Mars: Real-time Spatio-temporal Queries on Microblogs

Amr Magdy[1§], Ahmed M. Aly[2], Mohamed F. Mokbel[3§], Sameh Elnikety[4], Yuxiong He[5], Suman Nath[6]
[1,3]*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455*
{[1]amr,[3]mokbel}@cs.umn.edu
[2]*Department of Computer Science, Purdue University, West Lafayette, IN 47907*
[2]aaly@cs.purdue.edu
[4,5,6]*Microsoft Research, Redmond, WA 98052-6399*
{[4]samehe,[5]yuxhe,[6]suman.nath}@microsoft.com

***Abstract*— *Mars* demonstration exploits the microblogs location information to support a wide variety of important spatio-temporal queries on microblogs. Supported queries include range, nearest-neighbor, and aggregate queries. *Mars* works under a challenging environment where streams of microblogs are arriving with high arrival rates. *Mars* distinguishes itself with three novel contributions: (1) Efficient in-memory digestion/expiration techniques that can handle microblogs of high arrival rates up to 64,000 microblog/sec. This also includes highly accurate and efficient hopping-window based aggregation for incoming microblogs keywords. (2) Smart memory optimization and load shedding techniques that adjust in-memory contents based on the expected query load to trade off a significant storage savings with a slight and bounded accuracy loss. (3) Scalable real-time query processing, exploiting Zipf distributed microblogs data for efficient top-k aggregate query processing. In addition, *Mars* employs a scalable real-time nearest neighbor and range query processing module that employs various pruning techniques so that it serves heavy query workloads in real time. *Mars* is demonstrated using a stream of real tweets obtained from Twitter firehose with a production query workload obtained from Bing web search. We show that *Mars* serves incoming queries with an average latency of less than 4 msec and with 99% answer accuracy while saving up to 70% of storage overhead for different query loads.**

I. Introduction

Microblogs, e.g., tweets, Facebook comments, and Foursquare check-in's, are among the most popular web services nowadays. For example, Twitter has 500 Million users who generate 400+ Million daily tweets [12], while Facebook has 1+ Billion users who post 3.2+ Billion daily comments [4]. The richness of such data has attracted a plethora of research efforts that address various aspects for microblog streams, e.g., see [2], [3], [5], [8], [10]. Combined with the advances in wireless communication and GPS-equipped handheld devices, microblogs locations can be attached to each microblog to indicate the whereabouts of the microblog issuer. However, such location information are not yet exploited beyond data visualization [9], [11] or discovering localized events [13].

Mars exploits the microblogs locations to support a variety of important spatio-temporal queries on microblogs. With the plethora of incoming microblogs in real time, *Mars* supports its queries over the time window of the last T time units. Typical values of T spans from several hours to several days based on the available memory resources and the application requirements. Supported queries are categorized into three main types: (a) *Range queries*, where users request a set of recent microblogs in a certain area, (b) *Nearest-neighbor queries*, where users request a set of recent microblogs close to a certain location, and (c) *Aggregate queries*, where users request the set of top-k frequent or trending keywords within a certain area.

Mars works under a challenging environment due to the high arrival rates of microblogs (e.g., Twitter arrival rate is 4,600 tweets/sec) and the requirements of fast query response. This calls for the following three main features that are all incorporated inside *Mars*:

(1) ***Efficient digestion/expiration techniques***, to support high arrival rates of incoming microblogs. For range and nearest-neighbor queries, *Mars* spatio-temporal index is equipped with various techniques for batch updates, lazy deletions, and efficient index structure maintenance in real time. For aggregate queries, *Mars* employs an efficient and highly accurate aggregation technique that efficiently updates a set of counters, for each incoming microblog keyword, over a hopping-window of the last T time units. *Mars* can support up to 64,000 microblog/sec.

(2) ***Efficient memory management***. For real-time processing and querying, *Mars* opts to operate only with in-memory contents. Thus, it is crucial to manage the scarce memory resource efficiently. We develop and integrate various techniques to smartly decide the in-memory indexed contents to balance between memory consumption, query response time, and query accuracy. With high query accuracy, *Mars* shows memory savings up to 70%.

(3) ***Scalable real-time query processing***, that is able to process heavy query workloads in real time. To this end, *Mars* bounds its search horizon to the last T time units. In addition, *Mars* employs a set of spatio-temporal pruning techniques and exploits Zip distribution properties to minimize the number of visited microblogs/keywords significantly and hence provide average query latency of 4 msec.

