Scheduling Algorithms for Web Crawling

Carlos Castillo
Center for Web Research
Universidad de Chile
ccastill@dcc.uchile.cl

Mauricio Marin
Center for Web Research
Universidad de Magallanes
mauricio.marin@umag.cl

Andrea Rodriguez
Center for Web Research
Universidad de Concepción
andrea@udec.cl

Ricardo Baeza-Yates
Center for Web Research
Universidad de Chile
rbaeza@dcc.uchile.cl

***Abstract*— This paper presents a comparative study of strategies for crawling the Web. In particular we show that a combination of breadth-first ordering with the largest sites first is a practical alternative since it is simple to implement, it is fast and is able to retrieve the best ranked pages at a rate that is closer to the optimal than the other alternatives. Our study was performed on an actual sample (90%) of the Chilean Web which was crawled by using simulators so that all strategies were compared under the same conditions and actual crawls to validate our conclusions. We also explored the effects of large scale parallelism in the page retrieval task and multiple-page requests in a single connection for effective amortization of waiting and latency times.**

I. Introduction

Search engines play a central role in today's Web. Web search is currently generating more than 13% of the traffic to Web sites [1]. The main problem search engines have to deal with is the size of the Web, which currently is in the order of thousands of millions of pages. This large size induces a low coverage, with no search engine indexing more than one third of the publicly available Web [2].

In the early days of the Web, most search engines and directories included a prominent link to add your own Web page: Web site administrators were encouraged to submit their Web sites to the search engines. Today, that is no longer necessary, as search engine administrators consider that if the page cannot be easily reached by following links from the existent Web sites, then it is not "valuable" enough to be downloaded by the search engine's crawler. URLs of new pages are no longer a scarce resource.

We would like to develop a scheduling policy for downloading pages from the Web which guarantees that, even if we do not download all of the known pages, we still download the most "valuable" ones. As the number of pages grows, it will be increasingly important to download the "better" ones first, as it will be impossible to download them all.

The main contributions of this paper are:

- We propose several strategies for Web page ordering during a Web crawl, and compare them using a Web crawler simulator.
- We choose a strategy which is very competitive and is also very efficient.
- We test this strategy using a real Web crawler and show that it achieves the objective of downloading important pages early in the crawl.

Next section presents previous work on Web crawling, and the rest of this paper is organized as follows: section 3 outlines the problems of Web crawling, section 4 presents our experimental framework, sections 5 and 6 compare different scheduling policies. In section 7 we test one of these policies using a real Web crawler. The last section presents our conclusions.

II. Previous work

Web crawlers are a central part of search engines, and details on their crawling algorithms are kept as business secrets. When algorithms are published, there is often an important lack of details that prevents other from reproduce the work. There are also emerging concerns about "search engine spamming", which prevent major search engines from publishing their ranking algorithms.

However, there are some exceptions: some descriptions of crawlers (in chronological order) are: RBSE [3], the WebCrawler [4], which was the first publicly available search engine based on an automated agent, the World Wide Web Worm [5], the crawler of the Internet Archive [6], which is now used to store periodic snapshots of a large portion of the Web, the personal search agent SPHINK [7], an early version of the Google crawler [8], the CobWeb [9], a modular crawler called Mercator [10], Salticus [11], the WebFountain incremental crawler [12], and the WIRE crawler [13] used for this research. Descriptions of crawler architectures includes a parallel crawler architecture by Cho [14] and a general crawler architecture described by Chakrabarti [15].

Some crawlers released under the GNU public license include Larbin [16], WebBase [17] and HT://Dig [18]. For commercial products, see [19], [20].

Besides architectural issues, studies about Web crawling have focused on parallelism [14], [21], discovery and control of crawlers for Web site administrators [22], [23], [24], [25], accessing content behind forms (the "hidden" web) [26], detecting mirrors [27], keeping the freshness of the search engine copy high [28], [29], long-term scheduling [30], [31], [32] and focused crawling [33].

There have been also studies on characteristics of the Web, which are relevant to the crawler performance, such as detecting communities [34], characterizing server response time [35], studying the distribution of web page changes [36], [37], [38], [39], studying the macroscopic web structure [40], [41], and proposing protocols for web servers to cooperate with crawlers [42].

III. The problem of Web crawling

A Web crawler works by starting with a set of "seed" pages, downloading those pages and extracting links from them, and recursively following those links. Even if we don't account the technical difficulties of transferring, processing and storing a very large amount of data, there is still an important research problem, namely, the problem of scheduling the visits to the un-visited pages (we are considering in this article only the case of a complete crawl, with no re-visits).

