A New Model for Web Crawling[*]

Carlos Castillo Ricardo Baeza-Yates

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

Abstract

Web crawlers have to deal with a lot of challenges at the same time, and some of them contradict each other. They must keep fresh copies of Web pages, so they have to re-visit some of them, but at the same time they must discover new pages. They must use the available resources such as network bandwidth to the maximum extent, but without overloading Web servers as they visit them. They must get a lot of "good pages", but they cannot exactly know in advance which ones are the good ones. We present 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 by means of adjustable parameters. We show how this model generalizes some particular cases, and leads to a new crawling software architecture.

Keywords: Web crawling, search engines, crawling policies.

1 Introduction

Search engines have three standard components: crawlers, spiders or robots (input), collection and index (storage) and query resolver and interface (output). We focus on Web crawler design issues and its relation to the index.

As Web crawlers are mostly used by search engines, their inner components are usually well guarded secrets and only benchmarks [20] are revealed. Some exceptions (in chronological order) are: RBSE Spider [13], Internet Archive [2], SPHINX [16], Google Web crawler [18], Mercator [14], and parallel crawler [4]. Other papers deal with page ordering for crawling [8, 11, 17], mirroring [3], web server overloading [22, 21], keep the collection up to date [5, 9, 7], dynamic pages [19], and knowing how Web pages change over time [6, 1, 12].

Previous work tends to separate two very similar problems and to mix two very different problems:

- The two similar problems are the index freshness and the index quality according to other metrics (eg. link analysis). It will be better to think in terms of a series of scores associated to different characteristics of the documents in the collection, weighted accordingly to some priorities that will vary depending on the usage context of the crawler. As some goals are

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

contradictory, the software must decide amongst, for instance, trying to discover new pages or updating existing ones. In this case, we need some way to tell the crawler how to decide between these alternatives.

- The two different problems that are commonly mixed are the problem of short-term efficiency, that is, maximizing the bandwidth usage and be polite with servers [15](probably accounting some stationary variations of the Web servers during the day or the week [10]) and long-term efficiency (ordering the set to favor some important pages).

The main contributions of this paper are a model that uses the previous two observations and generalizes many specialized crawlers, where the crawling process can be parallelized more efficiently, and where the proposed overall architecture gives a framework that accounts for the different topics present on recent research.

2 The Proposed Crawling Model

The main goals of a crawler are the following:

- The index should contain a large number of Web pages that are *interesting* to the search engine's users.
- Every object on the index should *accurately represent* a real object on the Web (content through time).

It must be noticed that these two goals compete between them, because the crawler must decide between going for a new page, not currently on the index, or refreshing some page that is probably outdated in the index. There is a trade-off between quantity (more objects) and quality (more up-to-date objects). In addition, many new pages will appear as URLs in modified pages.

The problem of short-term efficiency is an important and difficult one. It can be stated as follows. There is bandwidth available B (bytes/second) to fetch a number of Web pages. If the total size of all pages is S (bytes) then the crawler should be able to download all pages in exactly $T = S/B$ (seconds). It cannot be done faster than that, and it seems possible to achieve this level of efficiency, but usually, it takes longer than that, mainly because Web sites have variable transfer rates (we have measured a variation around 60% between accesses). Even worse, latency times are unpredictable (around 120% of variation for the same Web page in two different times). A realistic diagram is depicted in Figure 1.

We propose that the *value* of an index $I = o_1, o_2, ..., o_n$ will be the sum[1] of the values of the objects o_i stored on the index, where the *value* of an object in the index, $V(o_i)$, is a product function, as follows:

$$V(I) = \sum_{i=1}^{n} V(o_i) \ , \qquad V(o_i) = q_i^a \times r_i^b \times p_i^c$$

where:

[1]Another function could be used, but it has to be non-decreasing on every component.