We demonstrate *Mars* with a system prototype using a stream of real tweets obtained from Twitter firehose, with an actual query workload obtained from Bing web search.

§The work of these authors is partially supported by the National Science Foundation, USA, under Grants IIS-0952977 and IIS-1218168.

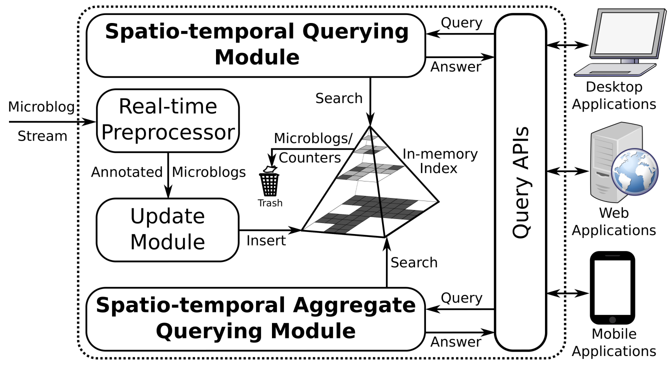


Fig. 1. System Architecture.

II. System Overview

Figure 1 presents *Mars* system architecture that mainly includes two querying modules, centered around a space-partitioning in-memory index. Incoming microblogs to *Mars* go through a preprocessor that attaches location and extracts keywords before forwarding microblogs to the update module. Microblog location could be associated from a GPS-enabled device or extracted from either the user profile or the microblog contents. Microblog locations can be either precise *latitude/longitude* coordinates or a city/county boundary. *Mars* users can issue their queries through either a web service, a desktop application, or a mobile app. *Mars* receives the submitted queries through its query APIs, and internally forwards them to the appropriate query module.

III. Spatio-temporal Pyramid Index

Index structure. *Mars* employs an in-memory partial pyramid index [1]. The pyramid index is spatio-temporal: the whole space is spatially partitioned into cells with different granularity (pyramid levels); within each cell, microblogs and aggregate information are indexed temporally.

Index update. Incoming microblogs, and their aggregate information, are inserted in *bulk* to the pyramid index. The bulk insertion is triggered every few seconds to flush thousands of incoming microblogs to the index, which reduces significant overhead over inserting microblogs one by one. Unlike insertion, deletions are done in a *lazy* way, mostly piggy backed on insertions to avoid any extra overhead.

Index maintenance. The index dynamically adapts its structure based on the incoming microblogs via split and merge operations. As splitting and merging pyramid cells are expensive operations, they are done only if necessary. Basically, four cells are merged only if three of them are empty. A cell is split only if it is over capacity and splitting its contents span at least two children cells. Details of *Mars* index are presented in [6].

IV. Spatio-temporal Queries

Supported queries. *Mars* supports spatio-temporal top-k queries based on the spatial proximity and temporal recency of each microblog. Due to the large numbers of microblogs, *Mars* limits its answer size to only k microblogs and bounds its search horizon to spatial and temporal boundaries, R and T, respectively. Users issue queries like "*Retrieve k microblogs near location P*". Then, *Mars* reports the top-k microblogs according to a spatio-temporal ranking function that combines the spatial proximity of each microblog M to P and the temporal recency of M with a parameter $0 \leq \alpha \leq 1$ that weights the importance of the spatial and temporal aspects. This allows *Mars* to support both range and k-nearest-neighbor queries. When $\alpha = 0$, *Mars* reports the k most recent microblogs in range R. When $\alpha = 1$, *Mars* reports the closest k microblogs to P that were posted in the last T time units.

Query processing. *Mars* query processor limits its search to microblogs in area R and posted in the last T time units. Thus, index cells within R are processed prioritized by their spatio-temporal scores. In each cell, microblogs are processed in a chronological order. As the answer size is only k, once *Mars* has an initial answer of k microblogs, it exploits the spatio-temporal ranking function to calculate new tightened search boundaries R' and T' that contain the least acceptable microblogs, based on their ranking scores, to be included in the answer. The goal is to minimize the number of visited microblogs while providing the answer as the highest-ranked top-k microblogs with respect to the querying user.