At a first glance, it might seem that this scheduling problem has a trivial solution. If a crawler needs to download a series of pages whose file sizes in bytes are P_i, and has B bytes per second of available bandwidth for doing it, then it should download all the pages simultaneously at a speed proportional to the size of each page:

$$B_i = \frac{P_i}{T^*} \tag{1}$$

In which T^* is the optimal time to use all the available bandwidth:

$$T* = \frac{\sum Pi}{B} \tag{2}$$

This optimal scenario is depicted in Figure 1.

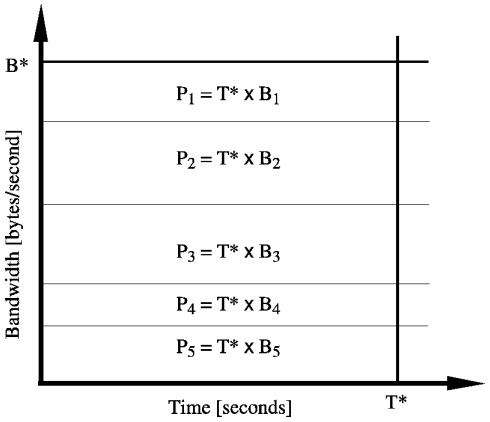

Fig. 1. Optimal scenario for downloading pages. The areas represent page sizes, as $size = speed \times time$.

However, there are many restrictions that forbid this optimal scenario. The main restriction of any scheduling policy is that it must avoid overloading Web sites. A Web crawler can impose a substantial amount of work on a Web server, specially if it opens many simultaneous connections for downloading [25]. It is customary that a Web crawler don't download more than one page from the same Web site at a time, and that it waits between 30 and 60 seconds between accesses. This, together with the fact that Web sites have usually a bandwidth B_i^{MAX} that is lower than the crawler bandwidth B, originate download time-lines similar to the one shown in Figure 2. In the Figure, the optimal time T* is not achieved, because some bandwidth is wasted due to limitations in the speed of Web sites (in the figure, B_3^{MAX}, the maximum speed for page 3 is shown), and to the fact that we must wait between accesses to a Web site (in the figure, pages $1-2$ and $4-5$ belong to the same site).

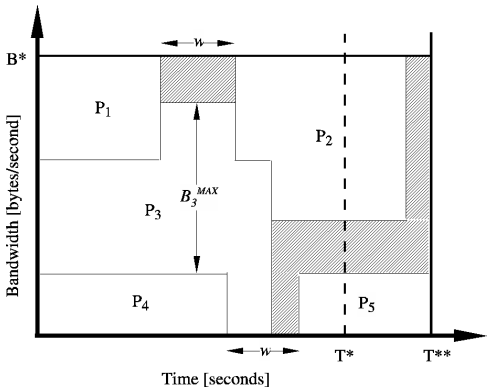

Fig. 2. A more realistic scenario while downloading Web pages. The hatched portion is wasted bandwidth due to limitations in the download rate from Web sites. The optimal time T* is not achieved.

To overcome the problems shown in Figure 2, it is clear that we should try to download pages from many different Web sites at the same time. Unfortunately, most of the pages are located in a small number of sites: the distribution of pages to sites, shown in Figure 3, is very bad in terms of crawler scalability. Thus, it is not possible to use productively a large number of robots and it is difficult to achieve good use of the network bandwidth.

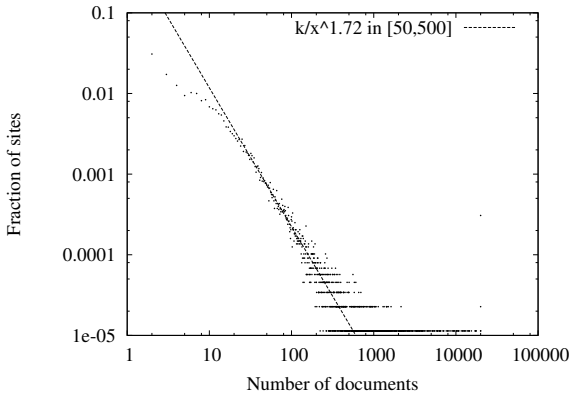

Fig. 3. Distribution of site sizes.

There is another serious practical restriction: the HTTP request has latency, and the latency time can be over 25% of the total time of the request [43]. This latency is mainly the time it takes to establish the TCP connection and it can be partially overcome if the same connection is used to issue several requests using the HTTP/1.1 "keep-alive" feature.

IV. Experimental setup

We downloaded 3.5 million pages in April 2004 from over 50,000 Web sites using the WIRE crawler [13]. We estimate that this is more than 90% of the publicly indexable Chilean Web pages, as the number of Web sites remaining in the queue

was very low. We restricted the crawler to download only the first 25,000 pages from each Website.

Using this data, we created a Web graph, and ran a simulator by using different scheduling policies on this graph. This allowed us to compare different strategies under exactly the same conditions. This simulator also considers bandwidth saturation of the internet link when too many connections are in process.

