Balancing Volume, Quality and Freshness in Web Crawling*

Ricardo Baeza-Yates   Carlos Castillo

Center for Web Research
Dept. of Computer Science, University of Chile
Blanco Encalada 2120, Santiago, Chile
E-mail: {rbaeza,ccastill}@dcc.uchile.cl

Abstract

We describe a crawling software designed for high-performance, large-scale information discovery and gathering on the Web. This crawler allows the administrator to seek for a balance between the volume of a Web collection and its freshness; and also provides flexibility for defining a quality metric to priorize certain pages.

Keywords: Web crawling, search engines, crawling policies.

1 Introduction

Search engines are the most used tools to find Websites and information in the Web, and they account for a significant percentage of through traffic in many Web sites. These engines have three standard components: crawlers, spiders or robots (input), collection and index (storage) and query resolver and interface (output). For more information we refer the reader to [8, 6].

Nowadays is difficult to build a crawler that can accurately traverse all of the Web because of its volume, pace of change and growth. In fact, Web crawlers have to deal with a lot of challenges at the same time:

a) They must keep fresh copies of Web pages, so they have to revisit some of them, but at the same time they must discover new pages.

b) They must use the available resources such as network bandwidth to the maximum extent, but without overloading Web servers as they visit them.

*Funded by Millenium Nucleus Center for Web Research, Mideplan, Chile.

c) They must get a lot of "good pages", but they cannot exactly know in advance which ones are the good ones.

In [15] we presented a model that tightly integrates crawling with the rest of a search engine and gives a possible answer about how to deal with these (contradicting) goals. This model generalizes many particular cases, and leads to a new crawling software architecture that has been evolving and that we describe in this paper by focusing in its design issues.

The rest of the paper is organized as follows. The next section covers previous work. Then we describe the software architecture of our crawler followed by the description of the main scheduler, how to be polite to Web servers and some performance issues. We conclude with future planned work.

2 Previous work

Crawlers are an important part of some search engines, and as such, their internals are business secrets, and when they publish information, there is an important lack of details that prevents other to reproduce the work.

Some descriptions of crawlers (in chronological order) are: RBSE [26], Internet Archive [14], SPHINX [31], Google [34], Mercator [32] and Salticus [13].

Some crawlers released under the GNU public license include [5] and [23]. They have less features than their commercial counterparts, but are very fast. For commercial products, see [4].

Other studies focus on parallelization [18, 37], discovery and control of crawlers [39, 38, 3], accessing contents behind forms (the "hidden" Web) [36], mirror detection [20], freshness [17, 22], URL ordering [33, 21], focused crawling [24], and characteristics of the Web that are relevant to the crawler performance such as server response time [30], Web page changes [16, 25, 11], Web structure [12, 7], and how Web servers can help crawlers [10]

3 Software Architecture

The original architecture for this crawler was presented in [15], but it has experienced some changes. The system is designed to be:

- *Fast*: all of its data structures are specially designed to be fast, accounting for the specific access patterns of the crawler.
- *Robust*: it is designed as a modular set of very specific tasks. For instance, the program that does the network input/output is different from the program that parses pages.
- *Smart*: the crawler has access to all the information about pages, so it can take better decisions about which pages to visit next.

- *Flexible*: most of the parameters, including the score function, are configurable.

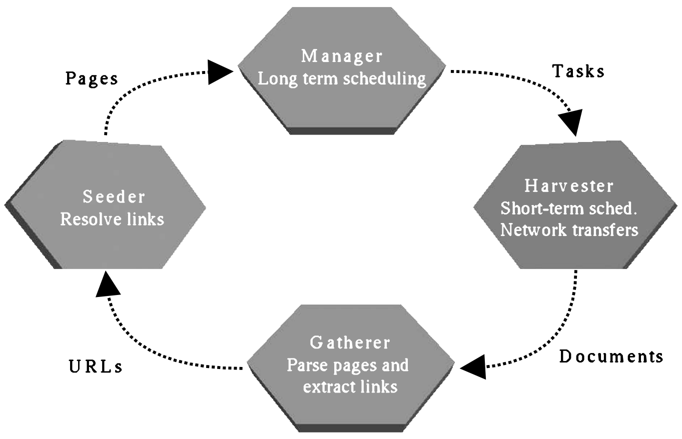

Figure 1: Modules of this Web crawler.

