Performance of System for Analyzing Diffusion of Social Media Messages in Real Time

Miki Enoki
Ochanomizu University
IBM Research – Tokyo
Tokyo, Japan
enomiki@jp.ibm.com

Issei Yoshida
IBM Research – Tokyo
Tokyo, Japan
issei@jp.ibm.com

Masato Oguchi
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku
Tokyo, Japan
oguchi@is.ocha.ac.jp

***Abstract*—A social media message that is forwarded (e.g., retweeted in Twitter) by many users may diffuse widely. Such a 'viral' message may have a major impact in the real world. If the content maligns a company, the company must respond as quickly as possible to defend its brand image. Timely knowledge about what kinds of information are diffusing in social media is thus quite important. We have developed a system that enables real-time analysis of information diffusion in social media. We defined typical data access patterns and created a special view for complicated queries such as top-k ranking queries to improve query performance. The time window is customized by estimating diffusion extinction, which enables retention of diffusion data in the system as long as it is being retweeted frequently. Evaluation showed that the processing of simple access patterns was extremely fast, that the special view effectively improved performance, and that the time-window customization improved accuracy.**

Keywords- social media; information diffusion; stream processing; in-memory database; query processing

I. INTRODUCTION

A. Information Diffusion in Social Media

A microblogging service such as Twitter characterized by frequent message posting and transient topics. Many tweets are replies or messages shared among users ("retweets"). If a retweeted message (RT) is shared widely so that it is read by many users, it may have a large impact in the real world. If the content maligns or criticizes a company, the company will normally want to respond as quickly as possible to protect its brand image and reputation. In contrast, if positive information is often retweeted, there is a positive impact on the marketing efforts of the company. Knowing what kinds of information are being widely disseminated through social media is thus essential in a company's efforts to protect its corporate brand value.

In Twitter-like services, many messages are posted in real time around the world. This means retweets can diffuse rapidly within hours or even minutes. There is a need to respond quickly on the basis of an analysis of user behaviors and to quickly identify trending messages because much of the information shared through social media quickly loses its impact. In the current Twitter system, tweets can be searched for by using keywords related to a company and monitored manually. Alternatively, there are commercial services that can check if certain topics are going viral by tracking the numbers of tweets containing target keywords [1].

Several research groups are studying trend and/or event detection through real-time monitoring of the entire Twitter stream. One group, for example, developed TwitterMonitor [2], a tool that identifies hot topics on Twitter by detecting bursts of keywords that arrive at unusually high rates.

We have developed a system for analyzing the diffusion of information through retweeted tweets. The retweeting of a tweet message by many users indicates that they find the content interesting and/or entertaining. By identifying the key people whose messages are frequently retweeted or who retweet messages about particular topics, we can answer such questions as "Who is interested in this topic?" and "Who are the influencers for this topic?"

Visualizing the networks through which retweeted tweets are diffused is helpful for finding the diffusion routes among users and the influencers whose messages were retweeted by many users. The profiles of the key people can be analyzed [3] to estimate their characteristic features, such as their interests, age, gender, and location.

B. Stream Processing of Information Diffusion Data

Twitter messages can be treated as streaming data because large volumes of message data are posted continuously. In stream processing, the streaming data are usually divided into segments called windows. Window size is usually measured in either the amount of data or the length of time. Various operations are applied to the data within a window. When information diffusion data are being handled in a stream processing system with time windows, the incoming tweets are collected and processed in batches.

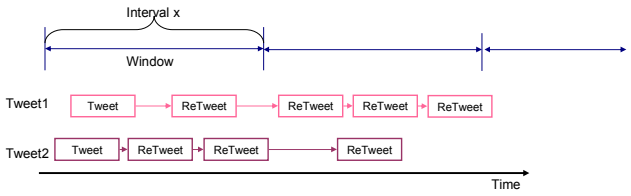

Figure 1. Stream processing of tweets.

An example of this processing for retweets is shown in Figure 1. Each box represents a tweet message, either a tweet or a retweet. The sequentially arriving tweets are divided into the defined time windows and partitioned by tweet IDs. If a tweet is a retweet, it is partitioned by the tweet ID of the original tweet.

