International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:6, 2016

Proxisch: An Optimization Approach of Large-Scale Unstable Proxy Servers Scheduling

Xiaoming Jiang, Jinqiao Shi, Qingfeng Tan, Wentao Zhang, Xuebin Wang, Muqian Chen

Abstract—Nowadays, big companies such as Google, Microsoft, which have adequate proxy servers, have perfectly implemented their web crawlers for a certain website in parallel. But due to lack of expensive proxy servers, it is still a puzzle for researchers to crawl large amounts of information from a single website in parallel. In this case, it is a good choice for researchers to use free public proxy servers which are crawled from the Internet. In order to improve efficiency of web crawler, the following two issues should be considered primarily: (1) Tasks may fail owing to the instability of free proxy servers; (2) A proxy server will be blocked if it visits a single website frequently. In this paper, we propose Proxisch, an optimization approach of large-scale unstable proxy servers scheduling, which allow anyone with extremely low cost to run a web crawler efficiently. Proxisch is designed to work efficiently by making maximum use of reliable proxy servers. To solve second problem, it establishes a frequency control mechanism which can ensure the visiting frequency of any chosen proxy server below the website's limit. The results show that our approach performs better than the other scheduling algorithms.

Keywords—Proxy server, priority queue, optimization approach, distributed web crawling.

I. Introduction

A Web crawler [1] is a program which downloads web pages from World Wide Web. Many companies and organizations are using this technology to collect the web pages from www. The Google search engine [2] is a distributed system that uses multiple machines for crawling. Attributor [3] is a company that monitors the web for copyright and trademark infringements by web crawler. Meanwhile, web crawling and network security has a great relevance. For example, many researchers and hackers use web crawling to crawl data from websites, such as LinkedIn, Facebook. There are many researches about connecting heterogeneous social networks [4]-[5]. Hackers can get comprehensive personal information by connecting different social networks, which is a disaster for users.

Focused (topical) crawlers [6] are a group of distributed crawlers that specialize in certain specific topics. Each crawler will analyze its topical boundary when fetching web pages. Many researchers have published studies about distributed crawler [7]-[8]. There are two types of distributed crawlers [9]: The crawlers distributed in a Local Area Network (LAN) and the crawlers distributed in Wide Area Network (WAN). Since LAN only uses one global IP address, a crawler distributed in a LAN tends to be detected as impolite crawler. On the other hand, implementing crawler distributed in WAN is costly. To avoid the problem, Harry et al. [9] proposed a method to run hundreds of threads in a single web crawler and to distribute the threads into hundreds or thousands publicly available proxy servers. Moreover, public proxy servers can hide the IP address and other personal information. However, they did not use a scheduling approach to schedule these unstable proxy servers.

Most free proxy servers are not always available, and they may be broken during a period of time. And most websites have a limit of visiting frequency. A focused web crawler has to run carefully. If a user visits a website too frequently, the IP address of this user may be blocked for a long period of time during which the user cannot get access to this website. Therefore, to ensure that the speed and efficiency of crawling will not be affected, the following two critical issues should be firstly considered. One is that many tasks may fail owing to the instability of free proxy servers. The other is that a proxy server will be blocked if it visits a single website frequently. These two issues which can be said to be critical problems seriously affect the efficiency of mining data.

To solve these two critical problems, we propose Proxisch: An optimization approach of large-scale unstable proxy servers scheduling to use free public proxy servers efficiently and save time of mining data in parallel. To solve the first problem, it selects the proxy servers by probability of their success rates; to solve the second problem, it launches a priority queue based on time. Both of the two methods will be introduced in detail later.

Contributions: To address the above open problems, we systematically study, implement, and evaluate Proxisch. Specifically, our main contributions are as follows:

1) We design and implement a distributed web crawling system. It enables users to quickly crawl data from a single website by 2000 public proxy servers in parallel. To the best of our knowledge, the work of Harry et al. [9] is closest to our work. But they did not use a scheduling approach to schedule these unstable proxy servers while we use Proxisch to schedule unstable proxy servers.
2) In our system, we systematically study the proxy servers life span and verify that the life span of proxy servers obeys to exponential distribution. We come to know when most proxy servers are unavailable and how often we should update proxy servers by our analysis.

