IEICE TRANSACTIONS

on Information and Systems

VOL. E100-D NO. 4
APRIL 2017

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. INF. & SYST., VOL.E100–D, NO.4 APRIL 2017

PAPER Special Section on Data Engineering and Information Management

Capacity Control of Social Media Diffusion for Real-Time Analysis System

Miki ENOKI[†a)], **Issei YOSHIDA**[††], ***Nonmembers***, ***and*** **Masato OGUCHI**[†††], ***Member***

SUMMARY In Twitter-like services, countless messages are being posted in real-time every second all around the world. Timely knowledge about what kinds of information are diffusing in social media is quite important. For example, in emergency situations such as earthquakes, users provide instant information on their situation through social media. The collective intelligence of social media is useful as a means of information detection complementary to conventional observation. We have developed a system for monitoring and analyzing information diffusion data in real-time by tracking retweeted tweets. A tweet retweeted by many users indicates that they find the content interesting and impactful. Analysts who use this system can find tweets retweeted by many users and identify the key people who are retweeted frequently by many users or who have retweeted tweets about particular topics. However, bursting situations occur when thousands of social media messages are suddenly posted simultaneously, and the lack of machine resources to handle such situations lowers the system's query performance. Since our system is designed to be used interactively in real-time by many analysts, waiting more than one second for a query results is simply not acceptable. To maintain an acceptable query performance, we propose a capacity control method for filtering incoming tweets using extra attribute information from tweets themselves. Conventionally, there is a trade-off between the query performance and the accuracy of the analysis results. We show that the query performance is improved by our proposed method and that our method is better than the existing methods in terms of maintaining query accuracy.

key words: *information diffusion, social media, in-memory database, microblogging, stream processing*

1. Introduction

In Twitter-like services, countless messages are being posted in real-time every second all around the world. In emergency situations such as earthquakes, users provide instant information on their situation through social media. Speed is of the essence here. As another example, if a user posts a tweet with content that maligns or criticizes a company, that company will normally want to respond as quickly as possible to protect its brand image and reputation. In contrast, if positive information is retweeted, the marketing efforts of the company in question are positively affected. Knowing what kinds of information are being widely disseminated through social media is thus essential for a company to protect its corporate brand value. Timely knowledge about what kinds of information are diffusing in social media is therefore quite important.

Several research groups are studying trend and/or event detection through real-time monitoring of the entire Twitter stream. Mathioudakis and Koudas, for example, developed TwitterMonitor [1], a tool that identifies hot topics on Twitter by detecting bursts of keywords that arrive at unusually high rates. TwitInfo [2] provides an event-tracking interface that can collect, aggregate, and visualize tweets about user-specified events as they unfold in the stream. These approaches are based on the numbers of tweets containing particular keywords. However, it is difficult to know how these keywords are disseminated and become popular in social media simply by detecting them. To address this, we focus on a system for monitoring and analyzing information diffusion data by tracking the retweeted tweets. A tweet retweeted by many users indicates that they find the content interesting and impactful. Sondy [3] provides network analysis and a visualization tool, but it is not for real-time use. We have developed a system for monitoring and analyzing the diffusion data interactively against a real-time tweet stream. Many users are interested in hot topics, so the system should be able to handle bursting tweets in real-time.

On social media, tweets about particular topics are often posted by many users within a short time frame. When this happens, our system must process thousands or tens of thousands of tweets simultaneously, which can cause performance degradation if there is a lack of memory or CPU resources in the stream server. Since our system is designed to be used interactively in real-time by many analysts, waiting more than one second for results in one query is not acceptable.

To maintain an acceptable query performance, we propose a capacity control method for filtering incoming tweets using extra attribute information of the tweets themselves. A tweet includes various pieces of information: the user who posted the tweet, the number of followers, and so on. Hence, we define a weight value representing the importance of a tweet with such extra information. We assume that popular tweets are important for analysis because they are impactful. If the weight value of an incoming tweet is lower than a specified threshold, the tweet is filtered out from the system.

Conventionally, there is a trade-off between query performance and accuracy of analysis results. We show that query performance is improved by our method and the method is better than the existing methods in terms of main-

Manuscript received June 29, 2016.
Manuscript revised November 7, 2016.
Manuscript publicized January 17, 2017.

[†]The author is with IBM Research – Tokyo, Tokyo, 103–8510 Japan.
[††]The author is with IBM Tokyo Software & Systems Development Laboratory, Tokyo, 103–8510 Japan.
[†††]The author is with Ochanomizu University, Tokyo, 122–0012 Japan.
a) E-mail: enomiki@jp.ibm.com

DOI: 10.1587/transinf.2016DAP0029

taining query accuracy.

We make three contributions in this work.

- Our diffusion analysis system enables real-time analysis of streaming social data to let users know how reshared messages are disseminated. We introduce typical query patterns for such analysis.
- Our system features capacity control of incoming messages to adjust for bursts by filtering less important diffusion data using extra attribute information of the messages.
- Query performance evaluation of our system using typical analysis scenarios with real bursting data from Twitter shows that our capacity control method outperforms existing methods in terms of maintaining query accuracy.