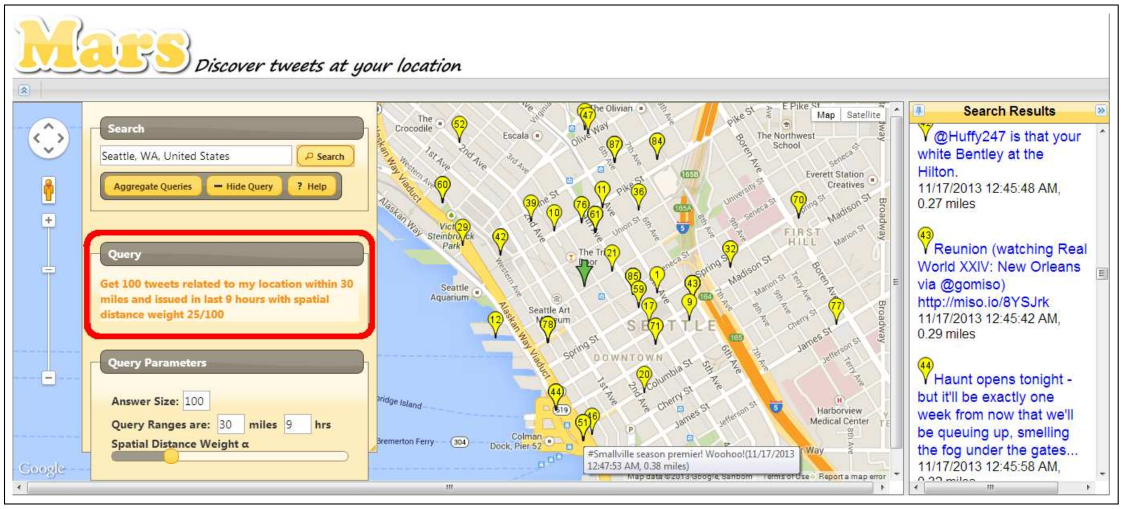
Memory management. Supported queries are answered from microblogs posted in the last T time units. However, storing all microblogs during T time units could be prohibitively expensive. Hence, *Mars* goes two major steps beyond such expensive storage. First, we exploit the fact that different parts of the space have different microblog arrival rates. Thus, in dense areas, e.g., city centers, the top-k microblogs would have arrived more recently than urban areas. Hence, in dense areas it is sufficient to cover smaller time horizons. This technique saves 50% of memory consumption while still offering 99% accuracy. Second, we employ a load shedding technique that trades significant reduction in memory footprint (up to 75% less storage) for a slight decrease in query accuracy (more than 95% accuracy). The idea is to expel a set of victim microblogs that are less likely to contribute to a query answer.

The details of *Mars* spatio-temporal queries processing and its memory management are presented in [6].

V. Aggregate Queries

Supported queries. *Mars* supports aggregate queries, e.g., "*Retrieve the top-k most frequent/trending keywords within a certain area R*". The aggregate function is computed based on either keyword frequency (top-k frequent) or a trending measure (top-k trending) within the last T time units. In particular, top-k trending keywords are ranked based on a linear regression slope of the keyword frequency across time sub-intervals in the last T time units.

Query processing. *Mars* supports aggregate queries in real time by maintaining a set of aggregate information on each microblog keyword in corresponding index cells. For each keyword, the aggregate information is maintained in a set of N counters, that split the time window T into equal sub-intervals, where each counter maintains information about its corresponding sub-interval. As time advances, new counters are included and older counters are expired using a low-overhead expiration technique. As real microblogs data follows Zipf distribution, where most of keywords are infrequent, our theoretical analysis and experimental results show that we can



Seattle, WA, United States

Get 100 tweets related to my location within 30 miles and issued in last 9 hours with spatial distance weight 25/100

Answer Size: 100
Query Ranges are: 30 miles 9 hrs
Spatial Distance Weight α

@Huffy247 is that your white Bentley at the Hilton.
11/17/2013 12:45:48 AM, 0.27 miles

43 Reunion (watching Real World XXIV: New Orleans via @gomiso) http://miso.io/8YSJrk
11/17/2013 12:45:42 AM, 0.29 miles

44 Haunt opens tonight - but it'll be exactly one week from now that we'll be queuing up, smelling the fog under the gates...
11/17/2013 12:45:58 AM,

#Smallville season premier! Woohoo!(11/17/2013 12:47:53 AM, 0.38 miles)

Fig. 2. Spatio-temporal Nearest-neighbor Queries.

therefore compute the final top-k answer list using only top-k keywords in each index cell which significantly reduces query processing time.

