Workload-Aware Web Crawling and Server Workload Detection

Shaozhi Ye
Department of Electronic Engineering
Tsinghua University
Beijing, P.R.China
ys@compass.net.edu.cn

Guohan Lu
Department of Electronic Engineering
Tsinghua University
Beijing,P.R.China
lvguohan@tsinghua.org.cn

Xing Li
Department of Electronic Engineering
Tsinghua University
Beijing,P.R.China
xing@cernet.edu.cn

ABSTRACT

With the development of search engines, more and more web crawlers are used to gather web pages. The rising crawling traffic has brought the concern that crawlers may impact web sites. On the other hand, more efficient crawling strategy is required for the coverage and freshness of search engine index. In this paper, crawlers of several major search engines are analyzed using one six-months access log of a busy web site. Surprisingly, we find that none of these crawlers pays attention to the workload of web site, which may hurt both server performance and crawling efficiency. Based on this observation, a server workload-aware crawling strategy is proposed. By measuring the web service time with a hybrid back-to-back packets pair, server workload is detected on the client side, thus crawler can adapt its crawling speed to web server. The experiment results show the power of our workload detection approach. This paper concludes with a discussion of future work on server workload detection and its applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; H.3.5 [**Online Information Services**]: Web-based services; H.4.3 [**Communication Applications**]: Information browsers

General Terms

Algorithms, Measurement, Performance

Keywords

Web Crawler, Server Workload, Back-to-back Packets

1. INTRODUCTION

World Wide Web(WWW) has grown rapidly in the past decade. Both the web page volume and web traffic are increasing explosively. Now large scale web search engines index billions of web pages. Google[1], the largest one, has indexed 4.28 billions web pages at Mar.1, 2004.

Search engines use web crawlers, which are also called web spiders or robots, to gather web pages. To keep the index coverage, crawlers have to gather as many web pages as possible, while to keep the index freshness, crawlers have to get the index updated as soon as possible. The tremendous volume of web pages poses challenges to web crawlers. Clearly all major search engines have highly optimized crawlers, although few technical details have been reported for commercial privacy.

On the other hand, the rising crawler traffic has brought the concern that crawlers will greatly hurt the performance of web sites. A popular web site serves thousands of client requests at the same time when busy hours. If these powerful crawlers are not configured properly, they may generate a lot of requests to the already heavy-loaded web servers in a short time, even result in server crash. And the crawlers can not archive a high download speed when the web server is already heavy-loaded. So it is important for crawlers to adapt their downloading speed to the workload of web sites.

This paper investigates the crawling behaviors of several major search engines by analyzing a six-months access log of a busy web site. To our surprise, none of these crawlers pays attention to the server workload. And some of them even do more crawling when server is heavy-loaded, which may impair both server performance and crawling efficiency. To our knowledge, there is no previous work reporting this phenomena. Based on this observation, a server workload-aware crawling strategy is proposed. We use the difference of HTTP response time and ICMP response time to estimate the service time of web server, which is a metric of server workload. Thus crawlers can do adaptive crawling suited for web sites workload. The experiment results show the power of our workload detection method.

The rest of this paper is organized as follows. Section 2 reviews the previous work. Section 3 reports our observations on server workload and crawling behaviors by mining real web access log. Then a server workload detection method using back-to-back packets is proposed in Section 4, results of simulation experiment and real web site measurement are presented in Section 5. We discusses future work on server workload detection and its possible applications in Section 6, then concludes this paper with Section 7.

2. PREVIOUS WORK

Much work has been done on web crawlers, including crawler architecture[1, 2, 3, 4], page selection strategy[5, 6, 7], up-

[1]http://www.google.com

Asia Pacific Advanced Network 2004, 2-7 July 2004, Cairns, Australia.
Network Research Workshop 2004, 2-7 July 2004, Cairns, Australia.

dating crawling[8, 9, 10, 11, 12] and dynamic web crawling[13]. But little has been discussed on server workload issue. This section reviews the previous work aware of it.

In 1993 and 1994, the early time of web, there were occasions that crawlers visited web servers where they were not welcome for various reasons. Thus in Jun 1994, *Robots Exclusion Protocol*[14] was suggested for webmasters to indicate crawlers which parts of the servers should not be accessed. And later in May 1996, *Robots META Tag*[15] was proposed for HTML authors to tell crawlers if a document could be indexed or used to harvest more links.

