International Journal of Grid Distribution Computing
Vol.7, No.4 (2014), pp. 149-156
http://dx.doi.org/10.14257/ijgdc.2014.7.4.14

Design and Implementation of A Focused Crawler — TargetCrawler

Feng Jian, Chen Jing-zhou and Cao Lei

College of Computer Science and Technology, Xi'an University of Science and Technology, Xi'an, China 710054
actour@163.com

Abstract

Adopting focused crawler to search web sites is the trend of next generation search engines. Design and implementation of a focused crawler - TargetCrawler is introduced in detail, including its overall architecture, main modules, working processes and two key algorithms, duplicate removing algorithm based on the Bloom filter and ranking algorithm based on priority which are designed to ensure accuracy and efficiency of web search. Experimental results show the effectiveness of the scheme.

Keywords: *Focused crawler; duplicate removing; priority ranking*

1. Introduction

With the rapid development of Web technology, management and search for massive data become particularly important. Crawler is a program or script which can automatically obtain information on the World Wide Web. It traverses the Internet relying on the link relationship between the pages, and downloads information stored dispersed on the Internet to the local, in order to make search engines to classify the Web pages according to indexes. Among all kinds of crawlers, focused crawler for a specific topic has become the development trend for the next generation of search engines, the design goal of it is to crawl the network and to get a theme related resources as efficiently as possible, and then to strategically filter out the irrelevant or poor-related resource.

This paper describes the system framework, the main modules and their functions and the key algorithms of a focused crawler - TargetCrawler, and then validates it through experiments. The key algorithms are designed to address the key issues of focused crawler: URLs analysis, filtering and prioritization.

2. System Frameworks

2.1. The Overall Design

The main task of the software is: according to the set of keywords, start crawling from the initial URLs and parse pages obtained. On the one hand extracts new URLs and updates the URLs list to continue crawling until reaching a certain standard; on the other hand analyzes to obtain the performance report according to crawling process and crawling results.

The main function modules of TargetCrawler include download module, parsing module and optimization module, and the system structure shown in Figure 1.

ISSN: 2005-4262 IJGDC

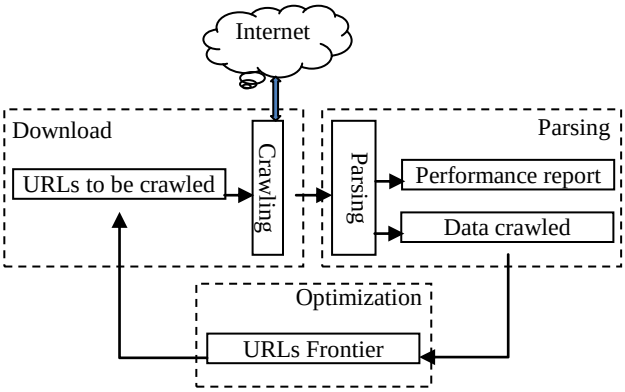


Figure 1. The System Architecture of TargetCrawler

(1) Download module design

Download module is responsible for grabbing Web pages. The function of the module is to generate (in the first time) or update URLs queue to be crawled, and then assign URLs in the queue to different crawling threads to complete the web fetching through the HTTP protocol. URLs are stored by decreasing the quality to the updated queue for URLs to be crawled.

(2) Parsing module design

The main work of parsing module is web page analysis and link extraction. And page analysis is divided into two processes - page preprocessing and information extraction. Among them page preprocessing process determines whether the format of Web page is in keeping with standards or not, and makes non-standard Web page to conform to the standards. Information extraction process use HTML tags to parse contents on Web pages, to extract key contents and related links based on predetermined theme, while to get crawling results and performance statistics reports. The module also realizes duplicate removing function through comparing extracted links to URLs which have been crawled and then passing the URLs which have not been crawled to the optimization module.

(3) Optimization module design

The module's function is to sort the URLs provided by parsing module based on the quality and then to trigger update of URLs which wait for been crawled. The module adopts the principle of combined priority algorithm and polite access policies, which includes through the process of URLs Frontier to ensure that only one connection to access a Web server one time, and wait for a few seconds to continuously visit the same Web server, while high priority is given to page which need to be crawled first.

There are four kinds of strategies [1-4] usually been adopted when Web crawler traverses nodes: depth-first traversal, breadth-first traversal, the best-first traversal and random traversal. Considering the depth-first traversal usually causes trapped and random traversal collects pages with lower relevance, TargetCrawler adopts the principles of breadth-first and best-first combination, having both character of generic Web crawler and focused crawler. Its goal is to cover much of the pages in crawling, and then according to the link content ratings to select one or more URLs predetermined subject highly relevant to continually crawling.