Section 2 of this paper describes the background and problem definition. Our diffusion analysis system is introduced in Sect. 3. Section 4 provides typical data access patterns for diffusion analysis. We introduce our capacity control method in Sect. 5. Section 6 presents our experimental environment and describes the evaluation. We discuss improvements to our method in Sect. 7. We review related work in Sect. 8 and conclude the paper in Sect. 9.

2. Background and Problem Definition

2.1 Performance Degradation When Bursting

One of the characteristics of social media is that thousands of social media messages are suddenly posted simultaneously. Such bursting creates a strain of machine resource and lower the query performance in analysis systems. Figure 1 shows an example of how tweets spread. Day 14 was the election day of the House of Representatives in Japan, 2014. The x-axis shows time and the y-axis shows the number of retweets including specific political terms. We can see that the number of tweets was bursting just after eight o'clock that night. This was when a quick flash report of the vote count was broadcasted, so many users tweeted about the election result.

This type of situation tends to occur whenever a particular topic is bursting. (e.g., an election, the World Cup, the Olympics, and disasters such as earthquakes). In these cases, our system must process thousands or tens of thousands of tweets simultaneously, which can create a strain of memory or CPU resources in the stream server.

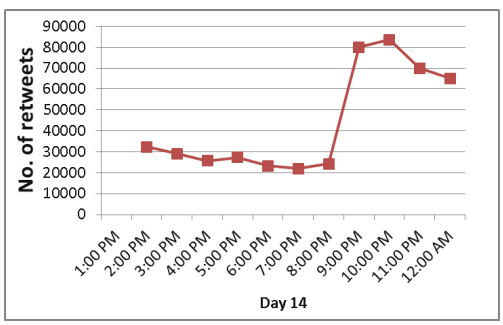


Fig. 1 Example of bursting tweets.

Figure 2 shows the response time of Queries 2 and 3 (details discussed in Sect. 3) and the CPU utilization of the stream server (detailed configuration discussed in Sect. 5). Query 2 is obtaining the rankings of retweet counts. Query 3 is obtaining the user rankings.

In the example in Fig. 2, after the vote counting, the response time lengthened to over one second and then became slower and slower. One cause of this was the lack of CPU resources to handle queries with a large number of incoming retweets.

2.2 Problem Definition

Since our analysis system is designed to be used interactively in real-time by many analysts, waiting more than one second for a query result is not acceptable. For example, a particular response time is specified by means of a service level agreements (SLA) in the cloud environment. In our system, (1) it is important to guarantee a certain response time in the form of an SLA.

As an existing solution, incoming tweets are filtered randomly to keep the number of tweets per second under control [4], [5]. Bosch et al. [6] provide a monitoring tool that filters incoming messages with keywords. They calculate a weight value for each search keyword on the basis of co-occurrences with an initial seed keyword. If the weight value of a keyword is lower than a predefined threshold value, that keyword is removed from the search keyword list. However, it is unclear that such methods are effective for our system, too.

In this work, we present our system for analyzing information diffusion data by tracking retweeted tweets. A useful query for our system involves not only obtaining retweet ranking, but also obtaining data from a whole retweet network. Such a retweet network can be used for analysis such as clustering and visualizing the diffusion path. Therefore, (2) it is important for our diffusion network analysis system to maintain high accuracy of the query results.

Conventionally, there is a trade-off between query performance and accuracy of analysis results. Our goal is to

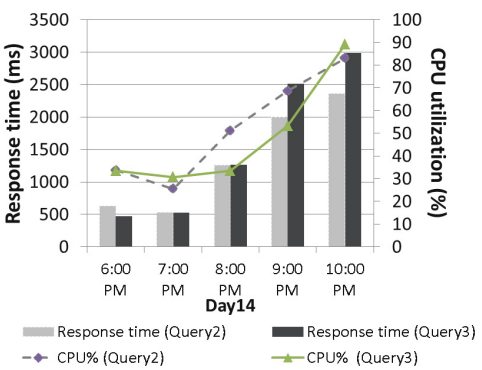


Fig. 2 Query performance when bursting.

maintain high accuracy of all query results as much as possible while honoring the SLA.

3. Real-Time Diffusion Analysis System

3.1 Framework of the Diffusion Analysis System

Our information diffusion data involves a collection of reshared messages. In this paper, we introduce our system with Twitter as a use case. Figure 3 shows our system operating as a real-time diffusion analysis system. The framework consists of the stream server and the application server. Incoming retweets are sequentially inserted into an in-memory data store running on the stream server. The application server provides various analysis modules for the diffusion data. For example, the "Ranking" module creates a ranking of retweet counts to detect the current hot tweets or the ranking of influential users for a specified topic, and the "Visualization" module displays a diffusion network for a specified tweet. We can add other modules for diffusion analysis, such as a profile analysis to detect users' hobbies or the locations of specified users. Each module retrieves diffusion data from the stream server.