At the same time, the concern that crawlers will overload servers also raises with the develop of crawlers[16]. As far as we know, [17] is the first published crawler in the literature which is aware of the impact of server.

In [4], the notion *politeness* is used for server-aware crawling and the details are described as *weak politeness guarantee* and *strong politeness guarantee*. [3] suggests one domain-based throttling strategy to distribute the crawling job in domains, which breaks the locality of crawling and thus slows the crawler.

However, few factors on server side have been considered, except for some naive approaches to avoid overloading the site, such as setting interval time of retrieval and limiting the number of crawling threads. These methods do not adapt for the change of server workload. The problem turns out to be more serious with the growth of web traffic, as shown in the next section.

3. SERVER WORKLOAD AND CRAWLING BEHAVIORS OBSERVATION

In this section, we investigate one six-months access log of a busy web site, the homepage of CERNET[2]. The log is provided with a non-disclosure agreement for protection of the privacy of end-users visiting the site.

3.1 Overview of Log Data

This web site is built up by a cluster of several servers balanced with dns and their logs are merged together. The access log is logged by *Apache Http Server*[3] in its *Combined Log Format*. Each line of the access log contains information of a single request. One example log entry is like the following:

127.0.0.1 - - [10/Oct/2000:13:55:36 -0700] "GET /apache.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"

From each log entry, it is possible to determine the ip address of the client making the request, the time that the request was made, the name of the requested document, and the version of http protocol it used. The entry also provides information about the server's response to this request, such as the response code identifying the status of request process, the number of bytes transmitted by the server. In the *Combined Log Format*, there are two additional fields. One is called *Referrer*, which gives the URL that the client reports having been referred from, and the other is *User-Agent*, which gives the identification information that the client's browser reports about itself. In the example entry, the *Referrer* is "http://www.example.com/start.html" and the *User-Agent* is "Mozilla/4.08 [en] (Win98; I ;Nav)." Refer [18] for a more detailed description of *Combined Log Format*.

[2]http://www.edu.cn
[3]http://httpd.apache.org

Table 1 shows the summery of this log.

Table 1: Summery of CERNET Access Log

Access Log Start Date	Aug.13 2003
Access Log End Date	Jan.31 2004
Total Log Days[a]	155
Total Requests	1,178,270,826
Average Requests/Day	7,601,747
Total Bytes Transferred(GB)	5,367.4
Average Bytes Transferred/Day(GB)	34.6

[a]We lost 13-days log in the total 168-days period due to various errors.

As shown in Table 1, it is a large log set. The largest web log set used in the published work is [19], which contains 1.35 billions requests collected in three months, just a little larger than ours.

3.2 Server Workload

Then we investigate the server workload. We static the average requests and traffic(transferred bytes) distributions in different hours of a day.

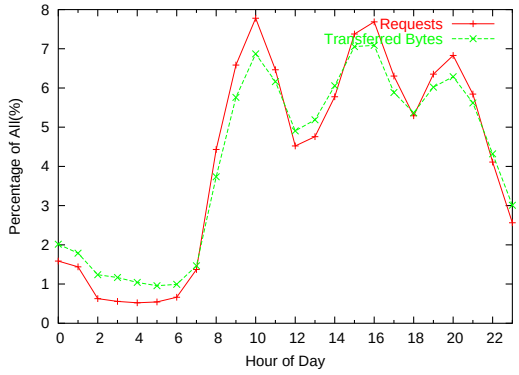

Figure 1: Average Server Workload in Different Hours

As shown in Figure 1, there are more requests in day time than night. And the distribution of traffic(transferred bytes) is similar with the distribution of requests. Therefore we get the conclusion that the server has a much heavier workload in the day than the night, which coincides with our intuition. For example, there are about 2.5% requests in five *silent* hours, from 2:00pm to 7:00pm, while there are about 8% requests in just one *busy* hour, from 10:00pm to 11:00pm. In other words, there are about 15 times more requests in busy hours than those of silent ones. In some peak time, the gap might be widened greatly.