The main modules of the crawler are depicted in Figure 1:

- *Robot Seeder*: this module adds URLs to a structure specially designed to be fast in determining if a Web page has been seen before.
- *Robot Manager*: this module calculates scores and freshness for each page, and determines which pages must be crawled or re–crawled in the next batch.
- *Robot Harvester*: this module fetches pages from the Web. It can create a short–term schedule to make efficient use of the network resources available and to avoid overloading Web sites. Many harvesters can be running in different machines with one manager, as in [32].
- *Robot Gatherer*: this module parses the contents of Web pages, creates summaries of them, and passes the discovered URLs to the seeder program.

Some important features of the crawler we have designed and implemented are:

- Entirely written in C++.
- Uses a three-level cache for IP addresses.
- Implements pagerank, and multiple ways of calculating the priority for each page.

- Detects slows sites and re-schedules them.
- HTML parser to keep mainly the structural tags, removing most of the formatting tags.
- Duplicate detections by text analysis.
- Completely configurable with a configuration file in XML, using LibXML2 [2], including maximum depth for static and dynamic pages and weights in score calculation.
- Robot exclusion protocol [29], with extra time between requests for smaller sites.

4 Long-term Scheduling

This crawler relies on assigning a score (between 0 and 1) to each page; the higher the score, the better the page. The score summarizes the known metadata about a Web page in a single number.

Useful score functions include evidence about:

- *Link analysis measures*: pagerank [35], hubs and authorities [28], backlink count, or another measure of the link popularity of this page [19].
- *Text analysis*: the textual content can be compared with a query or a set of queries that try to define a category of pages, as in focused crawlers.
- *Access patterns*: if the logs of a search engine can be used, then the crawler can know which pages are more important for the users.
- *Site analysis*: all of the above, applied to the site where the page belongs. Also, pages belonging to fast Websites can be considered more important that those belonging to slower sites.
- Other factors.

The score function $score(p)$ and an estimation of the freshness, or probability that a page in the index is up-to-date, $freshness(p)$, are used to compute the *value* of a Web page in the index.

$$value(p) = score(p) \times freshness(p)^{\frac{1}{\alpha}} \tag{1}$$

The parameter α can be used to make the crawler re-visit pages more often. We use $\alpha = 1$ as default value.

The long-term scheduling algorithm we use generates batches of documents to be visited. In each round, we want to increase the total value of the index; every time we visit a page, its freshness is 1, and its value equals its score. Freshness decreases as time goes by if the

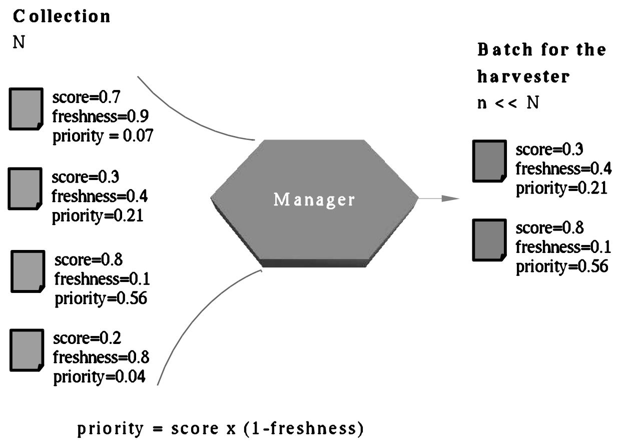

Figure 2: The process of creating a batch involves calculating estimators of freshness and score for each page.

page is not revisited. We use equation 1 to create a set of pages that will increase the value of the index more, as shown in Figure 2 the gain in value when bringing a fresh page p is $score(p) \times (1 - freshness(p))$ for $\alpha = 1$. Many batches can be created at the same time, because this can be an expensive calculation.

