Scheduling Web Crawl for Better Performance and Quality

Fengyun Cao Dongming Jiang Jaswinder Pal Singh

{fcao, dj, jps}@cs.princeton.edu
Department of Computer Science, Princeton University
Princeton, NJ 08540, USA

Abstract

Web crawler is an essential component of search engines, data mining and other Internet applications. Scheduling Web pages to be downloaded is an important aspect of crawling. Previous research on Web crawl focused on optimizing either crawl speed or quality of the Web pages downloaded. While both metrics are important, scheduling using one of them alone is insufficient and can bias or hurt overall crawl process. This paper explores the design space of crawl scheduling to balance performance and quality factors and optimize the global crawl efficiency. We design a network-efficient scheduling framework and use it to evaluate various scheduling strategies. We also define a new scheduling algorithm that factor both network performance and Web page quality into scheduling decision-making. Real world experiments clearly demonstrate the effectiveness of the two-level scheduling scheme and the new algorithm in improving overall crawl efficiency. Experiments also show that crawl-scheduling design can always be optimized based on full understanding of application properties.

1. Introduction

Web crawler is an essential component of search engines, data mining and other Internet applications. It recursively downloads Web pages into local storage, as shown in Figure 1. The operations can be briefly described as the following four steps:

a. Take a set of *seed URLs* as initial *task URLs*.
b. Select a URL from task URLs and download the page from the Web.
c. Extract hyperlinks contained in the downloaded page and if desirable add the new URLs into task URLs in strategic order.
d. Repeat step b and c until either task URLs become empty or the crawl is stopped by the application.

The strategy to determine the order of task URLs to crawl is *crawl scheduling*. Given a time window *T,* different scheduling strategies can lead to very different sets of Web pages crawled in *T*.

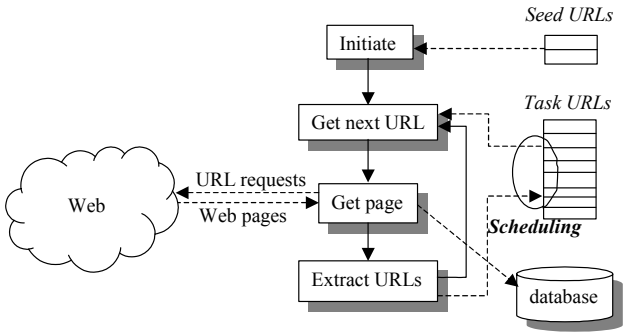


Figure 1. Operational model of a web crawler.
(Control flows are shown by solid lines and data flows are shown by dotted lines.)

Due to the explosive growth of the World Wide Web, crawling even a significant part of all pages available on the Internet has become very challenging: the typical time for a major search engine to crawl one billion pages is more than one week [14][15]; In the meanwhile, a large number of new pages have been created, and many crawled

pages have changed [2][9]. Therefore, a Web crawler can only access those Web pages that are scheduled to crawl early.

In this paper, we define the overall efficiency of a Web crawler as the total utility of the content crawled in limited time. For the reason stated above, this metric is of great importance and generality. To achieve the overall efficiency, the crawler faces two challenges: it should download pages at high speed, and also selectively prioritize crawling the most valuable Web pages. We call these two optimization metrics performance and quality metrics. These metrics have been studied mostly in separate.

While both metrics are important, scheduling by one of them alone can dramatically bias the crawl. For instance, a crawler guided by performance metric only can be trapped into a well-connected group of junk pages, while crawling for quality only may block the progress by waiting for responses from extremely slow servers that happen to have a few high quality pages. These scenarios are unacceptable from the perspective of global efficiency.

In this paper, we explore the design space of Web crawl scheduling to optimize the global crawl efficiency. In next section we briefly review related research works on Web crawl. In Section 3, we propose a novel two-level scheduling architecture. In Section 4, we define three scheduling algorithms to represent the strategies of breadth-first crawl and crawl with performance-only and quality-only optimization. We also design a new comprehensive strategy, called *crawl-ability based crawl*, which factors both performance and quality in scheduling decisions. We implement a Web crawler with the two-level scheduling and experiment with the algorithms designed. In Section 5, experimental results are presented and analyzed. The algorithms are shown to be highly effective in improving crawl under the corresponding metrics. Specifically, the new strategy proposed successfully improves the overall crawl efficiency more than all the others. Finally we conclude with future research directions in Section 6.