3.3 Crawler Workload

Table 2: Top-Five Crawlers

Crawler	Search Engine	Requests	Transferred Size(GB)
Googlebot	http://www.google.com	4,079,459	43.1
Slurp	http://www.inktomi.com	1,703,670	14.7
Baidu Spider	http://www.baidu.com	1,569,575	22.2
IBM	http://www.almaden.ibm.com/webfountain/	849,455	4.15
Scooter	http://www.altavista.com	685,665	5.48

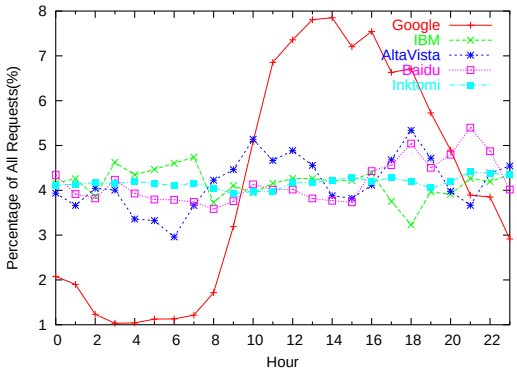

Figure 2: Crawler Requests in Different Hours

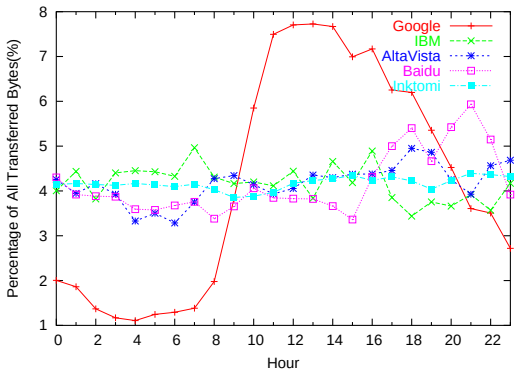

Figure 3: Crawler Traffic in Different Hours

While there is already some analysis of crawlers behaivior[20], we focus on the workload of crawlers in different hours of a day.

We have to identify requests of crawlers from all the requests in the access log, which is not an easy job. Fortunately, almost all the large scale commercial search engines fill the *User-Agent* field in the requests of their crawlers with the trouble contact information, such as the following record from Google:

"Googlebot/2.1 (+http://www.googlebot.com/bot.html)"

We check all the *User-Agent* fields in the access log and extract crawler records, then select the most frequently appearing crawlers. Whois service is used to verify whether their ip addresses belong to the corresponding search engines. The top-five crawlers, which visit the web site most frequently, are listed in Table 2.

Figure 2 shows the average requests distribution by hours of each crawler, and Figure 3 shows their average traffic, transferred size in bytes, in different hours.

To our surprise, none of these major commercial crawlers pays attention to the busy hours of web site. Most of them crawl at about the same speed, and some of them even do more crawling at day time than night.

One noticeable crawler is Google's Googlebot, which crawls much more at day time. The most possible reason might be that Googlebot is configured to avoid the busy hours but it does not check the time zone of the web site. When it is day time in China, where CERNET homepage is located, it is night in United States, where Google is located. So the naive strategy, which makes crawlers active in night and inactive in the day, may not adapt for the world-wide crawling.

Another noticeable crawler is Baidu Spider, which belongs to a large scale search engine in China. It also does more crawling at busy time without aware of server workload.

To our knowledge, there is no previous work reporting this phenomena that most crawlers are not aware of server busy time.

4. SERVER WORKLOAD DETECTION

4.1 Server Workload-Aware Crawling

Now popular web sites serve thousands of clients at the same time when busy hours. Crawlers may impact the already heavy-loaded servers, as shown in the previous section. On the other hand, the limited bandwidth and capacity of server also greatly slows the crawlers in busy hours, when the crawlers have to compete with about 20 or more times users than in silent hours.

Some search engines crawl in the night to avoid the busy time, while most web pages are updated during day time especially news pages. To keep up with the change of pages, search engines can not only crawl in the night. For example, in our log, it is observed that Google crawls the web site at an average speed of 0.2 page/second. Considering there are so many search engines working independently, crawlers might overload web server if they are not aware of server workload. Moreover, politeness crawling behavior is always welcomed by webmasters.

If the server workload is detected, crawlers can avoid the busy hours and crawl more efficiently in the lightly occupied time. Thus we propose our server workload-aware crawling strategy: detect the server workload and adjust the request

rate based on server workload. When busy hours, crawlers should use a slower request rate, and in silent hours, a faster request rate can be used for faster crawling.

