한국산업정보학회논문지 제8권 제3호(2003. 9)

동적 스케줄링 기반 웹 크롤러의 성능분석[†]
(Preliminary Performance Evaluation of a Web Crawler with Dynamic Scheduling Support)[†]

Yong-Doo Lee*, Soo-Hwan Chae**

Abstract A web crawler is used widely in a variety of Internet applications such as search engines. As the Internet continues to grow, high performance web crawlers become more essential. Crawl scheduling which manages the allocation of web pages to each process for downloading documents is one of the important issues. In this paper, we identify issues that are important and challenging in the crawl scheduling. To address the issues, we propose a dynamic crawl scheduling framework and subsequently a system architecture for a web crawler subject to the framework. This paper presents the architecture of a web crawler with dynamic scheduling support. The result of our preliminary performance evaluation made to the proposed crawler architecture is also presented.

Key Words: Web Crawler, Dynamic Scheduling, Performance Evaluation

1. Introduction

A web crawler is a software that collects web pages over the Internet. A crawler starts off its operation by taking a set of seed URLs in a queue. Retrieving a URL from the queue, the crawler downloads the web page and then extracts URLs from the content of the downloaded pages. These URLs are again placed into the queue. By repeating the above process, the crawler downloads web pages until the queue becomes empty[3,7,8,9]. The pages downloaded by a crawler are used by a number of applications such as search engines and web cache.

To get a high performance, the design and implementation of a crawler is usually made by using mutiprocesses rather than a single process. Indeed to meet the demand for high performance, it is essential to choose the multiprocesses based approach. In commercial sectors. there already exist a number of multiprocess-based web crawlers. Compelled to put the highest priority on getting fast prototypes, almost all the implementations of these crawlers hardly consider the tradeoff which arises in the design space of mutiprocess-based crawlers. Little effort are made in the scientific research to analyze them and any sound result is not yet published in open literature.

In the design of the multiprocess-based crawler, it should be clearly addressed to ensure efficient crawl scheduling[1,5,6]. Crawl scheduling is to assign a set of URLs to each process for download[4]. The design space for crawl scheduling is composed of a number

† This research was supported by IRC(Internet Information Retrieval Research Center) in Hankuk Aviation University. IRC is a Kyounggi-Province Regional Research Center designated by Korea Science and Engineering foundation and Ministry of Science & Technology

* School of Computer and Communication Engineering, Daegu University

** School of Computer and Communication Engineering, Hankuk Aviation University

of alternatives with respect to centralized versus distributed in the scheduling decision and static versus dynamic in partitioning URLs assigned for each crawl process to download.

Almost all the crawlers used for commercial search engines are implemented via multiprocesses[2]. In these crawlers, the task to download web pages as well as the task to extract the URLs inside newly downloaded web pages are respectively implemented by using multiprocesses. Their operations are controlled under a *fork-join* principle in their process creation and coordination. In the stage of downloading web pages, a set of downloader processes, each of which is assigned with a disjoint set of URLs, are created, *i.e., forked* and then each process conducts downloading web pages assigned to itself. When all the processes finish the download task, *i.e., joined,* the crawler starts off the extraction stage controlled in the fashion similar to the downloading stage. With some auxiliary operations such as uniqueness inspections, the above process are repeated in the execution of the crawler.

From the crawl scheduling perspective, the above model is regarded as taking a static scheduling because a downloader process is assigned with a URL set statically before its creation. The degree of multiprocessing reflected by the number of processes is set adequately for sufficient utilization of system resources under which the crawler is executed.

A crawler with static crawl scheduling suffers from low system utilization. The time spent for downloading a web page shows a wide range of variance affected by the target host on which the page resides. This is due particularly to the network, host load as well as the page availability on the host. Even though the same number of URLs is assigned to each process, some processes thus finish the download very lately than the others. It is clear that the idle time caused by the unbalance of process termination hinders the sufficient utilization of system resources and causes to decrease the performance of the crawler. Internally, it means that the degree of multiprocessing set for the optimal system utilization is not maintained under the static scheduling. To overcome this situation, it is essential to explore a dynamic crawl scheduling because any static crawl scheduling scheme with reasonable complexity in its implementation entails the idle time.