We considered a number of scheduling strategies whose design is based on a heap priority queue whose nodes represent sites and whose priorities are given by the pages associated with the sites. For every site-node we have another heap or a queue depending on the strategies we employ wherein the page in the top/front of the heap/queue determines the priority of the node in the sites heap, as depicted in Figure 4.

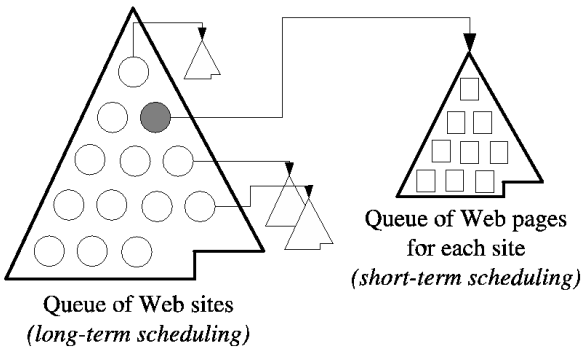

Fig. 4. Queues used during the scheduling.

At each simulation step, the scheduler chooses the topmost Website from the queue of Web sites and sends this site's information to a module that will simulate downloading pages from the Website. The parameters for our different scheduling policies are the following:

- The policy for ordering the queue of Web sites (long-term scheduling).
- The policy for ordering the queues of Web pages (short-term scheduling).
- The interval w in seconds between connections to a single Web site.
- The number of pages k downloaded for each connection when re-using HTTP connections.
- The number r of simultaneous connections, i.e.: the degree of parallelization. Although we used a large degree of parallelization, we restricted the robots to never establish more than 1 connection to a Web site at a given time.

A. Interval between connections (w)

The first proposal for the interval w between connections was given in [25] and was $w = 60$ seconds. If we download pages at this rate from a Web site with more than 100,000 pages over a perfect connection with zero latency and infinite bandwidth, it would take more than 2 months to download the entire Web site; also, we would be permanently using one of the processes or threads from that Web server. This does not seems acceptable.

Today, it's common to consider less than $w = 15$ seconds impolite. Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. We use $w = 15$ seconds in our simulations and real-life experiments.

B. Number of pages per connection (k)

Most Web crawlers download only one page per each connection, and don't re-use the HTTP connection. We considered downloading multiple pages in the same connection to measure the impact of this in the quality of the scheduling, to avoid the problem of latency in the HTTP connection.

The protocol for keeping the connection open was introduced as the `Keep-alive` header in HTTP/1.1 [44]; the default configuration of the Apache web server enables this feature by default, and allows for a maximum of 100 objects downloaded per request, with a timeout of 15 seconds between requests, so when using $k > 1$ in practice, we should also set $w \leq 15$ to prevent the server from closing the connection. This is also a good reason to set $w = 15$ during the experiments.

C. Number of simultaneous requests (r)

All of the robots currently used by Web search engines have a high degree of parallelization, downloading hundreds or thousands of pages from the same computer at a given time. We used $r = 1$, serialization of the requests, as a base case, $r = 64; r = 256$ during the simulations, and $r = 1000$ during the actual crawl.

As we never open more than one connection to a given Web site, r is bounded by the number of Web sites available for the crawler, i.e., the Web sites that have pages which have not yet been visited. If the number of Web sites is too small, we cannot make use of a high degree of parallelization and the crawler performance in terms of pages per second drops dramatically. This is specially critical at the end of a large crawl, when we have covered a substantial fraction of Web pages, or by the end of a batch of pages, when downloading pages grouped by batches.

In the second case, when we download the pages in batches, the batch of pages must be carefully built to include pages from as many Web sites as possible; this should be a primary concern when parallelism is considered.

V. Long-term scheduling

We tested different strategies for downloading pages from the stored Web graph. The complete crawl on the real Chilean Web takes about 8 days, so, apart from comparing strategies under exactly the same scenario, it is much more efficient to test many strategies using the crawling simulator. Retrieval real-time for Web pages is simulated by considering the real-crawl observed latency, transfer rate, page size for every downloaded page, and (possible) saturation of bandwith in accordance with the number of active connections at a given

moment of the simulation. We also validated our results with measures from the real 8-days crawl. For the experiments which do not consider retrieval time but rate of advance in terms of cumulative pageRank values, there is no difference between the simulator and the real crawler.

As mentioned, for evaluating the different strategies, we calculated before hand the Pagerank value of every page in the whole Web sample and used those values to calculate the cumulative sum of Pagerank as the simulated crawl goes by. We call those values "oracle" ones since in practice they are not known until the complete crawl is finished. The strategies which are able to reach faster values close to the target total value 1.0 are considered the most efficient ones.