These servers can run in the cloud environment. The stream server runs in one cloud instance and provides calculated results both in real-time and off-line. We can provide multiple application server instances for analysts. For example, one user might be focused on election topics for the analysis, while another user might want to retrieve messages related to a disaster topic.

To handle all the fresh diffusion data for each tweet retweeted by users, we use an in-memory data store to collect all of 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.

However, continuously storing all the diffusion data for many days is impractical since memory resources are limited. The stale data should be deleted or moved to a disk-resident database. To determine the staleness, we estimate retweet diffusion extinction for each tweet [7]–[9]. The disk-resident database stores historical diffusion data for use in offline data analysis. We focus on the real-time processing in this paper.

There are various candidates for implementing the in-memory data store, such as an in-memory database, a key/value store, and a graph data base [10], [11]. In our application, the primary scenarios involve finding popular tweets and aggregating or sorting the users who retweeted them in the data store. The in-memory database is best for our purpose since it can handle complicated queries using SQL.

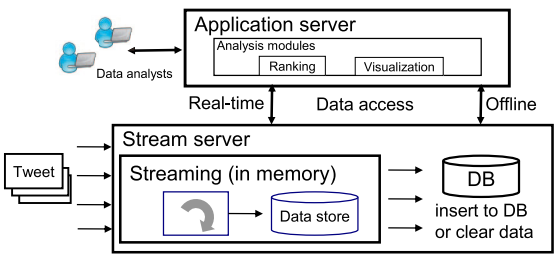


Fig. 3 Diffusion analysis system.

3.2 Tables in the In-Memory Database

The in-memory database contains two tables: a RETWEET table and an ORIGIN_TWEET table, as shown in Fig. 4. The RETWEET table stores retweeted messages. It has fields for the tweet ID of a retweet (TweetID), the tweet ID of the original tweet (RTID), the retweeted time (Time), the user name of the source (Src) and destination (Dst), the retweeted user's language (Lang), and the location information (Location). The value of Dst is the name of the user who retweeted. The value of Src is the user who was retweeted. The pair of Src and Dst in each row represents an edge in the diffusion network. The ORIGIN_TWEET table contains a tweet ID (TweetID), the tweeted time (Time), the name of the user who posted the tweet (User), the tweet message (Msg), and the retweeted count (RTcount). We increment the number of RTcount if we receive corresponding retweeted messages. The RTID in the RETWEET table can be joined with the TweetID in the ORIGIN_TWEET table if we want to obtain information of an original tweet.

4. Data Access Patterns

In this section, we describe typical data access patterns for our diffusion analysis system.

[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?" This query pattern can be interpreted using the following SQL example:

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

Figure 5 shows an example of the information diffu-

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	..
..	..	..	..	..	..

Fig. 4 Tables in the in-memory database.

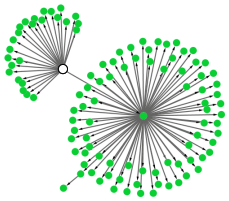


Fig. 5 Information diffusion network.

sion network of one tweet. The white node indicates the user who posted the original tweet and the green nodes are the users who retweeted it. The edges represent diffusion routes. For example, if user @u2 retweets a tweet posted by user @u1, an edge between user @u1 and user @u2 is created. From that network, we can find out if there is any user whose retweet was re-tweeted more times than the original tweet. If that user's name was @u3, user @u3 may have many more readers than the original user, which means user @u3 is the main influencer in the network.

If we input several tweet IDs (e.g., tweets including a specified keyword) in the WHERE clause, we can create more complicated diffusion network. The obtained network can then be used for network analytics such as clustering and frequent path detection.

[Obtain ranking results for specified tweets]
(1) Simple sort
Example: Find trending tweets
This query pattern is used to obtain the current rankings of retweet counts. This SQL query returns the top hundred most retweeted tweets:

```
[Query 2]
SELECT          *
FROM            ORIGIN_TWEET
WHERE           TweetID in (tweet IDs)
ORDER BY        RTcount    DESC
FETCH FIRST 100 ROWS ONLY
```

(2) Aggregation and sort
Example: 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 main influencer for this topic?" by using this pattern. The SQL query returns the top hundred users who retweeted most often from among the specified tweets:

```
[Query 3]
SELECT          Src, count(Src)
FROM            RETWEET
WHERE           RTID in (tweet ids)
GROUP BY        Src
ORDER BY        count(Src)    DESC
FETCH FIRST 100 ROWS ONLY
```

When Src is specified in the GROUP-BY clause, Query 3 returns the most influential users. Another possible scenario of this query pattern is obtaining the location ranking by referring to the user location information in the RETWEET table.

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, such as fetching data using a primary key and its associated values, are extremely fast. 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 [12], [13]. For example, Queries 2 and 3 may be slow when data becomes large because they use aggregation operations.

5. Capacity Control to Adjust for Bursts of Streaming Data with Attribute-Based Filtering

Here, we consider capacity control to adjust for bursts of streaming tweet data. As described in Sect. 2, our system should be able to maintain an acceptable query performance when bursting occurs.