Xiaoming Jiang, Muqian Chen are with the Institute of Information Engineering, CAS, National Engineering Laboratory for Information Security Technologies, and University of Chinese Academy of Sciences, China (e-mail: jiangxiaoming@iie.ac.cn, chenmuqian@iie.ac.cn).

Qingfeng Tan is with the Institute of Information Engineering, CAS and National Engineering Laboratory for Information Security Technologies, China (corresponding author, e-mail: tanqingfeng@iie.ac.cn).

Jinqiao Shi, Wentao Zhang, and Xuebin Wang are with the Institute of Information Engineering, CAS and National Engineering Laboratory for Information Security Technologies, China (e-mail: shijinqiao@iie.ac.cn, zhangwentao@iie.ac.cn, wangxuebin@iie.ac.cn).

3) In proxisch, we estimate the reliability of proxy servers and the stabler proxy servers are used preferentially. In this way, the success rate of network tasks is higher than other algorithms when used in our system.
4) In proxisch, we propose a frequency control mechanism which can ensure the visiting frequency of chosen proxy server below the limit of a single website, thus we can avoid proxy servers being blocked by any website.
5) We introduce a method to help researcher or research groups with small infrastructures to build an efficient distributed web crawler. And it has proved to be a good method.

Roadmap. Section II presents the summary of the most representative related work. Section III describes the system architecture of our distributed web crawler. Section IV focuses on the frequency control mechanism and proxy servers selection mechanism which are the two cores of system architecture. The other core of system architecture which is named IP proxy span analysis, according to which we update our proxy servers regularly, is discussed in Section V. The most important results are summarized in Section VI where we give a description of our experiments. Finally, we make a conclusion and give an introduction of future work in Section VII.

II. Related Work

A great deal of work on web crawling, especially distributed web crawling has been done and is being done in the world. Distributed web crawling can increase the speed of crawling. Resource scheduling algorithm can help to make full use of proxy servers properly. In this section, we survey about the previous research on distributed web crawling and resource scheduling algorithm.

Distributed web crawling. Salvatore et al. [10] crawled part of data from Facebook for social network analysis with a single thread in 10 days. It is too slow for collecting data from a website. And other researches also show that it is necessary to run a distributed system for collecting a great deal of data [7]-[11]. Researchers have proposed lots of methods for distributed web crawling. A distributed web crawling [12] which can speed up this activity was put forward by Ming Ke. However, it is not a good crawler when used to get data from a certain website. Mining data from a single website differs from mining data from many websites because the former is intended to collect specific data from a certain place. Wang [13] crawled Twitter from January 3rd to 24th 2010. During this period, he collected around 500 thousand tweets and 49 million relations from 25 thousand users. Shkapenyuk et al. [8] implemented a high-performance distributed web crawler which can scale to (at least) several hundred pages per second. But collecting a very large data in a limited time tends to be detected as a cyber attack and will be banned from connecting into the web server. To avoid the problem, Harry et al. [9] proposed a method to run hundreds of threads in a single web crawler and to distribute the threads into hundreds or thousands public available proxy servers. However, considering that most of public available proxy servers are unstable, a reasonable and effective method to dispatch proxy servers is necessary.

Resource scheduling algorithm. Because proxy servers may become unavailable during certain periods of time, we study papers with the similar case here. Liu et al. [14] proposed an effective hybrid algorithm based on particle swarm optimization for permutation flow shop scheduling problem with the limited buffers between consecutive machines to minimize the maximum completion time. Similar to Liu whose algorithm is with the limited buffers, Gunter [15] presented a paper which reviewed results related to deterministic scheduling problems with limited machine availability. McCoy et al. [16] presented Proximax, a robust system that distributes the proxies among the different channels in a way that maximizes the usage of these proxies while minimizing the risk of having them blocked. Wang et al. [17] proposed rBridge user reputation system for bridge distribution; it assigned bridges according to the past history of users to limit corrupt users from repeatedly blocking bridges. Au et al. [18] presented BLACR, which constituted a first attempt to generalize reputation-based anonymous revocation, where negative or positive scores can be assigned to anonymous sessions across multiple categories. But all of these methods cannot be used to schedule proxy servers directly.