2. Related Work

Literatures on Web crawl can be roughly divided into two categories:

Major search engines [4][15] designed high performance crawlers to download large number of pages per unit time. Although page rankings such as PageRank [4][21] are of great importance to the search application, it is not clear whether they are considered in search engine crawls, and if so, how.

Other research works have focused on scheduling Web pages (before these pages are downloaded, they are represented by the URLs in the crawl task list) by their quality rankings: pages more valuable to the application are ranked higher and are scheduled to download before the less-valuable ones. Page qualities are defined and computed in the context of specific applications. In [6], a *focused crawler* seeks out pages that are relevant to a pre-defined set of topics. In [8], web pages that are referenced by hyperlinks in many other pages are deemed important and given high rankings. Other page quality measurements include page freshness [9][10] or even arbitrary predicates defined by users [1]. In [11], the URL ordering is studied within the context of multiple parallel crawl processes. While these works are shown to be highly effective in selectively downloading good-quality pages early, it is not clear how they perform with respect to the time constraint, and whether incorporating performance aspect of the pages into the priority ranking would further improve the global efficiency of the crawl. In fact, many experiments were simulation of "virtual crawls" on a set of local Web page collections, and therefore it is not clear how the algorithms would interact with network and performance impacts when used in real crawls.

3. Architecture Design

In this section, we present our design of the scheduling framework. We first review the network protocol features to understand that it is important for the scheduler to measure the value and cost of a Web page with explicit awareness of its server status and network conditions. Based on this understanding, we propose the idea of scheduling crawl tasks on both Web server (TCP connection) level and Web page (individual URL) level.

3.1 Network protocol implication

A Web crawler downloads Web pages though HTTP protocol on top of TCP connections. Traditional HTTP 1.0 protocol has well-known performance problems [20]: A separate TCP connection is opened for each HTTP request and closed after receiving the response. Because HTTP messages are relatively small, a large fraction of network traffic is TCP management packets. Also, the TCP connections are always running in their slow start stage, leading

to underutilization of network capacity. To address these problems, HTTP1.1 protocol offers two performance optimization techniques: *persistent connection*, which allows TCP connections to remain open for consecutive object transfers between the same server/client peer, and *pipelining*, which allows client to send more HTTP requests before receiving response for the previous ones.

Clearly, a high performance crawler should take advantage of HTTP 1.1 optimizations. To reduce connection management overhead and slow start effect, a crawler should be encouraged to *batch* downloading of multiple Web pages over one TCP connection. The time cost of downloading a Web page should include not only the time to send URL request and receive the respond from the server, but also the time for opening and closing connections if such operations are needed. Therefore, a URL from an already connected Web server should be considered faster or "cheaper" to get than one from a similar speed but unconnected server. In all, the performance crawl scheduling decisions should not be made strictly on a per URL base, but rather within a TCP connection context.

3.2 Two-level scheduling

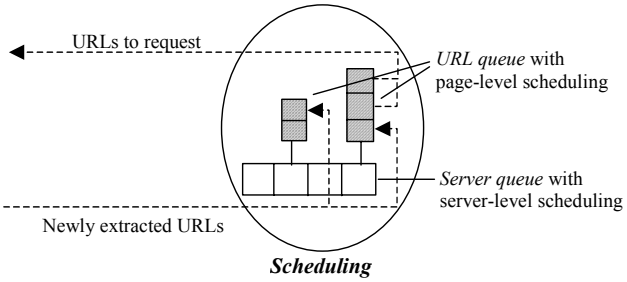


Figure 2. Task URLs are organized into two levels for two-level scheduling.

To accommodate underlying network performance characteristics, we propose a novel scheduling architecture of two levels: scheduling is done on both URL level and Web server level. In Figure 2 we redraw the scheduling circle shown in Figure 1. Task URLs, i.e. URLs that have been discovered but not downloaded yet, are ranked and ordered in URL queues associated with their Web servers. Web servers in turn are ranked and sorted in a server queue. Both URL queues and server queues are implemented as priority queues, with ranking functions plugged in based on specific crawl strategies. We will present four such strategies, together with their ranking algorithms on both levels in the next Section.

The scheduling process works as follows: to achieve network and computation parallelism, the crawler crawls from multiple Web servers at the same time. The maximum number of simultaneously open TCP connections remains at a system-specific, constant level. When an old connection is closed, the crawler opens a new connection to the top-ranked unconnected server and downloads top URLs from that server. Downloaded Web pages are parsed and the extracted URLs are ranked and inserted into appropriate URL queues. URL queue updates may also trigger ranking update of corresponding Web servers. To avoid excessive connection overhead and the danger of swapping between servers, server-level scheduling is implemented in a non-preemptive way: an established connection would not be closed simply because an unconnected server gets a higher ranking than the connected one.