As an existing solution [4], [5], incoming tweets are filtered randomly to keep the number of tweets per second under control. Morstatter et al. [5] reported that the number of top hashtags of tweets filtered randomly was highly correlated with the non-filtered original result. The results are considered to be reasonable, as popular tweets are mostly retained because the number of such tweets is originally large. The random sampling rate can be determined by monitoring the CPU resources and response time periodically.

Another solution is filtering with keywords. Bosch et al. [6] provide monitoring tool that filters incoming messages with keywords. They calculate weight value for each search keyword on the basis of co-occurrences with the initial seed keyword. A threshold value is determined by users. If the weight value of a keyword is lower than a predefined threshold, that keyword is removed from the search keyword list. As a result, the number of incoming tweets is reduced.

However, if we remove tweets randomly, the diffusion network can become fragmented. Our objective is to analyze diffusion data created by retweets, so all of the diffusion data relating to each tweet must be stored. When we create a diffusion network as shown in Fig. 5, some edges might be lost if incoming tweets are filtered at random, but if we remove tweets with keyword-based filtering, popular retweets including only minor keywords will be filtered out.

We propose another solution. A tweet includes various pieces of information (the user who posted the tweet, the number of followers, etc.), so in our system we define the importance of a tweet on the basis of this extra information. We assume that popular tweets are important for analysis because they are impactful, so we want to avoid filtering out any tweet (and its retweets) that has a large total retweet count.

We calculate the importance of each tweet by

$$Weight(t) = \begin{cases} getUserInfo(t_{RTID}), & if\ t\ is\ retweet \\ getUserInfo(t_{ID}), & otherwise \end{cases}$$

The weight of a tweet t represents the importance level

of the tweet. The *getUserInfo* method returns a weight value related to the original user or, if t is retweeted, to the user who posted the original tweet. If the weight value of an incoming tweet is lower than a specified threshold, the tweet is filtered out from the system. For example, we refer to the number of followers a user has as a weight value. Let us assume a user posts a retweet (ID = 11). If the number of followers of the user who posted the original tweet (ID = 1) is 500, *Weight*(ID11) = *getUserInfo*(ID1) = 500. When the threshold is 1000, the retweet is filtered out and not stored in the data store. This is based on the assumption that if the number of followers is large, the user is likely to be retweeted by many people because the user's tweet would catch many users' attention.

The procedure of our proposed filtering method is as follows.

```
1.  threshold = x;
2.  input_t = (incoming tweet or retweet);
3.  if(input_t == tweet) {
4.    //get weight value (e.g. # of followers)
5.    weight = getUserInfo(input_t);
6.  }else if (input_t == retweet){
7.    // get original tweet
8.    // it is included in the retweet in Twitter
9.    origin_t = getOriginalTweet(input_t);
10.   weight = getUserInfo(origin_t);
11. }
12. //filtered out
13. if(weight < threshold) return NULL;
14. else return input_t;
```

However, removing too many tweets would affect the accuracy of the analysis result (e.g., the retweet or user ranking results). This is a trade-off between capacity control and accuracy of analysis results. We can consider scaling out with several servers for capacity control. For example, incoming tweets are dispatched with the weight value, but we have to handle distributed query processing. This would negatively affect query performance.

Our proposed system is not restricted to Twitter service: it can be applied not only to such microblogging services, but also to real-time streaming data such as GPS and sensor data, whose incoming rates vary depending on time, location, and so on. For example, the GPS data of cars on expressways is collected to determine the traffic situation and control traffic volume. The amount of GPS data typically increases during holiday season and in particular time slots. In these cases, we can filter out data by using location information, as data from sparse areas may less important for traffic control. The criteria of importance differ depending on the analysis scenario.

6. Experiments

6.1 Evaluation Methodology

We filter tweets on the basis of weight (as described in Sect. 5) and evaluate the query performance to see how often important retweets are erroneously filtered out (ideally, we want this to happen as little as possible). We set the SLA for the query response time of Queries 1, 2, and 3 to one second. The system keeps filtering out until it satisfyies the SLA. The filtering rate will increase little by little just after the system detects failure of the SLA. We compare our method with the two existing methods described in Sect. 5.

(a) Random filtering (Wei [4], Morstatter [5])
Filter tweets randomly to maintain the specified filtering rates. We begin the rate from 10% and then increase it by 5% until the average query response time is less than one second.
(b) Keyword filtering (Bosch [6])
Calculate weight value for every search keyword. The value is calculated on the basis of co-occurrence with the initial seed keyword. In this experiment, the initial seed keyword is "election". We use Jaccard similarity coefficient, a common method for co-occurrence.
(c) Attribute filtering (proposed method)
Use number of followers to calculate the weight value. If the number of followers is less than the threshold, the tweet is filtered out.

We used Japanese tweets/retweets from 12/14 2014, the House of Representatives election day in Japan, that included political terms such as names of political parties and official accounts of candidate users. The total number of tweets/retweets was 616,148. Our test server used two Xeon X5670 CPUs (2.93GHz, 6 cores) with 32 GB of RAM, and Red Hat Linux 5.5. The H2 database [14] was used as the in-memory mode. Indices were created for the TweetID field in the ORIGIN_TWEET table and for the RTID and SRC fields in the RETWEET table to optimize Queries 2 and 3.