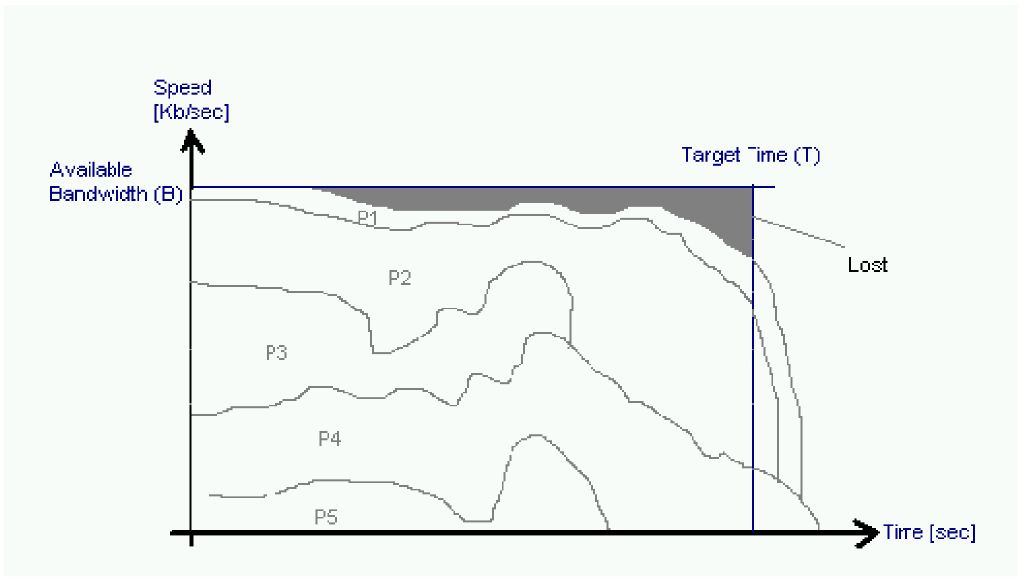

Figure 1: A realistic crawling scenario.

- q_i (intrinsic quality) approximates the importance of the object o_i by itself (how interesting it is?).
- r_i (representation quality) is the quality of the representation of the object, putting aside its intrinsic quality, considering the space needed and the rendering time of the object. For example, compression uses less space but increases the rendering time.
- p_i (freshness quality) is the probability that this representation coincides with the represented object.
- a, b and c are adjustable parameters of the crawler, that depend on the objective and policies of it.

The variable q_i can be estimated in many ways [8]: link analysis such as Pagerank [18](link popularity), similarity to a driven query, accesses to that page on the index (usage popularity), and location-based: by the perceived depth (eg. number of directories on the path to the Web object) or by domain name, IP address or geography.

On the other hand, r_i depends mainly on the quantity and format of the information being stored for every object. If the complete object is stored, it is high.

The probability that an object is up to date, p_i (freshness), as Web updates are common, decreases with time. Freshness can be estimated quite precisely if the last modification date of the Web page is informed by the Web server [5], using the work in [1].

The proposed model covers many particular cases, as shown in Figure 2.

