Chapter 3

A New Crawling Model and Architecture

Web crawlers have to deal with several 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 pages, but at the same time they must discover new pages (which are found in the modified 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 many “good pages”, but they cannot exactly know in advance which pages are the good ones.

In this chapter, we present a model that tightly integrates crawling with the rest of a search engine and gives a possible answer to the problem of how to deal with these contradicting goals, by means of adjustable parameters. We show how this model generalizes several particular cases, and leads to a new crawling software architecture.

The rest of this chapter is organized as follows: Section 3.1 discusses the problems of the typical crawling model. Section 3.2 shows how to separate short-term and long-term scheduling, and Section 3.3 shows how to combine page freshness and quality to obtain an efficient crawling order. Section 3.4 introduces a general crawler architecture which is consistent with these observations.

Note: portions of this chapter have been presented in [CBY02, BYC02].

3.1 Problems of the typical crawling model

Crawling literature emphasizes on the words “crawler” and “spider”, and those words suggests walking through a directed graph. That is very far from what is really happening, because crawling is just automatic page downloading which does not need to follow a browsing-like pattern: in most cases a breadth-first approach is favored or something entirely different that has no direct topological representation on the Web graph.

The typical crawling algorithm comes from the early days of the World Wide Web, and it is given by Algorithm 1. We consider that this algorithm can be improved, because during crawling it is not necessary to add the newly found URLs to Q each and every time a Web page is parsed. The new URLs can be added in groups or “batches”, because:

Indexing is done in batches. The crawling process adds information to a *collection* that will be *indexed*. The indexing process is done in batch, many megabytes of text at a time, and with current algorithms it is very inefficient to do it one document at a time, unless you can achieve an exact balance between the incoming stream of documents and the processing speed of the index [TLNJ01], and in this case,

Algorithm 1 Typical crawling algorithm

```
Require: $p_1, p_2, ..., p_n$ starting URLs
1:  $Q = \{p_1, p_2, ..., p_n\}$, queue of URLs to visit.
2:  $V = \emptyset$, visited URLs.
3:  while $Q \neq \emptyset$ do
4:      Dequeue $p \in Q$, select $p$ according to some criteria.
5:      Do an asynchronous network fetch for $p$.
6:      $V = V \cup \{p\}$
7:      Parse $p$ to extract text and extract outgoing links $\rightarrow \Gamma^+(p)$.
8:      for each $p' \in \Gamma^+(p)$ do
9:          if $p' \notin V \wedge p' \notin Q$ then
10:             $Q = Q \cup \{p'\}$
11:         end if
12:     end for
13: end while
```

the index construction becomes the bottleneck. Thus, in most search engines, the index is not updated continuously but entirely at the same time (to the best of our knowledge, this is the case for most large search engines – there is even a term coined for the update of Google's index, which is called "Google dance"). When the index is updated in batches, it is not important which URL were transferred first.

Distributed crawlers exchange URLs in batches. If the crawler is distributed, then it has to sent the results back to a central server, or it has to exchange results with other crawling processes. For better performance, it must send many URLs at a time, as the exchange of URLs generates an overhead which is mostly given by the context switches, not for the (relatively small) size of the URLs [CGM02]. This means that how *locally* the URLs are ordered should not impact the *global* crawling order.

The important URLs are seen earlier in the crawl. If some URL ordering is done and if this ordering is not based on text-similarity to a query, then in permanent regime a page that we have just seen is a very unlikely candidate to be downloaded in the near future: "good" pages are seen early in the crawling process [NW01]. Conversely, if a URL is seen for the first time in a late stage of the crawling process, there is a high probability that it is not a very interesting page. This is obviously true if Pagerank [PBMW98] is used, because it reflects the time a random surfer "spends" at the page and if a random surfer spends more time in a page, he surely will find the page early.

We have noticed that previous work tends to separate two similar problems and to mix two different problems:

- The two different problems that are usually mixed are the problem of short-term efficiency (maximizing the bandwidth usage and being polite with servers) and long-term efficiency (ordering the crawling process to download important pages first). We discuss why these two problems can be separated in Section 3.2.
- The two related problems which are usually treated as separate issues are the index freshness and the index intrinsic quality. We consider that it is better to think in terms of a series of scores related to different characteristics of the documents in the collection, *including freshness*, which should be weighted accordingly to some priorities that vary depending on the usage context of the crawler. This idea is further developed in Section 3.3.