All of the considered strategies are based on the two-level scheduling shown in Figure 4. We named our strategies *Optimal*, *Depth*, *Length*, *Batch* and *Partial*. Their description is the following:

Optimal Under this strategy, the crawler visits the pages in Pagerank order. To do that, it asks for the Pagerank to an "oracle" which knows the final value of the Pagerank for each page. Note that during the crawl, a crawler does not have access to this information, as it only knows a portion of the Web and therefore can only estimate the final Pagerank. The next page to be retrieved is the one which is on top of the two heaps, namely the page that have the greatest oracle's Pagerank in the short-term web pages heap, which in turn makes the corresponding site heap node to be the one with the greatest Pagerank value in the entire web sites heap (long-term scheduling).

Depth Under this strategy, the crawler visits the pages in breadth-first ordering. To this end, every web page heap is kept in such a way that the pages with the smallest depth in the Web graph are the ones with the greatest heap's priorities.

Length Pages associated with every web site are kept in a FIFO queue so this strategy is almost *Depth* but it extends it by considering the number of pages in the respective queues as the policy for the sites heap queue at any given moment of the crawling process. In this case, the web sites' heap nodes get higher priority of becaming the heap's top as their respective web pages queues get more and more pages. Node in the sites heap are re-arranged dynamically to follow changes in their priorities as new pages are inserted in the pages FIFO queues.

Batch Similar to Optimal but the crawler takes not a single page but a batch of m pages and once all of those pages are retrieved, the general Pagerank algorithm is run on the subset of known web pages (i.e., no oracle Pagerank values are used). The batch is formed with m pages sorted by the Pagerank value at that moment. Initially all pages (home-pages) are assumed to have the same Pagerank value. We performed experiments for m= 10K, 50K, and 100K.

Partial Similar to Optimal but the Pagerank algorithm is re-executed every time m pages are retrieved. No batch is formed as new pages are inserted inmediately in their respective web page heaps. Before a Pagerank is calculated, new pages are assigned with Pagerank values equivalent to the sum of the normalized rankings of the pages that point to them. Once the Pagerank is calculated and pages' rankings are updated, pages are sorted in their respective web page heaps. Like the previous strategies, all pages (home-pages) are initially assumed to have the same pageRank value. We performed experiments for m= 10K, 50K, and 100K.

All of the strategies are bound to the restrictions stated in the previous section: $w = 15$ waiting time, $k = 1$ pages per connection, r simultaneous connections, and no more than 1 connection to each Web site at a time. In the first set of experiments we assume a situation of high-bandwidth for the Internet link, i.e., the bandwidth of the Web crawler B is larger than any of the maximum bandwidths of the Web servers B_i^{MAX}.

Figure 5 shows the results for the cumulative sum of the Pagerank values considering one robot ($r = 1$, a single network connection at a time; we show the effect of parallelizing the downloads with more robots in the following figures). The figure shows that *Length* is more efficient than the strategies based on periodical Pagerank calculations and *Depth*. Notice that the implementation and processing requirements for *Length* are significantly simpler and lower than for all others. In particular, calculating periodical Pageranks takes a significant amount of resources in running time and space.

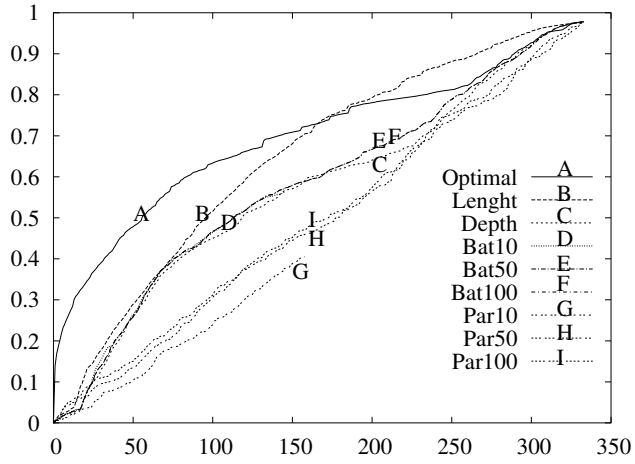

Fig. 5. Cumulative sum of Pagerank values vs number of retrieved web pages. Case for $r = 1$, one robot.

An important observation is that the "Optimal" strategy is, at the same time, too greedy, as it downloads all the pages with good Pagerank very early, but later it has only a few Web sites to choose from, so it is then surpassed by other strategies due to the restrictions of Web crawling.

Figure 6 shows the effect of increasing the number of robots to $r = 64$ and $r = 256$. Observing the rate of growth of the cumulative Pagerank sum, the results show that *Length* is more stable than *Depth*, though it improves as the number of robots increases.