Our expectation going into the experiments was that our proposed method would be able to deliver a query performance at least as good as the comparative methods and with better accuracy of the results.

6.2 Experimental Results for Query Performance

We evaluate the query performance results of Queries 2 and 3 and omit the results of Query 1, as it can return results much faster than the other two. We simulated queries being issued by 100 analysts simultaneously. The targeted tweet IDs specified in the WHERE clause are tweets containing "Liberty Democratic Party of Japan" (LDP, the current ruling party in Japan)-in other words, users who want to investigate retweet and user rankings related to the LDP. We measured the response time per query. Each user issues a query ten times in each time slot. The SQLs used in the

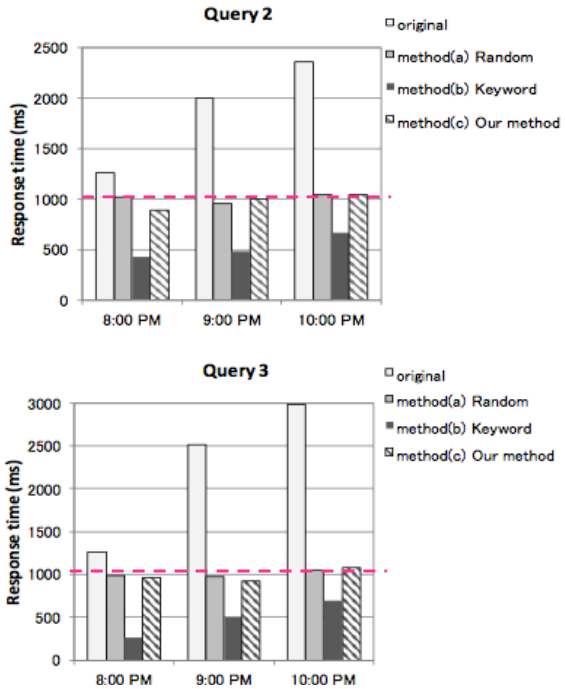


Fig. 6 Response time of Query 2 and Query 3.

	8:00 PM	9:00 PM	10:00 PM
(a) Random filtering rate	30% cut	35% cut	70% cut
(b) Keyword co-occurrence threshold	0.1	0.1	0.1
(c) Attribute filtering threshold	400	1000	1400

Fig. 7 Filtering rate and threshold in each time slot.

experiment are the same as those in Sect. 4.

Figure 6 shows the median value in each time slot for Queries 2 and 3. The response time at 10 p.m. is the longest in the original results because the total number of rows in the in-memory database is largest. In all time slots, method (a), method (b), and method (c) are able to maintain a response time under one second.

Figure 7 shows the filtering rate and threshold in each time slot. The rate is determined to satisfy the SLAs of the results of all Queries. We filtered out incoming tweets to a maximum of 70% in method (a). Because the total number of rows at 10 p.m. in the in-memory database is largest, 70% of incoming tweets had to be filtered out to meet the SLA. The response time of method (b) was much faster than one second, but if we set a lower threshold value (e.g., 0.09), the response time became longer than one second. We therefore adjusted the threshold to 0.1. In the case of method (c), the system set the threshold value from 400 to 1400, meaning the system filtered out tweets/retweets whose weight was lower than the threshold.

Figure 8 shows the CPU utilization of each query. Our system consumed over 80% of CPU resources with the original data, which was reduced to under 50% after filtering. This shows that we can reduce performance degradation by filtering tweets during bursting. At the same time, this would affect the accuracy of query results. We evaluate the

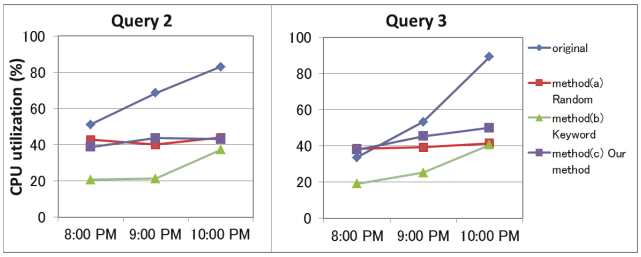


Fig. 8 CPU utilization.

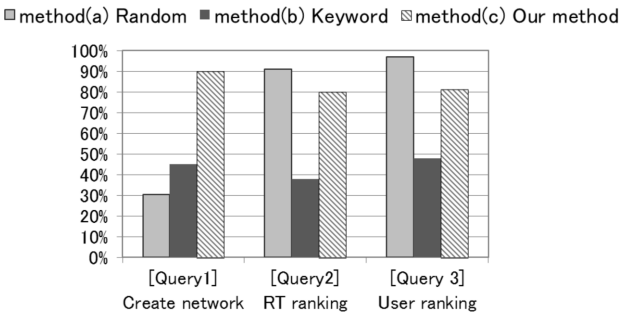


Fig. 9 Coverage of the original result.

accuracy of the query results in the next experiment.