If a tweet with an ID of 1 (Tweet1) is retweeted, a thread for Tweet1 is created, and its retweets are collected.

If the window size is set to one hour, and the number of retweets in each tweet thread is counted, the retweet count for each tweet that was retweeted in the past hour is obtained. We are interested in the tweets currently being retweeted by many users, but the total retweet count for each tweet cannot be directly obtained. This means that we have to use fragmented diffusion data for our diffusion analysis. We may also fail to create a complete information diffusion network due to missing retweets.

To handle all of the diffusion data for each tweet retweeted by users, we use an in-memory data store to collect the diffusion data in the stream system. In stream processing, a query is usually defined in advance and used for a period of time with the incoming streaming data. In contrast, for an in-memory data store, various queries can be issued interactively against the stored data. The queries are usually processed quickly since the data is stored in memory.

Continuously storing all of the diffusion data for many days is impractical since memory resources are limited. In addition, there is no reason to store stale diffusion data (i.e., information in tweets that are no longer being retweeted). Therefore, we retain diffusion data in the data store for only as long as it is being retweeted frequently. When the data becomes stale, they are removed from the data store.

To facilitate the handling of complicated queries, such as a user ranking query of the top-k influencers by topic, we created a special view.

Here are the main contributions of our work.

- Our diffusion analysis system enables real-time analysis of streaming social data. A special view for ranking-related queries enables rapid data access.
- Customization of the time window enables the freshness of the stored data to be assessed by estimating retweet diffusion extinction.
- Query performance evaluation using typical analysis scenarios with real diffusion data from Twitter showed that the query processing is extremely fast. For a complicated query such as ranking-related query, the special view effectively improves query performance. The accuracy of diffusion extinction was much better than with a fixed time-window size.

The outline of the rest of this paper is as follows. Section 2 describes the diffusion analysis system. Section 3 provides data access patterns and optimizations. We introduce the maintenance method for the data store in Section 4. Section 5 presents our experimental environment and describes the performance evaluation. We review related work in Section 6 and conclude the paper in Section 7.

II. REAL-TIME DIFFUSION ANALYSIS SYSTEM

An overview of the real-time diffusion analysis system is shown in Figure 2. The framework consists of a stream server and an application server. Incoming retweets are sequentially inserted into an in-memory data store running on the stream server. The application server provides various modules for analyzing the diffusion data. For example, the "Ranking" module creates a ranking of retweet counts to enable detection of the current hot tweets or the ranking of influential users for a specified topic. The "Visualization" module displays the diffusion network for a specified tweet. Other modules for diffusion analysis can be added, such as a profile analysis module for estimating users' interests or locations. Each module retrieves diffusion data from the stream server.

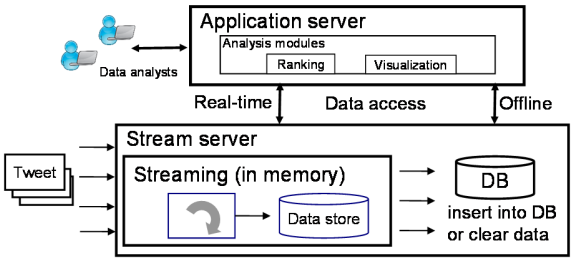

Figure 2. Real-time diffusion analysis system.

The stream processing runs in memory on the stream server. Since the size of the memory is limited, stale data must be deleted or moved to a disk-resident database. The database stores historical diffusion data for use in offline data analysis. We focus on real-time processing in this paper.

There are various candidates for the in-memory data store such as an in-memory database, a key/value store, and a graph database [4,5]. In our application, the primary scenarios involve finding trending tweets or popular users by aggregating or sorting data. The in-memory database is best for this since it can handle complicated queries using SQL.

III. DATA ACCESS PATTERNS AND OPTIMIZATIONS

In this section, we describe typical data access patterns for diffusion analysis. We also describe our special view for ranking-related query processing.