To overcome the above problems, this paper proposes Proxisch: An optimization approach of large-scale unstable proxy servers scheduling. It can speed up multi-threading web crawling and make full use of proxy servers which are crawled from the public websites.

III. System Architecture

As mentioned above, mining big data without proxy servers from a single website is very difficult. If it's done too fast, the IP address of users may be blocked for a certain of time. To solve this problem, it is advisable to turn to a fast distributed crawler with proxy servers. Large-scale stable and rechargeable proxy servers are too exorbitant for researchers, so we tend to crawl free proxy servers from websites. However, free proxy servers are always unstable or may be prohibited due to the limit of visiting frequency of most websites. Even if a batch of proxy servers work well in the beginning, most of them may be broken after a period of time. Therefore, we propose Proxisch and implement a web crawling system by Proxisch. The system architecture is shown in Fig. 1.

A. Free Public Proxy Servers

There are lots of publicly free proxy servers on the Internet. Some websites provide an interface by which we can get a large number of proxy servers. Others only provide a few proxy servers and we must analyze their pages to get their proxy servers. We crawl about 8000 proxy servers from the listed websites in Table I.

B. Checking Public Proxy Servers

Since lots of the public proxy servers are unreliable, they must be checked before used. In our program, we test every proxy server for 3 times at most. If a proxy server can get

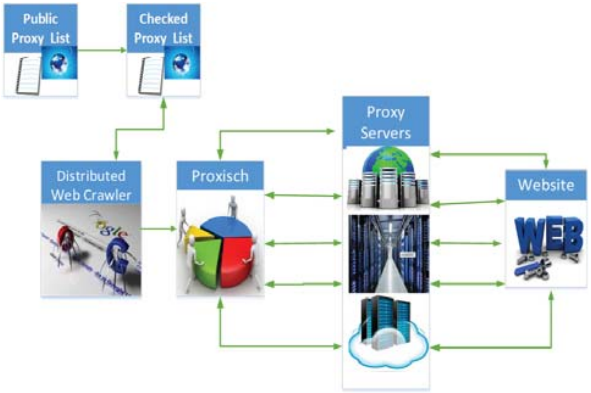

Fig. 1 System Architecture

TABLE I
WEBSITES FROM WHICH WE CAN GET PUBLICLY FREE PROXY SERVERS

Home	Have a interface	Type
www.89ip.cn	Yes	http/https
www.shifengsoft.com	Yes	http
www.66ip.cn/pt.html	Yes	http
http://www.xicidaili.com	No	http/https

access to global.bing.com for less than 4 times, then it is considered as an available proxy server and will be stored in the database. Otherwise, it will be discarded. The results of our experiments show that only about 25 percent of them are available.

C. Updating Public Proxy Servers

The available time of public proxy servers is always very tight. If it takes 2 hour to complete our task and the proxy servers do not work after 40 minutes, most of the task is not completed. Therefore, it is meaningful to study the life span of proxy servers. We verify that the life span of proxy servers obeys to exponential distribution and use the model to compute the interval we should change our proxy servers.

D. Proxisch

In order to improve efficiency of web crawler, the following two issues should be considered primarily: (1) Tasks may fail owing to the instability of free proxy servers; (2) a proxy server will be blocked if it visits a single website frequently. Proxisch exactly deals with the issues. To solve the first problem, it sets up a mechanism, under which the stabler proxy servers are used preferentially. To solve the second problem, it establishes a frequency control mechanism which can ensure the visiting frequency of any chosen proxy server below the website's limit. The detail of Proxisch will be explained in Section IV.

E. Distributed Web Crawler

In order to enhance the parallelism of system, we use gevent [19] which is a coroutine-based Python networking library and it uses greenlet to provide a high-level synchronous API on top of the libev event loop. In our system, the size of thread pool is 2000. It means that there are 2000 threads running simultaneously.