6.3 Evaluation of Accuracy of Query Results

Figure 9 shows coverage of the analysis results compared with the original results. The results of Query 1 show coverage of the total number of diffusion network edges in the top 100 tweets. We counted the total number of edges (one edge = a pair of Src and Dst) of a diffusion network created by tweets in the ranking. Each tweet in the ranking has a diffusion network of retweets, as described in Sect. 4. We calculated the top 100 tweets (Query 2) and users (Query 3) after applying methods (a), (b), and (c), at 10 p.m. Then we measured how many tweets/users are retained in the top 100 results compared with the original top 100 results. Here, 100% mean that all results of the original top 100 are retained.

In Query 1, methods (a) and (b) failed on about 70% and 55% of edges, respectively, while method (c) maintained at about 90%. This means that the diffusion network after filtering with methods (a) and (b) failed at 70% and 55% of the edges, respectively. In the case of method (c), the diffusion network whose original user had a large number of followers was not filtered out. Method (a) could keep over 90% in Queries 2 and 3, which we consider reasonable since the 70% sampling reduced the number of tweets while mostly retaining popular tweets because the number of retweets is originally large. The accuracy of method (b) for Queries 2 and 3 were under 50%. This indicates that many popular retweets were filtered out when the search keywords were reduced. Method (c) was able to maintain an accuracy of about 80%, but this is lower than method (a). One reason is that some users who have a small num-

ber of followers are retweeted widely but are filtered out by method (c). Take the 12th-place user in the original user ranking as an example. This user was filtered out by method (c) since its weight value was lower than the threshold. However, this user was retweeted by a user with 30,498 followers, which resulted in the tweet having the chance to be retweeted by many more users than usual.

7. Improved Method for Attribute-Based Filtering

7.1 Improvement of Attribute Filtering

The coverage rate of method (c) can be improved by considering the information of users who retweeted. Currently, we refer only to user information of the original tweet. In this section, we modify the procedure of our filtering method.

The *getUserInfo* method returns a weight value related to the user who posted the original tweet if t is retweeted. We also calculate a weight value for the user who retweeted. If the weight value is more than a threshold, the retweet is not filtered out. Furthermore, we retain subsequent retweets even if their weight value is lower than the threshold.

For example, assume a tweet is retweeted like in Fig. 10. We assume that the weight value of the user who posted the original tweet is lower than the threshold. The whole diffusion data is filtered out by our original filtering method because we refer only to the information of the user who posted the original tweet. In the improved method, if the weight value of user A is more than the threshold, the user's subsequent retweets are not filtered out. Specifically, the retweets framed by the red rectangle in Fig. 10 are retained in the in-memory database.

The procedure of the improved method is as follows. If the weight value of a retweeted user is more than the threshold, the tweetID of the original tweet is stored in retainList to prevent the subsequent retweets from being filtered out.

We assume that popular retweets will be retained more frequently than with the original method, but we have to increase filtering ratio to meet the SLA. We applied the improved method (= method (c)') at 10 p.m. The threshold was set to 2500 to keep the query response time under one second.

Figure 11 shows coverage of the analysis results with the improved method. Method (c)' demonstrated improved accuracy in all queries. More specifically, method (c)' had results comparable with those of method (a) in Queries 2 and 3 and outperformed method (a) in Query 1.

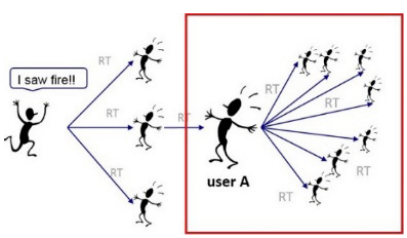


Fig. 10 Tweet diffusion example by retweets.

7.2 Evaluation of Our Improved Method with Other Scenario's Data Set

Up to this point, we have experimented with an election data set. Such burst event is predictable since it is scheduled in advance, but sometimes unpredictable events such as disasters or blackswan occur, too. In this section, we evaluate our improved method with unpredictable scenario's data set. Our capacity control system is applicable to any situation because the system always monitors query performance, and when the performance does not meet the SLA, the system adapts the threshold immediately.

In this experiment, we used Japanese tweets/retweets including disaster terms related to "earthquake" from Twitter on 4/14 2016, which is the date the 2016 Kumamoto

```
1.  threshold = x;
2.  input_t = (incoming tweet or retweet);
3.  if(input_t == tweet) {
4.     weight = getUserInfo(input_t);
5.     if(weight < threshold) return NULL;
6.     else return input_t;
7.  }else if (input_t == retweet){
8.     // get tweetID of original tweet
9.     rtID = getOriginTweetID(input_t);
10.    if(retainList.contains(rtID) return input_t;
11.   else{
12.    origin_t = getOriginalTweet(input_t);
13.    weight = getUserInfo(origin_t);
14.    if(weight < threshold) return NULL;
15.    weight = getUserInfo(input_t);
16.    if(weight < threshold) return NULL;
17.    else{
18.      retainList.add(rtID);
19.      return input_t;
20.   }
21. }
```

