# Towards Privacy-Preserved Query Optimization on Microblog Data

Jie Zhao

*School of Business, Anhui University*
*zj_teacher@126.com*

***Abstract***

*Microblog platform, such as Twitter and Sina, has been one of the major ways of information diffusion in modern society. However, although microblog has been proven to contain lots of information, it is really hard for people to find useful information on it. Besides, some information in microblog such as user IDs is not allowed to publish due to privacy policies. Thus the queries on microblog data can be regarded as privacy-preserved queries. One of the challenged issues is the poor performance of answering privacy-preserved queries over microblog data, which owes to the large and increasing volume and the complex social network structure of microblog data. In this paper, we propose a basic idea to optimize the privacy-preserved queries on microblog data. We use a query-specific approach to treat the queries, i.e., the microblog data is first preprocessed according to the specific requirements of different types of queries, which are then organized through some indexing structures. Our preliminary experiments on real microblog data show that this approach has reasonable performance.*

***Keywords:*** *microblog data, query, optimization, privacy preservation*

## 1. Introduction

Nowadays, microblogging service such as Twitter [1] and Sina Weibo [2] has been widely used all over the world. It has been demonstrated that microblog usually has a big impact on a lot of social and political events. Therefore, recently research on microblog data has been a hot topic.

While microblogging services can continuously produce a huge amount of data, how to use those data is still a challenging problem. The hardest issue is the data volume, which makes traditional query processing techniques difficult to answer queries over microblog data. Another issue is the privacy problem. As microblog data contains a lot of personal private information about users as well as their social network, many countries have laws that disallow people to directly use the microblog data. From this perspective, we can only perform *privacy-preserved queries* over microblog data, which means a query can not reveal any private information about users.

However, even though the privacy-preserved queries can not retrieve user names as well as their personal messages, we are still able to find lots of useful information from microblog data, which can be further used to server many applications including event prediction, sentiment analysis, and so on. This is because microblog data contains rich information about social network structure.

In this paper, we focus on the optimization issues for the privacy-preserved queries over microblog data. We will first state the problem regarding privacy-preserved queries, and then propose a query-specific approach to optimizing those queries. We also introduce some index structures to improve the query performance.

## 2. Problem Statement

Microblog platform now has a big impact on people's daily life. Typical microblog service providers, such as Twitter (U.S.A.) and Sina Weibo (China) allow users to get the original data via some APIs. Thus we can build a spider to continuously obtain microblog data and therefore construct a real-time database for microblog data. However, due to privacy preservation policies in most countries, we are not allowed to directly use the contents as well as user identifiers information in our research. We name such search behaviors on microblog data as *privacy-preserved queries*.

In this paper, we mainly focus on the optimization techniques for privacy-preserved queries over microblog data. It should be noted that privacy-preserved queries can also bring a lot of new insights to many applications, due to the rich information introduced by the social network information among microblog data. For example, we can predict the retweeted count of a specific tweet based on the followers and followee list, which can be used to conduct micro/macro economics analysis, e.g., car sales, consumers' confidence, and so on.

To answer privacy-preserved queries, we can create some relational tables to store the acquired microblog data except the private parts in them. Figure 1 shows a design of such a database, where we use five tables to represent microblog data.

| MICROBLOG | | |
|---|---|---|
| **Field Name** | **Field Definition** | **Comments** |
| MID | Variable text, size 64 | Message ID |
| UID | Variable text, size 200 | User ID |
| TIME | Date and time | Timestamp |
| Primary Key: MID | | |

(a) microblog

| FRIENDLIST | | |
|---|---|---|
| **Field Name** | **Field Definition** | **Comments** |
| UID | Variable text, size 200 | User ID |
| FRIENDID | Variable text, size 200 | Friend ID |
| Primary Key: (UID, FRIENDID) | | |

(b) friendlist

| EVENT | | |
|---|---|---|
| Field Name | Field Definition | Comments |
| MID | Variable text, size 64 | Message ID |
| TAG | Variable text, size 200 | Event Tag |
| Primary Key: (MID, TAG) | | |
| (MID) Foreign Key, references MICROBLOG(MID) | | |

