Jia *et al. Computational Social Networks* (2015) 2:5
DOI 10.1186/s40649-015-0014-7

Computational Social Networks
a SpringerOpen Journal

RESEARCH Open Access

Handling big data of online social networks on a small machine

Ming Jia*, Hualiang Xu, Jingwen Wang, Yiqi Bai, Benyuan Liu and Jie Wang

*Correspondence: mjia@cs.uml.edu
Department of Computer Science, University of Massachusetts, Lowell, MA 01854, USA

Abstract
Dealing with big data in computational social networks may require powerful machines, big storage, and high bandwidth, which may seem beyond the capacity of small labs. We demonstrate that researchers with limited resources may still be able to conduct big-data research by focusing on a specific type of data. In particular, we present a system called MPT (Microblog Processing Toolkit) for handling big volume of microblog posts with commodity computers, which can handle tens of millions of micro posts a day. MPT supports fast search on multiple keywords and returns statistical results. We describe in this paper the architecture of MPT for data collection and phrase search for returning search results with statistical analysis. We then present different indexing mechanisms and compare them on the microblog posts we collected from popular online social network sites in mainland China.

Keywords: Microblog posts; Phrase search; Nextword indexing; Lucene-based indexing; Mongo DB; Statistical analysis; Precision rate; Recall rate

Introduction

Dealing with big data in computational social networks may require big machines with big storage. This may seem that only large companies or organizations with lucrative budgets can afford big-data research. We show that, by focusing on a specific type of data, it is possible to carry out big-data research on online social networks (OSNs) using commodity computers in a small lab environment. In particular, we present a system for handling big volume of microblog posts (MBPs). Our system is called MPT, which stands for Microblog Processing Toolkit.

MPT collects MBPs from popular OSN sites in mainland China, identifies interesting topics from MBPs, and carries out statistical analysis on each topic. The statistical analysis includes gender and location distributions, frequent words, and trends. We have collected on average approximately 4.5 million (sometimes over 10 million) MBPs a day since September 2012. We stored these MBPs in Mongo DB running on a commodity computer.

To make use of these data, MPT is required to support, among other things, phrase search that will quickly return the set of MBPs that contain a set of words entered by the user and the statistical results of these posts displayed in various graphs. For this purpose, we would need to create an appropriate indexing mechanism and update the indexing content regularly. In addition, we also want to retrieve MBPs in real time while we are collecting them, so that we may detect unexpected social events and perform other tasks.

In this paper, we present three indexing methods deployed on commodity computers. Without using clusters of computers, we were able to build a system suitable for implementing a fast search engine and carrying out topic modeling and statistical analysis on large volume of MBPs.

The rest of the paper is organized as follows: In the 'Data collection' section, we will describe the data source and the API (Application Programming Interface) we used to collect MBPs. In the 'Database' section, we will introduce the database we use to store the data and describe some of the problems we encountered when storing MBPs. In the 'Data retrieval' section, we will describe a number of indexing mechanisms, including the default Mongo DB queries using regular expressions, our own implementation of the nextword indexing [1], and an indexing system we built based on Lucene [2]. In particular, we will describe the structures of the systems for indexing, searching, and carrying out statistical analysis. In the 'Statistical analysis' section, we will compare the speed of querying, the speed of performing statistical analysis, and the accuracy of each method on real data sets. We conclude the paper in the 'Experiments' section.

Data collection

We have collected MBPs continuously since September 2012 from popular OSN sites in mainland China, including Sina, Tencent, and Renren. Both Sina and Tencent provide open API to developers. They provided different APIs for different purposes. Since we were focusing on discovering topics from daily MBPs with statistical analysis, rather than on a specific user or a specific topic, we used the public-timeline interface [3] to collect MBPs (see Figure 1 for an example of such interface).