4.2 HTTP Response Delay

To implement our crawling strategy, the server workload has to be detected first. Although it is convenient to measure the workload on server side, as we do in the previous section, there is no direct way to detect on client side.

HTTP Response Delay can be used as a metric of server workload. It is defined as the interval between the time when one HTTP request is sent and the time when the fist packet of the corresponding HTTP response is received. It can also be considered as the *Round Trip Time*(RTT) of HTTP, corresponding to the notion of RTT in TCP.

If the server is heavy-loaded, it takes more time to process the HTTP requests of clients. At the same time, due to the limit of server bandwidth, it also takes more time to transmit the packets in busy hours. Thus *HTTP Response Delay* will be larger when server is heavy-loaded. So it can be used as a metric of server workload.

However, *HTTP Response Delay* is composed of network delay and server HTTP transaction delay, we cannot distinguish whether its increase is caused by network or server workload.

4.3 Revised HTTP Response Delay

To reduce the network affection as much as possible, back-to-back packets[21] are used in our measurement. A back-to-back packets pair are two packets sent one immediately after the other, which means the interval of their sent time is small. Thus these two packets are expected to experience almost the same network condition and are probably processed by routers and destination host one immediately after the other. TCP packets pair and ICMP packets pair have been used in priori work on network bottleneck detection and bandwidth measurement[22, 23]. But the packets pair we used in this paper are not made up by two same packets, both TCP packets or both ICMP packets.

Our pair consists of a TCP packet, which sends HTTP request, and an *ICMP Echo* packet[24]. In most cases, an *ICMP Echo* packet will be processed and echoed by kernel in the IP layer as soon as it is received, while an HTTP request will be sent to the application layer for web service processing. *HTTP Response Delay* is supposed to be greater than *ICMP Response Delay* for the former may be composed of I/O operation or dynamic content generation. And when heavy loaded, web service time, the process time of this request, is expected to be greater, thus it can be used as a metric of server workload.

At client side, the interval arrival time of responses of our back-to-back packets can be used to estimate the web service time at server side. We define this interval time as *Revised HTTP Response Delay*. Figure 4 shows the measurement based on our hybrid back-to-back packets pair.

Actually our proposed *Revised HTTP Response Delay* is the difference of *HTTP Response Delay* and *ICMP Response Delay*, which is a more precise metric for server workload than the original *HTTP Response Delay*, shown as following:

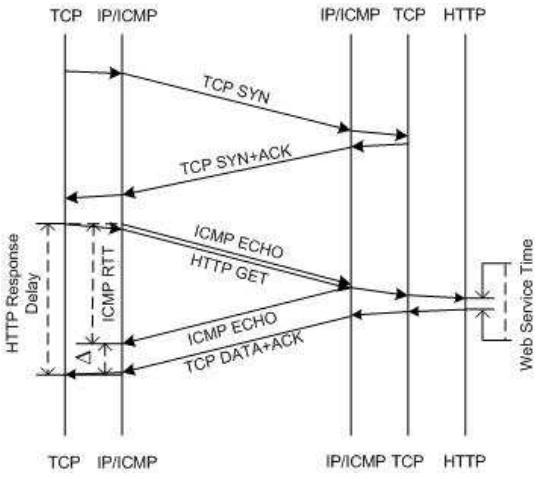

Figure 4: Back-to-Back ICMP and HTTP Packets Pair Measurement

$$\begin{aligned} RTT_{Revised\ HTTP} &= RTT_{HTTP} - RTT_{ICMP} \\ &\approx Web\ Service\ Time \end{aligned}$$

However, the network still affects our measurement. One of the most significant affection factors is packet loss and retransmission of TCP, which results in a large delay.[4] Reordering of TCP may also affect our experiment. We have to discard these retransmission or reordering packets.

5. EXPERIMENT RESULTS

In this section, we present the results of simulation experiment and real web site measurement.

5.1 Simulation Experiment

First we design a simulation experiment to validate our workload detection method. This experiment involves three hosts, shown as Figure 5.

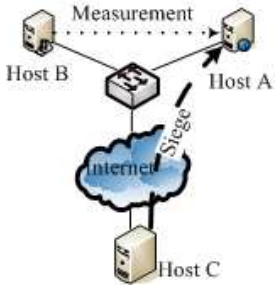

Figure 5: Siege Experiment

[4]In our experiment, the delay dominated by retransmission will be greater than 200 milliseconds, comparing with the average delay of 30 milliseconds.