(c) event

| MENTION | | |
|---|---|---|
| Field Name | Field Definition | Comments |
| MID | Variable text, size 64 | Message ID |
| UID | Variable text, size 200 | User ID |
| Primary Key: (MID, UID) | | |
| (MID)Foreign Key, references MICROBLOG (MID) | | |

(d) mention

| RETWEET | | |
|---|---|---|
| Field Name | Field Definition | Comments |
| MID | Variable text, size 64 | Message ID |
| TAG | Variable text, size 200 | Event Tag |
| PATH | Variable text, size 700 | Retweet Path |
| Primary Key: (MID, REMID) | | |
| (MID,REMID) Foreign Key, references MICROBLOG (MID,MID) | | |

(e) retweet

**Figure 1. Tables used to Store Crawled Microblog Data**

Based on the database design shown in Fig.1, now we can formulate the typical privacy-preserved queries over microblog data. Table 1 summarizes those queries.

How to answer the above queries over microblog data? This paper is mainly aimed at this issue. The biggest challenge is due to the large data volume. Hence we have to design efficient techniques to optimize the queries over microblog data.

**Table 1. Typical Privacy-preserved Queries over Microblog Data**

| Query | Description |
|---|---|
| Q1 | Find top-x suggested followees for user A. |
| Q2 | Find top-x users for user A, who are A's followees and have top-x followers' count. |
| Q3 | Find top-x users for user A, who are A's followees and have top-x followees' count. |
| Q4 | Find out users followed by User A and B together. |
| Q5 | Find out users who are B's followers and A's followees. |
| Q6 | Find top-x users who are mentioned in all microblogs in a time range. |
| Q7 | Find top-x users who are mentioned in user A's microblogs in a time range. |
| Q8 | Show top-x latest microblogs from user A's followees or the followees of them. |
| Q9 | Find out top-x users most interested the tag "?". |
| Q10 | Find out top-x users order by the num. of their microblogs being retweeted by others in a time range. |
| Q11 | Find out top-x users order by the num. of their microblogs retweeting A's microblogs in a time range. |
| Q12 | Find out top-x microblogs which are from A's followees or followees of them and ordered by num. of being retweeted by others in a time range. |
| Q13 | Find out top-x users (not A's followees) order by num. of tags being mentioned by A's microblogs and their microblogs in a time range. |
| Q14 | Find out top-x tags order by the num. of being mentioned by the microblogs in a time range. |
| Q15 | Find out top-x users order by num. of their microblogs containing tag "?" in a time range. |
| Q16 | Find out top-x users order by num. of microblogs being retweeted by user A's followees in a time range. |
| Q17 | Find out top-x users (of A's followees) order by num. of being mentioned by all the microblogs in a time range. |
| Q18 | Find out top-x users(of A's followers) order by their microblogs mentioning A in a time range |
| Q19 | Find out top-x trending tags, which are came from A's followees or followees of them and ordered by num. of being mentioned by the microblogs of A's followees or followees of them. |

## 3. Rules for Query Optimization

### 3.1 Classification on Privacy-Preserved Queries

We analyze the original dataset and queries carefully and find that we can group the queries into several classifies according to the disparate attributes they need while processing queries.

(1) *User-based queries*: this group includes Q1, Q2, Q3, Q4, Q5. Those queries only use the user followship data to obtain result.

(2) *Mention-based queries*: this group includes Q6, Q7, Q17, Q18. Those queries are the only ones that need the mention information. Besides, they may need user followship data as well.

(3) *Retweet-based queries*: this group includes Q10, Q11, Q12, Q16. Those queries are the only ones that need the retweet information. Besides, they may need user followship data as well.

(4) *Event-based queries*: this group includes Q9, Q13, Q14, Q15, Q19. Those queries are the only ones that need the event information. Besides, they may need user followship data as well.

(5) *All Microblog-based Queries*: this group only contains one query, Q8. We cannot put it into any of the groups above, so it is handled individually in our system.