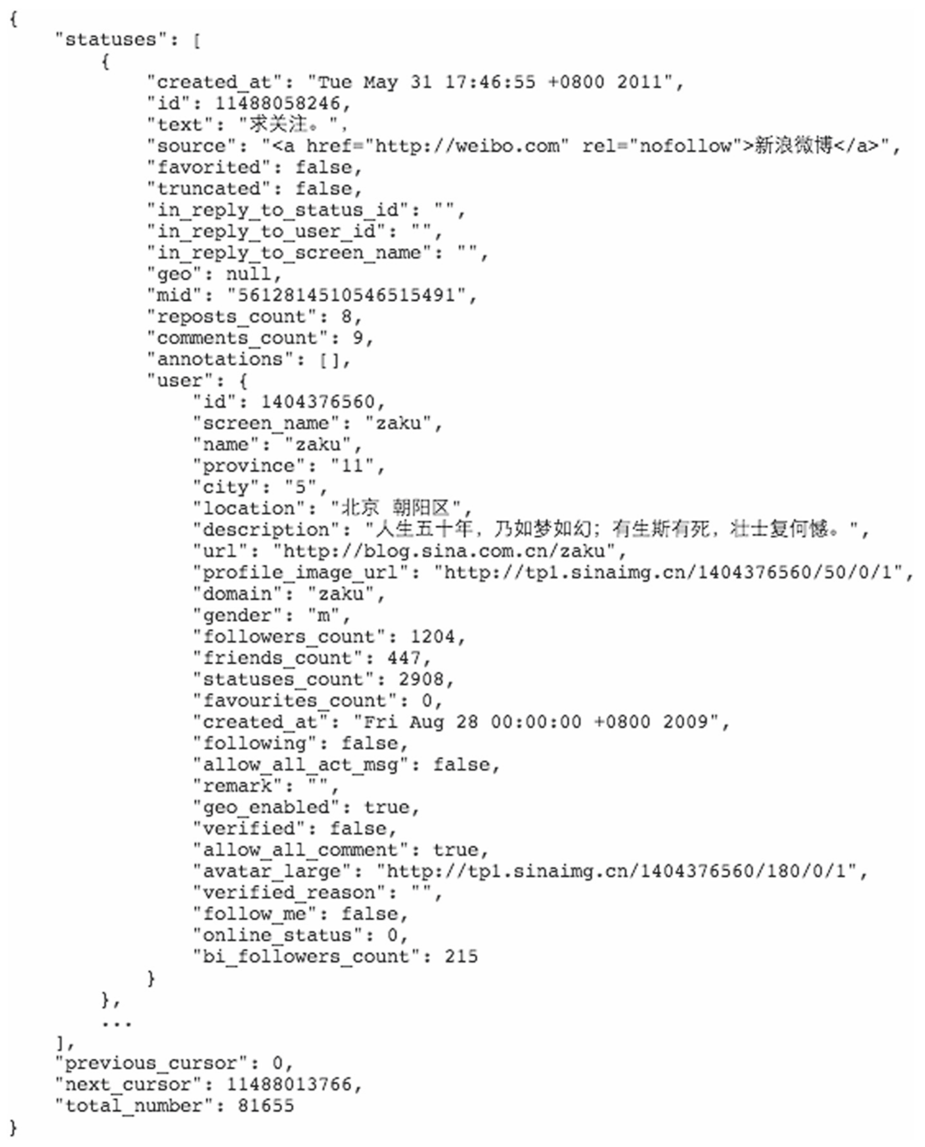
Under the restriction of the user privilege given to us, we collected on average about 4.5 million (sometimes over 10 million) MBPs a day from these OSN sites. These MBPs were semi-structured JSON style records.

Database

To handle unstructured MBPs in large quantity, we would need a high-performance, steady, and flexible database system. Because MBPs are unstructured, we chose Mongo DB [4] for this purpose, which is a common choice for storing unstructured data.

Different MBPs from different sources use different data structures. With Mongo DB, we can store data in different data structures in the same collection. This makes it convenient to manage the data. Moreover, Mongo DB is a database scheme with high performance on the operations of both read and write, which meets our need of intensive writing and querying MBPs.

Initially, we stored in one collection the MBPs posted on the same date and stored all the collections in the same database. After running the system for a few weeks, we experienced unexpected system crashes. The reason was that Mongo DB would write data in the same database into the same file and load all the files that are accessed frequently into the main memory. As more MBPs were stored in the same database, the file grew larger quickly, causing Mongo DB to consume almost all the RAM and crashing the system. To solve this problem, we divide the collections of MBPs according to a fixed time interval of 1 week into different databases. Because writing to the database was the main operation



```
{
    "statuses": [
        {
            "created_at": "Tue May 31 17:46:55 +0800 2011",
            "id": 11488058246,
            "text": "求关注。",
            "source": "<a href="http://weibo.com" rel="nofollow">新浪微博</a>",
            "favorited": false,
            "truncated": false,
            "in_reply_to_status_id": "",
            "in_reply_to_user_id": "",
            "in_reply_to_screen_name": "",
            "geo": null,
            "mid": "5612814510546515491",
            "reposts_count": 8,
            "comments_count": 9,
            "annotations": [],
            "user": {
                "id": 1404376560,
                "screen_name": "zaku",
                "name": "zaku",
                "province": "11",
                "city": "5",
                "location": "北京 朝阳区",
                "description": "人生五十年，乃如梦如幻；有生斯有死，壮士复何憾。",
                "url": "http://blog.sina.com.cn/zaku",
                "profile_image_url": "http://tp1.sinaimg.cn/1404376560/50/0/1",
                "domain": "zaku",
                "gender": "m",
                "followers_count": 1204,
                "friends_count": 447,
                "statuses_count": 2908,
                "favourites_count": 0,
                "created_at": "Fri Aug 28 00:00:00 +0800 2009",
                "following": false,
                "allow_all_act_msg": false,
                "remark": "",
                "geo_enabled": true,
                "verified": false,
                "allow_all_comment": true,
                "avatar_large": "http://tp1.sinaimg.cn/1404376560/180/0/1",
                "verified_reason": "",
                "follow_me": false,
                "online_status": 0,
                "bi_followers_count": 215
            }
        },
        ...
    ],
    "previous_cursor": 0,
    "next_cursor": 11488013766,
    "total_number": 81655
}
```

Figure 1 JSON example of public-timeline API.

of the system and the system only collected real-time data, Mongo DB would load the file of the most recent week into RAM and thus consume much less RAM than before. The system has never crashed after we made this change.

We note that we may also consider using an SQL-based database for reliability.

Data retrieval

Our Microblog Processing Toolkit performs the following three types of search:

1. Given a keyword, retrieve all MBPs that contain the keyword.
2. Given a set of keywords, retrieve all MBPs that contain at least one of these keywords (this is the logical OR operation).

3. Given a set of keywords, retrieve all MBPs that contain all of the keywords (this is the logical AND operation).

We approached these tasks using the following three methods.

Mongo DB regular expressions

Mongo DB provides a built-in regular expression searching method. Given the regular expression we want the text to match, Mongo DB returns all the records that match the regular expression. Then, the system will go over all the MBPs and count MBPs for each statistical feature we are interested in, such as gender, location, and sentiment. This search method, however, is inefficient and does not meet our needs. To speed up the search process, we developed an indexing system based on the nextword indexing scheme [5].