3.2 Separating short-term from long-term scheduling

If we are going to separate short-term scheduling from long-term scheduling in a crawler's architecture, first we must prove that these two problems are not related. We must confirm that the intrinsic quality of a Web page or a Web server is not related to the bandwidth available to download that page. If that were the case, then the two problems would be tightly coupled and no separation would be possible.

We designed and ran the following experiment to validate this hypothesis. We took one thousand Chilean site names at random from the aproximately 50,000 currently existing. We accessed the home page of these Web sites repeatedly each 6 hours during a 2-weeks period, and measured the connection speed (bytes/second) and latency (seconds). To get a measure of the network transfer characteristics and avoid interferences arising from variations in the connections to different servers, pages were accessed sequentially (not in parallel).

From the 1000 home pages, we were able to measure succesfully 750 of them, as the others were down during a substantial fraction of the observed, or did not answered with an actual Web page to our request. In the analysis, we consider only Web sites which answered to the requests.

As a measure of the "importance" of Web sites, we used the number of in-links from different Web sites in the Chilean Web, as this is a quantitative measure of the popularity of the Web site among other Web site owners.

We measured the correlation coefficient between the number of in-links and the speed ($r = -0.007$), and between the number of in-links and the latency ($r = -0.016$). The correlation between these parameters is not statistically significant. These results show that the differences in the network connections to "important" pages and "normal" pages are not relevant to long-term scheduling. Figure 3.1 shows a scatter plot of connection speed and number of in-links.

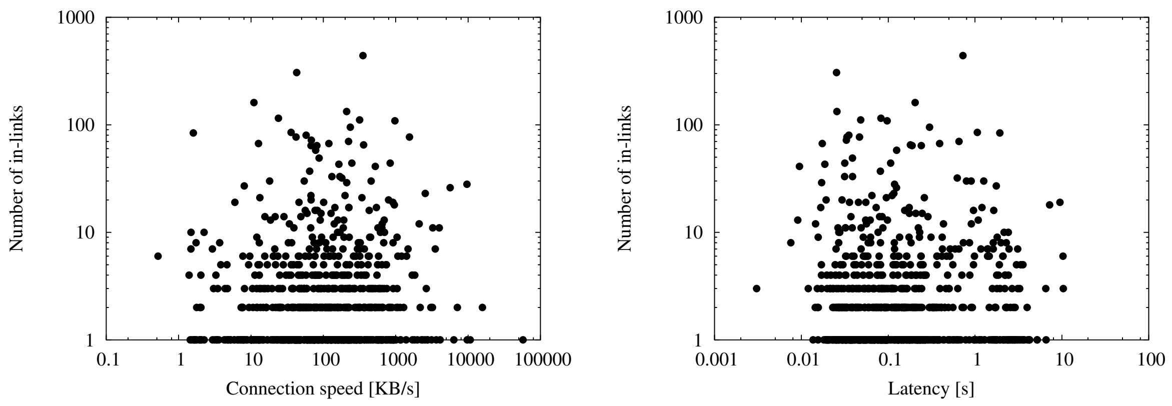

Figure 3.1: Scatter plot of connection speed (in Kilobytes per second) and latency (in seconds) versus popularity (measured as the number of in-links). In our experiments, we found no significant correlation between these variables.

We also used the data gathered during this experiment to measure the correlation between the connection speed and latency ($r = -0.063$), which is still low, as shown in Figure 3.2. In the graph, we can see that Web sites with higher bandwidths do not have high latency times.

Another interesting result that we obtained from this experiment was that connection speeds and latency times varied substantially during the observed period. We found on average a relative deviation of 42% for speed and 96% for latency, so these two quantities cannot be predicted based only on their observed mean values. It is likely that the daily and weekly periodicity in Web server response time observed by Liu [Liu98]

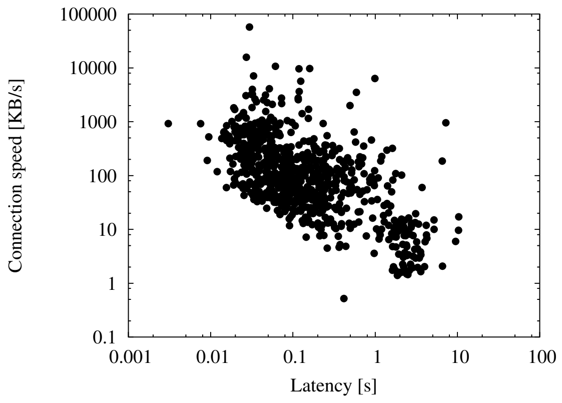

Figure 3.2: Scatter plot of connection speed versus latency.

has to be considered for a more accurate prediction.

