반도체디스플레이기술학회지 제18권 제3호(2019년 9월)
Journal of the Semiconductor & Display Technology, Vol. 18, No. 3. September 2019.

An Implementation and Performance Evaluation of Fast Web Crawler with Python

Cheong Ghil Kim†

† Department of Computer Science, Namseoul University

ABSTRACT

The Internet has been expanded constantly and greatly such that we are having vast number of web pages with dynamic changes. Especially, the fast development of wireless communication technology and the wide spread of various smart devices enable information being created at speed and changed anywhere, anytime. In this situation, web crawling, also known as web scraping, which is an organized, automated computer system for systematically navigating web pages residing on the web and for automatically searching and indexing information, has been inevitably used broadly in many fields today. This paper aims to implement a prototype web crawler with Python and to improve the execution speed using threads on multicore CPU. The results of the implementation confirmed the operation with crawling reference web sites and the performance improvement by evaluating the execution speed on the different thread configurations on multicore CPU.

Key Words : Web Crawler, Web Crawling, Web Scraping, Search Engine, Web Bot, Thread

1. Introduction

Over the past decade, an astronomical amount of information has been published on the Web. As well, Web services such as Twitter, Facebook, and Digg reflect a growing trend to provide people and applications with access to real-time streams of information updates [1]. Together, these two characteristics imply that the Web has become an exceptionally potent repository of programmatically accessible data [1]. Due to the extremely large number of pages present on Web, the search engine depends upon crawlers for the collection of required pages [2].

Considering the situation that our computing platforms are shifting to mobile form desk top PCs rapidly, web crawling, also known as web scraping, which is an organized, automated computer system for systematically navigating web pages residing on the web and for automatically searching and indexing information, has been inevitably used broadly in many fields today. Fig. 1 shows the percentage of all global web pages served to mobile phones and we can easily find that mobile access has begun to outpace online access since 2017 [3-6].

In general, search engines are used to extract valuable information from the Internet and the principal part of them is known as web crawler. It is a computer program or software that is able to browse the World Wide Web in a methodical, automated manner or in an orderly fashion. It is an essential method for collecting data on, and keeping in touch with the rapidly increasing Internet [7].

Another utilization of web crawler is web archiving which collects large sets of web pages periodically and archives them for posterity. Web data mining also can be executed effectively using web crawler to analyze web pages for various purposes. Finally, web monitoring services allow their clients to submit standing queries, or triggers, and they continuously crawl the web and notify clients of pages that match those queries [9].

This paper reviews the concepts of web crawler with its architecture and implements a prototype web crawler using an open source [9]. The source code was programed with

†E-mail: cgkim@nsu.ac.kr

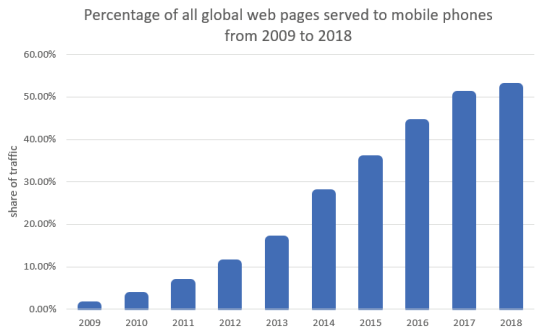

Fig. 1. Percentage of all global web pages served to mobile phones from 2009 to 2018.

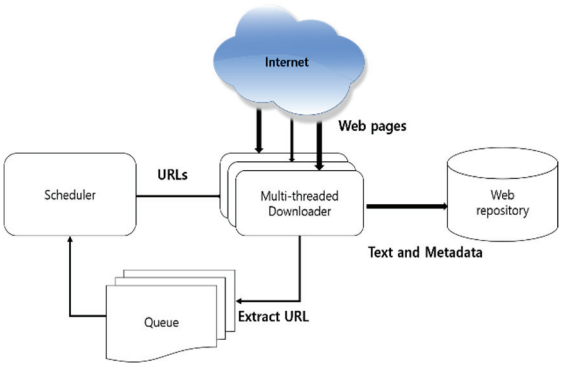

Fig. 2. Web crawler architecture.

Python 3.6. At the beginning a sequential crawler was implemented and the performance improvements were made using thread on multicore CPU. The crawler implemented is automatically scrapping the data into files and adding threads and multiprocessing functions. The implementation results were verified by measuring the crawl rate of various sites by changing the number of threads and the number of processors.

The rest of this paper is organized as follows. Section 2 introduces the web crawler; Section 3 introduces its serial and parallel implementations; Section 3 shows the simulation results using multi-threads on multicore CPU, and Section 4 brings out the conclusion.

2. Web Crawler

Fig. 2 shows the overall architecture of web crawler consisting of three main components: a schedule, multi-threaded downloader, and web repository. A scheduler stores the list of URL's to visit. Multi-threaded downloader downloads web pages from the Internet. Web repository receives web pages from a crawler and stores them in the database. More details on the functions of the main components are follows [8].