Nextword indexing

The nextword index consists of a vocabulary of distinct words and, for each word w in the vocabulary, a nextword list and the position list. The nextword list consists of each word s that succeeds w in an MBP in the database. For each triple of (w, s, M), where M is an MBP that contains the phrase ws, the position list consists of the list of positions where the phrase ws appears in the MBP M and a pointer that links (w, s) to M. We also store in the position list the necessary information of each statistical feature. This structure will be used in the statistical analysis component of the system. Figure 2 depicts an example of the nextword structure of MBPs with the gender value, where 'female' in a triple (w, s, M) means that M was posted by a female user.

Based on the nextword indexing, we built a system consisting of two parts: (1) an index server and (2) a search server. The index server indexes in real time the collected MBPs. In particular, we store the word pair list in the main memory and the information of each

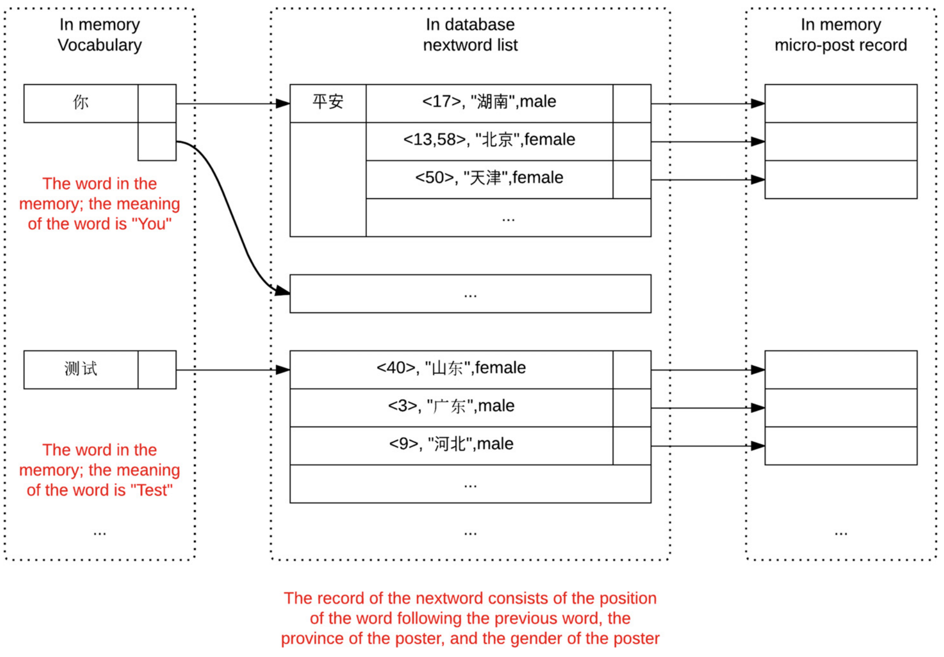

In memory Vocabulary
In database nextword list
In memory micro-post record
你
平安
<17>, "湖南",male
<13,58>, "北京",female
<50>, "天津",female
...
The word in the memory; the meaning of the word is "You"
测试
<40>, "山东",female
<3>, "广东",male
<9>, "河北",male
The word in the memory; the meaning of the word is "Test"
The record of the nextword consists of the position of the word following the previous word, the province of the poster, and the gender of the poster

Figure 2 Nextword index structure.

word pair in the database. This list contains the position of the word pair in the text and the information of the MBPs. The position of the word pair (w, s) in M is the same as the position of w in M.

Our early version of the system stored each word pair in RAM. We observed that, for less frequent words w, if we only stored the word-pair position of w in RAM but not the word pairs (w, s), then the search speed would only be mildly affected, while the consumption of RAM would drop tremendously. Because retrieving a word pair (w, s) not stored in RAM will incur more time, to balance between the search speed and the RAM occupancy by the nextword indexing, we set a threshold value on word-pair counts, so that the system only stores in RAM the nextword list of words with frequencies over the threshold. This measure cuts down the RAM usage significantly.

After several weeks of running the system, we encountered another problem of data explosion: The number of MBPs provided through the APIs suddenly increased significantly, more than twice the size of the data we normally collected in 1 day. Likewise, the index size was also increased to occupy about 5 GB of RAM. Since we deployed the system on a commodity computer, we could not afford such RAM usage. This motivated us to devise a low-cost RAM solution. We accomplished this using Apache Lucene.

Lucene-based indexing

Apache Lucene is a library for a high-performance, full-featured text search engine, which performs indexing and searching using small RAM, and generates an index file with reasonable size. We customized the Lucene core and built a data retrieval system that solved the problem of memory blowup. Our system consisted of two parts: (1) a real-time index server and (2) a search server. The real-time index server refreshes the database frequently and indexes dynamically the newly collected MBPs on the fly. The search server returns the MBPs that contain all the keywords entered by the user. These two parts may work simultaneously without conflicting each other, and so the search server can return real-time posts the system collects. Figure 3 shows the structure of our Lucene-based indexing structure.