In our study, we focus on the comparative performance between static and dynamic scheduling schemes. Initially, we believe that dynamic crawl scheduling presents a number of benefits; The system utilization becomes higher by reducing the idle time caused from unbalanced workload which occurs frequently under static crawl scheduling.

Our work contributes to the design of high performance crawlers with the dynamic crawl scheduling, more precisely as below:

- We identify the drawbacks of the static crawl scheduling and suggest that some dynamic scheduling support must be adopted to get higher performance.
- We explore a dynamic crawl scheduling scheme and provide a system architecture for a crawler with the dynamic scheduling support.
- Even it is preliminary, we show, through the experimental performance evaluation, the clue that dynamic scheduling support is essential for high performance crawlers.

The rest of the paper is organized as follows. In section 2, the proposed crawler architecture is explained. In section 3, the result from a preliminary performance evaluation is presented. Finally, concluding remarks are provided.

2. Crawler Architecture with Dynamic Scheduling Support

This section presents the proposed dynamic scheduling scheme and its model architecture for a crawler. The implementation issues and directions are briefly discussed.

2.1 Dynamic Scheduling Scheme

The benefit of the mutiprocess-based implementation of a web crawler is to get higher performance. To this end, it should be guaranteed during the execution of the crawler to maintain the degree of multiprocessing set to adequately beforehand subject to the hardware resources of the system. In earlier discussion, it is pointed out that the inherent drawback of the static crawling scheduling policy prevents the crawler from maintaining the sufficient degree of multiprocessing set for the platform hardware.

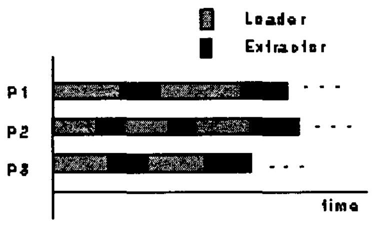

Figure. 1. Example dynamic schedule

To guarantee a sufficient degree of multiprocessing for crawling, we choose a dynamic scheduling approach. The underlying rationale of our approach is that at runtime any available process actively fetches URLs from the queue whenever the queue is not empty with URLs to download. It entails that the processes in our dynamic scheduling scheme are *persistent* during the whole runtime, while they keeps busy downloading web pages as long as the queue is not empty.

As an implementation model for the above dynamic scheduling support, we use a shared memory used in the general multiprocessing paradigm. Queues accessed by multiple processes are implemented by the shared memory accessed via 'lock" and "release" principles to guarantee mutual exclusive accesses. Whenever a process finishes downloading a URL, it takes a new URL from the queue while conforming to the shared memory access.

2.2 Proposed Crawler Architecture

In the previous section, we discuss the rationale of the dynamic crawl scheduling with a brief description of the implementation model. The crawler with dynamic scheduling capability has conceptually a different architectural model from the crawler with static scheduling.

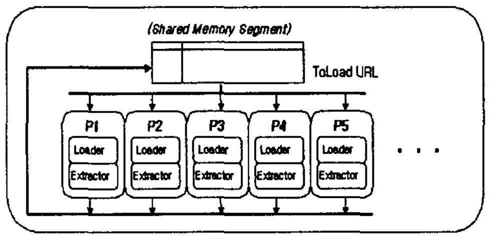

Figure. 2. Conceptual model of the proposed web crawler

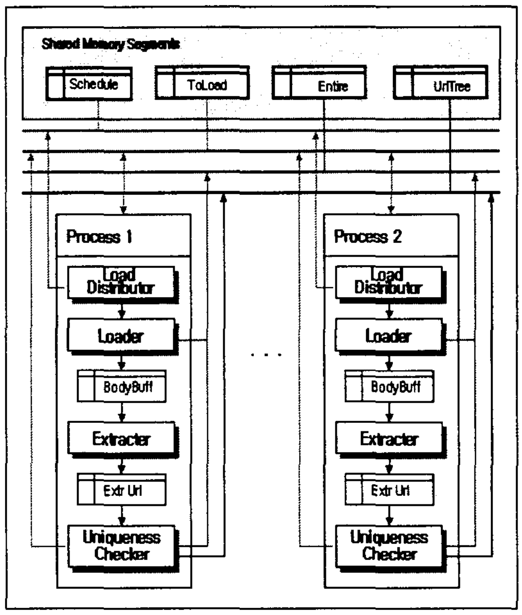

Figure. 3. Architecture of the crawler with the proposed dynamic scheduling support