A Web server usually allows transferring of no more than a fixed number of Web pages within one connection. This number is a parameter the server configures and will be further discussed in Section 5. The number of Web pages the crawler actually downloads over one connection is a parameter configured by the crawler to balance performance optimization with scheduling granularity. Setting this number to one implies scheduling with strict URL ordering and loss of HTTP 1.1 performance benefits. In our work, we let the crawler greedily download as many task URLs as the server allows. More finely tuning of this parameter is an interesting future work.

4. Scheduling Strategies

In this Section, we define two-level ranking algorithms for four crawl strategies with different optimization metrics. Specifically, these strategies represent the cases of breadth-first crawl, crawling for high page fetching

speed, for good page quality, and for high aggregate page quality achieved in limited time. In Section 5, we will present experimental results of crawling based on these strategies.

4.1 Breadth-first scheduling

In breadth-first scheduling, Web pages are crawled in the order of the discovery of their URLs. Therefore, we order both URL queues and the server queue in First-In-First-Out (FIFO) fashion. Breadth-first strategy involves no explicit scheduling preference but has been shown to yield pages with good link-structure qualities [19]. Therefore, we include it as a meaningful baseline study.

4.2 Performance based scheduling

Performance based scheduling aims at crawling as many pages as possible in unit time. It prioritizes crawling from fast connections and do not distinguish page qualities.

A Web server *s* is ranked for its next TCP connection (without loss of generality, we assume it is the *i*th connection of server *s*) as:

$$R(s,i) = \frac{P(s,i)}{T(s,i)}$$

where *P(s, i)* is the number of pages to be downloaded from server *s* over the *i*th TCP connection, and *T(s, i)* is the time to download them. Servers with high rankings are expected to serve large number of pages in short time. Note that to avoid overloading any Web server, the crawler opens no more than one connection to a server at any time.

Apparently, both *P(s, i)* and *R(s, i)* can only be known after actually downloading the pages over the connection. For scheduling purpose we need to estimate the value *a priori*. As server performance and network conditions change constantly, this can be challenging. We define

$$P(s,i) = \min\{RequestsPerConnection(s,i), TaskURLs(s,i)\}$$

$$T(s,i) = 2 * ConnectionTime(s,i) + \frac{P(s,i) * ResponseTime(s,i)}{p}$$

P(s, i) is set as the smaller of two numbers: *ReqestsPerConnection(s, i)* is the number of URL requests server *s* serves per connection, as estimated from the previous connections from the same server. For a new server, the number is set to a default value of 50. *TaskURLs(s, i)* is the length of the URL queue on server *s*.

T(s, i) consists of time to open and close the connection (*2 * ConnectionTime(s, i)*) and the total transfer time to fetch *P(s, i)* Web pages. *p* is a saving factor for Web servers that support HTTP 1.1 pipelining. In our experiments, *p* is set to 1.2. *ConnectionTime* and *ResponseTime* are adaptively estimated as:

$$ConnectionTime(s,i) = ConnectionTime(s,i-1) * \alpha + MeasureConnectionTime(s,i-1) * (1-\alpha)$$

$$ResponseTime(s,i) = ResponseTime(s,i-1) * \alpha + MeasureResponseTime(s,i-1) * (1-\alpha)$$

where *ConnectionTime(s, i-1)* and *ResponseTime(s, i-1)* are old estimates, and *MeasureConnectionTime* and *MeasureResponseTime* are new observations. We set parameter α to 0.8 to smooth the estimations.

In performance based scheduling, server queue is ordered by *R(s, i)* of each server, while URL are ordered as FIFO in each URL queue.

4.3 Quality based scheduling

Quality based strategy simulates strict URL ordering by prioritize crawling good quality URLs before lower ones. For simplicity and generality, we assume Web page quality is computed by another application-specific module and is transparent to scheduling. We also assume that the quality of a Web page is available as soon as its URL is discovered and stays constant during the scheduling phase. Although in some applications the value of a page changes as more information are gathered in the crawl process, such change is orthogonal to the scheduling scheme studied in this paper and is not considered for the sake of clarity.

In quality based scheduling, URL queues are ordered by page qualities. Web servers are ranked by the quality of the best URL on the server:

$$R(s,i) = Quality(Top\_entry\_in\_the\_URL\_queue)$$

In this way, the crawler always eagerly crawls the best-quality URLs.