A. Data Access Patterns

We use two tables, a RETWEET table and an ORIGIN_TWEET table, as shown in Figure 3. The RETWEET table stores retweeted messages. It has fields for the ID of the retweet (TweetID), the ID of the original tweet (RTID), the retweet time (Time), the source user name (Src) and destination (Dst), the retweet user's language (Lang), and the location information (Location). The value of Dst is the name of the user who retweeted the message. The value of Src is the name of the user whose message was retweeted. The ORIGIN_TWEET table has a tweet ID (TweetID), the tweet time (Time), the name of the user who posted the tweet (User), the tweet message (Msg), and the retweet count (RTcount). The RTcount is incremented when a corresponding retweet message is received. RTID in the RETWEET table can be joined with TweetID in the ORIGIN_TWEET table to get information about the original tweet.

RETWEET table

TweetID	RTID	Time	Src	Dst	Lang	Location	‥
100	1	Time data	u1	u2	Ja	GPS	‥
101	1	Time data	u2	u4	Ja	GPS	‥
‥	‥	‥	‥	‥	‥	‥	‥

ORIGIN_TWEET table

TweetID	Time	User	Msg	RTcount	..
1	Time data	u1	message	29	..
2	Time data	u5	message	14	..
..	..	..	..	..	..

Figure 3. Tables in in-memory database.

Here are typical data access patterns.

(1) Obtain retweet diffusion network data for specified tweets

With this pattern, diffusion data are obtained to create the diffusion network. We can then answer "How did the information flow among users?" A network can be used for network analytics such as clustering and frequent path detection. This query pattern can be interpreted as the following SQL example:

```
[Query 1]
SELECT    Src, Dst
FROM      RETWEET
WHERE     RTID in (tweet ids)
```

(2) Find trending tweets

This query pattern is used to obtain the current rankings of retweet counts. This SQL query returns the top ten most retweeted tweets:

```
[Query 2]
SELECT    *
FROM      ORIGIN_TWEET
WHERE     TweetID in (tweet ids)
ORDER BY  RTcount  DESC
FETCH FIRST 10 ROWS ONLY
```

(3) Find influential users

This query pattern is used to obtain user rankings. We can answer questions such as "Who is interested in this topic?" and "Who is the influencer for this topic?" by using this pattern. The SQL query returns the top ten users who retweeted most often from among the specified tweets:

```
[Query 3]
SELECT    Src, count(Src)
FROM      RETWEET
WHERE     TweetID in (tweet ids)
GROUP BY  Src
ORDER BY  count(Src)  DESC
FETCH FIRST 10 ROWS ONLY
```

When Src is specified in the GROUP-BY clause, Query 3 returns the most influential users.

In these ways, we can obtain various types of information from the in-memory database. The data access speed is higher than that of a disk-resident database because the results are returned directly from memory. In particular, simple queries are extremely fast, such as fetching data using a primary key and its associated values.

However, for some complicated queries using sort, count, join, and subqueries, the overall query performance can be worse than that of a disk-resident database [6]. For example, Query 3 may be slow because it uses aggregation operations. Therefore, we need to use data access optimization.

B. Data Access Optimization for Top-k Ranking Queries

We created a special view for rapid processing of ranking-related queries. For example, a query to find influencers calls for obtaining in rank order the names of users who are frequently retweeted. This is a top-k retrieval query, so we can use a rapid top-k retrieval algorithm with a special view that has keys and values.

An example view for calculating user rankings is shown in Figure 4. The key is the tweet ID, and the value is the list of (u, w). The u is the name of the user whose tweet was retweeted, and the w is the retweet count for u. The u represents the Src user in the RETWEET table. The value of w is the result of "SELECT Src, count(Src) FROM RETWEET GROUP BY Src." For example, for the tweet with an ID of 1, *u3* was retweeted by 100 users, and u1 was retweeted by 40 users.