### 3.2 Data Preprocessing Rules

The original microblog data is not efficient to answer most of the given queries. For example, the user followship network data contains user information like "USERID\tFOLLOWEEID\n", so if we want to search for the friends of a user's friends, we might traverse the dataset twice and result in poor performance. Besides, the tweets dataset contains different kinds of information, e.g., event tag, mention tag, and retweet tag. However, those information is not necessary for all the queries. So in the data preprocessing step, we make preprocess for each group of queries. Our preprocessing mainly contains two parts: general preprocessing and query-specific preprocessing.

#### 3.2.1 General Preprocessing

The general preprocessing rules are as follows.

(1) We first split the original microblog data into many small files according to different time frames. After considering about different time span of queries, we chose one day as the time frame. Therefore, the original data set is partitioned into a set of files, each of which contains tweets posted in a specific day.

(2) In the original tweets dataset, a tweet may have ten tags at most, and each tag contains distinct attributes. For each group of queries, it is not necessary to scan the whole dataset because only a part of data may be involved. So we first extract tweets with specific attributes (e.g. mention, retweet or event) for each group of queries, then create different data formats for tweets with different attributes.

(3) In the original tweets dataset, many tags have extra information that is redundant for querying (e.g. the tag name "time:" is redundant in the tag "time: 2009-09-09 09:09:09"), so we simply remove those information. Besides, as we have partitioned the original dataset by one day granularity, the time information we need is only the hour time "09:09:09", so we only retain the hour time information in the partitioned file. Furthermore, we transform the string formatted hour time into an Integer (e.g. "09:09:09" into 90909) to accelerate querying.

#### 3.2.2 Query-Specific Preprocessing

The general preprocessing is not enough for query accelerating because the queries in different group varies a lot, and even queries in the same group may need different kinds of data formats. So we need to employ extra preprocessing based on different queries, which is called *query-specific preprocessing* in this paper. As a result, we design different preprocessing rules for different types of queries.

**(1) User-based queries**

The data format the original user followship dataset provides is not enough for efficiently querying. We observe the queries in this group and found that there are mainly three targets we need to achieve: finding a user's followee, follower and r-friend. So we first statistic a user's follower, followee and r-friend information and create three files with specific formats that contains the three information accordingly. This data formats are shown in Table 2.

We take userfile1 as an example to illustrate the new data format. In userfile1, each line of the file contains a user's followers information, The userID, number of followers and follower list are separate by tag "%". Different followers in the followers list are separate by tag "#". In this paper we call such a line of data a "record". With an efficient indexing mechanism we will discuss in 3.3, we could obtain a user's record in millisecond level time, so this data format could accelerate user-based queries a lot.

**Table 2. User-based Query Preprocessing Rules**

| Rule | Data Format (each line) | File Type |
|---|---|---|
| 1 | UserID % Follower count % follower1# follower2# follower3#... | UserFile1 (Follower) |
| 2 | UserID % Followee count% followee1# followee2# followee3#... | UserFile2 (Followee) |
| 3 | UserID % R-friend count% r-friend1# r-friend 2# r-friend 3#... | UserFile3 (R-friend) |

**(2) Event-based Queries**

There are microblogs that contain event tags in the tweets dataset. We first extract all these microblogs then group them by tag name. Each group of event microblogs is splitted into small files by one day granularity. After observe the queries, we build three kinds of files with distinct data forma.

The data formats of the three files are shown in Table 3.

**Table 3. Event-based Query Preprocessing Rules**

| Rule | Data Format (each line) | File Type |
|---|---|---|
| 1 | UserID % Mention count%time1# time2 #time3#… | Event -User File |
| 2 | UserID \t Time \t Event tag | User-Event File |
| 3 | UserID \t time | User-Time File |

**Event-User File:** We found that most queries required us to find out the count of a user mentioning an event, from this point we build event-user file which contain information about which users mention an event tag in one day, as shown in Table 3. The value "count" represents how many times a user mentioned the event tag in one day, and "time1#time2…" is the timestamp of each mentioning action.