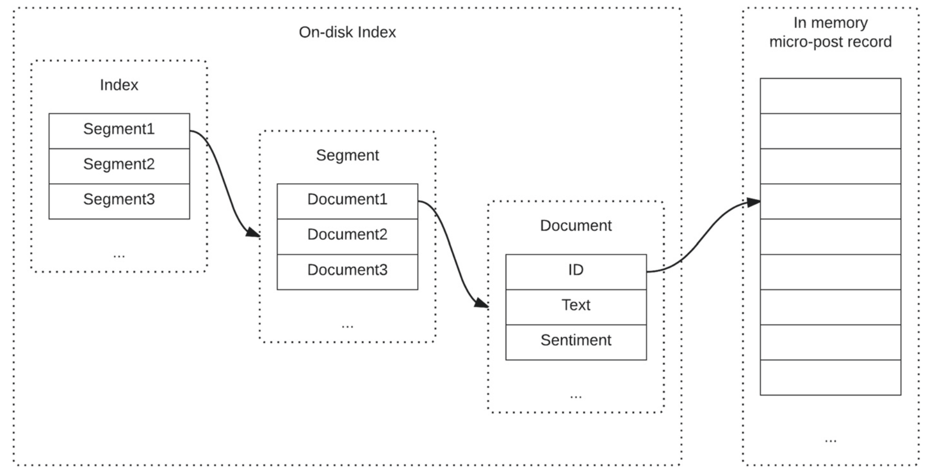


Figure 3 Lucene-based index structure.

For convenience, we will call a person who authors or retransmits an MBP a sender. For each MBP, we will include in one string its sender's location and gender, the posting time, and the OSN this MBP is transmitting within. This string is used as the indexing key of the MBP.

Statistical analysis

In addition to returning the MBPs, the Microblog Processing Toolkit also returns a number of statistical results of these MBPs, including the proportion of genders, the provinces of the senders, the sources of the MBPs, and the distribution of the posting time. Moreover, hot words (i.e., words with high frequencies) are generated while returning the MBPs.

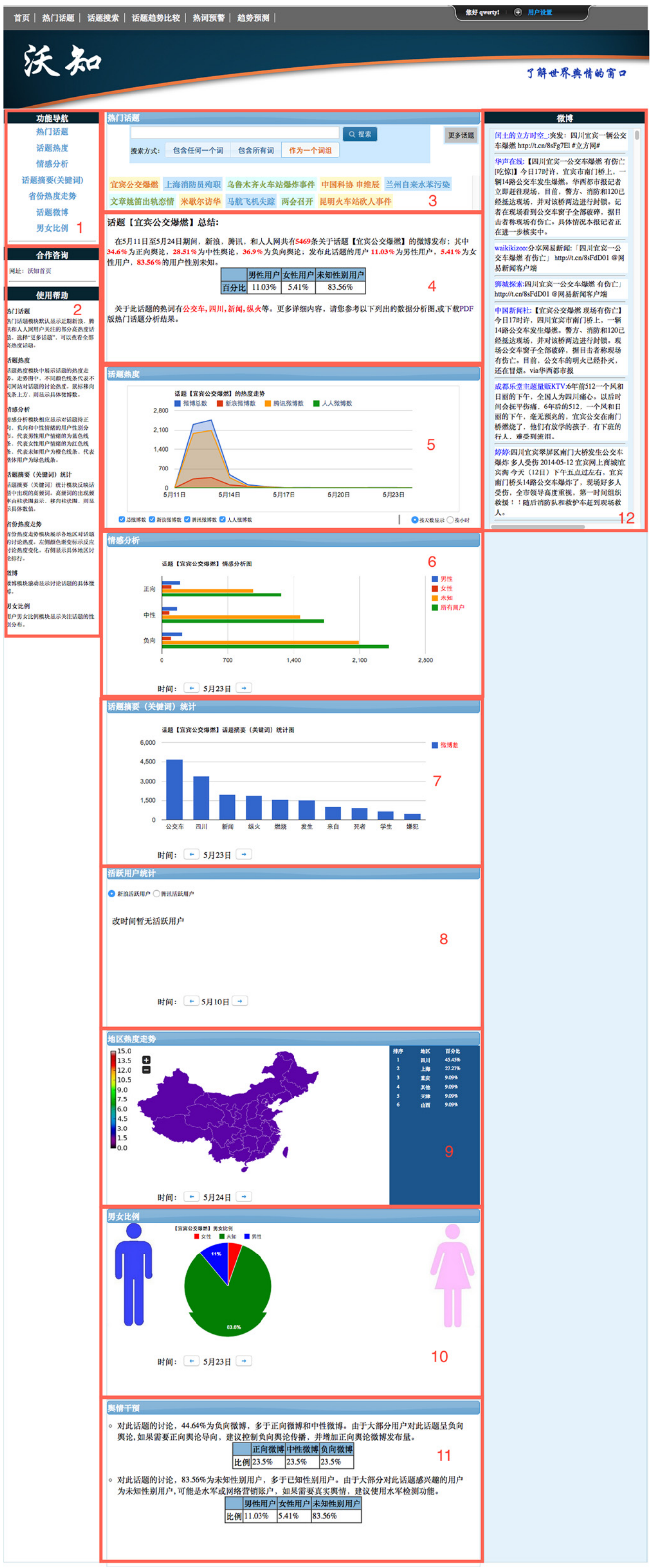
To carry out statistical analysis on the MBPs we have collected, we group them according to the following three attributes: 1) posting time, 2) sender's gender, and 3) sender's location.

Using the built-in regular expressions of Mongo DB, we would only need to traverse all the MBPs the database returned and then carry out the statistical analysis. This process, however, is extremely time consuming.

Using the nextword indexing, we could complete the statistical analysis in just a few steps. Since the nextword index file contains the position of each word pair and the required statistical features, the system can carry out statistical analysis by traversing the position list for the given input phrase to be searched for without querying the entire database.