IV. DESIGN

In this paper, we propose Proxisch which can enhance the effective utilization of proxy servers and save time of web crawling. The two cores of Proxisch are: 1) Frequency control mechanism: It controls the used speed of proxy servers by a priority queue; 2) Proxy servers selection mechanism: It selects a proxy server by probability of the proxy server's success rate. In this section, we amply describe the implementation of Proxisch.

A. Frequency Control Mechanism

Since most websites have visiting frequency limits (e.g., Google will block the IP address that visits it too often), we should choose those usable proxy servers whose visits are not frequent. Otherwise, many proxy servers will be blocked and we will lose our usable resources. So we need to create a new proxy servers' allocation mechanism that considers the visiting frequency limit.

TABLE II
PARAMETERS AND NOTATION USED IN SECTION IV

T	The frequency limit: once every T seconds
Et	The earliest available time of a proxy server
$Et^{'}$	The next earliest available time of a proxy server
Ct	The current time
$Max(Ct, Et)$	The bigger one between Ct and Et

Ten next earliest available time $Et^{'}$ is defined as:

$$Et^{'} = Max(Ct, Et) + T \tag{1}$$

This means that the time interval of one proxy server's two continuous visits is more than T seconds, so it meets the limit of visiting frequency. The pseudocode of frequency control mechanism is as Algorithm 1.

Algorithm 1 Frequency Control Mechanism

```
Get IP address of n proxy servers, and save them in ip[0...n-1]
for i = 0 → n − 1 do
    list[i] ← ip[i]
    queue[i] ← [list[i], Ct, 0, 0]
end for
function GETPROXYFREQUENCY()
    choice ← queue[0]
    tail ← [queue[0][0], Max(Ct, Et) + T, queue[0][2], queue[0][3]]
    del queue[0]
    queue[n − 1] ← tail
    return choice
end function
```

The third element of the $queque$ (defined in Algorithm 1) means the number of successful visits of the proxy server while the fourth element of $queque$ means the number of visiting failures of the proxy server. When we select and delete a proxy server from the head of $queque$, it is used in a network task at once. And we will put this proxy server into the tail of the $queque$ with the new earliest available time. So proxy

servers are used cyclically and can meet visiting frequency limit of websites.

B. Proxy Servers Selection Mechanism

Since all of the proxy servers are free, they are unreliable and unstable. If we choose a proxy server only based on its available time, then we will get many visiting exceptions. So every time we choose a proxy server to do the coming network job, we should consider its stability and reliability. So we create the proxy servers selection mechanism based on success rates. The better proxy servers are used more frequently and this way can solve the problem exactly.

TABLE III
PARAMETERS AND NOTATION USED IN SECTION IV

Ff	The number of visiting failures
Fs	The number of successful visits
Sr	The probability with which one proxy server is selected

Table III shows parameters and notation that will be used. Sr which means the success rate is computed as:

$$Sr = Fs/(Fs + Ff) \tag{2}$$

This means that the proxy server chosen from the queue is selected with Sr probability, and abandoned with $(1 - Sr)$ probability; if the current proxy server is abandoned, then another proxy server is chosen from the queue until a proxy servers is selected. The pseudocode of proxy servers selection mechanism is shown in Algorithm 2.

Algorithm 2 Proxy Servers Selection Mechanism

```
function GETPROXYPROBABILITY()
    choice ← GETPROXYFREQUENCY()
    if choice[2] + choice[3] < 10 then
        return choice
    else
        random = get a random number between 0 and 1
        if random <= choice[2]/(choice[2]+choice[3])
then
            GETPROXYPROBABILITY()
        else
            return choice
        end if
    end if
end function
function TASK()
    choice ← GETPROXYPROBABILITY()
    use the proxy server to run crawling job
    result = the result of job
    if result = 1 then
        choice[2] + = 1
    else
        choice[3] + = 1
    end if
end function
```

Every time a task is going to an end, it will update the Fs and Ff of its used proxy server. Only when a proxy server is used more than 10 times, it could be selected by its success rate. Otherwise, it will be selected directly.