**User-Event File:** In order to find out what events a user mentioned in a time period, we create the event-user file. We first employ a simple hash function to hash user into small files, than for each file we store the information of a user microblogs with event tag. Each line in the file contains user id, timestamp and the event tag name.

**User-Time File:** Since some of the queries may be related to large amount of users, searching by user may cost much time, so we create a User-Time file for each of the events by one day granularity, it's alight-weight file that only contains the timestamp and user id for each microblog mentioning the event, and the file is sort by timestamp.

**(3) Retweet-based and Mention-based Queries**

After observing the queries and attributes in the two groups of queries, we found that the two groups have a lot in common. Both groups of queries mainly focus on two actions: user A mention/ retweet user B or user A is mentioned/ retweeted by B. So in this part we illustrate the two groups together. We take retweet-based query as a representative. There are four queries (Q10, Q11, Q12, and Q16) in the retweet group. We find two features to describe a query: user set (one/ multiple/ all user) and retweet type (retweet/ being retweeted). User set means the least number of users we need to traverse in order to obtain the result. Retweet type means the information we need to find is about a user (in the user set) retweeting others tweets or a user's tweets being retweeted by others. Take Q11 for example, Q11 requires us to find out top-X users ordered by the number of their tweets retweeting user A's microblogs. So the least number of users we need to traverse is only one (A), and the retweet type is being retweeted. We list those features in Table 4.

**Table 4. Features of Retweet-based Queries**

| Mention Type | User Set | Query |
|---|---|---|
| Being Retweeted | All | Q10 |
| Being Retweeted | One | Q11 |
| Being Retweeted | Multiple | Q12 |
| Retweet | Multiple | Q16 |

According to the features listed in Table 4, we create three files with distinct format: user-retweeted file, user-retweet file and user file. User-retweeted file is aiming at handling queries which retweet type is being retweeted and user set is one or multiple. Each line of the file contains information about whom and when a specific user's tweets are retweeted by in a day. User-retweet file is aiming at handling queries which retweet type is retweet and user set is one or multiple. Each line of the file contains information about whose tweets a specific user retweets in a day. Source file contains only userID and time in one line, and the total count of a user appearing in the file is stored in the count file, the total count includes one day granularity and one month granularity. We create this file to handle queries which user set is all or multiple, such as Q10. The data formats of the three files are listed in Table 5 and Table 6.

**Table 5. Mention-based Query Preprocessing Rules**

| Rule | Data Format (each line) | File Type |
|---|---|---|
| 1 | UserID%Mention count\|uid1#time1\|uid2#time2\|uid3#time3\|… | User-Mention-File |
| 2 | UserID%Mentioned count\|uid1#time1\|uid2#time2\|uid3#time3\|… | User-Mentioned-File |
| 3 | UserID \t time | Source-File |
| 4 | UserID \t count | Count-File |

**Table 6. Retweet-based Query Preprocessing Rules**

| Rule | Data Format (each line) | File Type |
|---|---|---|
| 1 | UserID%Retweet count\|uid1#time1\|uid2#time2\|uid3#time3\|… | User-Retweet -File |
| 2 | UserID%Retweeted count\|uid1#time1\|uid2#time2\|uid3#time3\|… | User-Retweeted-File |
| 3 | UserID \t time | Source-File |
| 4 | UserID \t count | Count-File |

**(4) All Microblog-based Queries**

Since there is only one query in this group, we do not build index on the whole microblog dataset, instead we create a light-weight file that only contain user id and microblog id for each microblog, and the file is sorted by time.

### 3.3 Indexing

Preprocessing is done to provide well-designed data format for efficiency queries, but in order to obtain those data while querying efficiently, we need to employ a series of indexing mechanisms. We build various indexes for each group of queries. After experiments we found that different index performs different on different queries. So we choose the best index for each query based on experiment result. Here is the summarization of our indexes:

**(1) Use-Based Queries:** A modified B-tree index and an inverted index are constructed. As we knew, the B-tree index [3] is widely used in many DBMS, we first introduce the modified B-tree index used in user layer. The index is based on the lexicographic order of user ids and we modify the format of the leaf nodes. Figure 2 is the format of a B-tree index.