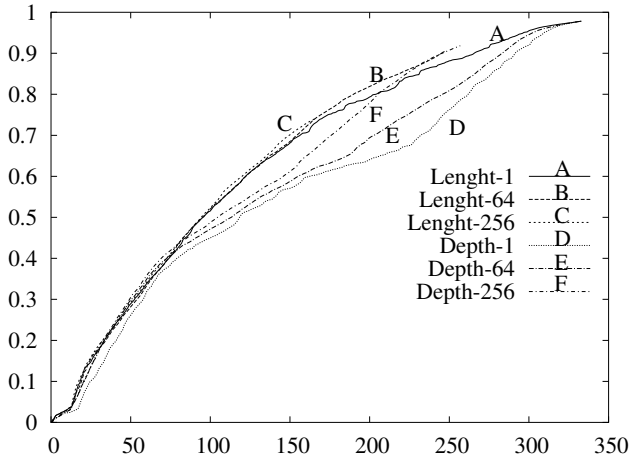

Fig. 6. Cumulative sum of Pagerank values vs number of retrieved web pages. Case for 1, 64 and 256 robots.

Finally table I shows the effects in retrieval time when we increase the number of robots for different bandwiths. The results shows how bandwith saturation makes pointless the use of several robots.

Bandwidth	r=1	r=64	r=256
20000000	1.0	54.6	220.1
2000000	1.0	54.3	204.3
200000	1.0	43.0	114.1
20000	1.0	27.0	83.3
2000	1.5	3.8	16.0
200	6.3	1.6	3.0

TABLE I

PREDICTED SPEED-UPS FOR PARALLELISM IN THE CRAWLING PROCESS.

VI. BETTER GROUPS THAN SINGLE PAGES

While crawling, specially in distributed crawling architectures, it is typical to work by downloading group of pages. That is, a batch of pages that is retrieved in parallel by several robots. For example, the Length strategy can be modified to retrieve several pages every time it visits a site. During that period of time no care is taken about Pagerank values. We have observed that on a typical batch, most of those pages are located in a reduced number of sites. We have shown the distribution of pages on sites for the whole Web in Figure 3; on Figure 7 we show page distribution on sites for a typical batch. Because of the rule telling that the crawler cannot visit the same site before $w = 15$ seconds, this distribution of pages to sites is very bad in terms of crawler scalability. That is, it is not possible to use productively a large number r of robots and it is difficult to achieve good use of the network bandwidth.

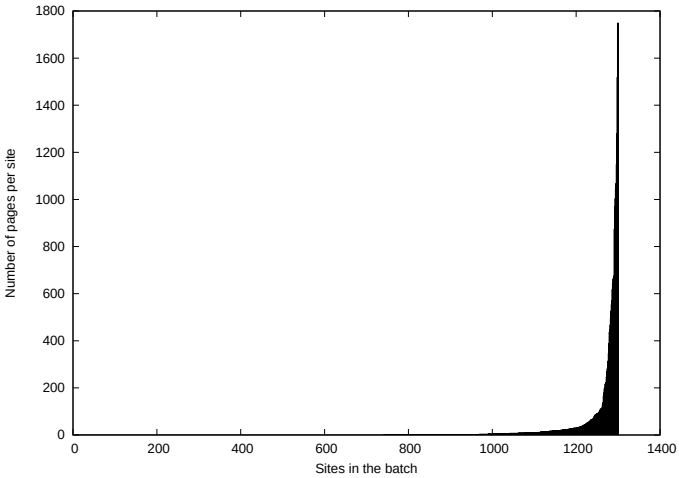

Fig. 7. A typical batch: distribution of pages onto sites.

Even if a batch involves a large number of Web sites, if a large fraction of the Web sites has very few pages, then quickly many of the robots will be idle, as the number of connections r is always smaller than the number of different active Web sites.

In practice, the robots tend to exhaust quickly the number of Web sites available. Figure 8 shows for this strategy the effective number of robots involved in the retrieval of a typical batch in the middle of the crawl.

A better approach is to increase k and let robots get more than one page every they visit a site. In figure 9 we show results for a case in which robots can download up to $k = 100$ pages per site in a single connection, using the HTTP/1.1 "keep-alive" feature.

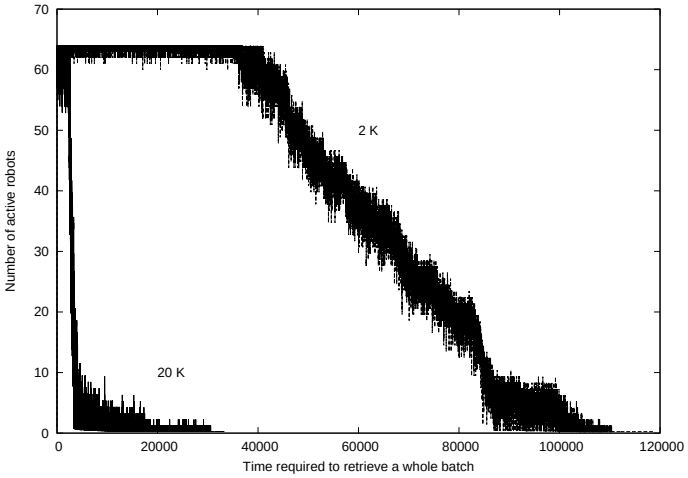

Fig. 8. Number of active robots vs batch's total retrieval time. The two curves are for small (2K) and large (20K) bandwidth. In either case, most robots are idle most of the time.