Memory management. *Mars* can satisfy all its aggregate queries form in-memory contents if it sores the aggregate information of all keywords that have arrived in the last T time units. However, that may be prohibitively expensive due to the large number of keywords. To minimize memory consumption, *Mars* exploits the fact that microblog keywords follow a Zipf distribution (which have just used for efficient query processing) to maintain only those keywords with a frequency greater than a certain threshold, as other keywords are unlikely to appear in any top-k result. *Mars* extends the Lossy Counting technique [7], that is designed mainly for counting frequent items in a data stream, to accommodate the time hopping-window behavior. Based on our experimental data, we save 70% of *Mars* storage, while still answer all queries with 100% accuracy.

VI. Demonstration Scenario

Mars functionality and system internals are demonstrated using a real data set obtained from Twitter as a prime example of a microblog stream. In particular, we use a rich set of 340 Million real tweets obtained from Twitter firehose. The real tweets are fed to *Mars* with the current Twitter rate of 4,600 tweets/second. Our demo attendees would be able to interact with *Mars* through one or more of the following five scenarios.

A. Scenario 1: Spatio-temporal Queries

Mars demo attendees would be able to issue spatio-temporal range and nearest neighbor queries on Twitter data. Figure 2 depicts the user query screen of *Mars* demonstration. To issue a spatio-temporal range or nearest neighbor query, a demo attendee should: (a) Navigate to a location of interest through the map or via the location text box, (b) *Optionally* change the default parameters values of k, R, T, and α through the query parameters box, and (c) Finally, click the search button to obtain the results. Our front-end interface submits the search query request to the back-end of *Mars*. Once the answer tweets are returned back, the front-end interface shows them to the user both on the map and on a right-side panel. Figure 2 gives the output of a spatio-temporal 100-nearest-neighbor query near Seattle, WA. The output shows each tweet text, location, timestamp, and distance to user location.

B. Scenario 2: Aggregate Queries

The demo attendees can also issue spatio-temporal aggregate queries through the main user query screen. To issue a spatio-temporal top-k frequent/trending keyword query, a demo attendee should: (a) click the "*Aggregate Queries*" radio button to switch the querying mode to either frequent or trneding queries, (b) navigate to the spatial range of interest through the map, (c) *optionally* change the default query parameters values through the query parameters box, and (d) click the search button to obtain the results. Figure 3 gives the output of a spatio-temporal top-100 frequent keywords query near Minneapolis, MN. The frequent keywords are displayed only in the side panel and do not show up on the map; as they have no exact locations. The output shows the keyword as well as the number of times it has been tweeted in the query area.

C. Scenario 3: Tweets Heatmap

Mars has an option to show a tweet heatmap where users can navigate and zoom the map to find out areas with dense tweets (marked in red), while areas with much less tweets (marked as green), and so on. The heat map can be either based on all US tweets or based on a certain keyword (e.g., the heat map for the hash tag #Obama). The heat map is updated online based on the incoming microblog stream. A fast forward run of a heat map over our 340 Million tweets gives an impressive visualization of the development of posting tweets in US.

Internally, *Mars* presents its heatmap based on the information of its spatio-temporal pyramid index structure. The heat map basically reflects the cells of the index. As each pyramid cell covers a geographical map area, we draw such an area with a certain color that reflects its aggregate information for all tweets, or counters for each keyword. Zooming in the map corresponds to going to deeper levels in the pyramid index.