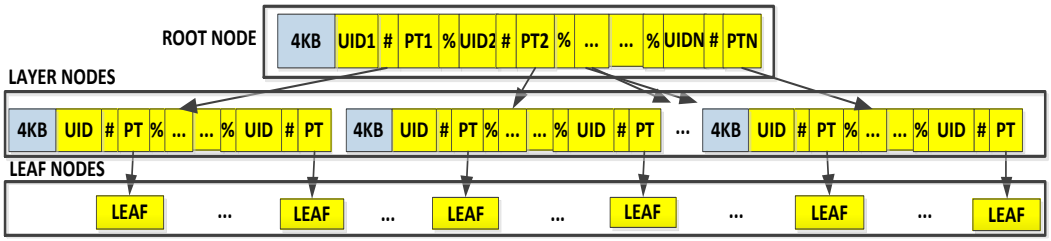


**Figure 2. The modified B-tree index Structure**

**(2) Event-based Queries:** A three-layer index (event - time - user) is built. The user layer is the same as in user-based queries and no index is used.

**(3) Retweet based & Mention based Queries:** A two layer index (time - user) is constructed for both user-retweet/ user-mention and user-retweeted/ user-mentioned file. The user layer is the same as in user-based queries and no index is used. There are 46 events and 916 days (from 2009-8-14 to 2012-2-17) in the dataset, we index the events by event name and index time by one day granularity, those are easy to understand. Here we introduce the user layer index in detail.

The basic data size in our index is 4KB. In particular, all the non-leaf files are organized as 4KB files. In order to minimize the time spent in open/ close files, we put

data into the leaf node. The first 4KB of the file is the file header which contains information of the offset and size of a data block, here offset represents the *i*-th 4KB in the file, and size represents how many 4KBs the data block contains, and a data block represents one specific line in the files described in 3.2. The non-leaf nodes could be loaded into RAM.

The general querying algorithm of the modified B-tree index is shown in Algorithm 1. This algorithm may be used in most of the 19 queries.

**Algorithm. 1. Querying Algorithm of Modified B-tree index**

**For** a specific user id A.

1. Read root node, get A's layer node number through pointer.
2. Read layer node, get A's leaf node number through pointer.
3. Read the file header of the leaf node, get the offset and size of A's data block.
4. Skip to the offset address, read the data block into RAM.

We use Lucene 3.5 [4], an open source search software developed by Apache software foundation to build a <line number, user id> inverted index [5] for the files describe in 3.2. We first employ a simple double hash function to hash users to smaller files. Then we build inverted index for each of those files. The querying algorithm is described in Algorithm. 2.

**Algorithm. 2. Querying Algorithm of Inverted index**

**For** a specific user id A.

1. Get the file number of A through the hash function.
2. Get the line number of A through inverted function.
3. Read the file by line until reach the A's line. Read the data block into RAM.

## 4. Execution of Privacy-Preserved Queries over Microblog Data

In this section we describe the query processing of each query by providing the pseudo-code. Limited by the length of article, here we just list the pseudo-code of some representative queries. Algorithm 3 to Algorithm 7 describes the process of those queries.

### 4.1 User-based Queries

Since the size of files of this group described in 3.2 is quite large and the total number of users is quite big, we select the modified B-tree index as the indexing mechanism of this group. We take Q2 as an example to illustrate the query process; the other queries have a similar process with Q2. The detailed process is shown in Algorithm 3.

**Algorithm. 3. Query processing of Q2**

**For** a specific user id A.

1. Empty the Result Set.
2. Get A's followee data block in UserFile2 by index.
3. Extract A's followees by processing the data block in RAM.
4. **For** each of A's followee B
   (a) Get B's follower data block in UserFile1 by index.
   (b) Extract B's followers by processing the data block in RAM.
   (c) **For** each of B's followers C,
   **If** C is in result set, increase the count of C.
   **Else** put C into result set, set count to be 1.
5. Sort the result set by a quick-sort algorithm and return the top-K users.