4.4 Crawl-ability based scheduling

Finally, we design a scheduling algorithm that combines performance and quality factors in scheduling rankings. We define *crawl-ability* of a server to be:

$$C(s,i) = \frac{AQ(s,i)}{T(s,i)}$$

where

$$AQ(s,i) = \sum_{p=1}^{P(s,i)} Quality(Entry\_p\_in\_URL\_queue)$$

and *P(s, i)* and *T(s, i)* are as defined in Section 4.2.

C(s, i) reflects the time-average page quality the crawler expects to download from Server *s*. High crawl-ability indicates the server can serve good-quality pages fast. Note that in this paper we use simple algebraic summation for page quality aggregation. Other definitions of aggregation are possible and would be an interesting future work.

In crawl-ability based strategy, servers are ranked by their crawl-ability, and URLs are ranked by quality in each URL queue.

5. Experiments and Evaluations

In this Section, we present our implementation of a Web crawler with two-level scheduling and the crawl experience of the four scheduling strategies defined in Section 4.

5.1 Experiment setup

5.1.1 Programming model

A high-performance Web crawler crawls from multiple Web servers at the same time. There are two programming alternatives to achieve this parallelism: single thread event driven (STED) model as used in [4], and multithread model as used in [15]. We choose STED model to avoid context switch and thread synchronization overhead. The disadvantage is a possibly more complicated program structure. The crawler is implemented in a single thread with an event loop. Asynchronous network events are registered with the *select* system call of UNIX and handled by callback functions. To avoid the well-documented DNS lookup bottleneck, we enable asynchronous DNS lookup by adding another thread to resolve IP address from hostname.

The downloaded Web pages are parsed and the extracted URLs are ranked and inserted into appropriated URL queues. The pages are then discarded. A list of hash values of URLs recently seen are maintained so that the same Web page won't be downloaded again within a short period of time.

5.1.2 Hardware environment

Our crawler is a C program running on a Pentium III 700 MHz PC with 512M RAM and 100Mbps Ethernet access. The Ethernet of our department is connected to the campus network through 100Mbps gateway, which in turn connects to Internet 2 backbone through 45 Mbps gateway [13]. Some of our network protocol management functions are modified from the protocol modules provided by LibWWW library from World Wide Web Consortium (W3C) [22].

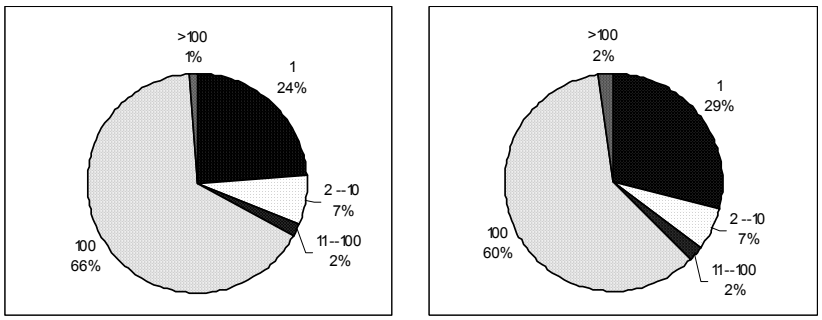
5.1.3 Data sets

To obtain comparable results of various scheduling strategies, the crawl experiments need to satisfy the following requirements:

(1) All crawls should visit the same set of pages.
(2) Each page should have a consistent quality value in different crawls.
(3) Network traffic and Web server load during each crawl should be comparable.

After a variety of tests, we found that the university Web servers are more stable and friendly to crawlers, with less dynamically generated or frequently changing pages, than most commercial Web sites. We choose a large university Web domain on the other coast from us, Stanford.edu, to be our test set. The domain has more than 30,000 Web pages on more than 500 Web servers. We understand that the test set is small compared to the typical workload a commercial Web crawler will experience; however, it allows repeatable experiments and detailed analysis of the characteristics of the scheduling algorithms. Because the scheduling scheme proposed in this paper is relatively new, we believe such detailed analysis and deep understanding to be very important. Pushing our experiments toward larger scale is one important future work.