We need to customize our own statistical features. As mentioned earlier, for each MBP, we include in the index its sender's location, gender, and posting time in one string. We set this string as the key of the MBP. This means that for two MBPs posted during the same hour, at the same location, and by the senders of the same gender, they will share the same key. When indexing the MBP, we include this key in the index. When searching for the data, we first carry out the group search by the key. After obtaining the groups with the same key, we traverse each group and perform the statistical analysis in the group (details of statistical analysis are not included in this paper).

We constructed a website [6] to perform opinion analysis using our MPT toolkit. For example, Figure 4 is a snapshot of the website after a keyword (or a keyword phrase) is entered to be searched. As marked in the graph, Part 1 is a navigation bar of the page, Part 2 is an introduction of the website, Part 3 shows recent hot topics, Part 4 shows a short summary of the statistical result which includes the gender distribution and sentiment distribution of the posts, Part 5 shows the trending of the topic, Part 6 shows the sentiment-gender distribution, Part 7 is a frequent relevant-keyword bar graph, Part 8 shows active users, Part 9 is a map of China that shows the location distribution of the MBPs containing the input keyword or keyword phrase, Part 10 is a pie chart of gender distribution, Part 11 shows statistical analysis detecting water-army, and Part 12 shows a list of hot posts. Figure 5 shows some of the statistical results, including a trending graph, a frequent relevant-keyword bar graph, a frequent-sender bar graph, a pie chart of gender distribution, and a map of China that shows the location distribution of the MBPs containing the input keyword or keyword phrase. The trending graph shows n lines, on a given set of n input keywords $w_1, w_2, \cdots, w_n$, such that the ith line displays the number of the MBPs containing keyword w_i at a given time from a starting time to an ending time.


首页 | 热门话题 | 话题搜索 | 话题趋势比较 | 热词预警 | 趋势预测 |
您好 qwerty! 用户设置
沃知
了解世界舆情的窗口
功能导航
热门话题
话题热度
情感分析
话题摘要(关键词)
省份热度走势
话题微博
男女比例
1
合作咨询
网址：沃知首页
使用帮助
2
热门话题
搜索
更多话题
搜索方式：包含任何一个词 包含所有词 作为一个词组
宜宾公交爆燃 上海消防员殉职 乌鲁木齐火车站爆炸事件 中国科协 申维辰 兰州自来水苯污染
文章姚笛出轨恋情 米歇尔访华 马航飞机失踪 两会召开 昆明火车站砍人事件
3
4
话题热度
话题【宜宾公交爆燃】的热度走势
微博总数 新浪微博数 腾讯微博数 人人微博数
5
情感分析
话题【宜宾公交爆燃】情感分析图
正向
中性
负向
男性 女性 未知 所有用户
6
时间：5月23日
话题摘要（关键词）统计
话题【宜宾公交爆燃】话题摘要（关键词）统计图
公交车 四川 新闻 纵火 微信 发生 来自 死者 学生 通知
微博数
7
时间：5月23日
活跃用户统计
新浪活跃用户 腾讯活跃用户
8
时间：5月10日
地区热度走势
9
时间：5月24日
男女比例
【宜宾公交爆燃】男女比例
女性 未知 男性
11%
83.6%
10
时间：5月23日
舆情干预
11
微博
12

话题【宜宾公交爆燃】总结：

在5月11日至5月24日期间，新浪，腾讯，和人人网共有5469条关于话题【宜宾公交爆燃】的微博发布；其中34.6%为正向舆论，28.51%为中性舆论，36.9%为负向舆论；发布此话题的用户 11.03%为男性用户，5.41%为女性用户，83.56%的用户性别未知。

	男性用户	女性用户	未知性别用户
百分比	11.03%	5.41%	83.56%

关于此话题的热词有公交车，四川，新闻，纵火等。更多详细内容，请您参考以下列出的数据分析图,或下载PDF版热门话题分析结果。

改时间暂无活跃用户

- 对此话题的讨论，44.64%为负向微博，多于正向微博和中性微博。由于大部分用户对此话题呈负向舆论，如果需要正向舆论导向，建议控制负向舆论传播，并增加正向舆论微博发布量。

	正向微博	中性微博	负向微博
比例	23.5%	23.5%	23.5%

- 对此话题的讨论，83.56%为未知性别用户，多于已知性别用户。由于大部分对此话题感兴趣的用户为未知性别用户，可能是水军或网络营销账户，如果需要真实舆情，建议使用水军检测功能。

	男性用户	女性用户	未知性别用户
比例	11.03%	5.41%	83.56%

闰土的立方时空_:突发：四川宜宾一辆公交车爆燃 http://t.cn/8sFg7El #立方网#

华声在线:【四川宜宾一公交车爆燃 有伤亡[吃惊]】今日17时许，宜宾市南门桥上，一辆14路公交车发生爆燃。华西都市报记者立即赶往现场，目前，警方、消防和120已经抵达现场，并对该桥两边进行封锁。记者在现场看到公交车窗子全部破碎，据目击者称现场有伤亡。具体情况本报记者正在进一步核实中。

waikikizoo:分享网易新闻:「四川宜宾一公交车爆燃 有伤亡」 http://t.cn/8sFdD01 @网易新闻客户端

狮城探索:四川宜宾一公交车爆燃 有伤亡」 http://t.cn/8sFdD01 @网易新闻客户端

中国新闻社:【宜宾公交爆燃 现场有伤亡】今日17时许，四川宜宾市南门桥上，一辆14路公交车发生爆燃。警方、消防和120已经抵达现场，并对该桥两边进行封锁。现场公交车窗子全部破碎，据目击者称现场有伤亡。目前，公交车的明火已经扑灭，还在冒烟。via华西都市报