Host A acts as web server, and its detailed configuration is AMD Athlon 1GHz processor/256M memory/100M Ethernet. Its operation system is Debian/Linux[5] with kernel version 2.4.18, and the web server is Apache 1.3.26. This server contains 11,555 web pages mirrored from the homepage of CERNET, 145M bytes in size.

Host B is used for detecting the delays of host A, which is on the same switch of host A. Thus the measurement will be little affected by Internet traffic. We implement our workload detection algorithm and measure the *Revised HTTP Response Delay*of host A every second.

Host C, one high performance pc, is used to stress host A with *siege*[6], an HTTP regression testing and benchmark utility. Although it is not in the same subnet with host A, the link between host A and C is good enough to impact host A.[7]

Host C is configured to visit host A with 25 processes. Each process reads URLs from a URL table which contains all the URLs of host A, and then gets the corresponding web pages from host A.

There are two modes in the experiment. One is moderate mode, and the other is turbulent mode. When host C is working in moderate mode, every process will wait one second after downloading a page from host A, and then request for another one. When it is working in turbulent mode, every process will request for the next page as soon as it finishes one. Therefore the server workload is rather different in these two modes. Actually, our experiment results show that there are 2.4 processes concurrently requesting to host A in moderate mode, while 24.7 processes in turbulent mode.

Our experiment starts with the moderate mode, and after 62 seconds, it is switched to the turbulent mode. The experiment result[8] is shown as Figure 6. Both *ICMP Response Delay* and *Revised HTTP Response Delay* increase with the workload. The maximum *Revised HTTP Delay* is 148.8ms and the maximum *ICMP Response Delay* is 8.2ms.

There is a major peak of *Revised HTTP Response Delay* between the 62nd and 83rd second with a mean of 39.7ms, which is probably caused by disk I/O, when the client starts to request for a lot of web pages. After most of these files are loaded in to memory/cache, *Revised HTTP Response Delay* becomes smaller. On the other hand, there is no peak at the curve of *ICMP Response Delay* at the beginning period of turbulent mode. The mean of this period is 1.0ms. *Revised HTTP Response Delay* seems to be more sensitive compared with *ICMP Response Delay*, for it corresponds to the workload of web service, which coincides with our assumption in the previous section.

It is also observed that the measured delay is busty when heavy-loaded, although the workload at server side keeps unchanged after the switch of stress mode. Actually, when heavy-loaded, the delay is sensitive to the server status when the request arrives. Although the workload dominates the delay, several other factors may also affect it, such as the cache status of disk and net adapter.

[5]http://www.debian.org
[6]http://joedog.org/siege/
[7]We get a 100Mbps bandwidth in our experiment.
[8]The delays greater than 20ms are set to 20ms for a better view.

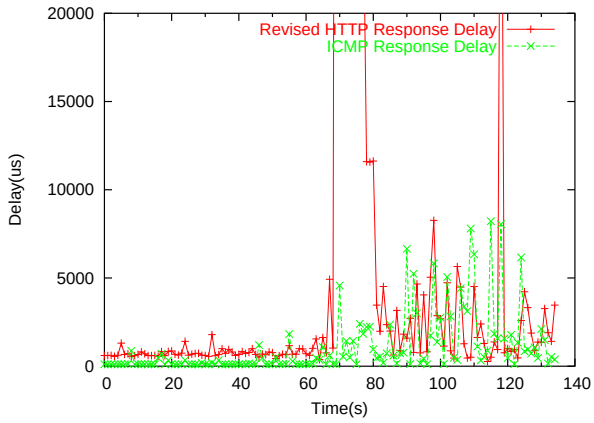

Figure 6: Simulation Experiment Result

5.2 Measurement on Real Web Site

Then we take a measurement on the homepage of CERNET. This experiment lasts for 24 hours by detecting the delays three times at the beginning of every minute. The median of three delays is used to allay the affection of network traffic. The delays larger than 200ms are discarded for they are most probably caused by retransmission of TCP in such a measurement environment. The result is shown as Figure 7. We also get the access log of that day, the workload and traffic are shown as Figure 8 and Figure 9 respectively for comparison.