3.3 Combining page freshness and quality

A search engine's crawler is designed to create a collection of pages which is useful for the search engine's index. To be useful, the index should balance comprehensiveness and quality. These two goals compete between them, because at each scheduling step, the crawler must decide between downloading a new page, not currently indexed, or refreshing a 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 the following, we propose a function to measure the quality of the index of a search engine. The crawler's goal is to maximize this function.

We start by stating three factors which are relevant for the quality of a Web page in the index:

Intrinsic quality of the page. The index should contain a large number of Web pages that are *interesting* to the search engine's users. However, the definition of what will be interesting for users is a slippery one, and currently a subject of intense research. A number of strategies have been proposed [CGMP98, DCL$^+$00, NW01], usually relying in a ranking function for ordering the list of objects found by the search engine.

We cannot known in advance the interest that a Web page will have to users, but we can approximate it [CGMP98] using a ranking function which considers the partial information that the crawler has obtained so far during its process.

The intrinsic quality of a page can be estimated in many ways [CGMP98]:

- Link analysis (link popularity).
- Similarity to a driven query.
- Accesses to the page on the index (usage popularity).
- Location-based, by the perceived depth (e.g. number of directories on the path to the Web object).
- Domain name, IP address or segment, geography, etc.

Representational quality of the page in the index. Every object on the index should *accurately represent* a real object on the Web. This is related to both the amount of data stored about the object (e.g.: it is not the same to index just the first 200 words than to index the full page) and to the rendering time of the object (e.g.: compression [WMB99] uses less space but increases the rendering time).

The representational quality depends mainly on the quantity and format of the information being stored for every object. In the case of Web pages, we can order the representational quality from less to more:

- URL.
- URL + Index of text near links to that page.
- URL + Index of text near links to that page + Index of the full text
- URL + Index of text near links to that page + Index of the full text + Text "snippet" (extract).
- URL + Index of text near links to that page + Index of the full text + Full text.

Rendering time depends on the format, particularly if compression is used. Some adaptivity can be used, e.g.: text or images could be compressed except for those objects in which a large representational quality is required, because they are accessed frequently by the search engine.

At this moment, Google [goo04] uses only two values, either RepresentationalQuality$(p_i) = high$ and the Web page is stored almost completely, or RepresentationalQuality$(p_i) = low$ and only the URL and the hyperlink anchor texts to that URL are analyzed. Note that in this case a page can be in the index without never have been actually downloaded: the index for these pages is built using the URL and a few words that appeared on the context of links found towards that page. In the future, the page can be visited and its representational quality can increase, at the expense of more storage space and more network transfers.

Freshness of the page. Web content is very dynamic, and the rate of change of Web pages [DFKM97, BCS$^+$00] is believed to be between a few months to one year, with the most popular objects having a higher rate of change than the others. We expect to maximize the probability of a page being fresh in the index, given the information we have about past changes: an estimator for this was shown in Section **??**.

Keeping a high freshness typically involves using more network resources to transfer the object to the search engine.

For the *value* of an object in the index, $V(p)$, a product function is proposed:

$$V(p) = \text{IntrinsicQuality}(p)^{\alpha} \times \text{RepresentationalQuality}(p)^{\beta} \times P(\text{Freshness}(p) = 1)^{\gamma} \tag{3.1}$$

The parameters α, β and γ are adjustable parameters of the crawler, and depend on the objective and policies of it. Other functions could be used, as long as they are increasing in the relevant quality measures, and allow to specify the relative importance between these values. We propose to use a product because the distribution of quality is very skewed and we usually will be working with the logarithm of the ranking function.

We propose that the *value* of an index $I = \{p_1, p_2, ..., p_n\}$ is the sum of the values of the objects p_i stored on the index:

$$V(I) = \sum_{i=1}^{n} V(p_i) \tag{3.2}$$

Depending on the application, other functions could be used to aggregate the value of individual elements into the value of the complete index, as long as they are non-decreasing on every component. For instance, a function such as $V(I) = \min_i V(p_i)$ could be advisable if the crawler is concerned with ensuring a baseline quality for all the objects in the index.

The proposed model covers many particular cases, which differ on the relative importance of the measures described above. In Figure 3.3, different types of crawlers are classified in a taxonomy based on our three factors.

