Chapter 13
MBCrawler: A Software Architecture for Micro-Blog Crawler

Gang Lu, Shumei Liu and Kevin Lü

Abstract Getting data is the precondition of researching on micro-blogging services. By using Web 2.0 techniques such as AJAX, the contents of micro-blog Web pages are dynamically generated rapidly. That makes it hard for traditional Web page crawler to crawl micro-blog Web pages. Micro-blogging services provide some APIs. Through the APIs, well-structured data can be easily obtained. A software architecture for micro-blogging service crawler, which is named as MBCrawler, is designed basing on the APIs provided by micro-blogging services. The architecture is modular and scalable, so it can fit specific features of different micro-blogging services. SinaMBCrawler, which is a crawler application based on MBCrawler for Sina Weibo, has been developed. It automatically invokes the APIs of Sina Weibo to crawl data. The crawled data is saved into local database.

Keywords Social Computing · Micro-blog · Crawler · Twitter

13.1 Introduction

As a fast developing and widely used new Web application, micro-blogging service such as Twitter and Sina Weibo, has attracted attention of users, enterprises, governments, and researchers. The first issue comes to researchers is

G. Lu (✉) · S. Liu
College of Information Science and Technology, Beijing University of Chemical Technology, 15 BeiSanhuan East Road, ChaoYang District 100029 Beijing, China
e-mail: sizheng@126.com

K. Lü
Brunel University, Uxbridge UB8 3PH, UK
e-mail: kevin.lu@brunel.ac.uk

W. Lu et al. (eds.), *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*, Lecture Notes in Electrical Engineering 212,
DOI: 10.1007/978-3-642-34531-9_13,

getting data of them. Generally, because of privacy and business reasons, micro-blogging service providers will not provide the data readily.

In 2001, the authors of [1] described the common architecture of a search engine, including the issues about selecting and updating pages, storing, scalability, indexing, ranking, and so on. Nevertheless, being different from traditional Web pages, Web 2.0 techniques such as AJAX (Asynchronous JavaScript and XML) are widely used in micro-blog Web pages, and the contents in micro-blog Web pages change too rapidly and dynamically for web crawlers. Traditional crawlers for static Web pages do not work well to them. Of course, there has been some research on getting web content from AJAX based Web pages [2–6]. However, all of the work is based on the state of the application, and the technique is not that easy to implement. Fortunately, to encourage developers to develop applications about micro-blogging services to make it used as widely as possible, the providers of micro-blogging services publish some APIs. By those APIs, well-formatted data of micro-blogging services can be obtained. Except some work in which Twitter APIs were not used [7], and the work in which the authors did not state how they got the Twitter data [8, 9], most of existing research about Twitter utilizes the provided APIs [10–15].

We can see that Twitter APIs are widely used in the research work on Twitter. We believe that using APIs is the most popular way to get data from micro-blogging services. That provides us the probability of constructing uniform and universal software architecture to utilize the provided APIs, to automatically download and save well-structured data into database. To make it convenient for researchers to obtain data from micro-blogging services, a software architecture named as MBCrawler is proposed. Basing on this software architecture, a crawler using APIs of micro-blogging service with multi threads can be developed. More functions can be added easily along with more APIs are added, and details can be designed to fit different online social network services.

13.2 MBCrawler

13.2.1 Basic Structure of MBCrawler

Software architecture named as MBCrawler is proposed. This software architecture presents a main framework by which crawlers for micro-blogging services data can be easily designed, developed, and expanded. MBCrawler is designed as multi-threaded, and consists of six components, which are UI, Robots, Data Crawler, Models, Micro-blogging APIs, and Database. The structure of MBCrawler is illustrated in Fig. 13.1.