Estimate the availability according to the visiting history of a proxy server and select a proxy server with probability according to its success rate. It is obvious that the better the proxy server is, the more frequently it is used. So the success rate of all network tasks is increased and the effectiveness of proxy servers is upgraded.

V. PROXY SERVERS LIFE SPAN ANALYSIS AND ITS APPLICATION

The available time of public proxy servers is always very tight. If it takes 2 hour to complete our task and the proxy servers do not work after 40 minutes, most of the task is not completed. Therefore, it is meaningful to study the life span of proxy servers, because it can tell us how often the proxy servers should be updated.

In this section, we first introduce exponential distribution to which the life span of proxy servers probably obeys approximately. And then we verify that the life span of proxy servers obeys to exponential distribution in reality and use the model to compute the updated time interval.

A. Proxy Servers Life Span

Here, qualities of the free proxy servers usually become worse with time, so we can generalize proxy servers as a kind of resource whose effectiveness and reliability are decreased with time. That is to say, the newer the resource is, the stabler it will be. And available time of them has no relation with the time they has been used. As this kind of resource is just like ideal lamps whose available time is decreased gradually over time. And life of ideal lamp obeys exponential distribution approximately, therefore we can suppose that life of this kind of resource also obeys exponential distribution approximately. The probability function of exponential distribution is:

$$f(x) = \lambda e^{-\lambda x}, \quad \text{if } x > 0. \tag{3}$$

Fig. 2 shows the exponential distribution where $\lambda = 0.5$. Suppose that X_1,X_2,$\cdots$,X_n are n samples which subject to exponential distribution and the parameter is λ. Then we can get maximum likelihood estimation of λ:

$$L(x_1, x_2, , x_n; \lambda) = f(X = x_1) * f(X = x_2) * ... * f(X = x_n); \tag{4}$$

Then we can get the result: $\lambda = 1/\frac{\sum_{i=1}^{n} x_i}{n}$

If we get the value of λ which can be calculated by experiments, then we can calculate the theoretical available time of resources, denoted Tt. If the value of Tt is too small, which means most resources are old and unstable, we should think about to update them.

B. Application

In order to know how long we should update our proxy servers and verify that life of proxy servers obeys to exponential distribution, we tested the life span of 299 proxy

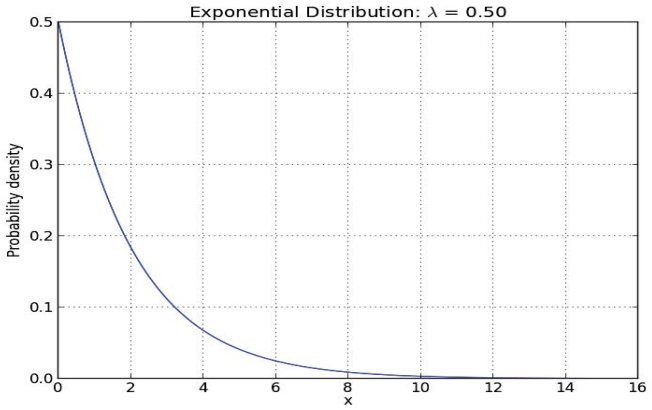

Fig. 2 Exponential distribution where $\lambda = 0.5$

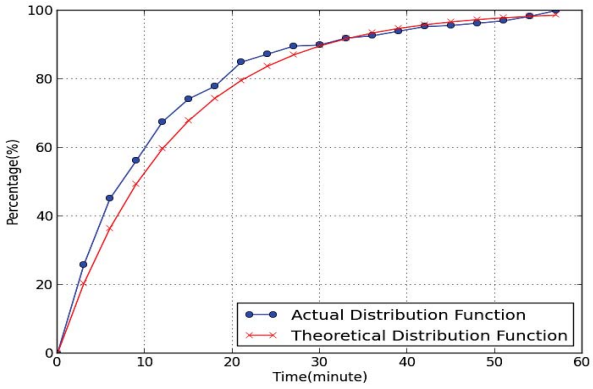

Fig. 3 Actual and theoretical life span distribution of proxy servers

servers, and the result showed that the value of λ is 13.2. And distribution function of exponential distribution is as:

$$F(x) = 1 - e^{-\lambda x}, \quad \text{if } x > 0. \tag{5}$$

The actual life span distribution of proxy servers and theoretical life span distribution are shown in Fig. 3.