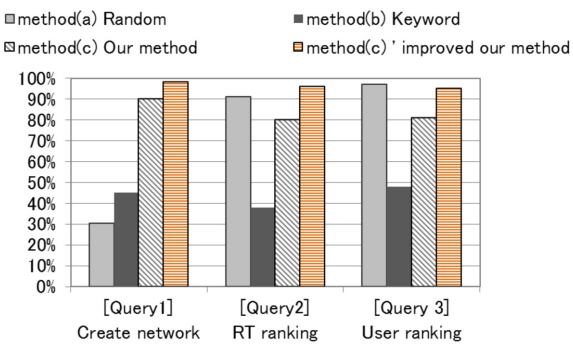
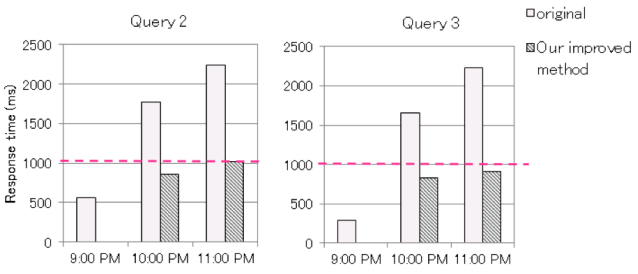


Fig. 11 Coverage of the original result compared with that of the improved method.



	10:00 PM	11:00 PM
Attribute filtering threshold	800	1200

Fig. 12 Response time of Query 2 and Query 3, and threshold.

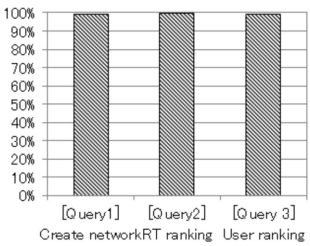


Fig. 13 Coverage of the original result.

Earthquake occurred in the Kyushu region of Japan. The earthquake occurred at 9:26 PM, so we measured the query performance from 9:00 p.m. to 11:00 p.m. The total number of tweets/retweets was 950,177. Our test environment is the same as in Sect. 6.

The targeted tweet IDs specified in the WHERE clause in Queries 2 and 3 are tweets containing "blackout" or "shelter".

Figure 12 shows the median value in each time slot of Queries 2 and 3. After the earthquake, the response time at 10 p.m. was over one second. Our improved method can maintain a response time under one second after 10 p.m. Figure 13 shows the coverage of analysis results with the improved method. As shown, it could cover 99% of the original results in all queries. Our system confirmed effectiveness with the unpredicted scenario.

8. Related Works

There have been various studies on the use of information diffusion analysis in social networks.

Truthy [15] is a Web service that tracks political information on Twitter. It provides real-time analysis of information diffusion on social media by mining and visualizing massive streams of microblogging events. Gupta et al. [16] tried to identify the important content and source-based features, seeking those that can predict the credibility of information in a tweet. They presented an algorithm to automatically calculate the credibility. TweeQL [17] is an SQL-like query interface for streaming Twitter data. The streaming API allows users to issue long-running HTTP requests with keyword, location, or user ID filters, and then to collect the matching tweets that appear in the stream. Taxidou et al. [18] introduces a system for real-time analysis of information diffusion on Twitter. It provided a visualization tool with their estimation algorithm of information cascade.

Although these works pertain to streaming social media analysis services, they focus on the algorithms used for analysis. In contrast, our focus is the development of a real-time analysis system and an efficient framework to enable a superior diffusion analysis performance.

Real-time analysis of streaming data has been studied by many researchers [19]–[21] who have set continuous queries over data streams to optimized query performance. The queries are fixed in advance and periodically used for incoming streaming data within a specified window.

In contrast, our system is developed for interactive use. We assume that many queries run simultaneously so that many analysts can issue various queries. It is important to return the required data rapidly since the system is used interactively in real-time. S-Store [22] is a streaming processing that uses an OLTP engine. It supports SQL and focuses on transactional support of streaming data.

9. Concluding Remarks

In this paper, we introduced a real-time monitor and analysis system for information diffusion data. Analysts can use our system interactively to find tweets retweeted by many users as well as to identify the key people who are retweeted frequently by many users or who have retweeted tweets about particular topics. On social media, tweets about a particular topic are often posted by many users in a short time. In this case, our system must process thousands or tens of thousands of tweets simultaneously, which can cause performance degradation stemming from a lack of memory or CPU resources in the stream server. It is important to maintain a nearly real-time response time against queries and to detect popular messages, as our system is designed to be used interactively in real-time by many analysts.

To maintain query performance, we developed a capacity control method that filters incoming tweets using extra attribute information of tweets. We evaluated the effectiveness of the filtering method for capacity control with Twitter data. Conventionally, there is a trade-off between query performance and accuracy of analysis results. We showed that the query performance can be improved by our method and that it is better than the existing methods in terms of maintaining query accuracy. We also confirmed that our method works well in two different types of bursting scenarios.