We have also examined various Web server properties that may affect scheduling effectiveness. For instance, we collected configurations of number of URL requests supported per HTTP connection by the Web servers. Distribution of this configuration number among 500 Stanford servers is shown in Figure 3(a): 24% of the servers only serve one request per connection, which implies that they are running HTTP 1.0; 66% of servers serve 100 requests per connection, which is a typical configuration for HTTP 1.1. The distribution indicates that a large number of Web servers are using HTTP 1.1 protocol, and this proportion is expected to continue to grow in the future. Therefore, the technique of exploiting HTTP 1.1 performance optimization is expected to be of great importance. In Figure 3(b), we present the same configuration of 3500 Web servers listed in Yahoo Web directory. The distribution is shown to be very similar to Figure 3(a). From this perspective, we believe that Stanford.edu domain can be a reasonable representative of the Web at large for crawl scheduling studies.



(a) For 500 Web servers from Stanford.edu domain

(b) For 3500 Web servers from Yahoo directory (4/5 .com, 1/5 others)

Figure 3. Proportion of Web servers supporting certain number of URL requests per HTTP connection.

5.2 Breadth-first crawl performance analysis

5.2.1 Breadth-first crawl on the test set

To understand the crawler performance and the impact of network conditions, we first launch breadth-first crawls with various protocol configurations. In particular, we crawled using HTTP 1.0, HTTP 1.1 single mode (with persistent connection but not pipelining technique), and HTTP 1.1 pipelining mode (with both persistent connection and pipelining techniques). We also varied the maximum number of simultaneously open TCP connections the crawler maintains.

The results are shown in Figure 4. With same level of parallelism, crawling in HTTP 1.1 mode is more than twice as fast as using HTTP 1.0. Pipelining further improves performance but the magnitude is not as large as expected. Looking into detailed trace data, we found that this is because the saving of round-trip delay between consecutive HTTP requests is small compared to the time the Web server takes to respond to requests, which may include time for disk access and serving other connections. Figure 4 also shows that crawl throughput increases with the number of open TCP connections. The improvement, however, disappears when crawling from more than 96 connections at the same time.

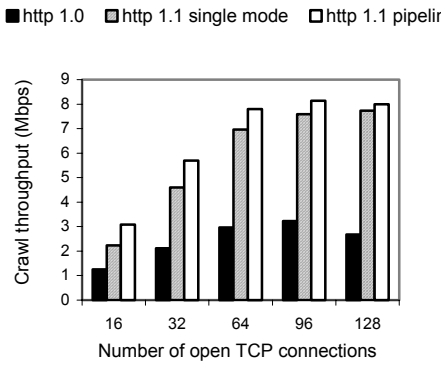


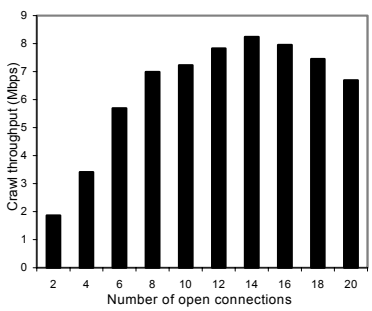
Figure 4. Performance of breadth-first crawls with various configurations.

5.2.2 Crawling synthetic test set of fast servers

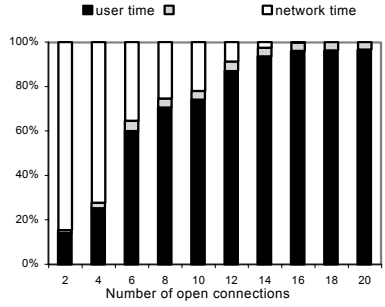
To fully understand the crawler behavior, we designed a synthetic test set consisting of 20 "fast" Stanford Web servers, which all have response time of less than100 milliseconds and serve 100 URL requests per TCP connection. The crawler then crawls only from these servers to avoid possible impacts from slow servers that may drag the overall crawl performance. The results of crawling fast servers are shown in Figure 5.

In Figure 5(a), the maximum crawl throughput of 8.4 Mbps is reached using only 14 TCP connections, compared with 96 connections in Figure 4. Figure 5(b) shows the time breakdown of each bar in Figure 5(a). The total crawl time is divided into three parts: system time is CPU time spent in operating system kernel space, user time is CPU time spent in user space, and the network time is the time that CPU is idle and waiting for the asynchronous I/O operations to become ready. It is easy to understand that CPU idle time decreases with more network connections. With 16 TCP connections, the CPU becomes saturated. After that, more network connections actually hurt the performance by adding more management overhead and consuming more memory space. Moreover, we found that CPU saturation reduces crawler's responsiveness – a large number of responses from servers are lost, and some connections are even killed by Web servers due to timeout.