The conceptual architectural and operational model for the crawler of our proposal is depicted in Figure 2. The loader module downloads web pages. The extractor module takes off URLs inside a web page. Different from fork-join multiprocess model as for static crawl scheduling where the download and the extractor module are implemented respectively via multiple homogeneous processes, the loader and the extractor module in the proposed model are merged in a single process which will be replicated during runtime for multiprocessing. In the operational perspective, the proposed model differ from the fork-join model in that processes in the proposed model are persistent until the end of crawling. The communication between processes are made via shared memories.

A detail description of our model is depicted in Figure 3. Apart from the loader and the extractor module, the uniqueness checker inspects if an extracted URL is already downloaded. Finally the load distribution module prepares and then allocates a new URL set to each process of the loader module. This architecture is composed of a set of shared entities which include queues for URLs to download and a system URL tree used for uniqueness inspection. Processing entities are composed of a set of homogeneous and persistent processes. These processes have both the download and extraction functions. This merge of the two functions in a process is possible due to the capability of shared

<Table 1> Collection rate

Excution time (Min) / Model	10	20	30	40	45
Dynamic Scheduling (Collected pages	10,449	19,563	30,333	34,516	36,022
Static Scheduling (Collected pages)	1805	2886	4391	5078	5713
Speedup	5.8	6.8	6.9	6.8	6.3

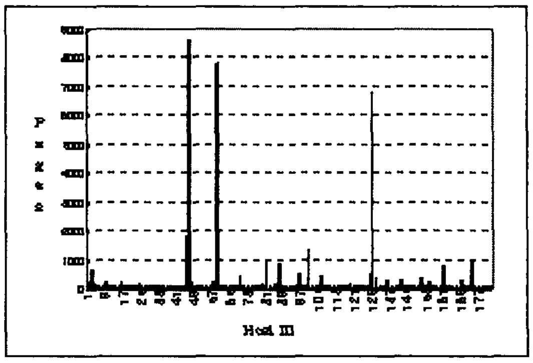

Figure 4 Page count of each host

accesses to the system URL tree which is not possible normal fork-join based static scheduling crawlers.As described previously, the multiple processes communicate and cooperate with one another by using shared memory. All the queues shared by more than one process are thus implement in the shared memory accessed under the restriction of mutual exclusion.

3. Preliminary Performance Result

This chapter presents the result of the performance evaluation which are ongoing for more thorough analysis.

<Table 2> Collection time

Excution time (Min) / Model	2	4	6	6	10	12	14	16	18	20
Dynamic Scheduling (Collected pages	3.7	6.0	8.7	11.(	13.)	14.8	16.5	18.(	19.5	21.7
Static Scheduling (Collected pages)	13.9	27.3	47.(	65.(	80.(	99.(	1200	1450	1750	2100
Speedup	3.8	4.6	5.4	5.9	6.1	6.9	7.3	8.0	8.9	9.7

