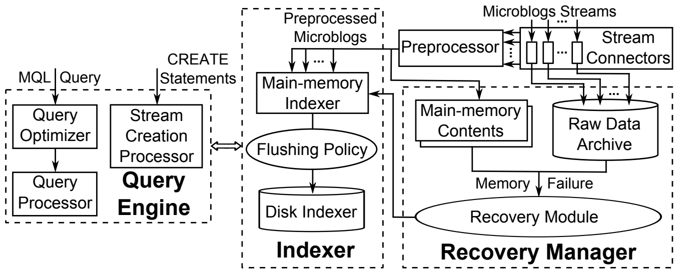

Fig. 1. MDMS System Architecture.

III. System Proposal

In this section, we propose a system architectural design to address the discussed challenges and requirements. Figure 1 gives our proposal for Microblogs Data Management System (MDMS) architecture. It consists of three core components and two auxiliary components. The core components include an indexer, a query engine, and a recovery manager. The auxiliary components include stream connectors and preprocessors. As the figure depicts, microblogs data comes to the system through stream connectors and firstly go to the preprocessor for parsing and any additional manipulation, e.g., keyword extraction. Then, the preprocessed microblogs are digested in system indexer efficiently with their high rates. System indexer handles real-time microblogs in a main-memory indexer. On full memory, a portion of main-memory data is flushed to disk indexer according to a certain flushing policy. On memory failure, the recovery manager combines raw data and partially-replicated main-memory contents to restore the system status. Meanwhile, users submit queries to system query engine that accesses data indexer to retrieve answers efficiently. The different components are briefly introduced below.

Stream Connectors. MDMS should support multiple microblogs streams simultaneously. Each incoming stream has a connector. The stream connector brings data from its source, e.g., Twitter firehose, to system buffering queues. Then, preprocessors consume data from the buffering queues to digest them into the system. Stream connectors is an auxiliary module that is not further discussed throughout the rest of the paper.

Preprocessor. The preprocessor module contains a sub-module for each connected stream. It gets the data from the system buffering queues, parses it, and performs any required additional manipulation based on the data type and format. For example, Twitter data preprocessor should parse JSON-formatted tweets into associative arrays and performs keyword and spatial information extraction. Then, preprocessed data are forwarded to system indexer to be available for incoming search queries. MDMS admin would have friendly interface to plug new stream connectors and preprocessors.

Indexer. MDMS indexer is the main module to handle microblogs data from its arrival to the system to long-term storage that keeps historical data. Thus, the indexer consists of three main sub-modules: a main-memory indexer, a disk indexer, and a flushing module. The main-memory indexer continuously digests incoming data that arrives with high rates in main-memory. In addition, with many queries come on recent data, main-memory indexer organizes recently arrived data in memory-resident indexes for efficient retrieval. Once main-memory becomes full, a portion of memory contents is flushed to disk-indexer according to a certain flushing policy. The flushing policy is responsible for deciding on which data should be kept in main-memory so that queries on recent data are highly likely to be answered from main-memory contents. This should reduce disk access during query processing and hence improve overall query performance. Like main-memory indexer, disk-indexer supports system indexes on disk-resident data. Such indexes should be efficient in retrieval of data that lie in arbitrarily long time horizons, e.g., several months. Disk contents represent an archival storage that keeps old data as long as the available disk storage allows. Our vision for MDMS indexer is presented in Section IV.

Query Engine. MDMS query engine is the main interfacing module with both system admins and system end users. It provides system admins with means to control and tune the system. In addition, it is responsible for processing end users queries on microblogs data. The query engine provides its interface as MQL statements: a SQL-like query language for defining and manipulating microblogs data. For system admins, MQL facilitates connecting new data sources, define data schemes, and tuning MDMS environment, e.g., index creation and tuning. For end users, MQL facilitates submitting SQL-like queries to express a myriad of useful functions on microblogs data. MQL queries can be posted on any attribute(s) and on both recent and old data. Facilitating such a rich set of queries introduces a need for query optimization. To elaborate, MDMS environment allows supporting multiple indexes and posing a single query on multiple attributes. Thus, a query may have multiple ways to be processed through different indexes and based on different attributes selectivity. As different query plans usually have different costs, it is essential to choose a cheap plan for efficient query processing. Consequently, to process end users queries, MDMS query engine employs two modules: a query optimizer and a query processor. The query optimizer, similar to a DBMS query optimizer, exploits catalog information about underlying data storage and system indexes to propose a cheap query plan. However, MDMS query optimizer is distinguished by employing novel selectivity estimators that address new challenges in MDMS indexes. Then, the query processor efficiently executes the plan against system indexes to retrieve the answer with low query latency. We present our vision for MDMS query engine in Section V and our proposal for MQL query language in Section VII.