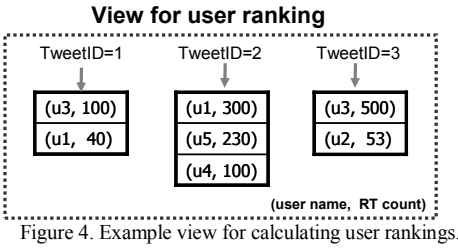

Figure 4. Example view for calculating user rankings.

Top-k retrieval is done using an optimized method called the threshold algorithm [7], which calculates the top-k rankings more rapidly than a non-optimized method

[create view for top-k query]

For each original tweet ID (= t), create a view S(t) and an index R(t).

S(t): List of (u, w) pairs for tweet t, sorted by retweet count in descending order. S(t) can be implemented using a standard list structure.

R(t): Index for random access to a (u, w) pair for an arbitrary user in S(t). R(t) can be implemented using a standard map structure.

In Figure 4, S(*1*) = [(*u3, 100*), (*u1, 40*)], S(*2*) = [(*u1, 300*), (*u5, 230*), (*u4,100*)], and S(*3*) = [(*u3, 500*), (*u2*, *53*)].

[top-k algorithm]

The algorithm for calculating the top-k users whose messages were retweeted the most for a specified set of tweets $t_1, t_2, \ldots t_m$ is as follows.

```
Input   S(t), R(t), Tweetid_list = [t1, t2, …, tm]
Output  result_candidate  //top k user list
1.   result_candidate = [];
2.   For each i=1,...,m, Do
   1.   If !S(ti).hasNext() Then continue;
   2.   Retrieve (ui, wi) from S(ti);
   3.   If result_candidate.contains(ui) Then continue;
   4.   V := sum of w in each R(tj) for ui // j=1,...,m
   5.   Vk := sum of w in the k-th candidate in
        result_candidate;
   6.   If (V > Vk) Then update result_candidate
        by (ui, V).
   7.   V' := sum of the minimal value of w in S(ti)
        that have been retrieved so far // i=1,...,m
   8.   If (V >= Vk) Then break;
3.   End For
```

Use of the special view consumes additional memory space for S(t) and R(t). There is thus a trade-off between memory usage and query performance.

IV. METHOD FOR MAINTAINING DIFFUSION DATA IN IN-MEMORY STORE

As described in Section 2, stale data must be deleted from the in-memory data store since the amount of memory is limited, but the diffusion data should be retained as long as it is being retweeted frequently. In this section, we introduce a customized timewindow for each tweet to determine when it is to be removed.

A. Time-window Customization

The time window is customized as shown in Figure 5. The start of the window is when the original tweet was posted. The end of the window should be when the last retweet was posted, but it is difficult to find that endpoint because even an old tweet could suddenly be retweeted.

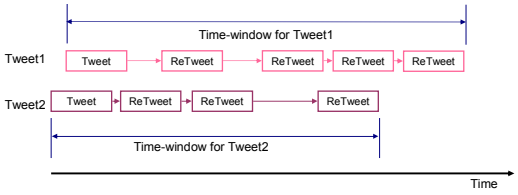

Figure 5. Customized time window.

Retweeted messages diffuse rapidly within hours or even minutes but usually not over many days. Therefore, we assume that, after a certain period from the original tweet posting, the occurrence probability of a retweet is low enough for the diffusion data to be safely removed from memory. As a simple technique, we use a fixed period of time to determine diffusion extinction for all of the diffusion data. However, the time lag between diffusion extinction and the original tweet varies [8].

A retweet propagation model [9] shows that retweets follow a log-normal distribution. Therefore, we regard the distribution of retweets over time in each time slot as a probability distribution. Figure 6 shows an example distribution of the retweet delay times.

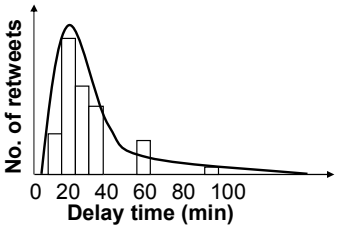

Figure 6. Example distribution of retweet delay times.