In all, the synthetic study tells us that the bottleneck of the crawler is at CPU, and increasing network parallelism cannot always improve performance. Instead, comparison with Section 5.2.1 shows that a different task set can lead to very different crawl performance.



(a) Performance of crawling fast servers



(b) Time break down for crawling fast servers

Figure 5. Performance analysis for the synthetic test.

5.3 Evaluation of scheduling strategies

Next, we present the crawl results of the four scheduling strategies designed in Section 4. To avoid CPU saturation, the crawler maintains no more than 64 TCP open connections at the same time. The evaluation metrics are:

(1) Crawl speed, as number of pages downloaded over time.
(2) Page quality, as aggregate page quality over the number of pages downloaded.
(3) Overall efficiency, as aggregate page quality downloaded over time.

As discussed before, we believe the metric (3) is of most practical importance. Metric (1) and (2) may also be of concern to various applications. We experiment with two page quality definitions, PageRank [21] and Zipf distribution [3][18], because of their practical interests. As stated in Section 4.3, the page qualities are pre-computed and stay fixed during the crawl.

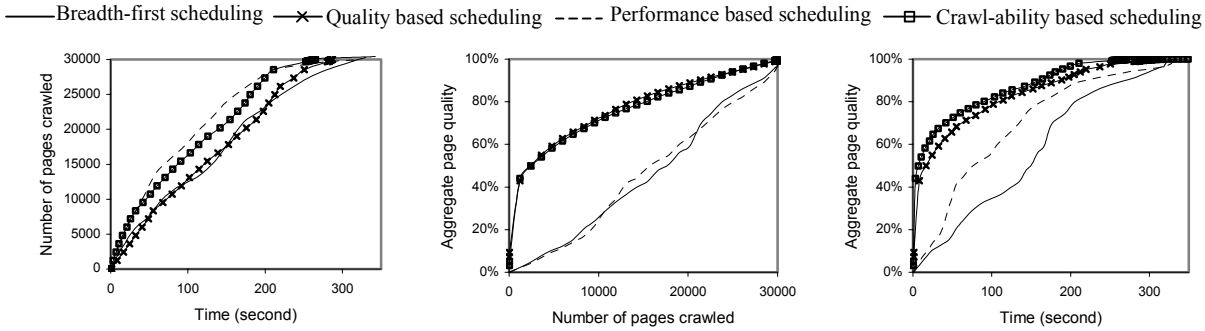
5.3.1 Page quality based on PageRank

PageRank is a popular page quality metric [4][8][19][21]. Google uses PageRank to rank search results for user queries. [8] has shown that PageRank can also be a good crawl metric to discover "important" URLs, such as the ones with high back link counts or PageRanks.

PageRank of one page is recursively defined as the weighted sum of the PageRank of pages that contain hyperlinks pointing toward this page. To formalize, let u be a Web page. Then let F_u be the set of pages u points to and B_u be the set of pages that point to u. Let $C_u = |F_u|$ be the number of links from u and let $0 < d < 1$ be a damping factor. Suppose there are T total pages on the Web. Then the PageRank R for u is defined as:

$$R(u) = d \cdot \frac{1}{T} + (1-d) \cdot \sum_{v \in B_u} \frac{R(v)}{C_v}$$

The crawl results of the four scheduling strategies are presented in Figure 6.



(a) Speed of page fetching.

(b) Aggregate page quality over pages crawled.

(c) Aggregate page quality over time.

Figure 6. Scheduling effectiveness with page quality based on PageRank.

Figure 6(a) shows that performance-based and crawl-ability based scheduling achieve higher crawl throughput than the other two strategies, especially in early stage of the crawls. Specifically, performance-based strategy has downloaded 50% more pages than breadth-first crawl at the midpoint of the crawl. This shows that both strategies are effective in prioritize crawling from fast Web servers. Compared with performance-based strategy, crawl-ability based strategy is penalized for also considering page qualities in server rankings. As discussed before, a crawler cannot touch all pages on the Web and can be seen as always running at "early stage". Therefore, a scheduling strategy that improves early performance would in fact improve performance of the crawler in its limited lifetime.