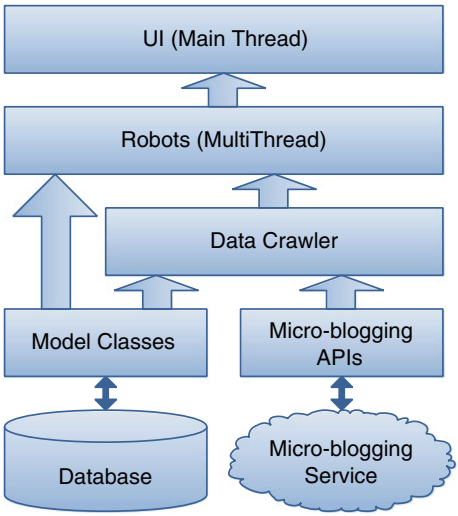

Fig. 13.1 Basic Structure of MBCrawler

(1) **Database** At the most bottom of MBCrawler is a relational database, in which crawled data is stored.
(2) **Model Classes** Each entity like user and status has a database table correspondingly. On the other hand, entities appear as Model Classes to the upper layers of the architecture. Model Classes provide methods for upper layers to manipulate data in database.
(3) **Micro-blogging APIs** Micro-blogging services provide some APIs which meet REST (REpresentational State Transfer) requirements. The layer of Micro-blogging APIs includes simple wrapped methods of the APIs. The methods submit URLs with parameters by HTTP requests, and return the result string in the format of XML or JSON.
(4) **Data Crawler** Data Crawler plays a role as a controller between Robots and Micro-blogging APIs. It receives commands indicating what data to crawl from Robots, and then invokes specific function in Micro-blogging APIs. It also transforms the crawled data from the format of XML or JSON into instances of certain model class, which will be used by Robots.
(5) **Robots** MBCrawler needs different robots crawl different data in multi-threads at the same time. User Relation Robot, User Information Robot, Status Robot, and Comment Robot are four main robots, which should be included at least. All robots have their own waiting queues, such as queues of users' ID or queues of status IDs. The waiting queues are all designed as circular queues to crawl the updated data repeatedly.

Generally, a crawler should run continuously. However, some cases may interrupt its running. For example, the network disconnects, or even the computer on which the crawler is running has to reboot. As a result, every robot will record the user ID or status ID before it start to crawl the data. The robots can start from the last stopped points when they restart.

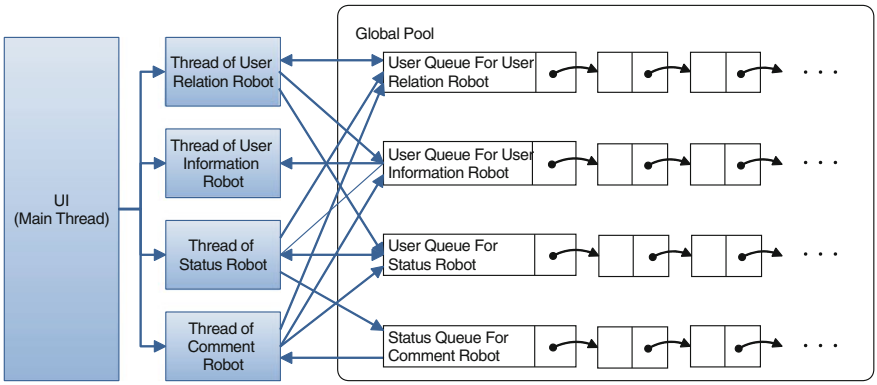

Fig. 13.2 Multi-threaded robots and their queues

(6) **UI** UI means User Interface. It's the layer of representation, which shows information from the crawler and makes users able to control the program. Necessary options of the program also can be set by UI. Robots can be started, paused, continued, or stopped at any time by the buttons on UI.

13.2.2 Multi-Threads Structure of Robots Layer

MBCrawler is multi-threaded. Besides the main thread of UI, more threads with robots working in them is generated, so that different robots can work in parallel. The structure of the multi-threaded robots and their queues is as in Fig. 13.2 shows.

Each robot has its own waiting queue. The queue of Comment Robot is of status IDs, and others are of user IDs.