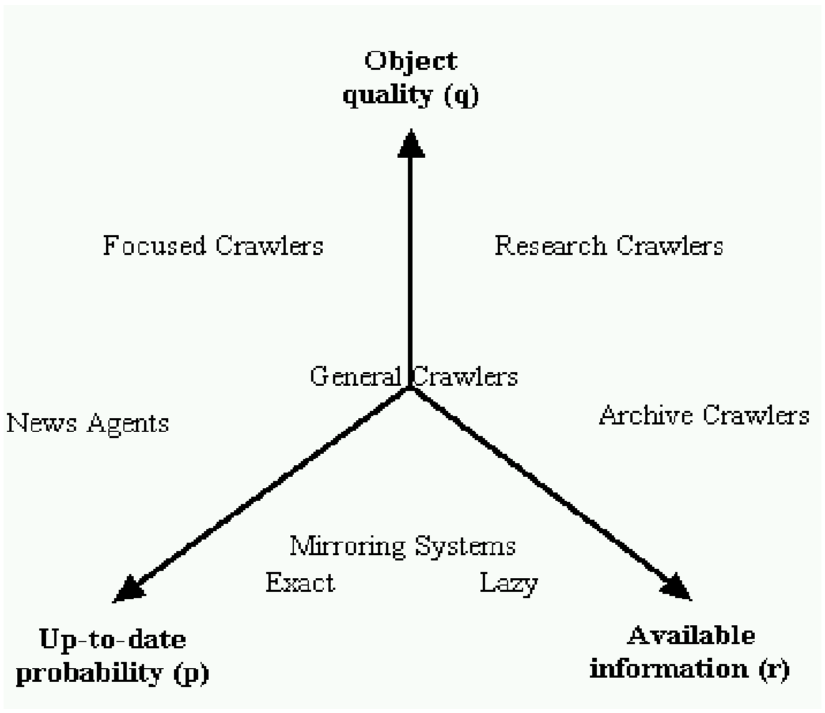

Figure 2: Different crawlers in our model.

3 A Software Architecture

We present a software architecture that comes from the model and from the observations of the introduction, that seems quite natural.

This architecture involves the use of *two* schedulers. One is the long-term scheduler, which we call the "manager". Its role is to order the URLs and create *batchs*, that is, collections of URLs of size K that, when crawled, will increase the value of the index.

The long-term scheduler estimates q_i, r_i and p_i for every object in the collection. Note that we do not have to make an exact estimation, because we are just interested in K pages with high q_i and currently low r_i or p_i.

The short-term scheduler, which we call "harvester", receives a list of K URLs to crawl. Its task is to sort them and control the "fetchers" tasks that do the network I/O. The harvester must avoid overloading sites and has to keep the network usage on its maximum. Many harvesters can be running on different machines across the network. They do not communicate back until the batch of work assigned by the manager is done. This helps to parallelize the process.

After that, a "gatherer" collects all pages fetched, generating the logical views and file formats according to the r_i values dictated by the manager and adds those views to the collection. This process also extracts all URLs on Web pages and stores them separately. If some harvester is late, the gatherer does not need to wait for it, and the harvester output (local copy of) will be processed on the next gatherer run.

The "seeder" has the mission of taking the list of URLs that the gatherer produced and add them to the index if they are new. Also, the seeder must help to keep the link structure for ranking purposes is needed. The seeder does not have to run on every (manager-harvester-gatherer) cycle.

Figure 3 shows the whole architecture.

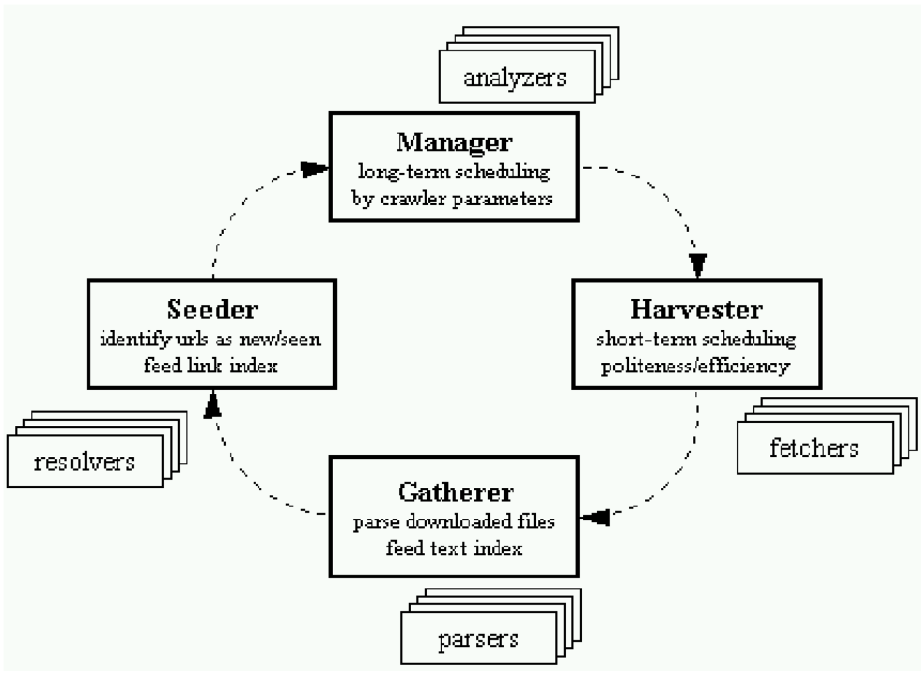

Figure 3: The proposed software architecture.