Interestingly, under the page quality metric in Figure 6(b), quality based scheduling and crawl-ability based scheduling outperform the other two strategies by a much larger scale. They selectively download high-quality pages at the beginning of the crawl and obtain more than 80% of the total page quality before half of the pages are downloaded. Looking into the detailed trace data, we found the significance of the improvement can be (partially) attributed to a special feature of PageRarnk: studies [7][16] found that Web pages reference pages from the same server more often than those from different servers. We call this "server locality" of the hyperlinks. When we use a citation-based page quality measure such as PageRank, pages on the same server tend to obtain similar quality values. Therefore, by aggressively seeking for the very best page, the quality-based strategy is likely to schedule a server with many good pages, which will be also downloaded over the same connection. At the same time, given the skewed distribution of page qualities, scheduling servers with better qualities becomes very important.

Figure 6(c) shows that crawl-ability based scheduling performs best under the comprehensive metric of overall efficiency. Quality-based scheduling closely follows the second. Both of them are able to obtain more than 80% of the total page quality in less than 1/3 of the time needed to crawl all pages. Performance-based scheduling also effectively outperforms breadth-first crawl by about 50% at the midpoint of the crawl.

Results in Figure 6 show that the scheduling strategies designed are highly effective in improving crawl behavior under their optimization metrics. However, we have also found that the magnitude of improvement is related to the special "server locality" property of PageRank. To understand how the scheduling algorithms would behave with page quality metrics without such property, we then experiment with another page quality metric.

5.3.2 Page quality based on Zipf distribution

Zipf's law is a classic observation that frequency of occurrence of some event (P), as a function of the ranking (i) when the ranking is determined by the above frequency of occurrence, is a power-law function with the exponent a close to unity:

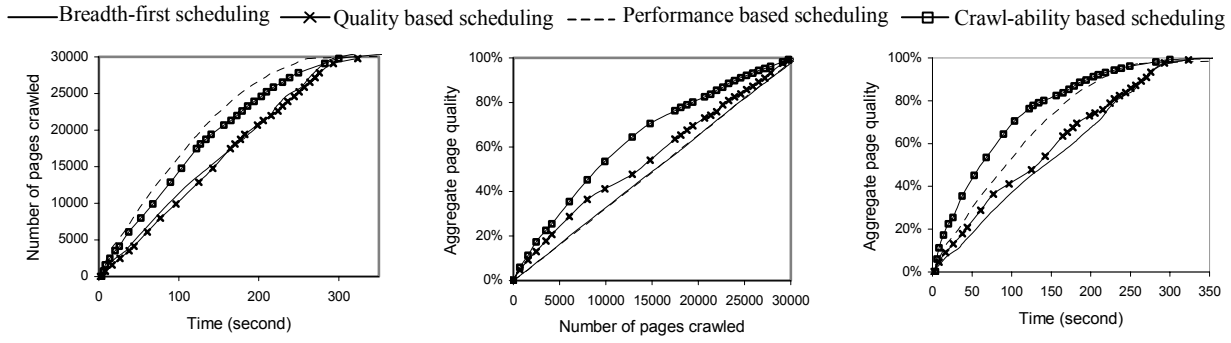
$$P_i = \frac{1}{i^{\alpha}}$$

Recent studies ([3, 17]) found that Zipf distribution characterizes Web access patterns and has great implication for Internet applications. In this experiment, we randomly assign quality values to all Web pages in the test set so that the overall quality distribution follows Zipf distribution (a is set to 1). Unlike PageRank, page qualities assigned in this way are independent of each other and "server locality" no longer exists.

Figure 7 shows crawl results for this quality metric. As expected, under the performance metric that is orthogonal to page quality measures, the results in Figure 7(a) are similar to Figure 6(a). It is rather counter-intuitive to see that crawl-ability based scheduling even outperforms quality-based scheduling in Figure 7(b). The reason is that when quality-based crawl schedules to crawl the top-quality page, unlike in Section 5.3.1, the other pages downloaded in the same connection no longer have the tendency to also possess high quality. In contrast, crawl-ability based scheduling ranks a server by total page qualities to download in one connection and tends to schedule a server with many good pages.

Figure 7(c) shows that crawl-ability based scheduling again outperforms all other strategies in optimizing overall crawl efficiency. It achieves more than 60% of the total page quality in less than 1/3 of the time to crawl all pages, and 80% before the midpoint. Scheduling based on performance or quality alone also outperforms breadth-first crawl, but with less scale. Without the server locality property of page qualities, quality-based scheduling lost its advantage to performance-based scheduling.

Because of its consistent effectiveness in improving overall crawl efficiency, we expect crawl-ability based scheduling to a good strategy to balance the performance and quality aspects of Web crawl.



(a) Speed of page fetching.

(b) Aggregate page quality over pages crawled.

(c) Aggregate page quality over time.