The arrows between robots and the waiting queues in Fig. 13.2 show the data flow direction between them. Basically, every robot fetches the head item from its own queue to crawl, and then move the item to the end of the queue. By crawling followers and followings IDs of a user, User Relation Robot processes BFS in the social network of micro-blogging service. User Relation Robot will make all waiting queues grow at the same time, by adding new items to them synchronously. User Information Robot does not add new items into any queues. Status Robot adds the ID of the crawled status into the waiting queue of Comment Robot. If it finds new user IDs, it will also add them to the queues of user IDs. Comment Robot also adds commenters' user IDs into the queues of user IDs.

13.2.3 Two-Part Queues Management

As it is shown in Sect. 12.2.2, each robot has a waiting queue of user IDs or status IDs. A whole waiting queue is designed as a circular queue, to ensure that every ID

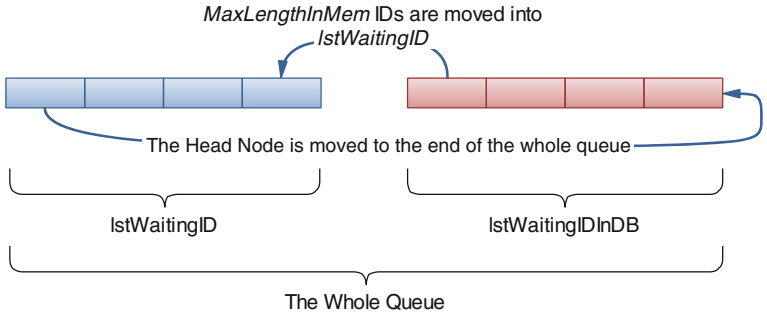

Fig. 13.3 Structure and working process of a queue

in it will be process repeatedly for information updating. As a result, the queues will grow longer and longer as the crawler works. However, the limited memory of computer cannot contain unlimited queues. To deal with that issue, a queue is departed into two parts. One part is maintained in memory, the other part is stored in disk, namely in database. The two parts are managed by two variable *lstWaitingID* and *lstWaitingIDInDB* respectively.

lstWaitingID is a linked list, whose length is limited to *MaxLengthInMem*. *lstWaitingIDInDB* manages the other part of the waiting queue in database. When a robot starts to work, it will create a temporary table in database in order to extend the queue into disk. If the length of the whole queue is longer than *MaxLengthInMem*, new IDs will be added into the temporary table. In this case, IDs in *lstWaitingID* will be moved into *lstWaitingIDInDB* at last. When there is no ID in *lstWaitingID*, the first *MaxLengthInMem* IDs in *lstWaitingIDInDB* will be moved into *lstWaitingID*, as Fig. 13.3 illustrates.

However, if *lstWaitingIDInDB* is too long, and *MaxLengthInMem* is set to a big number, the process of moving *MaxLengthInMem* IDs from *lstWaitingIDInDB* to *lstWaitingID* will take a long time. That may cause the response of database timeout. An alternate way to solve the problem, is adding an additional thread to take the charge of moving IDs between the two parts of the queue, as Fig. 13.4 shows.

As Fig. 13.4 illustrates, a model called *Queue Coordinator* and a queue named *Back Buffer* are added. *Queue Coordinator* fetches IDs from *lstWaitingIDInDB* to *lstWaitingID*. A crawling robot fetches an ID from *lstWaitingID* to crawl. After that, it sends the crawled ID to *Back Buffer*. At the same time, *Queue Coordinator* fetches crawled IDs from *Back Buffer* and pushes them to the back of *lstWaitingIDInDB*. The crawling robot and *Queue Coordinator* work asynchronously in parallel. Of course, each robot needs a *Back Buffer*, but only one *Queue Coordinator* is enough to manage all of the queues.