成都乐堂主题量贩KTV:6年前512一个风和日丽的下午，全国人为四川痛心。以后时间会抚平伤痛，6年后的512，一个风和日丽的下午，毫无预兆的，宜宾公交在南门桥燃烧了，他们有放学的孩子，有下班的行人，难受到流泪。

婷婷:四川宜宾翠屏区南门大桥发生公交车爆炸 多人受伤 2014-05-12 宜宾网上商城|宜宾淘 今天（12日）下午五点过左右，宜宾南门桥头14路公交车爆炸了，现场好多人受伤，全市领导高度重视，第一时间组织救援！！随后消防队和救护车赶到现场救人。

Figure 4 A website built with our system.

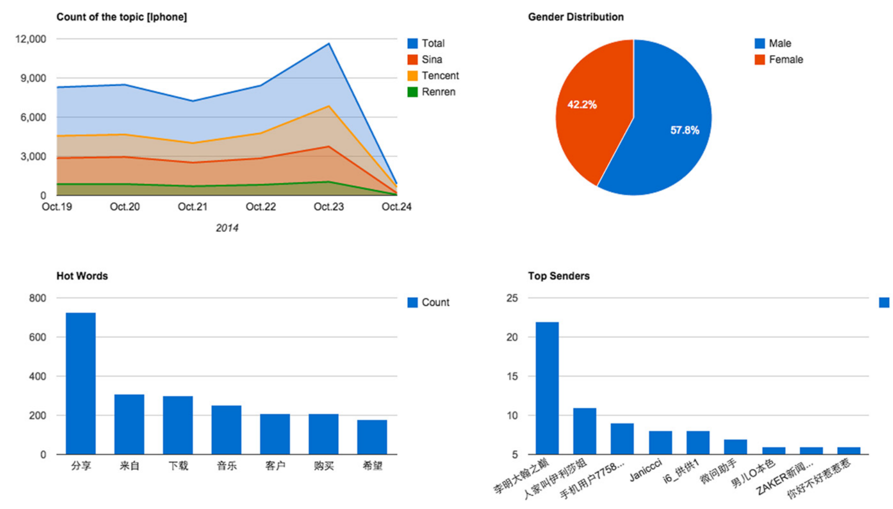


Figure 5 Sample of statistical analysis result.

Let *w* be the keyword or a keyword phrase to be searched. The frequent relevant-keyword bar graph shows the ten most frequent keywords (not equal to *w*) in the MBPs that contain *w*. These keywords are called hot words (with respect to *w*). The frequent-sender bar graph shows the ten most active senders who post the most MBPs containing *w*. These senders are referred to as top senders.

The trending graph, gender distribution, location distribution, and source OSNs can be easily calculated by a simple traversal of the the MBPs in each group. However, the hot words and the top senders cannot be obtained this way since each MBP may have its unique top tf-idf words. Instead, MPT traverses all the MBPs contained in each group and adds all the keywords and senders to two different maps. It then analyzes these two maps to obtain the list of hot words and the list of top senders.

Experiments

We designed an experiment to test the efficiency and accuracy of the three methods for data retrieval and statistical analysis mentioned in the previous sections. For convenience, we will refer to these methods of Mongo DB's built-in regular expressions, nextword indexing, and Lucene-based indexing as, respectively, Mongo, Nextword, and Lucene. We used the MBPs we collected in 1 day (the day was randomly chosen) from Sina and Tencent as the data set for comparing the three different methods. There were about 4.3 million MBPs in this 1-day collection. All experiments were executed on a commodity computer with a quad-core CPU and 16-GB RAM.

In practice, the system is designed to handle the posts of multiple days with volume much larger than the data used in the experiment. For a dataset of very large scale, our system will slice the dataset into multiple subsets. In particular, for a dataset that contains posts in multiple days, the system will slice the dataset into multiple subsets such that each subset contains the posts of the same day. Thus, our experiment does indicate the fundamental performance. We repeated the experiment on datasets from different dates with similar results.

The experiment consisted of two parts. In the first part, we examined how fast each method responded to user queries, as well as how fast the method carried out statistical analysis and returned the results. In the second part, we compared the accuracy of each method.

To run the experiment, we pre-counted all the keyword phrases in the data set and randomly selected a number of keyword phrases as our testing phrases, such that these phrases were made up of two or three keywords and appeared in the test data set for more than 100 times.

For each phrase we selected, we executed a query with each method. We recorded the time that each method incurred to respond to the query and finished up the statistical analysis. We tested Mongo, Nextword, and Lucene separately. For each test, we repeated the process twice and calculated the average time. The results are shown in Figures 6 and 7.

First, we compared the responding time of each method. The horizontal axis represents the actual frequency of keywords, and the vertical axis represents the running time of returning the MBPs that contain the keywords.

From Figure 6 , we can conclude the following:

1. Lucene offers the fastest responding time, and the responding time will increase as the phrase frequency increases.
2. Nextword is faster, which is slower but close to Lucene, but its time complexity is not as steady as Lucene.
3. Mongo is the slowest, which does not meet the real-time search requirement.