### 4.2 Event-based Queries

There are many queries in this group which combine query of user with query of other attributes, as well as in Retweet and Mention query group. Since the size of the users result set may be quite large (tens of thousand), there are two ways of querying other attributes: using user index and using user-time file (or source file for retweet/ mention file). For a large user set, using user index may be time-consuming, but for small user set, it's a good choice, and the choice is depend on specific query.

There are two ways to process Q9: using User-Event file (for each of a user, calculate the number of time mentioning the event) and using User-File (traverse the file and decide whether a user who mentioning the event is in the users result set). Since the size of the users result set may be quite large, processed with User-File may cost more than with User-File, the experiment result could prove this, so we use User-File to process this query. The detailed process is shown in Algorithm 4.

**Algorithm. 4. Query processing of Q9**

**For** a specific user id A.

1. Empty the user set.
2. Get A's followees and followees' followees using the algorithm described in 4.1, put them into the user set.
3. Traverse the User File of the event_tag_? in the given time span,
   **For** each user B
   **If** B in User Set, increase the count of B.
   **Else** continue
4. Sort the user set by the count with a quick-sort algorithm and return the top-K users.

There are three step when processing Q13: 1) find out A's followees. 2) find out events that user A mentioned and 3) find out the top-k users that mentioned the most number of events obtain in 2). We use the User-Event file to process step 2) and use the User-File to process step 3) because there may be so many users need to process. The process of this query is shown in Algorithm 5.

**Algorithm. 5. Query processing of Q13**

**For** a specific user id A.
1. Empty the user set.
2. Get A's followee using the algorithm 3.
3. Get events that user A mentioned using the User-Event file index.
4. **For** each of the event T obtain in 3
   Traverse User File of the eventtagT in the given timespan,
   **For** each user B
   **If** B is not A's followee,
   **If** B in user set, increase the count of B.
   **Else** put C into user set, set count to be 1.
5. Sort the user set by the count with a quick-sort algorithm and return the top-K users.

Since Q14 only acquire the number of being mentioned in microblogs, We just traverse the Time-User file of each event and calculate the number of lines in the given time span. Since the User file is quite not large, this approach could obtain an acceptable result.

Since Q15 only relates to one event but may relate to large amount of users, we use the User file of the event and could also obtain an acceptable result.

There are two step when processing Q19: 1) Find out A's followees and followeess' followees. 2) For each of the user B obtains in 1), get the events that B mentions, and calculate the count of each event being mentioned.

### 4.3 Retweet-based & Mention-based Queries

Since the there is a high similarity of the two group of queries, we just take the retweet-based queries as an example to illustrate those two group of queries.

Since the user set of Q10 is quite large, we could not use the user index to process this query, so we use the user-retweeted source file instead. We have calculated the count of a user appeared in the file beforehand, as we describe in 3.2, so we just read the file of corresponding date and complete the query. The detailed process is shown in Algorithm 6.

**Algorithm. 6. Query processing of Q10**

1. Empty the user set.
2. **If** time span=h or time span=d
   Read the source file.
   **Else**
   **If** date = first day or date = last day
   Read the source file.
   **Else** Read the count file.
3. Read the File obtain from 2 by line,
   **For** each user A
   **If** A in User Set, increase the count of A.
   **Else** put A into user set, set count to be one.
4. Sort the user set by the count with a quick-sort algorithm and return the top-K users.

For Q11, since we've created the User-Mentioned file and corresponding index, we just querying the user index in the given time span and obtain the users mentioning A.

Q12 is similar to Q10, the only different lies in that we need to obtain user A's followees and followees' followees using algorithm 2, and decide whether a user obtained from the source file is among those users. We do not use user-retweeted index because a user A's followees and followees' followees set may be quite large, it's time-consuming to process such a large user set.