Figure 7. Scheduling effectiveness with page quality based on Zipf distribution.

6. Conclusions and Future Work

In this paper, we studied how to schedule Web crawls for better performance and quality. Based on the understanding of underlying network performance characteristics, we proposed a novel scheduling framework that

schedules crawl tasks on both Web server level and individual URL level. The framework is flexible to accommodate various scheduling strategies. We then defined two-level ranking algorithms for strategies of breadth-first crawl, crawl for performance optimization, and crawl for page quality optimization. We also defined *crawl-ability* as a new ranking metric that combines performance and quality factors into scheduling decision-makings. We implemented a real Web crawler with the two-level scheduling and evaluated the four scheduling strategies. Experimental results show that all the intelligent strategies effectively improve the crawl under their optimization goals, especially in the early stage of the crawls. Among the strategies, crawl-ability based scheduling consistently outperformed others in achieving high aggregate page quality in limited time. Therefore, we expect crawl-ability to be a good scheduling metric to combine and balance the performance and quality aspects in crawl scheduling.

In our experiments, we found that crawl behavior can be sensitive to application-specific factors such as network attributes or page quality definitions. Therefore, crawl-scheduling design can always be optimized based on full understanding of relevant application properties.

There are a number of interesting extensions to our research. In section 4, we presented one way to estimate Web server performance and network conditions. It would be interesting to further explore the tradeoff between computation overhead and estimation accuracy. In crawl-ability scheduling algorithm, we calculated aggregation of page quality using algebraic summation. For applications in which page qualities are dependent of each other, this may not best reflect the total utility value of a set of Web pages. The implication of such application properties for crawl scheduling and the way to best utilize them deserves further investigation. We are also pushing our study towards larger scale.

References

[1] C. Aggarwal, F. Al-Garawi, P. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *Proceedings of the 9th International World Wide Web Conference (WWW9)*, 2000

[2] A. Arasu, J. Cho, L. Molina, et al. Searching the Web. In *ACM Transactions on Internet Technology*, 1(1), June 2001

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching, and Zipf-like distributions: Evidence, and Implications, In *Proceedings of IEEE INFOCOM*, 1999.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th World Wide Web Conference (WWW7)*, 1998.

[5] M. Burner. Crawling towards Eternity: Building an archive of the World Wide Web. Web Techniques Magazine, 2(5), May 1997

[6] S. Chakrabarti and M. Berg, B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of the 8th International World Wide Web Conference (WWW8)*, 1999

[7] S. Chakrabarti et al. Mining the Web's link structure. In *IEEE Computer*, 32(8):60-67, August 1999.

[8] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, 1998

[9] J. Cho, H. Molina. The evolution of the Web and implications for an incremental crawler. *In Proceedings of 26th International Conference on Very Large Databases (VLDB)*, 2000

[10] J. Cho, H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the ACM SIGMOD Conference*, 2000

[11] J. Cho and H. Garcia-Molina. Parallel crawlers. *In Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[12] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, M. Gori, Focused crawling using context graphs, Proceedings of International Conference on Very Large Databases (VLDB), 2000, pp. 527-534

[13] Global Internet Service at Princeton University, http://www.net.princeton.edu/internet.html

[14] Google Inc. press release: “Google launches world’s largest search engine.” June 26, 2000. Available at http://www.google.com/press/pressrel/pressrelease26.html

[15] A. Heydon, M. Najork. Mercator: A scalable, extensible Web crawler. In *Proceedings of the 8th World Wide Web Conference (WWW8)*, 1999

[16] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Procedings of ACM-SIAM Symposium on Discrete Algorithms*, 1998

[17] S. Lawrence, L. Giles. Accessibility of information on the Web. In *Nature*, Vol. 400, pg.107, July 8, 1999

[18] M. Levene and G. Loizou. Zipf's law for web surfers. In Knowledge and Information Systems, an International Journal, 3, (2001), 120-129.

[19] M. Najork, J. L. Wiener. Breadth-First search crawling yields high-quality pages. In *Proceedings of the 10th International World Wide Web Conference (WWW10)*, 2001

[20] H. Nielsen, J. Gettys, et al. Network performance effects of HTTP 1.1, CSS1, and PNG. In *Proceedings of ACM SIGCOMM*, Sep 1997

[21] L. Page, S. Brin, R. Motwani, Winograd. The PageRank citation ranking: Bring order to the Web. *Technical Report, Stanford University*, 1998.

[22] World Wide Web Consortium (W3C) http://www.w3.org