We then compare the speed of carrying out statistical analysis for each method. From Figure 7 , we can see that Lucene is still the fastest, Nextword is slower than Lucene, while Mongo is substantially slower. The time interval between responding to the search query

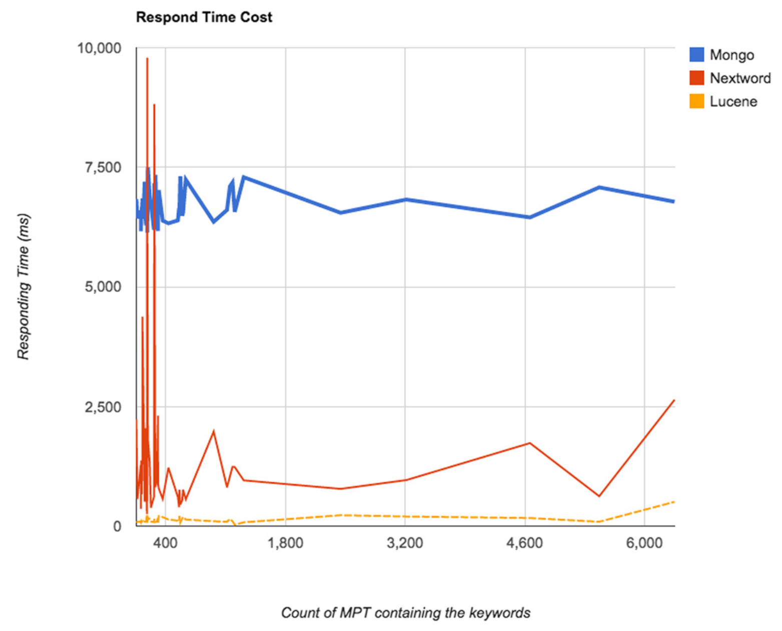


Figure 6 Responding time.

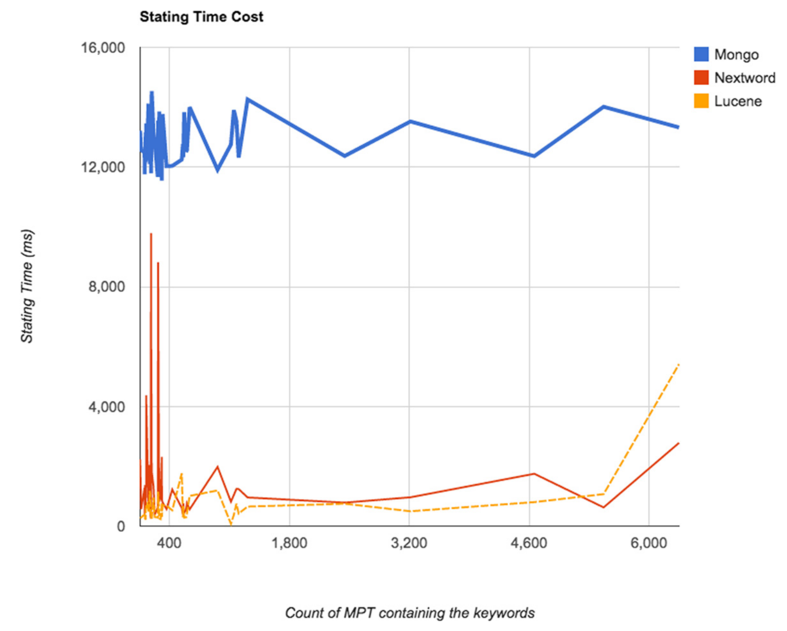


Figure 7 Statistical analysis time.

and finishing up statistical analysis is quite short for Lucene and is much longer for the other two methods. We note that the mechanism for carrying out the statistical analysis of each method is different. By querying Mongo, we would need to traverse all the MBPs returned on the query, which would incur significant computing time when the returning data set is large. The nextword indexing is a well-structured indexing mechanism, where no traverse is needed to perform statistical analysis, and so it can finish the statistical analysis quickly. Lucene finishes statistical analysis by traversing all the groups instead of all the posts.

We can see from Figure 7 that the time complexity of Mongo search does not increase much when the frequency of the keywords is increased, while the time complexity of Nextword search and the Lucene search is clearly related to the frequency of the keywords.

Finally, we compared the accuracy of each method (see Figure 8). The horizontal axis represents the actual frequency of the phrase, and the vertical axis represents the counts of the returning results.

From Figure 8, we can see that the accuracy of each method differs from each other, where Mongo is 100% accurate. In other words, its precision and recall rates are both equal to 1.

Nextword may miss some MBPs. The main reason of missing MBPs is due to the segmentation error of the keywords in the Chinese language. The keyword segmentation in the Chinese language is different from that of English, for the standard Chinese writing contains no space between characters. Thus, different segmentation tools may return different segmentation results. Even for the same keyword using the same segmentation tool, the results may still be different with different text. For the keywords we queried in the experiments, the segmentation result in the MBPs could differ from that in the search query. The MBPs with different segmentation results would be missed.