Recovery Manager. With heavily relying on main-memory in managing hundreds of millions of microblogs data, accounting for memory failures is necessary to prevent significant data loss. To this end, MDMS employs a recovery manager component. In a highly dynamic environment that continuously receives excessive numbers of data and queries, recovery management is required to be: (a) light enough so that it does not interrupt other components and thus limits system scalability, and (b) effective so that it does prevent significant data loss. Thus, MDMS recovery manager consists of two main modules: a raw data archive and a partial replicator for main-memory contents. Both modules work in isolation from other system components, i.e., indexer and query engine, to sustain system scalability. On memory failure, data from both modules are combined through a recovery procedure to restore the status of system main-memory contents. Our vision for MDMS recovery manager is presented in Section VI.

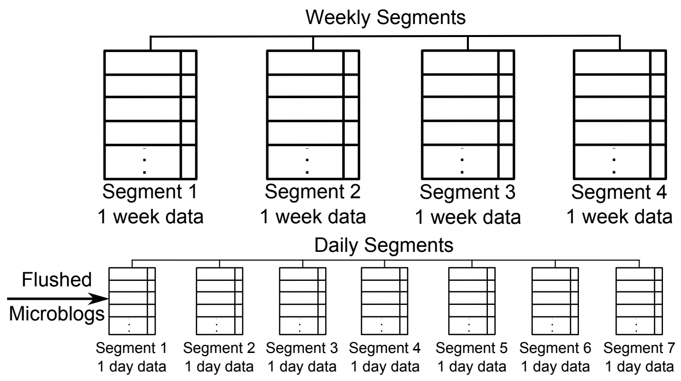

Fig. 2. Example of Disk Index Temporal Hierarchy.

IV. Indexer

Microblogs are user-generated data that come in excessive numbers with a rich set of attributes, e.g., timestamp, location, keywords, users information, and language. Microblogs Data Management System (MDMS) is a generic platform that should support queries, and hence indexes, on all microblog attributes and on both recent and old microblogs. To this end, MDMS should support two types of indexes: memory-resident indexes and disk-resident indexes. In each type, and motivated by exceptional importance of spatial and keyword attributes, MDMS should support at least three indexes: a spatial index, a keyword index, and a generic index for other attributes, e.g., hash index. All indexes though should incorporate temporal dimension as previously discussed (see Section II-B). Main-memory indexes are used for real-time indexing of recently posted microblogs while disk-resident indexes are used for microblogs that are flushed from memory to disk. We briefly outline our vision for both types of indexes and flushing management below.

Main-memory indexing. Memory-resident indexes should be equipped for digesting microblogs with high rates in real time. To support real-time digestion, each index should be: (1) organized in relatively small segments, i.e., each segment index relatively small number of microblogs, typically few tens of millions, and (2) equipped with efficient update techniques that reduce amortized insertion time per microblog. Existing proposals in the literature fit these requirements in spatial indexes [41], keyword indexes [16], and generic hash indexes [65]. For spatial indexes, MDMS should use space-partitioning index, e.g., quad-tree or gird, as proposed in [41, 56], because it provides lower restructuring overhead during real-time digestion and hence can support digestion rates higher than data-partitioning indexes, e.g., R-tree.

Disk indexing. 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. 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). Thus, the index has two levels of segments, namely, daily segments and weekly 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. 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 MDMS to support relatively long-period queries with minimal querying overhead. The temporal hierarchy is arbitrary, e.g., (week, month, year) instead of (day, week), and can be defined by system admins based on the applications requirements.

Although MDMS 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) MDMS 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.

Flushing management. MDMS flushing policies select which microblogs would be flushed from main-memory to disk when memory becomes full to ensure continuous digestion of new incoming data. This selection is very important as it controls main-memory contents from which a big family of queries are answered [15, 16, 36, 41, 56, 65]. Currently, motivated by the streaming nature of microblogs, the literature assumes only temporal flushing where the oldest data is flushed regardless its relevance to incoming queries. However, MDMS should employ flushing policies that take into account the characteristics of incoming queries. For example, pervasiveness of top-k queries can be exploited to keep only k relevant items, per search key, in main-memory. This would achieve significant improvements in memory consumption and memory hit. In general, flushing policies should take into account the context. Microblogs literature still has a room to study the flushing problem.

V. Query Engine

This section highlights challenges and opportunities in MDMS query engine. The query optimizer is discussed in Section V-A and the query processor is discussed in Section V-B.