For Q16, since the size of a user's followees set is not very large, we could use the User-Mention index to complete this query. The detailed process is shown in Algorithm 7.

**Algorithm. 7. Query processing of Q16**

**For** a specific user id A.

1. Empty the user set.
2. Get A's followees using the algorithm described in 4.1.
3. **For** each of A's followee B,
   Get the users that B retweets using the User-Mention index.
   **For** each user C obtained above
   **If** C in User Set, increase the count of C.
   **Else** put C into user set, set count to be one.
4. Sort the user set by the count with a quick-sort algorithm and return the top-K users.

### 4.4 All Microblog Queries

Since there's only one query in this group (**Q8**), we just traverse the light-weight file introduced in 3.2 from the latest date until the microblog count reaches the return count.

## 5. Experiment

To test the performance of privacy-preserved queries over microblog data, we use a real data set crawled from Sina Weibo (http://weibo.com), a popular microblogging service in China, via the API provided. In order to ensure privacy preservation, the dataset is preprocessed as follows:

(1) User IDs and message IDs are anonymized.

(2) Content of tweets are removed, based on Sina Weibo's Terms of Services.

(3) Some tweets are annotated with events. For each event, the terms that are used to identify the event and a link to Wikipedia (http://wikipedia.org) page containing descriptions to the event are given. Each event is identified with an event TAG.

A piece of preprocessed data is shown in Fig.3. The preprocessed dataset contains two sets of files:

(1) Tweets: It includes basic information about tweets (time, user ID, message ID etc.), mentions (user IDs appearing in tweets), re-tweet paths, and whether containing links.

(2) Followship network: It includes the following network of users (based on user IDs).

```
907334 time:2010-01-16 21:48:38|#|mid:510011007533653|#|uid:2257729460289142113 3|#|isContainLink:false
907335 time:2010-01-16 21:48:39|#|mid:510011007533682|#|uid:420100024494249|#|isContainLink:false
907336 time:2010-01-16 21:48:41|#|mid:510011007533621|#|uid:25343940102091820617|#|isContainLink:false
907337 time:2010-01-16 21:48:41|#|mid:510011007533696|#|uid:2100596701295809739029993|#|isContainLink:false
907338 time:2010-01-16 21:48:47|#|mid:510011007533669|#|uid:9478696163|#|isContainLink:false
907339 time:2010-01-16 21:48:50|#|mid:510011007533645|#|uid:9710328046|#|isContainLink:false
907340 time:2010-01-16 21:48:52|#|mid:510011007533305|#|uid:97266940112190194363|#|isContainLink:false
907341 time:2010-01-16 21:48:54|#|mid:510011007533355|#|uid:25343940102091820617|#|isContainLink:false
907342 time:2010-01-16 21:48:55|#|mid:510011007533353|#|uid:002018002002018002|#|isContainLink:false
907343 time:2010-01-16 21:49:00|#|mid:510011007533326|#|uid:286062543821893|#|isContainLink:false
907344 time:2010-01-16 21:49:00|#|mid:510011007533327|#|uid:25343940102091820617|#|isContainLink:false
907345 time:2010-01-16 21:49:01|#|mid:510011007533371|#|uid:2543896276221502584294828|#|isContainLink:false
907346 time:2010-01-16 21:49:04|#|mid:510011007533362|#|uid:05505521939493|#|isContainLink:false
907347 time:2010-01-16 21:49:05|#|mid:510011007533363|#|uid:294692992793883|#|isContainLink:false
907348 time:2010-01-16 21:49:17|#|mid:510011007533428|#|uid:050000000016012010|#|isContainLink:true
907349 time:2010-01-16 21:49:22|#|mid:510011007533432|#|uid:9785828953972852849396114209 29|#|isContainLink:false
907350 time:2010-01-16 21:49:36|#|mid:510011007534124|#|uid:971539715328768|#|isContainLink:false|#|rtTime:2009-11-
907351 time:2010-01-16 21:50:03|#|mid:510011007534040|#|uid:208439670896152931 95|#|isContainLink:false|#|eventList:
907352 time:2010-01-16 21:50:06|#|mid:510011007534512|#|uid:6701001301031012018|#|isContainLink:false
907353 time:2010-01-16 21:50:08|#|mid:510011007534558|#|uid:2339722586211339728529598|#|isContainLink:false
907354 time:2010-01-16 21:50:09|#|mid:510011007534584|#|uid:7201446001011463301200 1018|#|isContainLink:false
907355 time:2010-01-16 21:50:12|#|mid:510011007534568|#|uid:947282806197152|#|isContainLink:false
907356 time:2010-01-16 21:50:13|#|mid:510011007534573|#|uid:4401200905101200000 1018|#|isContainLink:false|#|eventLi
907357 time:2010-01-16 21:50:20|#|mid:510011007534805|#|uid:2873396882210062803722578962 98|#|isContainLink:false
907358 time:2010-01-16 21:50:33|#|mid:510011007534901|#|uid:4400946055050257689612 0|#|isContainLink:false
```