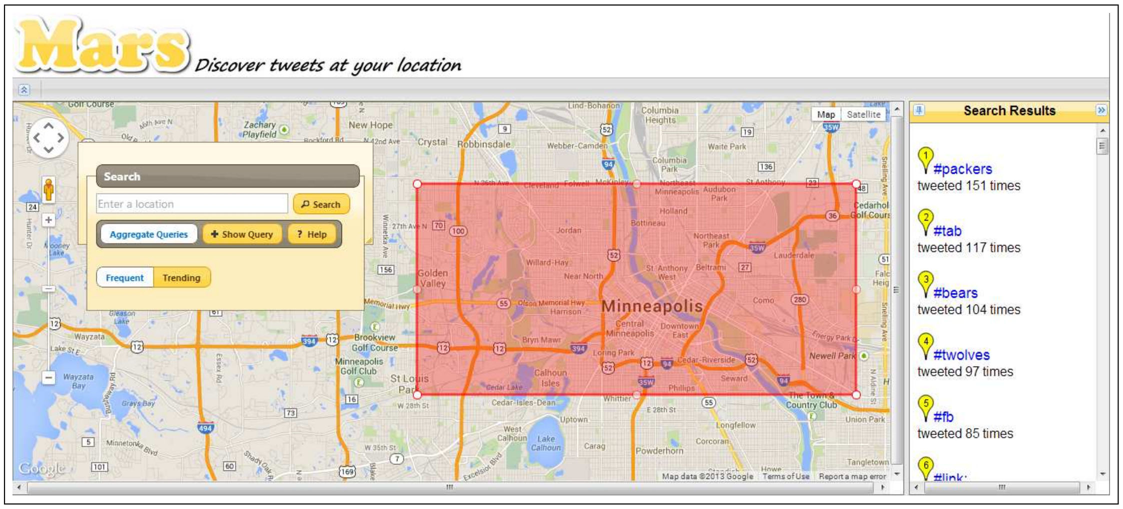


Fig. 3. Spatio-temporal Aggregate Queries.

D. Scenario 4: Admin View

In addition to issuing user queries as described in the previous scenarios, the demo attendees would be able to monitor the stream of queries that are received and handled by *Mars* through an administrative screen. The admin screen continuously shows the locations of the posted queries on the map interface as well as the log entries of receiving and finishing these queries. In addition to monitoring the queries that are posted by demo attendees, *Mars* automatically plays a stream of queries based on the locations of actual query workload of 1 Million Bing web search queries.

The admin view of *Mars* serves three main goals: (1) Monitoring the user activity in real time, (2) Continuously monitoring the system response times to users to quickly identify unexpected slowdowns, and (3) Analyzing system bottlenecks when certain regions may receive a large percentage of the queries and hence need a customized treatment, e.g., allocate more servers.

E. Scenario 5: Index Internals

Our demo shows the internals of *Mars* index by showing animated index update operations (i.e., bulk insertions and lazy deletions) on a geographical map. To show live index update operations, we visualize the steps to insert a batch of new tweets. This includes the arrival of the new tweets, enclosing them in a minimum bounding rectangle, and navigating through the pyramid levels to the corresponding spatial cells. When we reach a cell where we would insert some (or all) tweets, the lazy deletion checks for expired tweets to be deleted and displays the number of deleted tweets on the screen. The screen can also show a simple technique which updates *Mars* index with individual tweets, instead of bulk, and hence contrast the performance with our techniques. Figure 4 shows a step of such animated visualization when the bulk insertion is visiting the second level of the pyramid index to insert a set of new tweets in USA Midwest states. The red cells on the map represents *Mars* index cells at the second level while the markers indicate the tweets' locations that are enclosed within their minimum bounding rectangle.

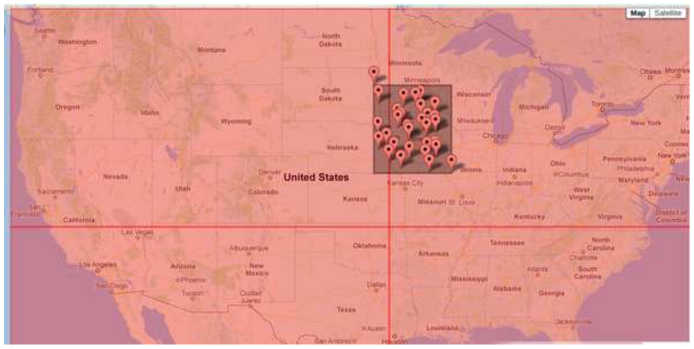


Fig. 4. Mars bulk insertion.

References

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

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

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

[4] Facebook Statistics. http://newsroom.fb.com/Key-Facts, 2012.

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

[6] 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.

[7] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *VLDB*, 2002.

[8] 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.

[9] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. Processing and Visualizing the Data in Tweets. *SIGMOD Record*, 40(4), 2012.

[10] M. Mathioudakis and N. Koudas. TwitterMonitor: Trend Detection over the Twitter Stream. In *SIGMOD*, 2010.

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

[12] Twitter Statistics. http://expandedramblings.com/index.php/march-2013-by-the-numbers-a-few-amazing-twitter-stats/, 2013.

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