A. Query Optimizer

Similar to a relational DBMS query optimizer, MDMS query optimizer selects a *cheap* query plan to execute the query, from alternative plans, e.g., multiple indexes to use or different operators order. The main challenge for MDMS query optimizer is to employ cost estimators that are: (a) temporal-aware for disk indexes, and (b) quickly-adaptive with real-time memory indexes. We briefly elaborate on both.

MDMS disk indexes are organized in a multi-resolution temporal hierarchy in which each level index same data and thus a query can be answered from any level. Consequently, MDMS query optimizer has to select a combination of index segments from different temporal levels to process query with minimum estimated cost. In addition, a single level may have a large number of segments. With plethora of segments in a single disk index, it is costly to store a traditional selectivity estimator for each segment separately. Instead, MDMS query optimizer should exploit the new index organization and extend current selectivity estimation techniques with a temporal dimension to minimize storage overhead of selectivity estimators and still provide high estimation accuracy.

Orthogonally, MDMS real-time indexes introduce a new direction to study, namely, *real-time selectivity estimation*. Selectivity estimators mainly maintain statistics about index contents to estimate for an incoming query the size of data processed through this index. Traditional selectivity estimators used to deal with indexes with fairly stable contents. However, in MDMS real-time indexes, the contents are changing very quickly. Thus, tradition selectivity estimators are not expected to be well-suited for real-time indexes. From one hand, the estimation accuracy is expected to degrade with quick index changes. From another hand, maintaining the estimation model may not scale for real-time digestion rates. This opens a room to explore new selectivity estimators that are able to adapt with fast index changes and keep high estimation accuracy while being equipped with low-overhead maintenance techniques.

B. Query Processor

MDMS query processor should execute both snapshot and continuous queries, each consists of one or more of following operations: selection, count aggregation, projection, and join. All operations, except projection, mostly require only top-k results. Thus, MDMS should support ranking-aware query processing. In this section, we briefly sketch broad lines for MDMS operations and continuous query processing.

Selection. With dominance of top-k queries, selection in MDMS should be top-k ranking-aware selection [33]. Incorporating top-k semantic inside query processor speeds up query latency significantly. MDMS could use top-k selection on hash indexes as proposed in [65] and on spatial indexes as proposed in [41]. The basic idea is similar to ones presented in DBMS literature [33]. 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.

Count aggregation. Our vision that MDMS should not support native indexes for count aggregation like proposed ones in [15, 56]. Instead, MDMS can exploit proposed indexes in Section IV to retrieve individual microblogs and then perform a counting operation. This is more scalable from a system point of view as the system supports all queries with lowest system-level overhead. Due to the discrete nature of microblogs attributes, e.g., keyword or language, MDMS could use 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 [12].

Projection. In MDMS, 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 MDMS because most of search queries ask about microblogs text which is rarely in full duplicated.

Continuous queries. In MDMS, continuous queries should be processed in two steps: (1) getting an initial answer, and (2) continuously reporting answer changes. Thus, all continuous queries are firstly processed like a snapshot query. Then, query is registered in the system to keep receiving answer updates. Continuous query processing in MDMS is similar to DSMS and should recycle ideas from its rich literature.

VI. Recovery Manager

Microblogs Data Management Systems (MDMS) would manage hundreds of millions of microblogs in main-memory structures. Therefore, accounting for memory failures is a must to prevent significant data loss. In a very dynamic environment like managing microblogs, recovery management has to be isolated from components that manage both data and queries so that it does not limit system performance and scalability. Thus, MDMS recovery management should be isolated a parallel universe to both indexer and query engine.

Our proposal for MDMS recovery management consumes a bit extra storage to replicate necessary data directly from data sources in total isolation from other core components. In particular, incoming raw data should be continuously flushed (one by one) to a large sequential disk archive that stores all raw data for whatsoever system failure. In addition, preprocessed data, that goes to main-memory indexer, should be replicated in main-memory of a configuration of machines that is similar to main-memory indexer machines. The amount of main-memory that is allocated for replication should be adjustable by system admins based on the allocated cost budget for recovery management. Thus, system admin should be able to replicate any percentage (0-100%) of memory contents.

On memory failure, the recovery machines would be able to partially process incoming queries during restoration of system status. Meanwhile, a recovery module would combine data from recovery machines and raw data archive to restore the system status. Any data that are not replicated in main-memory can be retrieved from the disk archive. However, accessing disk archive to retrieve, preprocess, and index missed data might lead to higher recovery time. Consequently, the larger replication memory, the faster recovery process.