5 Politeness

The crawler is designed to be polite with Web sites. It only opens one connection at a time to a Website, it waits by default 30 seconds between accesses, it uses the robot exclusion protocol for sitewise limits and the robots meta-tag for pagewise limits. Also, it uses HTTP `if-modified-since` headers to avoid downloading pages with no changes. Each site is assigned to a single harvester and to a single thread in the harvester, as shown in Figure 3.

The distribution of pages among Web sites on the Web is very skewed; its common for a vertical Web crawler to spend the last days of an entire crawl in a few Web sites. Based on that, we have implemented the crawler so it accesses pages from larger sites more often than smaller sites.

Mirrors on the Web are a significative challenge to Web crawlers, because a replicated collection usually has hundred or thousands of pages [20]. We compute a hash of the content of every downloaded page, and the crawler do not visit duplicates. The hashes are computed over the parsed contents (in which most of the tags are structural, and only a few formatting tags are allowed) so a page with minor stylistic changes is also considered as a mirror. The

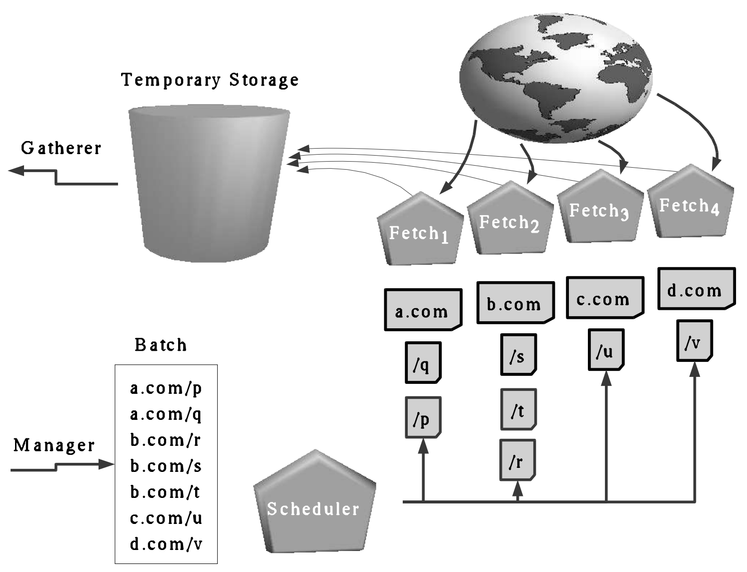

Figure 3: The harvester has a scheduler that creates a series of queues, one for each fetcher thread. At the end of the batch, a temporary storage is passed to the gatherer, that adds parses and adds pages to the main collection

process of storing the text of a page, which is done by the gatherer, is shown in Figure 4.

6 Performance Issues

One of the major performance bottlenecks is DNS resolving. This has led some designs to include DNS resolvers as a standard part of the crawler [37]. In our case, we use the ADNS library [27] for asynchronous DNS resolving, and a three-level cache of IP address, using DnsCache [9], the memory in the process of the harvester, and the disk, because we save the IP address along with the metadata.

Another critical process is checking the links of visited pages to verify if they have been seen before. This is done with two hash-tables, one for the site names, and one for the paths, as shown in Figure 5; this is done to exploit locality of references in Web sites (most of the references on a page point to the same Web site). This global id solution is split into site ids and local file ids when storing the link graph of the Web to reduce the memory usage due to the locality of reference.

We have done our own implementation of a lightweight HTTP/1.1 [1] client, and we start parsing pages only when the harvester is not running. This allows us to run up to 500 threads with a minimal impact in CPU load.