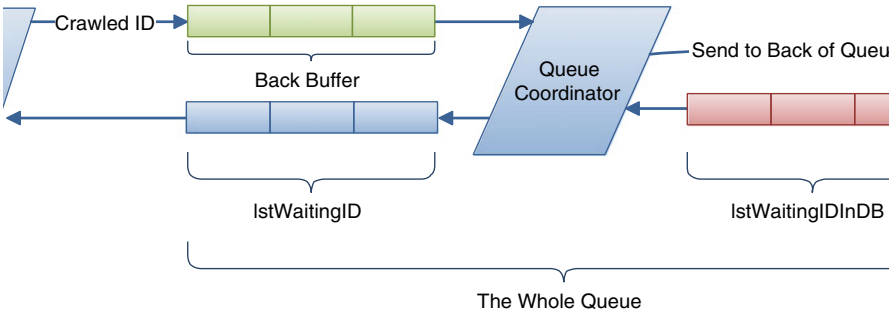

Fig. 13.4 Producer-consumer based queue management

13.3 Implementation as SinaMBCrawler

Basing on MBCrawler, a crawler program named as SinaMBCrawler for Sina micro-blog, which is called Sina Weibo, is developed. Sina Weibo is one of the most popular micro-blogging services in China. By SinaMBCrawler, data of Sina Weibo can be crawled into database.

Sina Weibo has some more features than Twitter. For example, users of Sina Weibo can label themselves with no more than 10 words, which can reflect the users' hobbies, profession, and so on. These words are called as tags. As a result, a new robot takes the charge of crawling the data about tags is added, which is named as UserTagRobot.

In our research work, SinaMBCrawler initially was running for months. During that time, it crawled 8,875,141 users' basic information, 55,307,787 user relationship, 1,299,253 tags, 44,958,974 statuses, and 35,546,637 comments. The data is stored in a SQL Server database. However, in that time, SinaMBCrawler was frequently stopped to be modified because of bugs and updates. That makes the data some dirty due to some testing result. So we abandoned that data. From 15:54:46 on May 30th, 2011 to 11:44:26 on January 7th, 2012, a stable version of SinaMBCrawler was running uninterruptedly. The data obtained this time is listed in Table 13.1.

ACR is the abbreviation for Average Crawling Rate, which is different for each robot, because they access different Sina Weibo APIs. For example, only one user's information can be obtained by UserInfoRobot for each invoking the relative API, while at most 5,000 user relationships can be obtained by UserRelationRobot for one time. ACR is also related to the data. For instance, many users don't set their tags, so UserTagRobot can't get their tags. That lowers the ACR of UserTagRobot.

Table 13.1 Crawled data by a stable version of SinaMBCrawler

Robot	Data content	Total records	ACR (records/minute)
UserInfoRobot	User information	6,571,955	20
UserRelationRobot	User relationship	37,902,219	118
UserTagRobot	Tag	1,068,060	3
	User owning tags	8,031,712	25
StatusRobot	Status	32,627,963	102
CommentRobot	Comment	26,884,365	84

13.4 Conclusion and Future Work

Software architecture for micro-blogging services crawler, called MBCrawler, is designed in our work. The whole architecture is designed in levels. By dividing the whole architecture into several levels and applying some simple design patterns, the structure of the architecture is highly modularized and scalable. The different parts of the architecture are loose coupling, and each part is high cohesion. That makes it easy to modify and extend the software.

Basing on MBCrawler, a crawler software for Sina Weibo named as SinaMBCrawler is implemented. Functions and a robot about users' tags are easily added according to Sina Weibo API. Because of the careful design of the architecture, we have easily upgrade SinaMBCrawler according to Sina Weibo API 2.0, in which only JSON format is used and some new properties are added to users, tags, and so on. SinaMBCrawler has been used to crawl a large number of data from Sina Weibo, which is used in our research work.

In summary, comparing to traditional Web page crawlers, MBCrawler mainly has the following features:

1. Because micro-blogging services APIs are the foundation of the design and implementation of MBCrawler, MBCrawler does not need to download Web pages to analyze. It obtains well-structured data by APIs directly. That makes MBCrawler avoid complex technical issues resulted from AJAX.
2. MBCrawler does not store dynamically generated Web pages, but directly stored the well-structured data obtained by APIs into database.
3. Because MBCrawler does not store Web pages, no index module for Web pages is needed. However, the indexing mechanism of the used database can be utilized to enhance the performance of the database.