VII. APPLICATION: DOWNLOADING THE REAL WEB

We started with a list of Web sites registered with the Chilean Network Information Center, and ran the crawler

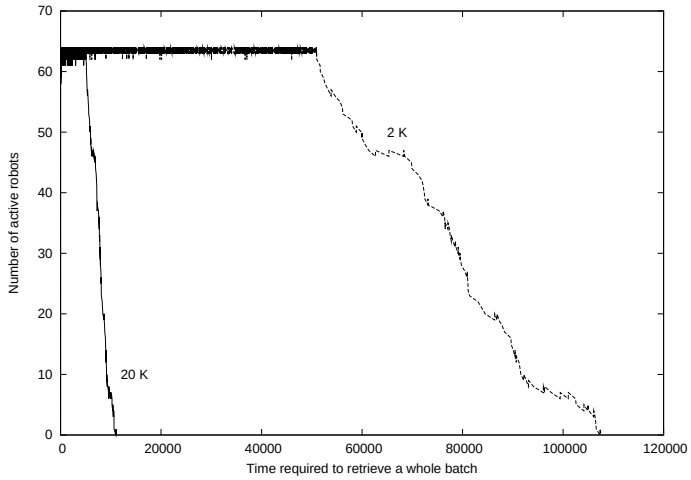

Fig. 9. Number of active robots vs batch's total retrieval time. The two curves are for small (2K) and large (20K) bandwidth. In this case robots are allowed to get 100 pages per site.

during 8 days with the *Length* strategy. The characteristics of the downloaded pages are summarized in Table II.

Pages visited	3 M
Successful downloads	(80%) 2.4 M
Bytes downloaded	57 GB
Web sites	53,196

TABLE II
DOWNLOADED PAGES

We ran the crawler in batches of up to 100,000 pages, using up to $r = 1000$ simultaneous network connections, and we waited at least $w = 15$ seconds between accesses to the same Web site. The crawler used both the `robots.txt` file and meta-tags in Web pages according to the robot exclusion protocol [25]. We didn't use keep-alive for this crawl, so $k = 1$.

We calculated the Pagerank of all the pages in the collection when the crawling was completed, and then measured how much of the total Pagerank was covered during each batch. The results are shown in Figure 10.

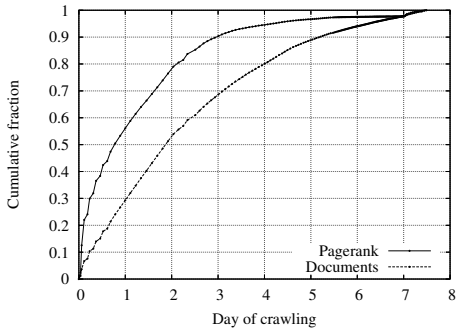

Fig. 10. Cumulative fraction of pages downloaded and cumulative Pagerank of the downloaded portion of the Web.

We can see that by the end of day 2, 50% of the pages were downloaded, and about 80% of the total Pagerank was achieved; according to the probabilistic interpretation of Pagerank, this means we have downloaded pages in which a random surfer limited to this collection would spend 80% of its time. By the end of day 4, 80% of the pages were downloaded, and more than 95% of the Pagerank, so in general this approach leads to "good" pages early in the crawl. In fact, the average Pagerank decreased dramatically after (very) few days (Figure 11), which is consistent with the findings of Najork and Wiener [30].

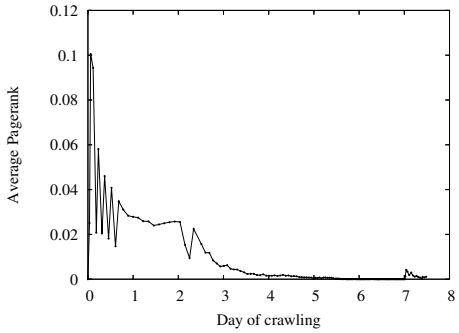

Fig. 11. Average Pagerank per day of crawl.