We fit the distribution of the probability density function to a log-normal distribution. The probability density function of a log-normal distribution with parameters μ and σ is given by

$$f(x)=\frac{1}{x\sigma\sqrt{2\pi}}e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}}.$$

The diffusion data for the first hour from when the original tweet was posted are used to estimate μ and σ. The 90 percentile point in the estimated distribution is then calculated, and that point is defined as the time-window size for the tweet. This is regarded as the time when the diffusion will be almost extinct. As a result, about 90% of the diffusion data is retained in the data store.

The system calculates diffusion extinction to determine the time-window size when it finds a tweet one hour after it was posted. If only a few retweets for a tweet are found, the tweet is unlikely to start being retweeted in the future. Hence, if the retweet count is less than a threshold, the corresponding diffusion data is considered to be extinct.

B. Workload Control to Adjust for Bursts of Streaming Data

As shown in Figure 7, the volume of retweets tends to peak at about 3 minutes following the posting of the original tweet. The peak is particularly high for certain topics such as an election, a World Cup match, an Olympics event, and a disaster. This burst of data could result in the system trying to process thousands or tens of thousands of tweets simultaneously, which would cause a lack of machine resources in the stream server. Workload control is used to adjust for such bursts of streaming data.

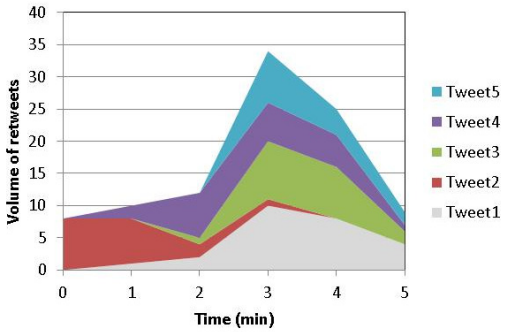

Figure 7. Volume of retweets within first five minutes of tweet posting.

This control includes removing less important diffusion data from the in-memory data store even if the corresponding tweet has not yet reached the estimated point of diffusion extinction. This is currently done by using the LRU (least recently used) method, a standard method of data eviction, to remove the

diffusion data that was least recently retweeted. This is not the optimal method, however, since other relevant information such as the total number of retweets and the user who posted the tweet are not considered. Future work includes defining the importance of each tweet and using that information instead of LRU information for data eviction.

V. PERFORMANCE EVALUATION

A. Methodology

(1) Query processing

We measured the response times for the three queries described in Section 3. For Query 3, we compared performance with and without using the special view. We used the same SQL statements and the database tables shown in Section 3. The response time was taken as the average response time for the same query repeated 10,000 times. (The first 1,000 times were ignored.)

(2) Customized time window

We determined the time of diffusion extinction by fitting a log-normal distribution, as described in Section 4.1. We estimated μ and σ using the diffusion data for the first hour from when the original tweet was posted.

Our test server had two Xeon X5670 CPUs (2.93 GHz, 6 cores) with 32 GB of RAM, running Red Hat Linux 5.5. The measurement client ran on the same server. The H2 database [10] was used as the in-memory mode. Indexes were created for the TweetID field in both the RETWEET and ORIGIN_TWEET tables and for the SRC field in the RETWEET table. The view for the top-k ranking in Query 3 was implemented in Java (IBM J9 VM JRE 1.7.0). The fit for the log-normal distribution was computed using R [11].

B. Experimental Data

We used retweets of Twitter messages in Japanese for the period from 1/19/2014 to 2/11/2014. This period included the Tokyo governor's election, which was one of the first elections in Japan for which "Internet campaign activities" were permitted. We collected the tweets and retweets that were posted from the five official accounts representing the candidates in the election as well as those of heavy social media users. A total of 1611 tweets were retweeted, and the total retweet count was 76,593.

C. Experimental Results

1) Query processing results

Figure 8 shows the average response times for Queries 1, 2, and 3. For Query 1, we specified one tweet ID in the WHERE clause. For Queries 2 and 3, we specified 100 different tweet IDs in each query. The tweet IDs were chosen at random. The response times for Queries 1 and 2 were 0.2 ms and 0.4 ms, respectively. This indicates that query processing was extremely fast for these queries. The response time for Query 3 was much longer, probably because the computational cost was higher for the aggregation operations.