2.2. Crawling Process

A simplified workflow of TargetCrawler is shown in Figure 2, where the Todo table is the queue which has not been crawled, and Visited table is the queue which has been crawled. Specific process is as follows:

● Initial setting: set crawling keywords, starting URLs, storage paths of crawling result and other basic information.

● Page download: using HTTP establish communication with hosts corresponding to URL, and obtain page returned by the interaction.

● Page parsing: analyze HTML pages and parse out different tags and URLs available.

● Performance report: through drawing graphs for crawling progress and URLs tree, visually observe crawling progress of TargetCrawler, and then save the crawling information to the database.

● Visited table elements adding: add the URLs which have been crawled to Visited table.

● Duplicate removing of URLs: compare extracted links from page parsing to URLs in Visited table and then pass the URLs which have not been crawled to URLs Frontier process.

● Updating Todo table: according to the keywords, URLs Frontier process filters URLs according to the correlation, and put URLs into the Todo table based on the principle of relevance descending.

● Execution loop: remove a URL from the updated ToDo table in, and then go to Step 2 to continue.

● Exiting the program: when the user chooses to exit the program, terminate crawling operation and exit the program.

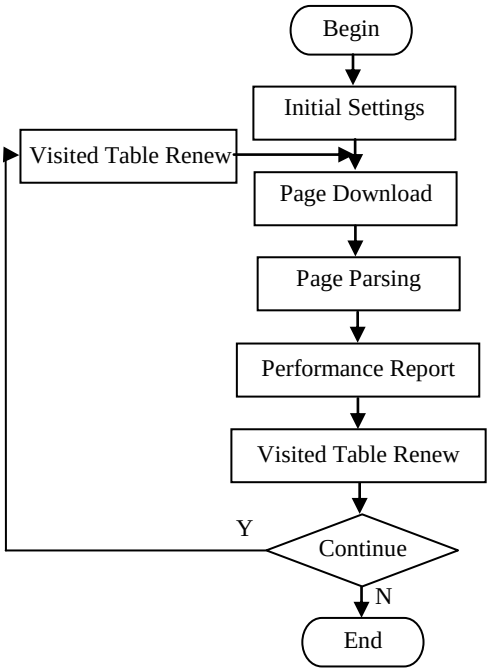


Figure 2. Simplified Workflow of TargetCrawler

3. Key Algorithms

Throughout the crawling process, duplicate removing of URLs and process for thematic relevance are especially important, its good design and implementation are fundamental guarantee to achieve a good crawling speed and effectiveness of information. TargetCrawler has taken the following key algorithms:

3.1. Bloom Filter-based Duplicate Removing Algorithm

In the download process will inevitably encounter duplicate links, how to eliminate these duplicate links will direct impact on the efficiency and effectiveness of crawling.

The current mainstream web crawler used to implement index of URLs by a tree or hash table, and to check duplicate URLs through comparing the index of a new URL to the indexes of URLs which have been crawled. TargetCrawler based on Bloom Filter [5] to achieve indexing and duplicate removing of the URLs. The algorithm works as follows:

Bloom Filter requires a bit array, named array (similar to a bitmap) and K mapping functions (similar to Hash table). In the initial state, length of the bit array is m, and all bits are set to 0, shown in Figure 3.

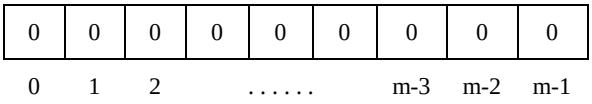


Figure 3. Bit Array of Bloom Filter (in the initial state)

The set S = {s1, s2 ....., sn} has n elements, and each element j (1<= j<= n) is mapped by a mapping function K {f1, f2 ....., fK} to K values {g1, g2 ....., gK}, and then the corresponding bits of array (array [g1], array [gK]) is set to 1, as Figure 4 shown.

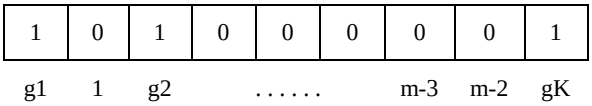


Figure 4. Bit Array of Bloom Filter (after mapped)

The same mapping method is used when needs to determine whether an element item in S or not. Firstly, by the mapping function {f1, f2, ..., fk} to obtain a map value K {g1, g2, ..., gk}, and then check the value of array[g1], array[g2], ..., array[gk], if all 1, it indicates that item is in the set S, or isn't.