It is reasonable to suspect that pages with good Pagerank are found early just because they are mostly home pages or are located at very low depths within Web sites. There is, indeed, an inverse relation between Pagerank and depth in the first few levels, but 3-4 clicks away from the home page the correlation is very low, as can be seen in Figure 12. Note that home pages have, *in average*, a low Pagerank as there are many of them with very few or no in-links: we were able to found them only by their registration under the `.cl` top-level domain database.

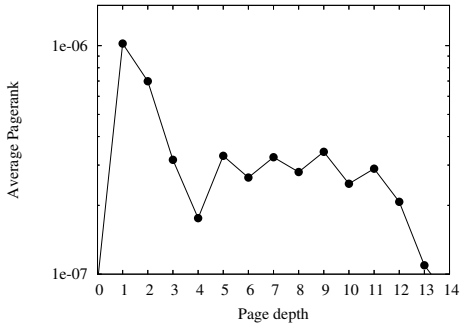

Fig. 12. Average Pagerank versus page depth.

VIII. CONCLUSIONS

The restrictions involved in Web crawling make this problem a very interesting one. In particular, a strategy which uses an "oracle" to detect pages with high Pagerank early, is not very good because as the crawl advances, few Web sites are available and inefficiencies arise.

For long-term scheduling, our results show that a really simple crawling strategy, such the one we called *Length*, is

good enough to efficiently retrieve a large portion of the Chilean Web. As the idea is to try to keep as many Web sites active as possible, this strategy prioritizes Web sites based on the number of pages available from them, such that is avoids exhausting Web sites too early.

Also our simulation results show that attempting to retrieve as many pages from a given site ($k >> 1$), allows one to effectively amortize the waiting time w before visiting the same site again. This certainly helps to achieve a better utilization of the available bandwidth.

Experiments with a real crawl using the *Length* strategy on the ever-changing Web validated our conclusions whereas simulation was the only way to ensure that all strategies considered were compared under the same conditions.

REFERENCES

[1] StatMarket, "Search engine referrals nearly double worldwide, according to websidestory," http://www.websidestory.com/pressroom/pressreleases.html?id=181, 2003.

[2] S. Lawrence and C. L. Giles, "Searching the World Wide Web," *Science*, vol. 280, no. 5360, pp. 98–100, 1998. [Online]. Available: citeseer.nj.nec.com/lawrence98searching.html

[3] D. Eichmann, "The RBSE spider: balancing effective search against web load," in *1st World Wide Web conference*, 1994.

[4] B. Pinkerton, "Finding what people want: Experiences with the WebCrawler," in *2nd World Wide Web conference*, 1994.

[5] O. A. McBryan, "GENVL and WWWW: Tools for taming the web," in *2nd World Wide Web conference*, 1994.

[6] M. Burner, "Crawling towards eternity - building an archive of the world wide web," *Web Techniques*, 1997. [Online]. Available: www.webtechniques.com/archives/1997/05/burner/

[7] R. Miller and K. Bharat, "Sphinx: A framework for creating personal, site-specific web crawlers," in *7th World Wide Web Conference*, 1998.

[8] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 107–117, 1998. [Online]. Available: citeseer.nj.nec.com/brin98anatomy.html

[9] A. S. da Silva, E. A. Veloso, P. B. Golgher, B. A. Ribeiro-Neto, A. H. F. Laender, and N. Ziviani, "Cobweb - a crawler for the brazilian web," in *SPIRE/CRIWG*, 1999, pp. 184–191. [Online]. Available: citeseer.ist.psu.edu/284260.html

[10] A. Heydon and M. Najork, "Mercator: A scalable, extensible web crawler," *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999. [Online]. Available: citeseer.nj.nec.com/heydon99mercator.html

[11] R. D. Burke, "Salticus: guided crawling for personal digital libraries," in *ACM/IEEE Joint Conference on Digital Libraries*, 2001, pp. 88–89. [Online]. Available: citeseer.nj.nec.com/burke01salticus.html

[12] J. Edwards, K. S. McCurley, and J. A. Tomlin, "An adaptive model for optimizing performance of an incremental web crawler," in *World Wide Web*, 2001, pp. 106–113. [Online]. Available: citeseer.ist.psu.edu/edwards01adaptive.html

[13] R. Baeza-Yates and C. Castillo, "Balancing volume, quality and freshness in web crawling," in *Soft Computing Systems - Design, Management and Applications*, 2002, pp. 565–572.

[14] J. Cho and H. Garcia-Molina, "Parallel crawlers," in *11th International World–Wide Web Conference*, 2002.

[15] S. Chakrabarti, *Mining the Web*. Morgan Kaufmann Publishers, 2003.

[16] S. Ailleret, "Larbin, a multi-purpose web crawler," larbin.sourceforge.net/index-eng.html, 2002.

[17] L. Dacharay, "Webbase web crawler," www.freesoftware.fsf.org/webbase/, 2002.

[18] "HT://Dig," www.htdig.org, 2004.