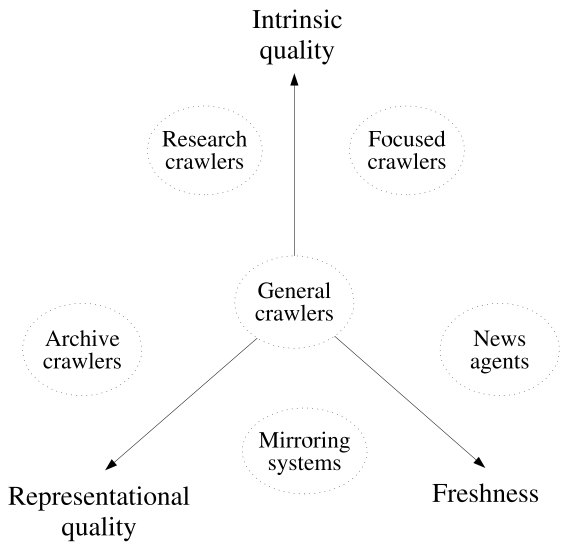

Figure 3.3: Crawler types can be classified based on the relative importance given to different objectives.

Research crawlers (e.g.: CiteSeer [cit04]) and focused crawlers are mostly interested in the intrinsic quality of the downloaded pages. Archive crawlers (e.g.: Internet Archive [arc04]) are mostly interested in keeping and accurate copy of the existing pages. News agents and mirroring systems are mostly interested in having fresh copies of the pages. General, large-scale crawlers are in the center of the graph, as they have to balance all the different aspects to have a good index.

3.4 A Software Architecture

The observations presented in the previous sections can be used to design a new crawling architecture. The objective of the design of this crawling architecture is to divide the crawling task into different tasks which will be carried efficiently by specialized modules.

We consider four main modules which cooperate to maintain the collection, as shown in Figure 3.4:

Manager: page value calculations and long-term scheduling.

Harvester: short-term scheduling and network transfers.

Gatherer: parsing and link extraction.

Seeder: URL resolving and link structure.

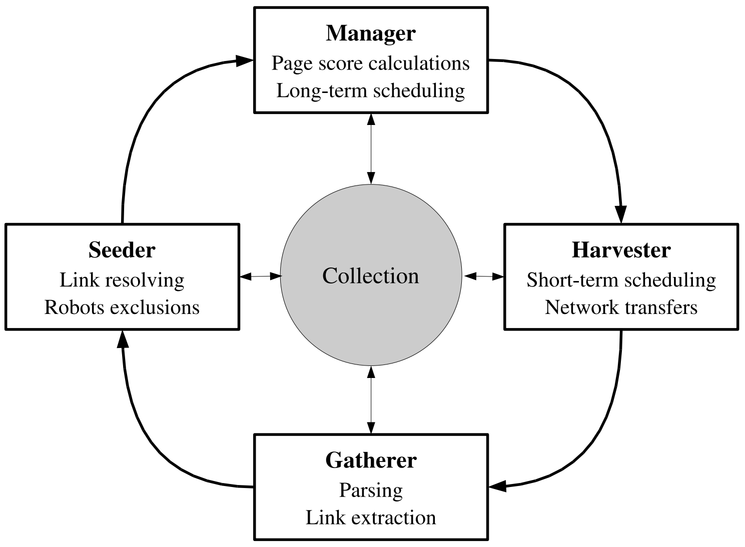

Figure 3.4: The proposed software architecture has a manager, that generates batches of URLs to be downloaded by the harvester. The pages then go to a gatherer that parses them and send the discovered URLs to a seeder.

Figure 3.5 introduces the main data structures that form the index of the search engine, and outlines the steps of the operation:

1. **Efficient crawling order** Long-term scheduling is done by the “manager” module, which generates the list with URLs that should be downloaded by the harvester in the next cycle (a “batch”). The objective of this module is to maximize the “profit” (i.e.: the increase in the index value) in each cycle.

2. **Efficient network transfers** Short-term scheduling is assigned to the “harvester” module. This module receives batches of URLs and its objective is to download the pages in the batch as fast as possible, using multiple connections and enforcing a politeness policy. The harvester generates a partial collection, consisting mostly of raw HTML data.

3. **Efficient page parsing** The extraction of the text and links is assigned to the “gatherer” module. This module receives the partial collections downloaded by the harvester(s) and adds the text to the main collection. It also generates a list of found URLs which are passed to the seeder.