Nevertheless, there is an important condition for using MBCrawler. To access the APIs, MBCrawler has to act as an application registered at the micro-blogging services provider. Fortunately, it is easy to register applications for it. After that, a unique pair of application ID and secret will be given to access APIs by the application. Because there are access frequency restrictions for an application, in our SinaMBCrawler, we registered five applications for the five robots. As a result, each robot can work as an independent application, and they will not share the

same access frequency restriction. That makes SinaMBCrawler work more efficiently.

In the future, MBCrawler can be improved mainly in two aspects. Firstly, a focused module can be designed, to tell the robots what type of data to crawl. For example, the robots can be told to crawl the information of users who are in a specific city, or the statuses including specific words. Secondly, we would like to design a ranking module. It is impossible to crawl the whole user relation graph due to the large scale of it. A ranking module will help the crawler to select more important users to crawl. Additionally, because database is loose-coupled with MBCrawler, it also can be considered that trying some NoSQL databases according to the practice requirements. For example, to research the user relation network, any one of graph databases can be selected to store the network.

Acknowledgments This work is supported by the Fundamental Research Funds for the Central Universities grants ZZ1224.

References

1. Arasu A, Cho J, Garcia-Molina H, Paepcke A, Raghavan S (2001) Searching the web. ACM Trans Internet Technol (TOIT) 1(1):2–43
2. Mesbah A, van Deursen A (2009) Invariant-based automatic testing of AJAX user interfaces. In: Proceedings of the 31st international conference on software engineering, Washington, USA, pp 210–220
3. Xia T (2009) Extracting structured data from Ajax site. In: First international workshop on database technology and applications, pp 259–262
4. Duda C, Frey G, Kossmann D, Matter R, Zhou C (2009) AJAX crawl: making AJAX applications searchable. In: IEEE 25th international conference on data engineering ICDE'09, pp 78–89
5. Peng Z, He N, Jiang C, Li Z, Xu L, Li Y, Ren Y (2012) Graph-based AJAX crawl: mining data from rich internet applications. In: 2012 international conference on computer science and electronics engineering (ICCSEE), vol 3. pp 590–594
6. Mesbah A, van Deursen A, Lenselink S (2012) Crawling ajax-based web applications through dynamic analysis of user interface state changes. ACM Trans Web 6(1):1–30
7. Weng J, Lim E-P, Jiang J, He Q (2010) TwitterRank: finding topic-sensitive influential twitterers. In: Proceedings of the third ACM international conference on web search and data mining, New York, USA, pp 261–270
8. Mendoza M, Poblete B, Castillo C (2010) Twitter under crisis: can we trust what we RT? In: Proceedings of the first workshop on social media analytics, pp 71–79
9. Sriram B, Fuhry D, Demir E, Ferhatosmanoglu H, Demirbas M (2010) Short text classification in twitter to improve information filtering. In: Proceeding of the 33rd international ACM SIGIR conference on research and development in information retrieval, pp 841–842
10. A. for C. M. S. I. G. on Security, Audit, and Control (2007) Why we twitter: understanding microblogging usage and communities. In: Proceedings of the ACM workshop on privacy in the electronic society, Washington, USA
11. Asur S, Huberman BA (2010) Predicting the future with social media. ArXiv 1003:5699
12. Kwak H, Lee C, Park H, Moon S (2010) What is twitter, a social network or a news media? In: Proceedings of the 19th international conference on World wide web, pp 591–600

13. Wu S, Hofman JM, Mason WA, Watts DJ (2011) Who says what to whom on twitter. In: Proceedings of the 20th international conference on World wide web, pp 705–714
14. Bakshy E, Hofman JM, Mason V, Watts DJ (2011) Everyone's an influencer: quantifying influence on twitter. In: Proceedings of the fourth ACM international conference on Web search and data mining, pp 65–74
15. Li R, Lei KH, Khadiwala R, Chang KC-C (2012) TEDAS: a twitter-based event detection and analysis system. In: International conference on data engineering, Los Alamitos, USA, vol 0. pp 1273–1276