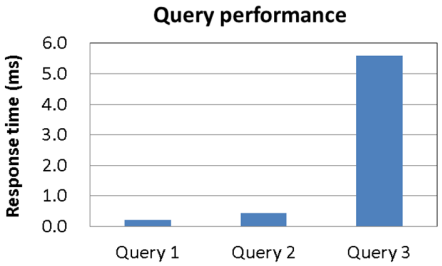

Figure 8. Query response times.

The average response time for Query 3 when the special view for user rankings was used is plotted in Figure 9. The number of specified tweet IDs was increased from 100 to 500. The blue dashed line shows the results without the view, and the red solid line shows the results with the view. The query performance was greatly improved by using the view. The maximum performance difference was 64 times faster when the number of tweet IDs was 100. The response time without the view increased in proportion to the number of specified tweet IDs. The response time with the view also increased, but it was still much shorter than without the view.

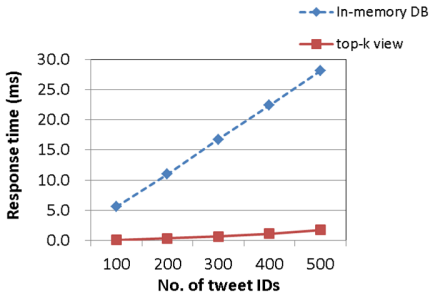

Figure 9. Query 3 response time using view for user rankings.

2) Diffusion extinction estimation results

To evaluate the effectiveness of diffusion extinction, we used the top 50 most retweeted tweets in our data set. We fitted the distribution of each tweet to the probability density function of the log-normal distribution, estimated the retweet curve, and calculated the 90 percentile point for that curve to determine the time-window size for the tweet. We counted the total number of retweets within the time window and calculated the percentage of those retweets to the total number of retweets. The total count was for one week's worth of data.

We calculated the average percentages for the 50 tweets. Since our method covers about 90% of the retweets, we retained about 90% of the diffusion data in the in-memory data store as expected. We also calculated the average percentages using a fixed period of time to determine the diffusion extinction for all 50 tweets. We set the fixed time (= t) to 1, 10, 11, and 12 hours.

Table 1. Average coverage of retweet diffusion

t=1	t=10	t=11	t=12	**our method**
55.0%	89.1%	89.9%	90.6%	**89.9%**

As shown in Table 1, when t was 1 hour, the average coverage was only 55%. This indicates that about a half of the retweets occurred after more than one hour from the original

tweet posting. When *t* was 11 hours, the coverage was the same as with our method. In this case, the diffusion data for each tweet was stored in the data store for 11 hours because the time-window size was set to 11 hours.

Figure 10 shows the time interval needed to cover 90% of all retweets for each tweet. The green and yellow bars show the actual 90% point and the point estimated with our method, respectively. The results with our method were reasonably close to the actual results for each tweet. For tweet 18, the actual point was about seven days later. The tweet was retweeted at least a few times every day, making it difficult to estimate the extinction because our method works with the retweet data posted within one hour from the original posting time. This weakness can be overcome by integrating another propagation estimation model. The average difference between the actual and estimated results was 368 minutes, while the difference between the actual results and 660 minutes (= time-window size of 11 hours) was 535 minutes. This indicates that the accuracy of our method is significantly better than that with a fixed time-window size.

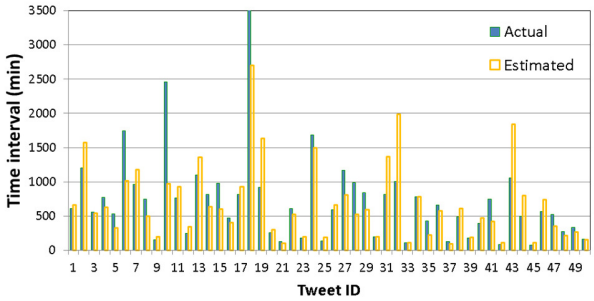

Figure 10. Time interval needed to cover 90% of all retweets.