Scheduler contains the list of unvisited URLs which is set with seed URLs. The list may be delivered by a user or from other programs. The initiation of crawler is made with the seed URL. And then the crawler retrieves a URL from the scheduler. The page corresponding to the URL is fetched from the Web, and the unvisited URLs from the page are added to the scheduler. This execution terminates upon the empty queue or with some other condition. Any prioritization scheme may decide the order of extracting URLs from the scheduler.

The main function of multi-threaded downloader is to download the page from the Internet based on the URLs from the scheduler. For this purpose, it has to equip with a HTTP client for HTTP request and response. Also, it may have the feature of timeout period in order not to take unnecessary time to read large files or wait for response from slow server.

Web repository: It use to stores and manages a large pool of web page data. The repository stores only standard HTML pages as forms of separate files. In this case, it does not store other media and document types data. And the storage manager stores the up-to-date version of every page retrieved by the crawler.

3. Implementation

Fig. 3 shows an overall operational diagram of the web crawler implemented with Python [10]. For implementation, two more modules are used. One is the requests module; and the other is Beautiful Soup [11]. The former allows easy loading of web pages via HTTP. The other automates the scraping of content from web pages. Follows are the basic operation sequences.

The first step is creating the queue. For this purpose, Python's built-in list data type can be used. In order to improve the performance when pulling items off the front of them repeatedly, an effective double-ended queue can be designed. Next is loading the page and finding the links taking advantage of requests and Beautiful Soup. After that we can do whatever we want with the web pages crawled by

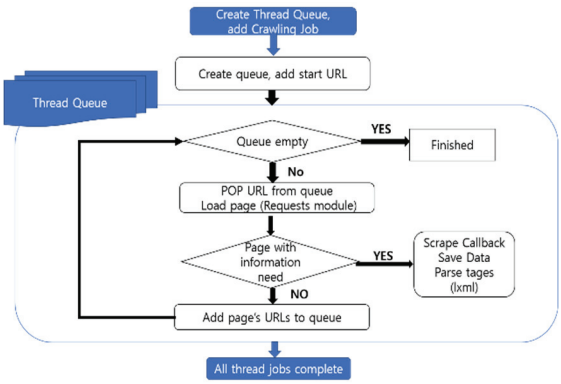

Fig. 3. Execution flow.

```
while crawl_queue:
    url = crawl_queue.pop()
    depth = seen[url]
    if rp.can_fetch(user_agent, url):
        html = D(url)
        links = []
        if scrape_callback:
            links.extend(scrape_callback(url, html) or [])
        if depth != max_depth:
            if link_regex:
                links.extend(link for link in get_links(html) if re.match(link_regex, link))
            for link in links:
                link = normalize(seed_url, link)
                if link not in seen:
                    seen[link] = depth + 1
                    if same_domain(seed_url, link):
                        crawl_queue.append(link)
    else:
        print('Blocked by robots.txt:', url)
```

(a) Serial execution implementation

```
def crawlqueue():
    url = crawl_queue.pop()
    depth = seen[url]
    if rp.can_fetch(user_agent, url):
        html = D(url)
        links = []
        if scrape_callback:
            links.extend(scrape_callback(url, html) or [])
        if depth != max_depth:
            if link_regex:
                links.extend(link for link in get_links(html) if re.match(link_regex, link))
            for link in links:
                link = normalize(seed_url, link)
                if link not in seen:
                    seen[link] = depth + 1
                    if same_domain(seed_url, link):
                        crawl_queue.append(link)

    else:
        print('Blocked by robots.txt:', url)
threads = []
while threads or crawl_queue:
    for thread in threads:
        if not thread.is_alive():
            threads.remove(thread)
    while len(threads) < max_threads and crawl_queue:
        thread = threading.Thread(target=crawlqueue)
        thread.setDaemon(True)
        thread.start()
        threads.append(thread)
    time.sleep(SLEEP_TIME)
```

(b) Parallel execution implementation with multi-thread

Fig. 4. Serial and parallel implementation source.

implementing specific functionalities. Here, we implemented code to find tour information. The final stage is adding the page's links to the queue.

Fig. 4 (a) shows sample source code for serial execution. For a fast execution we add some codes using multi-threaded feature on multi-core CPU shown in Fig. 4 (b). For the reference web pages for crawling, we choose three difference sites for tour information with different data sizes: myrealtrip [12], Tourtips [13], and EarthTory [14].

4. Results

For the performance evaluation, the hardware environment of PC is summarized as follows. CPU is CPU AMD Rygen 5 1600 with 3.2GHz clock speed. The number of core and thread are 6 and 12, respectively. Main memory is 16GB.

Core 6 Thread 12 Memory 16GB: Fig. 3 shows an overall operational diagram of the web crawler implemented with Python [10]. For implementation, two more modules are used. One is the requests module; and the other is Beautiful Soup [11]. The former allows easy loading of web pages via HTTP. The other automates the scraping of content from web pages. Follows are the basic operation sequences.