In the specific implementation process, the foundation class bitset is used to handle Bloom filter, and to realize insertion and retrieval capabilities of URLs. The following pseudo-code illustrates implementation of Bloom filter algorithm in the URLs extraction process.

```
Begin
   Seeds [7]: {3, 7, 11, 13, 31, 37, 61}
   BloomSet：bitset<NUM>
   IsUrlInBloomSet (<Url>):
   For i:= 0 to 7
          Hash := call <GetHashVal> (<Url, i>)
          If Hash = 0
              Return false
   If call<IsUrlInBloomSet> (<Url>) then
       Return False
   Else
       call<AddUrlIntoBloomSet> (<Url>)
       Return Ture
End
```

Among them, the array Seeds[] provides basic data for the creation of 7 mapping function, and <GetHashVal> (<Url, i>) is the process of mapping URL to 7 mapped values according

to mapping function. The basic idea of the algorithm is that when judging whether a URL in collection BloomSet, if the URL exists in the BloomSet already, it returns true, otherwise the URL is added to BloomSet and returns false.

3.2. Priority-based Sorting Algorithm

When the Web pages is crawled , in order to ensure that the entire crawling process has always focused on a specific subset of Web pages, focused crawler should filter URLs in the Web pages and extract URLs which are topics-related, and then crawl for subsequent URLs by the degree of correlation.

Priority-based sorting algorithm uses the classic Best-first approach [6], to evaluate the value of URLs which need to be downloaded based on the similarity of contents of the Web pages with the predetermined theme. Key words are used to describe crawling theme, and the priority of URLs which need to be downloaded is calculated according to those keywords and text contents of URLs which have been downloaded. Taking Web page p as an example, relevance of the link pointed by p to the theme should be estimated by relevance between text contents of p and the keywords. Web pages with high correlation point to the link with high priority, so as to determine priority order of the URL to be crawled. If the crawling queue is fully filled, then the URL with the lowest priority will be removed from the queue. The formula (1) is used to calculate the correlation between pages and themes.

$$\sin(q,p)=\left(\sum_{k\in q\cap p} f_{kq} f_{kp}\right)/\left(\sqrt{\sum_{k\in p} f_{kp}^2}\sqrt{\sum_{k\in q} f_{kq}^2}\right) \quad (1)$$

Wherein q represents theme, p represents crawled Web pages, f_{kq} represents the frequency which keyword k appears in the q, f_{kp} represents the frequency which keyword k appears in the p.

URLs Frontier process is used to handle optimizing and sorting of URLs in TargetCrawler. There are two kinds of FIFO queues, Front queues and Back queues. Front queues include 16 sub-queues with a priority of 0 to F, URLs in each sub-queue have the same priority. Back queues assign a set of sub-Back queues for each host, the sub-queues are stored in all URLs needing to be crawled from the same host, and priorities of the URLs are from the highest to the lowest, followed by the extraction of URLs one by one to achieve a courtesy visit. Firstly, priority of a new URL is calculated according to the theme, and then put the URL into corresponding sub-queue of Front queues based on priority; Secondly, distinguish different hosts based on IP address and remove URLs of the same host from Front sub-queue to Back sub-queue according to the priority. Namely a URL requires in turn through the Front queues and Back queues to enter URLs queue to be crawled. The following pseudo-code gives the specific implementation process of URLs Frontier:

```
Begin
  For i:= 15 to 0
     Url := call<PopFrontQueue>(<i>)
     If No Url then
          Continue
     HostName := call <getHostName> (<Url>)
     ID := call<FindBackQueueId>(<HostName>)
     call<PushUrlIntoBackQueue>(ID，Url)
     WaitTime := call<GetTimeMinHeap>(ID)
     Sleep(WaitTime)
     Call<PopUrlBackQueue>
```

End

Where the specific meanings of different calling processes are as follows: PopFrontQueue is used to extract URL with the highest priority from the Front queue; FindBackQueueId is for looking up Back sub-queue this URL belongs to, if the Back sub-queue doesn't exist then creates a new corresponding Back sub-queue for the host this URL belongs to; PushUrlIntoBackQueue is used to add the URL into the queue just been found; Through GetTimeMinHeap table, access time to wait for visiting the host mannerly can be known; PopUrlBackQueue is for removing the URL from Back sub-queue and then adding to crawling URLs queue after waiting for a specified time.