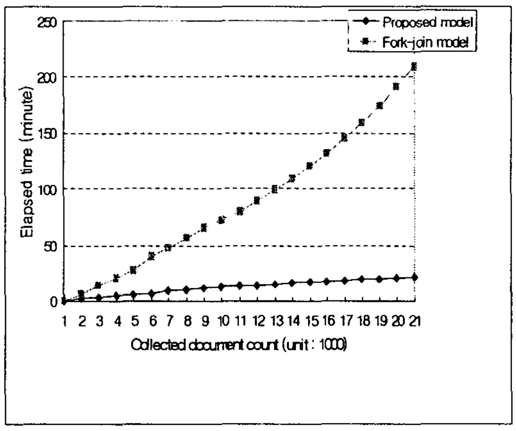

Figure. 5. Comparison of collection time over page counts

The performance metrics of crawlers consists mainly of collection efficiency and collection rate. Collection efficiency addresses how much completely a crawler collects the target web pages. It is represented by the percentage of the number of web pages, that a crawler successfully downloaded, over the number of target web pages to collect. Collection rate expresses how fast a web crawler to collect the target web pages. This metric is represented by average number of pages that a crawler collect during a unit time. We focused on the collection rate because the scope of this study is centered around enhancing the collection speed.

The experimental setup is composed of a two implementations of web crawlers: a fork-join based multiprocess crawler with static scheduling and a shared memory based multiprocess crawler with dynamic scheduling. Both implementations use the same crawler kernel in order to ensure the reasonable performance comparison. The code of the two implementation are of with the same quality except for the code relevant to crawl scheduling. The platform hardware on which crawlers run is a Dell PC server with two Pentium III CPU of 733 MHz, 2 Gbytes main memory and 90 Gbyteshard disk array. The network card in the server is 1 10M Ethernet adaptor and connects to the external network with T3 bandwidth. The operating system of the server is Linux RedHat 7.1.

In the experiment, we choose 21,000 URLs on 177 hosts from web sites of 17 universities as the target web pages to be downloaded by crawlers. The number of web pages of each host is depicted in Figure 4. In the execution of web crawlers, the number of processes which reflect the degree of multiprocessing is set to 5. The collection rate of each crawler is calculated by the arithmetic mean value for the ten collection rates which are obtained by executing a crawler ten times.

Figure 5 shows the result of the collection rate respectively with dynamic scheduling support and with static scheduling support. The average elapsed times of the dynamically scheduled crawler and those of the statically scheduled crawler are shown from 1K pages to 21K pages. The speedup of the dynamically scheduled crawler and those of the statically scheduled crawler is 9.5 for collecting 21K documents. This value indicates that the dynamic scheduling support is crucial for high performance web crawlers.

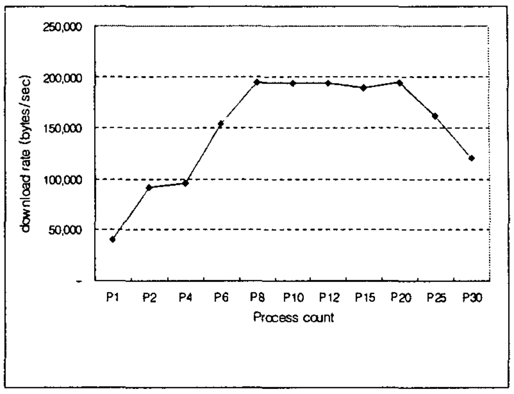

Figure. 6. Raw download rate of the proposed model

Figure 6 shows the raw download rates with respect to the number of processes. The raw download rate is measured by bytes per second. This figure presents how the degree of multiprocessing of our model affects the over all collection rate of the system. According to the figure, as the number of processes increases from 1 to 8, almost linear increase is shown for the raw download rate. For the range of 8 to 20 processes, the raw download rate does not increase. Finally, the raw download rate decrease when the number of processes becomes larger than 20.