VII. Query Language

In this section, we introduce microblogs query language (MQL); a proposal for SQL-like language to interact with Microblogs Data Management Systems (MDMS). MQL should consist of three main statements: (1) **CREATE**, (2) **SELECT**, and (3) **DROP** statements in addition to auxiliary statements and commands like **SHOW**, **DESC**, and **ALTER**. For space limitation, we introduce only **SELECT** statement that is used to pose queries on microblogs. **SELECT** statement is proposed as follows:

```
  ⋆ SELECT [CONTINUOUS] attr_list
FROM stream_name1 [,stream_name2,...]
[WHERE condition]
ORDER BY F(arg_list)
LIMIT k
ON {LAST T {MINUTES|DAYS} | (T_start,T_end)}
```

```
  ⋆ SELECT [CONTINUOUS] grouping_attr_list,
COUNT(attr_list)
FROM stream_name1 [,stream_name2,...]
[WHERE condition]
GROUP BY grouping_attr_list
LIMIT k
ON {LAST T {MINUTES|DAYS} | (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 (per Section II-B). If a query needs to omit declaring a specific time range or k, query should use special values ∞ and $-\infty$ to intentionally shows it wants to process all stored data or return all matching items. Basic search queries rank answers based on a user-defined function `F` while aggregate queries rank answers based on the count. We next present two examples of MQL queries.

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

```
  SELECT *
FROM twitter_stream
WHERE keyword CONTAINS ALL {Obama, Care}
ORDER BY Max(timestamp)
LIMIT 20 ON LAST ∞ DAYS
```

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