VI. RELATED WORK

There have been many studies of information diffusion analysis in social networks.

Truthy [12] is a Web service that tracks political information on Twitter. It provides real-time analysis of information diffusion in social media by mining, visualizing, mapping, classifying, and modeling massive streams of public microblogging events. TweeQL [13] is a SQL-like query interface for streaming Twitter data. The streaming API enables users to issue long-running HTTP requests with keyword, location, or user ID filters and collect the matching tweets that appear in the stream. It uses windowed select-project-join aggregate queries over the input stream and supports user-defined functions for deeper processing of tweets.

There are also information diffusion analysis services, but their descriptions do not reveal their performance for processing social media data and instead focus on the analysis algorithm used. Our focus here is on the development of a real-time analysis system and its query processing performance.

Gupta et al. [14] used materialized views to quickly access a copy of the target data. They are created by incrementally updating the differential data. Charles et al. [15] proposed storing database queries in cache and using offline analysis of the queries and updates. The structure of our special view is different, and an efficient maintenance method for updating the view would improve our system.

VII. CONCLUSION AND FUTURE WORK

Our proposed diffusion analysis system enables real-time analysis of streaming social data. It helps companies to respond quickly on the basis of an analysis of user behaviors and to quickly detect trending tweets, which is important because most of the information shared through social media quickly loses its impact. An in-memory database is used to collect the diffusion data, and the time window is customized by estimating diffusion extinction so that the best time to remove stale data can be effectively determined.

Measurement of the query performance of our system using typical analysis scenarios showed that the processing of simple access patterns was extremely fast. The special view we created for complicated queries such as ranking-related queries effectively improved performance. Evaluation of the time-window customization used to determine diffusion extinction showed that the accuracy was much better than with a fixed time-window size.

Future work includes evaluating the workload control method used to handle bursts of retweet data.

[1] Top 10 monitoring tools for Twitter & other social media platforms - http://socialmedia.biz/2014/02/12/top-10-monitoring-tools-for-twitter-other-social-media-platforms/

[2] M. Mathioudakis and N. Koudas, "TwitterMonitor: trend detection over the twitter stream." SIGMOD, pp. 1155–1158, 2010.

[3] John D. Burger, John Henderson, George Kim, and Guido Zarrella, "Discriminating gender on Twitter." EMNLP, pp. 1301–1309, 2011.

[4] HBase http://hbase.apache.org/

[5] Neo4j http://www.neo4j.org/

[6] Evaluation of in-memory database TimesTen http://zenodo.org/record/7566/files/CERN_openlab_report_Endre_Andras_Simon.pdf

[7] Ronald, F., Amnon, L., and Moni, N., "Optimal aggregation algorithms for middleware." PODS, pp. 102–113, 2001.

[8] Haewoon, K., Changhyun, L., Hosung, P., and Sue B., M, "What is Twitter, a social network or a news media?" WWW, pp. 591–600, 2010.

[9] Asur, S., Huberman, B. A., Szabo, G., and Wang, C., "Trends in social media: persistence and decay." ICWSM, 2011

[10] H2 Database Engine http://www.h2database.com/html/main.html

[11] R http://www.r-project.org/

[12] Ratkiewicz, J., Conover, M. Meiss, M. Goncalves, B. Patil, S., Flammini, A., and Menczer, F., "Truthy: Mapping the spread of astroturf in microblog streams." WWW, pp. 249–252, 2011.

[13] Adam, M., Michael, S. B, Osama, B., David R. K., Samuel, M., and Robert C. M., "Processing and Visualizing the Data in Tweets." SIGMOD, pp. 21–27, 2012

[14] Gupta, A., Mumick, IS., and Subrahmanian, VS.: "Maintaining views incrementally." SIGMOD, pp. 157–166 1993.

[15] Charles, G., Amit, M., Anastasia, A., Bruce, M., Todd, M., Christopher, O., and Anthony, T., "Scalable Query Result Caching for Web Applications." VLDB, pp. 550–561, 2008.