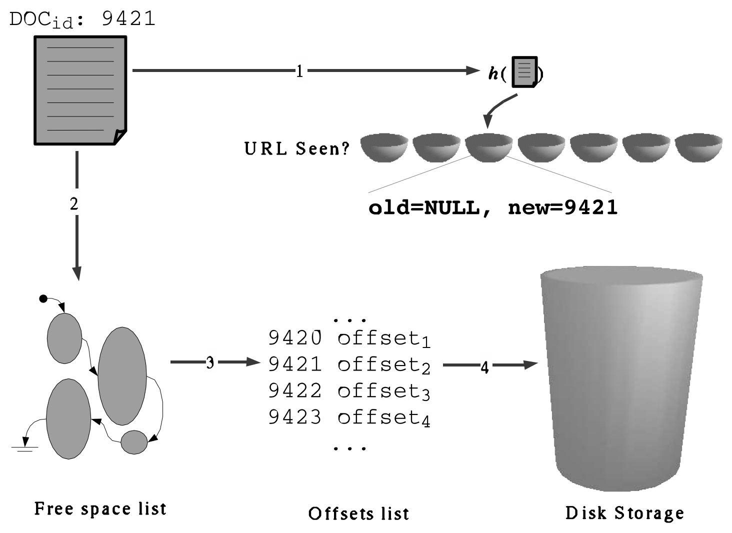

Figure 4: The storage subsystem, used by the gatherer, has a structure to determine if a page has been seen.

7 Final remarks and Future Work

Crawling is not as easy as using wget in mirror mode. To have a competitive performance, special software architectures that depend on networks, operating systems and programming environments must be designed. We have presented a crawler architecture that has evolved due to real experimentation that allows to find key issues that were not necessarily expected. For example, DNS resolving, dealing with large Web sites or storing the link graph of the Web.

There are four main areas in which we intend to do research in the short–term:

- Develop a framework to compare and optimize crawlers. This implies to answer questions like finding the optimal balance of processes and threads per CPU, the maximal number of threads to exploit a given bandwidth before saturation, or the optimal balance of processors and bandwidth.
- Web mining using this crawler; we intend to have a tool for gathering statistics about Web pages; we will use this to define a metric for evaluating different score functions and policies.
- Implement HTTP/1.1 pipelining for the largest Web sites.
- Develop and test a Web server plug-in to minimize the overhead due to request for non-changed pages, this plug-in can generate a lot of savings for both, the Web server and the Web crawler [10].

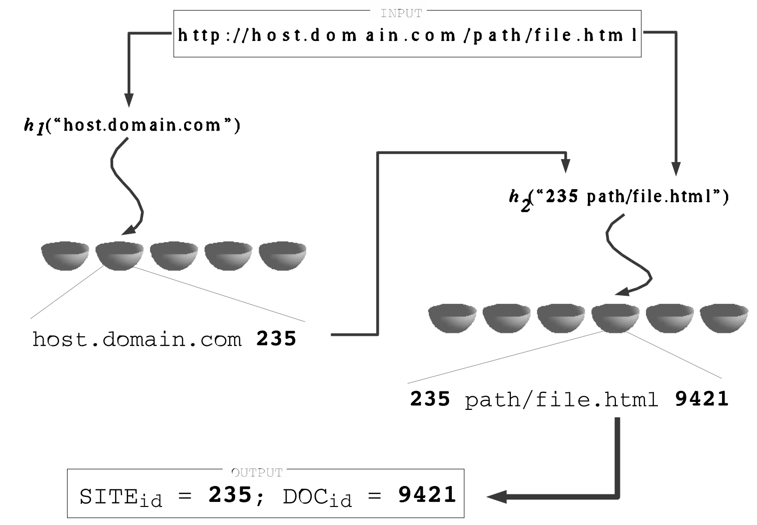

Figure 5: The URL index separates the URL in two parts; first is calculates a hash function of the host name, and then of the path. The most frequent operation is to resolve an unknown path in a known site.

References

[1] HTTP - Hypertext Transfer Protocol. w3.org/Protocols/rfc2616/rfc2616.html, 1999.

[2] LibXML - The XML C library for Gnome. www.xmlsoft.org, 2002.

[3] Robotcop. www.robotcop.org, 2002.

[4] Search engine watch. www.searchenginewatch.com/reports/, 2002.

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

[6] Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., and Raghavan, S. Searching the web. *ACM Transactions on Internet Technology 1*, 1 (August 2001), 2–43.

[7] Baeza-Yates, R., and Castillo, C. Relating web characteristics with link based web page ranking. In *String Processing and Information Retrieval* (2001), IEEE Cs. Press.