```
  SELECT CONTINUOUS keyword, COUNT(*)
FROM twitter_stream
WHERE location WITHIN (52,44.7,39.91,21.8)
GROUP BY keyword
LIMIT 10 ON ("18 Feb 2014",∞)
```

VIII. Conclusion

In this paper, we have presented our vision to build the first Microblogs Data Management System (MDMS). MDMS would serve as a backend for microblogs applications and hide all the complications of managing microblogs data from its users, either application developers or researchers who exploit microblogs. Throughout the paper, we have presented the challenges and requirements of MDMS. MDMS should provide the means to manage microblogs data for long time periods and support arbitrary SQL-like queries on them. To this end, MDMS should provide real-time ingestion, indexing, and querying for fast incoming microblogs in main-memory. Consequently, it needs to carefully manage memory resources for efficient utilization. Moreover, such heavy use of main-memory obligates to account for memory failures through recovery management techniques. In such rich querying and indexing environment, MDMS should employ new query optimization techniques that tackle the new challenges in microblogs indexes. We have presented our vision for a system proposal that satisfy these challenges and requirements. Our proposal consists of a scalable indexer, an efficient query engine, a light recovery manager, and a SQL-like query language. The proposed components, collectively, provide an end-to-end scalable and practical solution for managing microblogs and supporting queries on them.

REFERENCES

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

[2] D. J. Abadi and et. al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

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

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

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

[6] H. Andrade and et. al. Processing high data rate streams in System S. *Journal of Parallel and Distributed Computing*, 71(2), 2011.

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

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

[9] New Study Quantifies Use of Social Media in Arab Spring. www.washington.edu/news/2011/09/12/new-study-quantifies-use-of-social-media-in-arab-spring/, 2011.

[10] A. Arasu and et. al. STREAM: The Stanford Stream Data Manager. In *SIGMOD*, 2003.

[11] A. Archambault and J. Grudin. A Longitudinal Study of Facebook, LinkedIn, & Twitter Use. In *CHI*, 2012.

[12] 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.

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

[14] 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.

[15] C. Budak, T. Georgiou, D. Agrawal, and A. E. Abbadi. GeoScope: Online Detection of Geo-Correlated Information Trends in Social Networks. In *VLDB*, 2014.

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

[17] 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.

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

[19] S. Chandrasekaran and et. al. TelegraphCQ: Continuous Dataflow Processing. In *SIGMOD*, 2003.

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

[21] 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.

[22] K. C. Chretien, S. R. Greysen, J.-P. Chretien, and T. Kind. Online Posting of Unprofessional Content by Medical Students. *The Journal of the American Medical Association*, 2009.

[23] C. D. Cranor and et. al. The Gigascope Stream Database. *IEEE Data Engineering Bulletin*, 26(1), 2003.

[24] 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.

[25] Facebook Statistics. http://newsroom.fb.com/company-info/, 2015.

[26] W. Feng, J. Han, J. Wang, C. Aggarwal, and J. Huang. STREAMCUBE: Hierarchical Spatio-temporal Hashtag Clustering for Event Exploration Over the Twitter Stream. In *ICDE*, 2015.

[27] T. Ghanem, A. Magdy, M. Musleh, S. Ghani, and M. Mokbel. VisCAT: Spatio-Temporal Visualization and Aggregation of Categorical Attributes in Twitter Data. In *SIGSPATIAL*, 2014.

[28] R. Grover and M. Carey. Data Ingestion in AsterixDB. In *EDBT*, 2015.

[29] M. A. Hammad and et. al. Nile: A Query Processing Engine for Data Streams. In *ICDE*, 2004.

[30] J. Hannon, M. Bennett, and B. Smyth. Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches. In *RecSys*, 2010.

[31] Harvard Tweet Map. worldmap.harvard.edu/tweetmap/, 2013.

[32] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis. Discovering Geographical Topics In The Twitter Stream. In *WWW*, 2012.

[33] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACS*, 40(4), 2008.

[34] A Nobel Peace Prize for Twitter? www.csmonitor.com/Commentary/Opinion/2009/0706/p09s02-coop.html, 2009.

[35] J. Jiang, H. Lu, B. Yang, and B. Cui. Finding Top-k Local Users in Geo-Tagged Social Media Data. In *ICDE*, 2015.

[36] Y. Li, Z. Bao, G. Li, and K.-L. Tan. Real Time Personalized Search on Social Networks. In *ICDE*, 2015.

[37] J. Lin and A. Kolcz. Large-scale machine learning at twitter. In *SIGMOD*, 2012.

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

[39] A. Magdy, L. Alarabi, S. Al-Harthi, M. Musleh, T. Ghanem, S. Ghani, S. Basalamah, and M. Mokbel. Demonstration of Taghreed: A System for Querying, Analyzing, and Visualizing Geotagged Microblogs. In *ICDE*, 2015.

[40] A. Magdy, L. Alarabi, S. Al-Harthi, M. Musleh, T. Ghanem, S. Ghani, and M. Mokbel. Taghreed: A System for Querying, Analyzing, and Visualizing Geotagged Microblogs. In *SIGSPATIAL*, 2014.

[41] 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*, 2014.

[42] 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.

[43] 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.

[44] T. McCormick, H. Lee, N. Cesare, and A. Shojaie. Using Twitter for Demographic and Social Science Research: Tools for Data Collection. *Sociological Methods and Research*, 2013.

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

[46] E. Meij, W. Weerkamp, and M. de Rijke. Adding Semantics to Microblog Posts. In *WSDM*, 2012.

[47] G. Mishne, J. Dalton, Z. Li, A. Sharma, and J. Lin. Fast Data in the Era of Big Data: Twitter's Real-time Related Query Suggestion Architecture. In *SIGMOD*, 2013.

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

[49] New Enhanced Geo-targeting for Marketers. https://blog.twitter.com/2012/new-enhanced-geo-targeting-for-marketers.

[50] O. Phelan, K. McCarthy, and B. Smyth. Using Twitter to Recommend Real-Time Topical News. In *RecSys*, 2009.

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

[52] R. Ramakrishnan and J. Gehrke. *Database Management Systems (3rd ed.)*. McGraw-Hill, 2003.

[53] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake Shakes Twitter Users: Real-Time Event Detection by Social Sensors. In *WWW*, 2010.

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

[55] V. K. Singh, M. Gao, and R. Jain. Situation Detection and Control using Spatio-temporal Analysis of Microblogs. In *WWW*, 2010.

[56] A. Skovsgaard, D. Sidlauskas, and C. S. Jensen. Scalable Top-k Spatio-temporal Term Querying. In *ICDE*, 2014.

[57] Topsy Analytics: Find the insights that matter. www.topsy.com, 2014.

[58] TweetTracker: track, analyze, and understand activity on Twitter. tweettracker.fulton.asu.edu/, 2014.

[59] Twitter Historical Index. blog.twitter.com/2014/building-a-complete-tweet-index, 2014.

[60] Twitter Statistics. https://about.twitter.com/company, 2015.

[61] The Twitter War: Social Media's Role in Ukraine Unrest. news.nationalgeographic.com/news/2014/05/140510-ukraine-odessa-russia-kiev-twitter-world/, 2014.

[62] Twitter a Big Winner in 2012 Presidential Election. www.computerworld.com/article/2493332/social-media/twitter-a-big-winner-in-2012-presidential-election.html, 2012.

[63] Topsy Analytics for Twitter Political Index. topsylabs.com/election/.

[64] N. Vesdapunt and H. Garcia-Molina. Identifying Users in Social Networks with Limited Information. In *ICDE*, 2015.

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

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

[67] J. Zhao, J. C. Lui, D. Towsley, P. Wang, and X. Guan. Sampling Design on Hybrid Social-Affiliation Networks. In *ICDE*, 2015.