[19] D. Sullivan and C. Sherman, "Search Engine Watch," www.searchenginewatch.com/reports/, 2002.

[20] "BotSpot," www.botspot.com, 2004.

[21] V. Shkapenyuk and T. Suel, "Design and implementation of a high-performance distributed web crawler," in *ICDE*, 2002. [Online]. Available: citeseer.nj.nec.com/shkapenyuk01design.html

[22] P. Tan and V. Kumar, "Discovery of web robots session based on their navigational patterns," *Data Mining and Knowledge discovery*, 2001.

[23] J. Talim, Z. Liu, P. Nain, and E. G. C. Jr., "Controlling the robots of web search engines," in *SIGMETRICS/Performance*, 2001, pp. 236–244. [Online]. Available: citeseer.nj.nec.com/talim01controlling.html

[24] "Robotcop," www.robotcop.org, 2002.

[25] M. Koster, "Robots in the web: threat or treat ?" vol. 4, no. 4, 1999.

[26] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," in *Proceedings of the Twenty-seventh International Conference on Very Large Databases*, 2001. [Online]. Available: citeseer.nj.nec.com/article/raghavan01crawling.html

[27] J. Cho, N. Shivakumar, and H. Garcia-Molina, "Finding replicated web collections," in *ACM SIGMOD*, 1999, pp. 355–366.

[28] J. Cho and H. Garcia-Molina, "Synchronizing a database to improve freshness," in *ACM International Conference on Management of Data (SIGMOD)*, 2000, pp. 117–128. [Online]. Available: citeseer.nj.nec.com/cho00synchronizing.html

[29] A. Czumaj, I. Finch, L. Gasieniec, A. Gibbons, P. Leng, W. Rytter, and M. Zito, "Efficient Web searching using temporal factors," *Theoretical Computer Science*, vol. 262, no. 1–2, pp. 569–582, 2001. [Online]. Available: citeseer.nj.nec.com/137637.html

[30] M. Najork and J. L. Wiener, "Breadth-first crawling yields high-quality pages," in *10th World Wide Web Conference*. Hong Kong: Elsevier Science, May 2001, pp. 114–118. [Online]. Available: citeseer.nj.nec.com/najork01breadthfirst.html

[31] E. G. Coffman, Z. Liu, and R. R. Weber, "Optimal robot scheduling for web search engines, Tech. Rep. RR-3317, 1997. [Online]. Available: citeseer.nj.nec.com/coffman97optimal.html

[32] J. Cho, H. García-Molina, and L. Page, "Efficient crawling through URL ordering," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 161–172, 1998. [Online]. Available: citeseer.nj.nec.com/cho98efficient.html

[33] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs," in *26th International Conference on Very Large Databases, VLDB 2000*, Cairo, Egypt, 10–14 September 2000, pp. 527–534. [Online]. Available: citeseer.nj.nec.com/diligenti00focused.html

[34] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the Web for emerging cyber-communities," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 31, no. 11–16, pp. 1481–1493, 1999. [Online]. Available: citeseer.ist.psu.edu/kumar99trawling.html

[35] B. Liu, "Characterizing web response time (master thesis)," 1998.

[36] J. Cho, "The evolution of the web and implications for an incremental crawler," in *The VLDB Journal*, 2000.

[37] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul, "Rate of change and other metrics: a live study of the world wide web," in *USENIX Symposium on Internet Technologies and Systems*, 1997. [Online]. Available: citeseer.nj.nec.com/douglis97rate.html

[38] B. Brewington, G. Cybenko, R. Stata, K. Bharat, and F. Maghoul, "How dynamic is the web?" in *World Wide Web Conference*, 2000.

[39] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Agarwal, "Characterizing Web document change," *Lecture Notes in Computer Science*, vol. 2118, pp. 133–??, 2001. [Online]. Available: citeseer.nj.nec.com/452337.html

[40] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web: Experiments and models," in *9th World Wide Web Conference*, 2000, pp. 309–320.

[41] R. Baeza-Yates and C. Castillo, "Relating web characteristics with link based web page ranking," in *String Processing and Information Retrieval*. IEEE Cs. Press, 2001, pp. 21–32.

[42] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar, "Crawler-friendly web servers," in *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, California, 2000. [Online]. Available: citeseer.nj.nec.com/article/brandman00crawlerfriendly.html

[43] B. Liu and E. A. Fox, "Web traffic latency: Characteristics and implications," *J.UCS: Journal of Universal Computer Science*, vol. 4, no. 9, pp. 763–??, 1998. [Online]. Available: citeseer.ist.psu.edu/liu98web.html

[44] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "HTTP/1.1 - Hypertext Transfer Protocol," w3.org/Protocols/rfc2616/rfc2616.html, 1999.