According to Fig. 3, the actual life span distribution is close to the distribution function of exponential distribution and it can prove that our model is right. The available time of about 90% proxy servers is shorter than 30 minutes. So to ensure stability and availability of free proxy servers, we update our proxy servers per half hour.

VI. Implementation and Experiment

In this section, we firstly elaborate on the implementation of our program. Then a series of experiments are introduced to compare Proxisch with other scheduling approaches.

A. Implement

In order to verify the efficiency of Proxisch, we implement a test program to send http requests to Renren. Renren, whose home is www.renren.com, is a popular social network website. The related report [20] said that there was about 2.8 billion users in 2013. In our experiments, we did not crawl data from Renren, we only sent http requests to its home and we can know weather a proxy server is available by the http responses or not. The distributed program is written in python. In order to enhance the parallelism of system, we use gevent [19] which is a coroutine-based Python networking library. In our experiments, the size of thread pool is 2000. It means that there are 2000 threads running simultaneously. Furthermore, we use the Requests v2.7.0 library [21] to send http requests. In order to make a comparison, we selected 1000 checked proxy servers and sent 300,000 http requests. The key parameters of experimental environment are shown in Table IV.

TABLE IV
The Key Parameters of Experimental Environment

OS	Ubuntu 14.04.1
Cpu cores	4
Cache size	15360 KB
Cpu MHz	2400.000
MemTotal	4130476 kB
Tested website	www.renren.com
Programming language	python
High-level synchronous API	gevent
Time interval of updating proxy servers	half an hour
Average number of proxy servers	1000
The number of HTTP requests for each experiment	300,000
Timeout	40 seconds

It is regarded as a network error if we do not get a response 40 seconds after the request or we can not connect to the proxy servers. It is seen as an another error if we the http status code of a response is not 200. It is seen as a successful request only when we the http status code of a response is 200.

B. Experiment

1) Experiments of visiting frequency: To study how visiting frequency affects the results, we made a set of experiments in which only the visiting frequency is different and other parameters are same as shown in Table V. The value in Visiting frequency column of Table V means the visiting time interval of proxy servers (e.g, 7 seconds once). The unit of Time

column of Table V is second (e.g, 2905 seconds). And the visiting success/s column of Table V means the number of successful visits per second.

TABLE V
THE RESULTS OF EXPERIMENTS OF VISITING FREQUENCY

Visiting frequency	Total requests	Request success	Network error	Other error	Time	Visiting success/s
7	300,000	229,620	56,191	14,189	2905	79
5	300,000	229,387	52,372	18,241	2061	111
3	300,000	225,337	56,243	18,420	1819	123
1	300,000	226,712	30,947	42,341	1753	129

According to Table V, we can see that when we visited the home of Renren every 7 seconds, although the number of total successful visits is the highest, it took the longest time and the number of its successful visits per second is lowest. We can conclude that the number of network errors is not only related to visiting frequencies, it's also related to the stability of proxy servers. Most of other errors are due to frequent visits. The results show that the more frequent the visits are, the more other errors there will be.

2) Experiments of Different Scheduling Approaches: To study how scheduling approaches affect the results, we made a set of experiments in which only the selection mechanisms is different and other parameters are same as shown in Table V. The total number of requests is 300,000. The results are shown in Table VI. We made a comparison of Proxisch and Round-Robin scheduling, which is a classic scheduling algorithm.

TABLE VI
THE RESULTS OF EXPERIMENTS OF DIFFERENT SCHEDULING METHODS

Selection mechanism	Visiting frequency	request success	Network error	Other error	Time	visiting success/s
Proxisch	3	225,337	56,243	18,420	1819	123
Round-Robin	3	192,115	74,290	33,595	2308	83

According to Table V, our scheduling approach has a better result than polling scheduling. Our scheduling algorithm can get more successful visits within a shorter period of time. The number of successful visits per second of proxisch is 1.48 times of that in the polling scheduling.