**Figure 3. Illustration of the Dataset after Preprocessed**

We evaluate the performance of our method on a machine with configuration of Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz and 4GB RAM. The maximum heap space of Java virtual machine is 1.5GB.

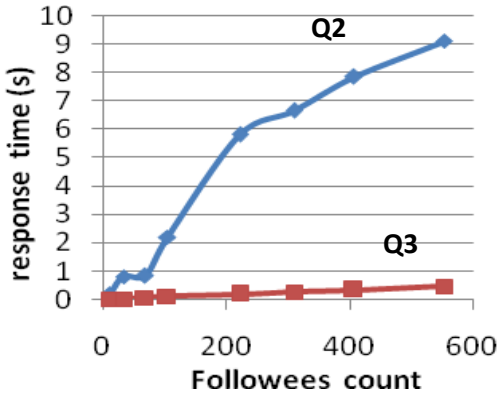


**Fig. 4. The Response Time of Q2 and Q3**

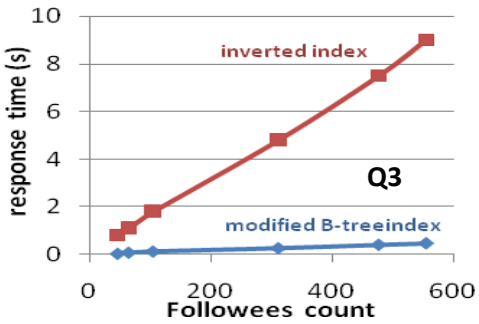


**Figure 5. The Response Time of Q3 with Different Indexes**

We evaluate our performance based on the response time. The user-based and event-based queries receive the best performance in our system, while the retweet-based queries show the worst response time in our experiment. Figure 4 shows the response time of the user-based queries. In Fig.4 we select some specific users based on their followees count and show the response time of Q2 and Q3 (since Q1, Q4, Q5 only need tens of millisecond to execute, we do not show the result here). We also compared two indexes: inverted index and modified B-tree index in Fig.5.

## 6. Conclusions

Microblogging service has been a hot topic in recent years. In this paper, we present a new approach to optimizing the privacy-preserved queries over microblog data, which is based query-specific preprocessing rules and index-based query algorithms. Our experiments on real microblog data crawled from Sina Weibo show that the proposed approach is helpful to improve the query performance of privacy-preserved queries.

## Acknowledgements

This work is supported by the National Science Foundation of Anhui Province (no. 1208085MG117), the National Science Foundation of China (no. 71273010), and the Soft Science Research Program of Anhui Province (grant no. 11020503056).

## References

[1] http://www.twitter.com.
[2] http://t.sina.com.cn.
[3] B. Rudolf and M. Edward, "Organization and Maintenance of Large Ordered Indexes", In Proc. Of SIGFIDET Workshop, **(1970)**, pp. 107-141.
[4] http://lucene.apache.org/.
[5] N. Chapin, "A Comparison of File Organization Techniques", Proc. ACM National Conference, **(1969)**, ACM press, pp. 273-286.