Table 1 summarized the execution times. Here, "C" and "T" denote Core and Thread, respectively. The numbers in front of "C" and "T" refer the number of Core and Thread, respectively. Fig. 5 depicts the result of Table 1 in order to compare the performance improvement easily. We can find that the linear performance gains as the increase the number of Core and Thread. Especially, the reference site with big data size shows much high performance gain.

5. Conclusion

This paper aims to implement a prototype web crawler with Python and to improve the execution speed using threads on multicore CPU. The results of the implementation confirmed the operation with crawling reference web sites and the performance improvement by evaluating the execution speed on the different thread configurations on multicore CPU.

Table 1. Performance evaluation results

Items	1C1T	1C5T	1C10T	12C10T	8C10T	4C10T
myrealtrip	3758.27	1173.17	643.59	383.87	397.92	399.82
TourTips	301.9	68.04	38.04	16.64	23.48	29.39
EarthTory	1374.91	387.52	204.79	82.77	83.63	84.6

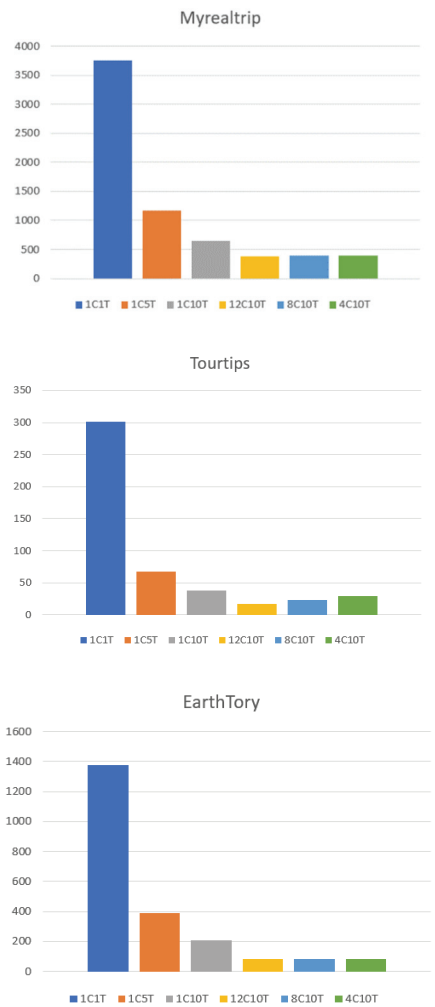

Fig. 5. Simulation results.

Acknowledgment

Funding for this paper was provided by Namseoul University year 2019.

References

1. Jonathan M. Hsieh, Steven D. Gribble, and Henry M. Levy, "The architecture and implementation of an extensible web crawler", NSDI'10 Proceedings of the 7th USENIX conference on Networked systems design and implementation, San Jose, California, April 28 - 30, 2010.
2. Keerthi S. Shetty, Swaraj Bhat and Sanjay Singh, "Symbolic Verification of Web Crawler Functionality and Its Properties", International Conference on Computer Communication and Informatics (ICCCI - 2012), Coimbatore, INDIA, IEEE Conference Publications, 2012.
3. How to Perform a Website Design & Optimization Audit. April 3, 2018. Available online: https://www.brightscout.com/how-to-perform-a-wesit e-audit/s
4. Cheong Ghil Kim, "A Study of the Performance Prediction Models of Mobile Graphics Processing Unit", Journal of the Semiconductor & Display Technology, Vol.18 No.1, pp. 1-5, 2019.
5. D. Choi, W. Han, Y. Lee, and Y. Kim, "Learning Methods for Effective Object Tracking in 3D Storytelling Augmented Reality", Journal of the Semiconductor & Display Technology, Vol. 15, No. 3., pp. 46-50, September 2016.
6. Yong-Hwan Lee and Heung-Jun Kim, "Evaluation of Feature Extraction and Matching Algorithms for the use of Mobile Application", Journal of the Semiconductor & Display Technology, Vol. 14, No. 4., pp. 56-60, December 2015.
7. Trupti V. Udapure, Ravindra D. Kale, Rajesh C. Dharmik, "Study of Web Crawler and its Different Types", IOSR Journal of Computer Engineering, Vol. 16, Issue 1, Ver. VI (Feb. 2014), PP 01-05
8. Christopher Olston and Marc Najork, "Web Crawling", Foundations and Trends in Information Retrieval, Vol. 4, No. 3, pp. 175–246, 2010.
9. Richard Lawson, "Web Scraping with Python", Packt Publishing, Birmingham, England, 2015.
10. Python: Available at https://docs.python.org/
11. Beautiful Soup: Available at https://www.crummy. com/software/BeautifulSoup/bs4/doc/
12. myrealtrip: Availabe at https://www.myrealtrip.com/
13. Tourtips: Availabe at http://www.tourtips.com/
14. EarthTory: Availabe at https://www.earthtory.com/

접수일: 2019년 9월 26일, 심사일: 2019년 9월 26일, 게재확정일: 2019년 9월 27일