4. **Efficient URL manipulation** The URLs found are processed by a “seeder” module, which searches for new URLs which have not been seen before. This module also check for URLs that shouldn’t be crawled because of the `robots.txt` exclusion protocol. The module maintains a data structure describing Web links.

3.5 Conclusions

The proposed model considers the index and the Web as a whole, and does not focus into separate problems, generalizing the current search engines. This model induces a crawler architecture, which we implemented in the WIRE crawler. The next chapter provides details on this implementation.

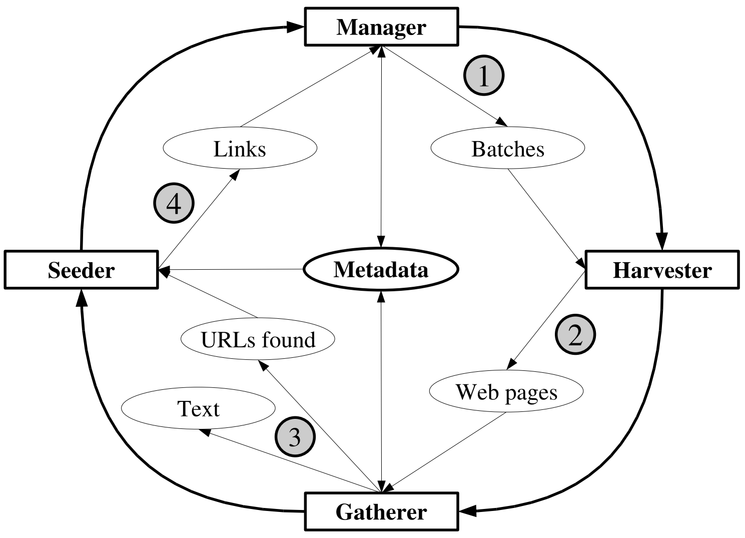

Figure 3.5: The main data structures, and the operation steps of the crawler: (1) the manager generates a batch of URLs, (2) the harvester downloads the pages, (3) the gatherer parses the pages to extract text and links, (4) the seeder checks for new URLs and maintains the link structure.

Bibliography

[arc04] Internet archive project. http://www.archive.org/, 2004.

[BCS$^+$00] Brian Brewington, George Cybenko, Raymie Stata, Krishna Bharat, and Farzin Maghoul. How dynamic is the web? In *Proceedings of the Ninth Conference on World Wide Web*, pages 257 – 276, Amsterdam, Netherlands, May 2000.

[BYC02] Ricardo Baeza-Yates and Carlos Castillo. Balancing volume, quality and freshness in web crawling. In *Soft Computing Systems - Design, Management and Applications*, pages 565–572, 2002.

[CBY02] Carlos Castillo and Ricardo Baeza-Yates. A new crawling model. In *Poster proceedings of the eleventh conference on World Wide Web*, Honolulu, Hawaii, USA, May 2002. (Extended Poster).

[CGM02] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the eleventh international conference on World Wide Web*, pages 124–135, Honolulu, Hawaii, USA, May 2002. ACM Press.

[CGMP98] Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Proceedings of the seventh conference on World Wide Web*, Brisbane, Australia, April 1998.

[cit04] Cite seer. http://citeseer.nj.nec.com/, 2004.

[DCL$^+$00] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proceedings of 26th International Conference on Very Large Databases (VLDB)*, pages 527–534, Cairo, Egypt, September 2000.

[DFKM97] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems*, pages 147–158, Monterey, California, USA, December 1997.

[goo04] Google search engine. http://www.google.com/, 2004.

[Liu98] Binzhang Liu. Characterizing web response time. Master's thesis, Virginia State University, Blacksburg, Virginia, USA, April 1998.

[NW01] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *Proceedings of the Tenth Conference on World Wide Web*, pages 114–118, Hong Kong, May 2001. Elsevier Science.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation algorithm: bringing order to the web. In *Proceedings of the seventh conference on World Wide Web*, Brisbane, Australia, April 1998.

[TLNJ01] Jerome Talim, Zhen Liu, Philippe Nain, and Edward G. Coffman Jr. Controlling the robots of web search engines. In *Proceedings of ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*, pages 236–244, Cambridge, Massachusetts, USA, June 2001.

[tod04] TodoCL - Todo Chile en Internet. http://www.todocl.cl/, 2004.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishing, 1999.