4. Experiments and Analysis

4.1. Experimental Environment

A HP desktop with 2GB RAM and 500GB hard drive was used for experiments, whose CPU is Pentium4 and operating system is Microsoft Windows 7. Pages are collected from China Education and Research Network. After entering the operational phase, detailed crawling results of TargetCrawler can be seen, including the number of URLs extracted, the number of URLs crawled and crawling URLs list based on keywords; graphical interface can also be used to display crawling results, such as the progress curve (x axis is time, y axis is the number of URLs crawled) and the relationship between URLs with different depths (URLs obtained from one URL).

4.2. Experimental Results and analysis

Table 1. Crawling Results with Different Numbers of Threads

	10s			20s			30s		
Data type	T	F	A	T	F	A	T	F	A
1 thread	39	4	265	49	19	2305	55	23	4541
3 threads	46	21	1836	61	30	5642	85	42	12315
	60s			90s			120s		
Data type	T	F	A	T	F	A	T	F	A
1 thread	85	31	10135	126	65	13726	154	120	14470
3 threads	131	67	14585	171	124	18472	213	203	23687

A number of experiments were conducted for examining the situation of crawling: different numbers of threads were started for crawling at the same time. Experimental results are shown in Tab. 1.

In the table, T represents the number of URLs successfully crawled, F represents the number of URLs filtered out by Bloom filter and A represents the number of URLs TargetCrawler got. From this table, crawling conditions at different time points can be observed:

Following time increase, the number of URLs crawled successfully is increasing, but crawling speed is not stable.

- The number of URLs filtered out by Bloom filter is in faster growth.
- Crawling faster when multithreads are used.
- There are two main reasons for crawling speed fluctuations: the same URL has multiple outbound links and URLs Frontier uses a courtesy visiting policy in the process.

From the data, it can be found that by increasing the number of crawler threads can significantly improve the rate of collection of pages. But if you want to further improve the

collection rate, it is necessary to increase the number of computer runs crawler or use of efficient parallel algorithms. By observing local URLs store results, it can be seen that most URLs stored contain crawling theme and correct filtration rate reached 92% according to a predetermined keywords.

In short, TargetCrawler can download HTML pages with high speed and efficiency based on predetermined theme, and store HTML document information to meet the requirements.

5. Conclusions

This paper designs and implements a topic-specific Web crawler – TargetCrawler. It can effectively remove duplicate links in crawling process, predict priority according to contents of the pages, and it has high efficiency for information collection. System scheme has been validated in the specific implementation. Currently the scheme takes only content relevance of the page into consideration in the calculation of priority of URL, and pays no consideration to the link relationship between pages. Research will focus on the comprehensive quality evaluation algorithm of pages in the future, to further improve the efficiency and quality of the search. Meanwhile parallel and distributed technologies will be used to further improve performance of the system, to adapt the needs of large-scale crawling and then provide more comprehensive information retrieval services.

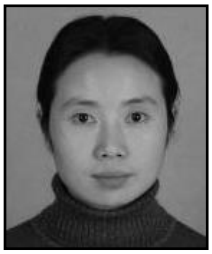
Acknowledgements

This work was supported in part by Shaanxi Provincial Natural Science Foundation Project (No. 2012JQ8030).

References

[1] D. Fetterly, N. Craswell and V. Vinay, "Search effectiveness with a breadth-first crawl", In Proc. of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, New York: ACM, **(2008),** pp. 755-756.
[2] M. J. Cafarella, J. Madhavan and A. Halevy, "Web-scale extraction of structured data", ACM SIGMOD Record, vol. 37, no. 4, **(2008),** pp. 55-61.
[3] A. Z. Broder, M. Najork and J. L. Wiener, "Efficient URL caching for World Wide Web crawling", In Proc. of 12th Int. Conf. World Wide Web, New York, ACM, **(2003),** pp. 680-689.
[4] F. McCown and M. L. Nelson, "Evaluation of crawling policies for a web-repository crawler", In Proc. of 17th ACM Conference on Hypertext and Hypermedia (HYPERTEXT 2006). New York: ACM, **(2006)** August, pp. 157-168.
[5] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey", Internet Mathematics, vol. 1, no. 4, **(2005),** pp. 485-509.

Author



Jian Feng, date of birth: August, 1973. She received her doctoral of Computer Software and Theory from Northwest University, Xi'an, China, in 2008. And research interests on computer network and communication, network security, distributed computing.

She is currently an Associate Professor with College of Computer Science & Technology, Xi'an University of Science and Technology, Xi'an, China.