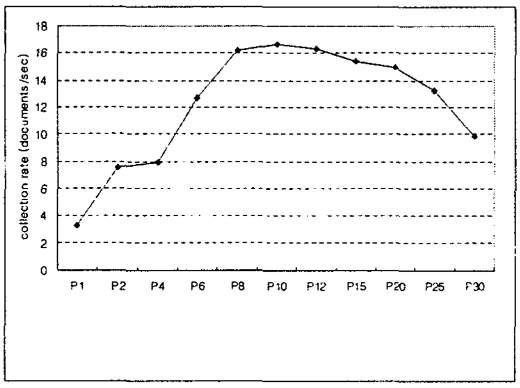

Figure. 7. Collection rate of the proposed model

Figure 7 shows the document collection rate with respect to the number of processes. The document collection rate is measured by average number of documents per second. Provided the number of documents collected by a crawler is sufficiently large,the raw download rate of a crawler reflects the raw document collection rate, even if the size of documents are differ one another. Indeed, the document collection rate in figure 8 almost matches with the raw download rate in figure 7. According to the figure, as the number of processes increases from 1 to 8, almost linear increase is shown for the collection rate, as does for the raw download rate. For the range of 8 to 12 processes, the download rate shows a slight increase, while it decrease very slightly during 12 to 20 processes. Finally, the collection rate decrease drastically when the number of processes becomes larger than 20.

Overall, both the values of the download rate and the collection rate indicates that the proposed model get a reasonable speedup with respect to the increase of the degree of multiprocessing. However, it also indicates that that the degree of the multiprocessing should be carefully adjusted subject to the resources of the system on which the crawler is executed.

As previously pointed out the experiment result presented in this paperis preliminary. We are currently investigating the operational behavior of a web crawler by analyzing the effect of timeout values with respectto the HTTP connection for page download, page head download, and page contents download. Also, we are evaluating the histogram of times spent for the major components internal to a crawler such as time spent for network activities and for the extraction of URLs as well as for uniqueness inspection, etc. These results will guide us to understand the internal behavior of the web crawler and thus to provide us with more opportunities of further optimization for higher performance.

4. Concluding Remarks

This paper presents a new approach to the implementation of a multiprocess crawler. The core of the study is to provide an efficient dynamic scheduling support mechanism. To this end, we propose a new web crawler architecture with a shared memory based dynamic scheduling support. The main benefit of this proposal is to guarantee the degree of multiprocessing during the execution of the crawler, which in trun yields higher collection rate. This is due to the removal of the idle time which occurs in the crawler model operated under the static scheduling paradigm. The result of our preliminary performance evaluation is very promising with speedup of 9.5. This indicates that the dynamic scheduling support is crucial for the high performance web crawler. More through study on the performance evaluation remains as further study.

References

[1] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery, In the 8th International World Wide web Conference, 1999

[2] J. Cho, H. Garcia-Molina. The evolution of the web and the implications for an incremental crawler. In proc. of the 26th international conference on Very Large Databases, 2000

[3] J. Cho and H. Garcia-Molina. Synchronizing database to improve freshness. In Proc of the 2000 ACM SIGMOD, 2000

[4] J. Cho and H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. Computers networks and ISDN systems, 30:161-172, 1998

[5] M. Diligenti, F. Coetzee, S. Lawrence, C. Giles, and M. Gori. Focused crawling using context graphs. In Proc. of the 26th international conference on Very Large Databases, 2000

[6] A. Heydon and M. Najork Mercator: A scalable, extensible web crawler. World Wide web 2(4):219-229, December 1999

[7] L. Page and S. Brin. The anatomy of a large-scale hypertextual web search engine. In Proc. of the 7th World-Wide web Conference, 1998

[8] R. Miller and K Bharat. SPHNIX: a framework for creating personal, site-specific web crawlers. In Proc. of the 7th World-Wide web Conference, 1998

[9] Robots exclusion protocol. http://infor.webcrawler.com/mak/projects/robots/exclusion.html

이 용 두 (Yong-Doo Lee)

1975: Hankuk Aviation Univ. (Communication. Eng. B.S.)
1982: Yeungnam University (Computer Eng. M.S.)
1995: Hankuk Aviation Univ. (Computer Eng. Ph.D.)
1982 ~ : Daegu University, Professor
Interest: GRID computing, Computer Architecture

채 수 환 (Soo-Hwan Chae)
1973: Hankuk Aviation Univ. (Communication Eng. B.S.)
1985: Univ of Alabama (Computer Science. M.S.)
1988: Univ of Alabama (Computer Science. Ph.D)
1989 ~ : Hankuk Aviation Univ., Professor
Interest: Computer Architecture, Distributed Computing