4 Conclusions

In this paper we have tried to formalize the goals of a key component of any search engine: the crawler. A crawler has to fulfill different objectives which cannot be all obtained simultaneously. We refine them in three parameters: object quality, object representation and availability, and object freshness. By setting those parameters according to the goals or policies of a specific crawler, we obtain different data, quality and usage of resources.

Based on this model we propose a software architecture that is currently under implementation. Preliminary testing have verified our hypotheses and the crawling performance obtained is quite good.

References

[1] B. Brewington, G. Cybenko: *How dynamic is the Web?*, Proc. WWW9, 2000.

[2] M. Burner: *Crawling towards Eternity - Building An Archive of The World Wide Web*, Web Techniques, May 1997. http://www.webtechniques.com/-archives/1997/05/burner/.

[3] J. Cho, N. Shivakumar, H. Garcia-Molina: *Finding replicated Web collections*, In Proc. of 2000 ACM International Conference on Management of Data (SIGMOD) Conference, May 2000.

[4] J. Cho, H. Garcia-Molina: *Parallel Crawlers*, Technical Report, Dept. of Computer Science, Stanford University, 2001.

[5] J. Cho, H. Garcia-Molina: *Estimating Frequency of Change*, Technical Report, Dept. of Computer Science, Stanford University, 2001.

[6] J. Cho, H. Garcia-Molina: *The Evolution of the Web and Implications for an Incremental Crawler*, The VLDB Journal, pages 200-209, 2000.

[7] J. Cho, H. Garcia-Molina: *Synchronizing a database to improve freshness.* Proc. of ACM SIGMOD, pages 117-128, 2000.

[8] J. Cho, H. Garcia-Molina: *Efficient crawling through URL ordering.* Proc. WWW7, 1998.

[9] E.G. Coan, Jr., Zhen Liu, Richard R. Weber: *Optimal robot scheduling for Web search engines.* Technical Report, INRIA, 1997.

[10] A. Czumaj, I. Finch, L. Gasienic, A. Gibbons, P. Leng, W. Rytter, M. Zito: *Efficient Web searching using temporal factors*, Theoretical Computer Science (2001) 262/1-2, p. 569-582.

[11] M. Diligenti, F. Coetzee, S. Lawrence, C. Lee Giles, M. Gori: *Focused Crawling using Context Graphs*, Proc. of 26th International Conference on Very Large Databases, VLDB 2000.

[12] F. Douglas, A. Feldmann, B. Krishnamurthy, J.C. Mogul: *Rate of Change and other Metrics: a Live Study of the World Wide Web*, USENIX Symposium on Internet Technologies and Systems, 1997.

[13] D. Eichmann: *The RBSE spider: Balancing effective search against Web load*, Proc. of 1st WWW conference, 1994.

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

[15] M. Koster: *Robots in the Web: threat or treat*, ConneXions, 9(4), 1995.

[16] R. Miller, K. Bharat: *SPHINX: A framework for creating personal, site-specific Web crawlers*, Proc. of WWW7, 1998.

[17] M. Najork, J. Wiener: *Breadth-first search crawling yields high-quality pages*, Proc. of WWW10, 2001.

[18] L. Page, S. Brin: *The anatomy of a large-scale hypertextual Web search engine.* Proc. of WWW7, 1998.

[19] S. Raghavan, H. Garcia-Molina: *Crawling the Hidden Web*, 27th International Conference on Very Large Data Bases, September 2001.

[20] Search Engine Watch, `http://www.searchenginewatch.com/reports/`.

[21] J. Talim, Z. Liu, Ph. Nain, E. G. Coffman: *Controlling the robots of Web search engines*, Joint international conference on on Measurement and modeling of computer systems, 2001.

[22] P.N. Tan, V. Kumar: *Discovery of Web Robots Session Based on their Navigational Patterns*, Available on-line at `http://citeseer.nj.nec.com/443855.html`