For future work, we will evaluate our method with additional data sets.

References

[1] M. Mathioudakis and N. Koudas, "TwitterMonitor: Trend detection over the twitter stream," Proc. 2010 International Conference on Management of Data, pp.1155–1158, 2010.
[2] A. Marcus, M.S. Bernstein, O. Badar, D.R. Karger, S. Madden, and R.C. Miller, "Twitinfo: Aggregating and visualizing microblogs for

event exploration," Proc. 2011 Annual Conference on Human Factors in Computing Systems (CHI), pp.227–236, 2011.
[3] A. Guille, C. Favre, H. Hacid, and D. Zighed, "Sondy: An open source platform for social dynamics mining and analysis," Proc. SIGMOD, pp.1005–1008, 2013.
[4] Y. Wei, V. Prasad, S.H. Son, and J.A. Stankovic, "Prediction-based QoS management for real-time data streams," 27th IEEE International Real-Time Systems Symposium (RTSS), pp.344–358, 2006.
[5] F. Morstatter, J. Pfeffer, H. Liu, and K.M. Carley, "Is the sample good enough? Comparing data from Twitter's streaming Api with Twitter's firehose," Proc. Seventh International AAAI Conference on Weblogs and Social Media (ICWSM), pp.400–408, 2013.
[6] H. Bosch, D. Thom, F. Heimerl, E. Püttmann, S. Koch, R. Krüger, M. Wörner, and T. Ertl, "ScatterBlogs2: Real-time monitoring of microblog messages through user-guided filtering," IEEE Conference on Visual Analytics Science and Technology (VAST), pp.13–18, 2013.
[7] W. Galuba, D. Chakraborty, K. Aberer, Z. Despotovic, and W. Kellerer, "Outtweeting the Twitterers – Predicting information cascades in microblogs," 3rd Workshop on Online Social Networks (WOSN), 2010.
[8] S. Asur, B. Huberman, G. Szabo, and C. Wang, "Trends in social media: Persistence and decay," Proc. Fifth International AAAI Conference on Weblogs and Social Media (ICWSM), 2011.
[9] M. Enoki, I. Yoshida, and M. Oguchi, "A practical framework for real-time diffusion analysis in social media," Proc. 12th ACM International Conference on Computing Frontiers (CF), 2015.
[10] HBase, http://hbase.apache.org/
[11] Neo4j, http://www.neo4j.org/
[12] Evaluation of in-memory database TimesTen, http://zenodo.org/record/7566/files/CERN_openlab_report_Endre_Andras_Simon.pdf
[13] Performance Benchmarks: TimesTen vs Oracle Database, http://www.peakindicators.com/index.php/knowledge-base/1136-performance-benchmarks-timesten-vs-oracle-database
[14] H2 Database Engine, http://www.h2database.com/html/main.html
[15] J. Ratkiewicz, M. Conover, M. Meiss, B. Gonçalves, S. Patil, A. Flammini, and F. Menczer, "Truthy: Mapping the spread of astroturf in microblog streams," Proc. 20th Intl. World Wide Web Conf. (WWW), pp.249–252, 2011.
[16] A. Gupta and P. Kumaraguru, "Credibility ranking of tweets during high impact events," Proc. 1st Workshop on Privacy and Security in Online Social Media (PSOSM), 2012.
[17] A. Marcus, M.S. Bernstein, O. Badar, D.R. Karger, S. Madden, and R.C. Miller, "Processing and visualizing the data in Tweets," Proc. SIGMOD, pp.21–27, 2011.
[18] I. Taxidou and P.M. Fischer, "RApID: A system for real-time analysis of information diffusion in Twitter," Proc. 23rd ACM International Conference on Information and Knowledge Management (CIKM), pp.2060–2062, 2014.
[19] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N.-E. Frantzell, "Supporting streaming updates in an active data warehouse," IEEE International Conference on Data Engineering (ICDE), pp.476–485, 2007.
[20] S. Babu and J. Widom, "Continuous queries over data streams," Proc. SIGMOD, pp.109–120, 2001.
[21] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: Semantic foundations and query execution," VLDB Journal, vol.15, no.2, pp.121–142, 2006.
[22] U. Cetintemel, T.J. Kraska, S. Madden, D. Maier, J. Meehan, and S. Zdonik, "S-Store: A streaming NewSQL system for big velocity applications," Int. J. Very Large Data Bases (VLDB), pp.1633–1636, 2014.

Miki Enoki received a Ph.D. from Ochanomizu University in 2016. She has been IBM Research – Tokyo since 2007. Her research focus is performance analysis of Java applications and middleware.

Issei Yoshida joined IBM Research – Tokyo in 2001. His research focus is text analytics and its application to question answering.

Masato Oguchi received a Ph.D. from the University of Tokyo in 1995. He joined Ochanomizu University as an associate professor in 2003. Since 2006, he has been a professor in the Department of Information Sciences, Ochanomizu University. His research focus is in network computing middleware, including high performance computing as well as mobile networking.