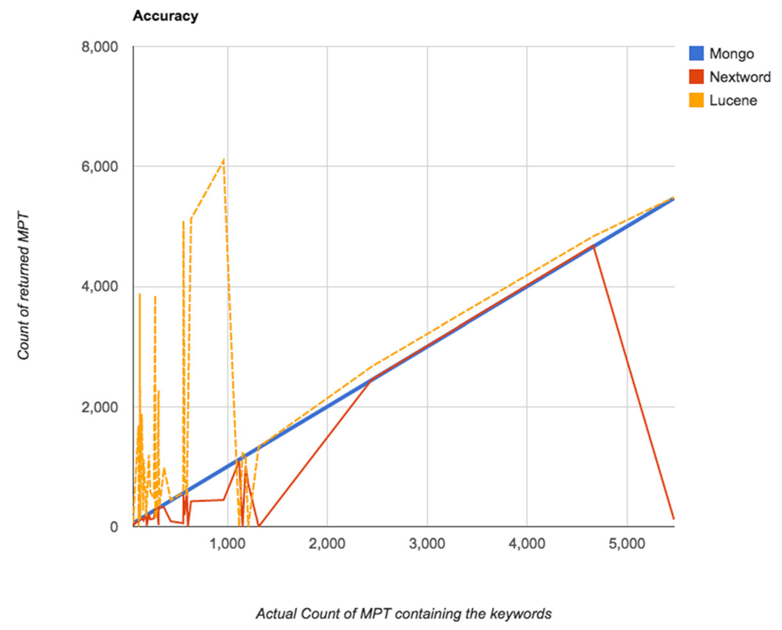


Figure 8 Accuracy.

Lucene, on the other hand, seems to have the worst precision and recall rates, where the number of returned MBPs is usually larger than the actual number that contains the phrase. This is caused by the Lucene indexing structure, where all the MBPs that contain a subset of the search keywords are returned. For example, if phrase X is made up of words A *and* B, when querying X, Lucene will return all the posts that contain A *or* B. So, the returning result usually has a larger-than-actual count.

Table 1 shows the precision and recall rates of the search results using the three different methods. We can see from the table that Mongo performs well on accuracy. Both Mongo and Nextword have a value of 1 on the precision rate, which means that the MBPs Mongo returned were exactly those that contain the search phrase. Since Nextword may miss some MBPs, this affected negatively its recall value. Lucene suffers precision loss compared to the other two methods. But it offers better performance in the recall value than Nextword, which means that it may miss fewer MBPs than Nextword.

From our experiment, we can see that each method has its pros and cons. In particular, Mongo provides the regular expression searching method with the perfect accuracy. But it is too slow to meet the needs of real-time querying and stat analysis. Nextword has the best performance in statistical analysis, and the responding time can meet the need of real-time search. But its memory consumption is high, which cannot meet the increasing data requirement. Lucene, on the other hand, has the fastest responding time and the fastest statistical analysis time, but it incurs low accuracy. This method is good for building a fast search engine but may not meet the requirement of high accuracy.

Table 1 Experimental results on precision and recall rates

Method	Precision	Recall
Mongo	1	1
Nextword	1	0.72
Lucene	0.53	0.81

We note that the statistical result returned by the system is in a tree-structured JSON format, which contains the information we are interested in. Such information is straightforward to obtain with little extra time. For example, for MBPs of a particular topic, it is easy to obtain from the statistical result the number of postings by users of a particular gender throughout a particular period of time of the day.

Conclusion

In this paper, we described and compared three methods we used to analyze big volume of microblog posts. We conclude the paper by summarizing our findings as follows:

1. Mongo is good for storing data but not suitable for carrying out search on big data.
2. Nextword offers good performance on real-time search with fast response time and statistical analysis time. But it would take up too much RAM. Nextword would be a good choice for analyzing moderate-size data (e.g., less than 3 million MBPs) with high requirement on statistical analysis.
3. Lucene is a low RAM-consumption and stable system. But it has the worst performance on precision and recall. For those who need to analyze big volume of data but do not require exact statistical results, Lucene would be a better choice.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
This project was directed by JW and co-directed by BL. MJ and JW designed the indexing mechanisms and drafted the manuscript. MJ implemented the indexing mechanisms and carried out numerical comparisons. HX and YB wrote and maintained the program to collect microblog posts. JWW developed the website and implemented the graphs. All authors read and approved the final manuscript.

Authors' contributions
JW is Professor and Chair at the Department of Computer Science at the University of Massachusetts Lowell (UMass Lowell), who received his PhD in Computer Science from Boston University in 1991. BL is Associate Professor at the Department of Computer Science at UMass Lowell, who received his PhD in Computer Science from the University of Massachusetts Amherst in 2003. MJ, JWW, and YB are PhD candidates at the Department of Computer Science at UMass Lowell under the supervision of JW. HX is Senior Software Engineer at EMC, who received his MS degree in Computer Science at UMass Lowell under the supervision of JW.

Acknowledgements
The authors were supported in part by the NSF under grants CNS-0953620, CNS-1018422, and CNS-1247875. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. The authors thank Peng Xia and Liqun Shao for participating in various discussions of the project.

Received: 2 November 2014 Accepted: 24 February 2015
Published online: 14 March 2015

References
1. Williams, HE, Zobel, J, Anderson, P: What's next? Index structures for efficient phrase querying. In: Australasian Database Conference, pp. 141–152, Auckland, New Zealand, (18-21/1/1999)
2. The Apache Software Foundation: Apache Lucene (2/1/2015). https://lucene.apache.org/
3. Sina corporation: Open Documentfor Sina Micro-blog API (2/1/2015). http://open.weibo.com/wiki/Statuses/public_timeline/en
4. MongoDB, Inc: MongoDB (2/1/2015). http://www.mongodb.org/
5. Bahle, D, Williams, H, Zobel, J: Compaction techniques for nextword indexes. In: International Symposium on String Processing and Information Retrieval, pp. 0033–0033, Laguna de San Rafael, Chile, (13-15/11/2001)
6. Wantology Corporation: Wocson (2/1/2015). http://www.wocson.net/