[8] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval.* Addison-Wesley & ACM Press, Harlow, UK, 1999.

[9] BERNSTEIN, D. J. DNS cache. cr.yp.to/djbdns.html, 2002.

[10] BRANDMAN, O., CHO, J., GARCIA-MOLINA, H., AND SHIVAKUMAR, N. Crawler-friendly web servers. In *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)* (Santa Clara, California, 2000).

[11] BREWINGTON, B., CYBENKO, G., STATA, R., BHARAT, K., AND MAGHOUL, F. How dynamic is the web? In *World Wide Web Conference* (2000).

[12] BRODER, A., KUMAR, R., MAGHOUL, F., RAGHAVAN, P., RAJAGOPALAN, S., STATA, R., AND TOMKINS, A. Graph structure in the web: Experiments and models. In *9th World Wide Web Conference* (2000).

[13] BURKE, R. D. Salticus: guided crawling for personal digital libraries. In *ACM/IEEE Joint Conference on Digital Libraries* (2001), pp. 88–89.

[14] BURNER, M. Crawling towards eternity - building an archive of the world wide web. *Web Techniques* (1997).

[15] CASTILLO, C., AND BAEZA-YATES, R. A new crawling model. In *World Wide Web Conference* (Honolulu, USA, 2002). (Extended Poster).

[16] CHO, J. The evolution of the web and implications for an incremental crawler. In *The VLDB Journal* (2000).

[17] CHO, J., AND GARCIA-MOLINA, H. Synchronizing a database to improve freshness. In *ACM International Conference on Management of Data (SIGMOD)* (2000), pp. 117–128.

[18] CHO, J., AND GARCIA-MOLINA, H. Parallel crawlers. In *11th International World–Wide Web Conference* (2002).

[19] CHO, J., GARCÍA-MOLINA, H., AND PAGE, L. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 161–172.

[20] CHO, J., SHIVAKUMAR, N., AND GARCIA-MOLINA, H. Finding replicated web collections. In *ACM International Conference on Management of Data (SIGMOD)* (1999).

[21] COFFMAN, E. G., LIU, Z., AND WEBER, R. R. Optimal robot scheduling for web search engines. Tech. Rep. RR-3317, 1997.

[22] CZUMAJ, A., FINCH, I., GASIENIEC, L., GIBBONS, A., LENG, P., RYTTER, W., AND ZITO, M. Efficient Web searching using temporal factors. *Theoretical Computer Science 262*, 1–2 (2001), 569–582.

[23] DACHARAY, L. Webbase web crawler. www.freesoftware.fsf.org/webbase/, 2002.

[24] Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. Focused crawling using context graphs. In *26th International Conference on Very Large Databases, VLDB 2000* (Cairo, Egypt, 10–14 September 2000), pp. 527–534.

[25] Douglis, F., Feldmann, A., Krishnamurthy, B., and Mogul, J. C. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems* (1997).

[26] Eichmann, D. The RBSE spider: balancing effective search against web load. In *1st World Wide Web Conference* (1994).

[27] Jackson, I. ADNS. www.chiark.greenend.org.uk/∼ian/adns/, 2002.

[28] Kleinberg, J. Authoritative sources in a hyperlinked environment. In *9th Symposium on discrete algorithms* (1998).

[29] Koster, M. Robots in the web: threat or treat ? In *ConneXions, 4(4)* (1999).

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

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

[32] Najork, M., and Heydon, A. On high-performance web crawling (using mercator). Tech. rep., 2001.

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

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

[35] Page, L., Brin, S., Motwain, R., and Winograd, T. The pagerank citation algorithm: bringing order to the web. In *7th World Wide Web Conference* (1998).

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

[37] Shkapenyuk, V., and Suel, T. Design and implementation of a high-performance distributed web crawler. In *ICDE* (2002).

[38] Talim, J., Liu, Z., Nain, P., and Jr., E. G. C. Controlling the robots of web search engines. In *SIGMETRICS/Performance* (2001), pp. 236–244.

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