To evaluate how well our method estimates the server workload, we calculate the coefficient of correlation between *Revised HTTP Response Delay* and requests and correlation between *Revised HTTP Response Delay* and transferred bytes. To smooth the bursts, the mean of past w samples are used, where w is the size of the window used for averaging. In other words, we compute the coefficient of correlation between the means of the past w samples of delays and requests/transferred bytes. Figure 11 shows the result. The coefficient of correlation between *Revised HTTP Delay* and requests is about 0.84 when the averaging window size is 20, which is rather small window comparing with the total 1440 sample points. Figure 10 shows the *Revised HTTP Delay* when w=20. The result indicates our method measures the server workload well.

In this experiment, the workload of server seems to be more affected by the number of requests than by the traffic, which indicates that the bottleneck of the server might be maintenance of file descriptors and sockets. On the other hand, it also means that additional crawler requests might affect the server performance even though the crawling traffic is small, which shows the significance of our server-aware crawling strategy.

6. FUTURE WORK

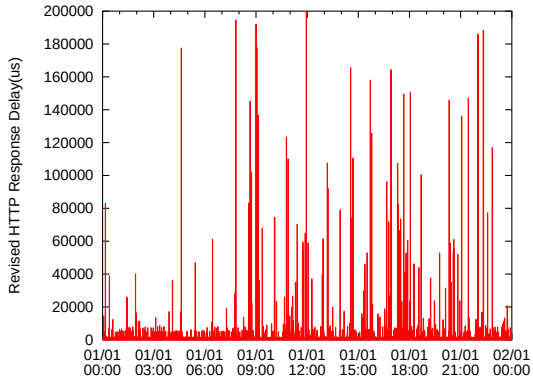

Figure 7: Revised HTTP Response Delay

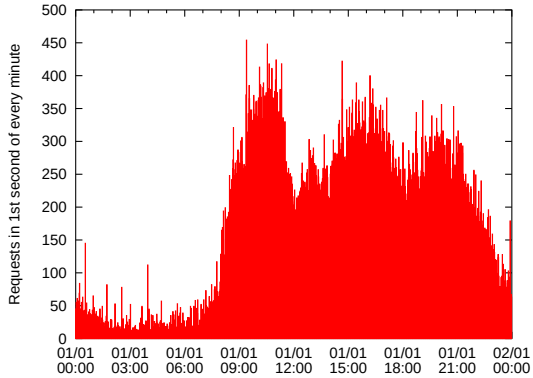

Figure 8: Requests

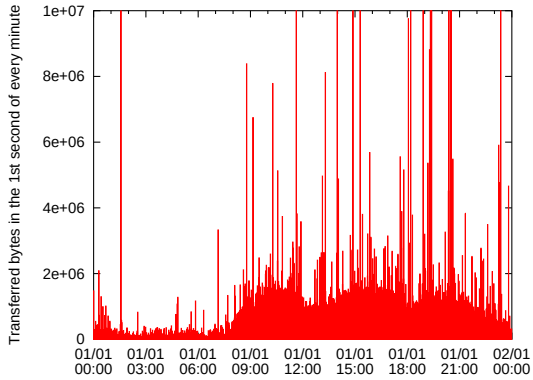

Figure 9: Transferred Bytes

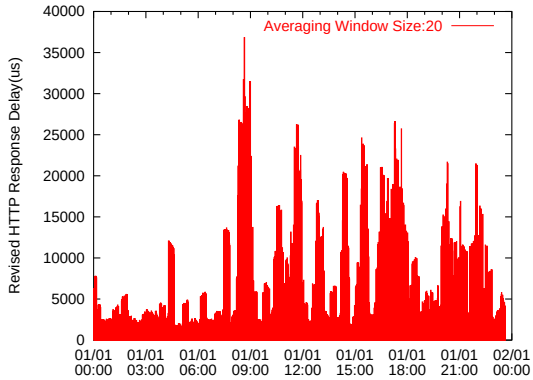

Figure 10: Revised HTTP Response Delay by Averaging Window

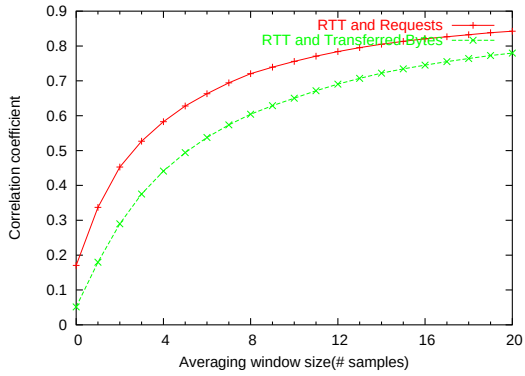

Figure 11: Correlation Coefficients computed over Various Averaging Window Sizes

Both simulation experiment results and real network measurement show the efficiency of our method in the previous section. While it is still a coarse measurement. E.g. the increase of delays with workload is not smooth but rather busty. There is much work to do for a better knowledge of how *Revised HTTP Delay* changes with the workload. We are planning to do more precise measurements on both client and server sides and try to explore the coherence between them in the future work.

Server workload detection also has many other possible applications. It can help a server in a cache cluster to determine which of other servers to request data first. And in Content Delivery Network, it is also important to know which server is less busy for decision of job distribution. In many cases, the servers cooperating in one cluster have no direct method to know the workload of others, such as P2P networks. Server workload detection may take an important part in these scenarios.

7. CONCLUSIONS

In summary, we believe we make the following contributions in this paper:

- We observe that most commercial web crawlers do not pay attention to the server workload. As far as we know, there is no previous study reporting it.
- Based on this observation, we argue that in order to be polite in crawling, web crawlers should employ some server workload-aware crawling strategies.