VII. CONCLUSION AND FUTURE WORK

In this paper we have presented Proxisch: An optimization approach of large-scale unstable proxy servers scheduling and it's used in distributed web crawling. We show how we get the free proxy servers and check them before they are used. We introduce the life span model of proxy servers according to which we update our free proxy servers regularly. Moreover, we explain our approach and make a set of experiments. In conclusion, our solution is a practical way to schedule large-scale unstable proxy servers.

Our future work is to make our approach adjust the visiting frequency automatically according to the results of previous tasks. There are only http proxy servers now and we attempt to get lots of Socks proxy servers in the future.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61303260; the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06030200.

REFERENCES

[1] S. Kaur and A. Gupta, "A survey on web focused information extraction algorithms," 2015.
[2] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.
[3] Attributor, "Attributor."
[4] Y. Zhang, J. Tang, Z. Yang, J. Pei, and P. S. Yu, "Cosnet: Connecting heterogeneous social networks with local and global consistency," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1485–1494.
[5] S. Ji, W. Li, P. Mittal, X. Hu, and R. Beyah, "Secgraph: A uniform and open-source evaluation system for graph data anonymization and de-anonymization," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 303–318.
[6] R. Patel and P. Bhatt, "A survey on semantic focused web crawler for information discovery using data mining technique," *International Journal for Innovative Research in Science and Technology*, vol. 1, no. 7, pp. 168–170, 2015.
[7] W. Galuba, K. Aberer, D. Chakraborty, Z. Despotovic, and W. Kellerer, "Outtweeting the twitterers-predicting information cascades in microblogs," in *Proceedings of the 3rd conference on Online social networks*, vol. 39, no. 12, 2010, p. 3âAS3.
[8] V. Shkapenyuk and T. Suel, "Design and implementation of a high-performance distributed web crawler," in *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 2002, pp. 357–368.
[9] H. T. Y. Achsan and W. C. Wibowo, "A fast distributed focused-web crawling," *Procedia Engineering*, vol. 69, pp. 492–499, 2014.
[10] S. A. Catanese, P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Crawling facebook for social network analysis purposes," in *Proceedings of the international conference on web intelligence, mining and semantics*. ACM, 2011, p. 52.
[11] L. F. Lopes, J. Zamite, B. Tavares, F. Couto, F. Silva, and M. J. Silva, "Automated social network epidemic data collector," in *INForum informatics symposium. Lisboa*, 2009.
[12] M. Ke, P. Zhang, and G. Chen, "The crawler of specific resources recognition based on multi-thread," in *Computational Sciences and Optimization (CSO), 2012 Fifth International Joint Conference on*. IEEE, 2012, pp. 569–572.
[13] A. H. Wang, "Don't follow me: Spam detection in twitter," in *Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on*. IEEE, 2010, pp. 1–10.
[14] B. Liu, L. Wang, and Y.-H. Jin, "An effective hybrid pso-based algorithm for flow shop scheduling with limited buffers," *Computers & Operations Research*, vol. 35, no. 9, pp. 2791–2806, 2008.
[15] G. Schmidt, "Scheduling with limited machine availability," *European Journal of Operational Research*, vol. 121, no. 1, pp. 1–15, 2000.
[16] D. McCoy, J. A. Morales, and K. Levchenko, "Proximax: A measurement based system for proxies dissemination," *Financial Cryptography and Data Security*, vol. 5, no. 9, p. 10, 2011.
[17] Q. Wang, Z. Lin, N. Borisov, and N. Hopper, "rbridge: User reputation based tor bridge distribution with privacy preservation." in *NDSS*, 2013.
[18] M. H. Au, A. Kapadia, and W. Susilo, "Blacr: Ttp-free blacklistable anonymous credentials with reputation," 2012.
[19] D. Bilenko, "gevent," *http://www.gevent.org/*, 2015.
[20] 199it, "Report about renren," *http://www.ebrun.com/20130507/72900.shtml*, 2013.
[21] K. Reitz, "Requests library," *http://www.python-requests.org/en/latest/*, 2015.