- Then a novel hybrid back-to-back packets pair is proposed for server workload detection.

8. ACKNOWLEDGMENTS

The authors would like to thank Shuguang Zhang of CERNET for providing the web log.

9. REFERENCES

[1] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

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

[3] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In *Proceedings of International Conference on Data Engineering(ICDE)*, 2002.

[4] M. Najork and A. Heydon. On high-performance web crawling. Technical report, Compaq Systems Research Center, September 2001.

[5] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[6] S. Chakrabarti, M. Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

[7] M. Najork and J. L. Wiener. Breadth-First Crawling Yields High-Quality Pages. In *Proceedings of the 10th International World Wide Web Conference*, pages 114–118, Hong Kong, May 2001. Elsevier Science.

[8] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases(VLDB)*, 2000.

[9] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology(TOIT)*, 3:256–290, 2000.

[10] J. Edwards, K. S. McCurley, and J. A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th International World Wide Web Conference*, pages 106–113, 2001.

[11] J.L. Wolf, M.S. Squillante, and P.S. Yu. Optimal crawling strategies for web search engines. In *Proceedings of the 11th International World Wide Web Conference*, pages 136–147, New York, NY, USA, 2002. ACM Press.

[12] J. Cho and A. Ntoulas. Effective change detection using sampling. In *Proceedings of the 28th International Conference on Very Large Databases(VLDB)*, September 2002.

[13] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Databases(VLDB)*, 2001.

[14] M. Koster. Robots exclusion protocol, Jun 1994. Http://www.robotstxt.org/wc/norobots.html.

[15] M. Mauldin and M. Schwartz. Spidering BOF report. Technical report, Distributed Indexing/Searching Workshop, May 1996.

[16] M. Koster. Robots in the web:threat or treat? *ConneXions*, 9(4), Apr 1995.

[17] D. Eichmann. The RBSE Spider - Balancing Effective Search Against Web Load. In *Proceedings of the First World Wide Web Conference(WWW94)*, 1994.

[18] http://httpd.apache.org/docs/logs.html#accesslog.

[19] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-1999-35, HP Laboratories, Palo Alto, 1999.

[20] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou. Characterizing crawler behavior from web server access logs. In *Proceedings of E-Commerce and Web Technologies, 4th International Conference*, pages 369–378, Sep 2003.

[21] S. Keshav. Packet-pair flow control. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey, USA, 1994.

[22] R. S. Prasad, M. Murray, C. Dovrolis, and K. C. Claffy. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network*, Jun 2003.

[23] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *Proceedings of IEEE Infocom 2001*, Anchorage AK, Apr 2001.

[24] J. Postel. *INTERNET CONTROL MESSAGE PROTOCOL*. RFC 792,IETF, http://www.ietf.org/rfc/rfc0792.txt, Sep 1981.