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.

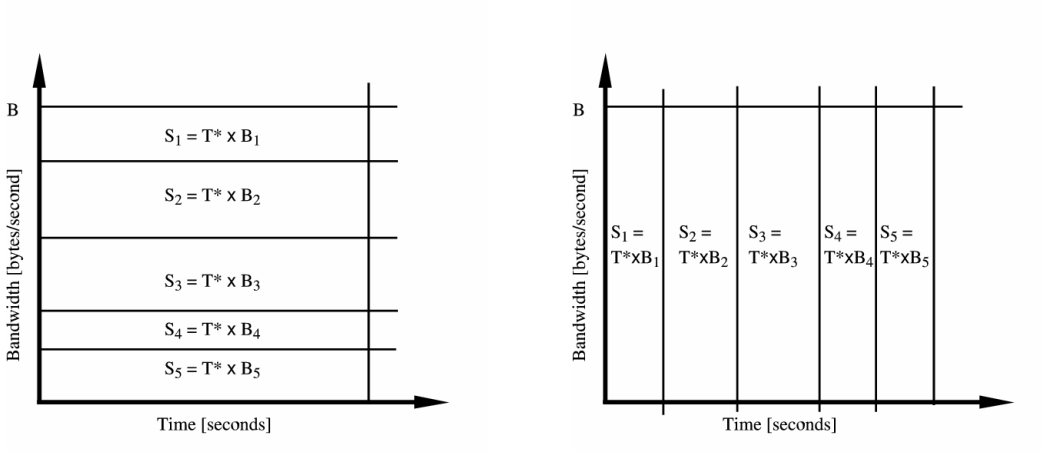
In this chapter, we present a model that tightly integrates crawling with the rest of a search engine and gives a possible answer to the problem of how to deal with these contradictory goals, by means of adjustable parameters. We show how this model generalizes several particular cases, and leads to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

The rest of this chapter is organized as follows: Section 3.1 presents the problem of crawler scheduling, and Section 3.2 discusses the problems of a typical crawling model. Section 3.3 shows how to separate short-term and long-term scheduling, and Section 3.4 shows how to combine page freshness and quality to obtain an efficient crawling order. Section 3.5 introduces a general crawler architecture that is consistent with these observations.

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

3.1 The problem of crawler scheduling

We consider a Web crawler that has to download a set of pages, with each page p having size S_p measured in bytes, using a network connection of capacity B, measured in bytes per second. The objective of the crawler is to download all the pages in the minimum time. A trivial solution to this problem is to download all the Web pages simultaneously, and for each page use a fraction of the bandwidth proportional to the size of each page. If B_p is the downloading speed for page p, then:



(a) Full parallelization (b) Full serialization

Figure 3.1: Two unrealistic scenarios for Web crawling: (a) parallelizing all page downloads and (b) serializing all page downloads. The areas represent page sizes, as s*ize* = s*peed* × t*ime*.

$$B_p = \frac{S_p}{T^*} \tag{3.1}$$

Where T^* is the optimal time to use all of the available bandwidth:

$$T^* = \frac{\sum_p S_p}{B} \tag{3.2}$$

This scenario is depicted in Figure 3.1a.

However, there are many restrictions that forbid this optimistic scenario. One restriction is that a scheduling policy must avoid overloading Web sites, enforcing a politeness policy as described in Section **??**: a Web crawler should not download more than one page from a single Web site at a time, and it should wait several seconds between requests.

Instead of downloading all pages in parallel, we could also serialize all the requests, downloading only one page at a time at the maximum speed, as depicted in Figure 3.1b. However, the bandwidth available for Web sites B_i^{MAX} is usually lower than the crawler bandwidth B, so this scenario is not realistic either.

The presented observations suggest that actual download time lines are similar to the one shown in Figure 3.2. 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 the crawler must wait between accesses to a Web site (in the figure, pages $1-2$ and $4-5$ belong to the same site, and the crawler waits w seconds between them).

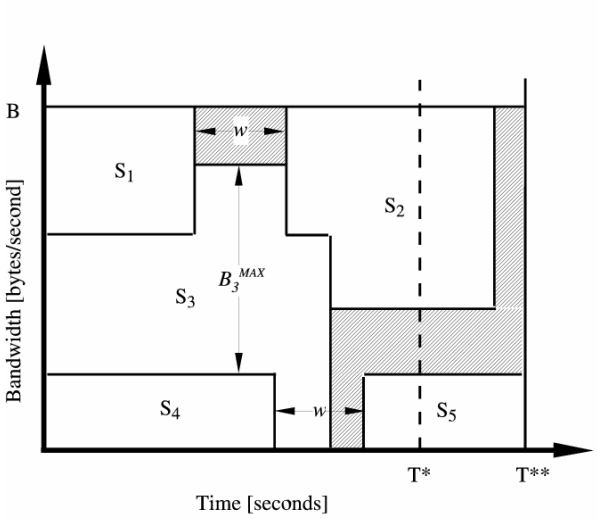


Figure 3.2: A more realistic download time line for Web crawlers. The hatched portion is wasted bandwidth due to the constraints in the scheduling policy. The optimal time T^* is not achieved.

To overcome the problems shown in Figure 3.2, it is clear that we should try to saturate the network link, downloading 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 3.3, 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 a high utilization of the available bandwidth.

There is another serious practical constraint: the HTTP request has latency, and the latency time can be over 25% of the total time of the request [**?**]. 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.

3.2 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 that does not need to follow a browsing-like pattern: in some cases a breadth-first approach is used, in other cases the crawling is done in a way that has not an obvious representation on the Web graph, and does not resembles a graph traversal.

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

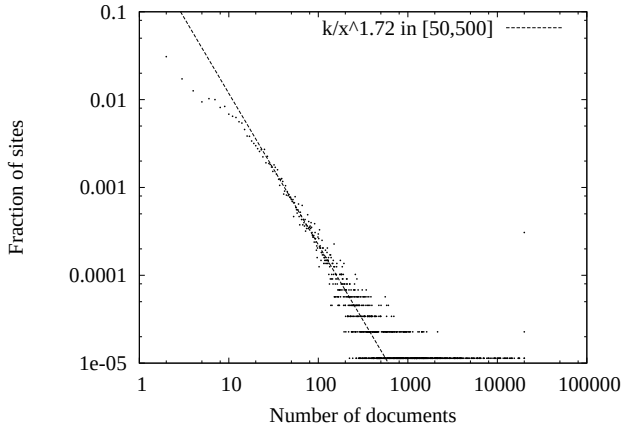


Figure 3.3: Distribution of site sizes in a sample of the Chilean Web. There are a few Web sites that are very large, and a large fraction of small Web sites. This poses a problem to Web crawlers, as they must download several pages from a small number of Web sites but at the same time they must avoid overloading them with requests.

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 one can achieve an exact balance between the incoming stream of documents and the processing speed of the index [TLNJ01], and in this case, the index construction becomes the bottleneck. Thus, in most search engines, the index is not updated continuously but completely at the same time. To the best of our knowledge, this is the case for most large search engines, and there is even a term coined for the update of Google's index ("Google dance"), when the new index is distributed to the different data centers [**?**]. When the index is updated in batches, it is not important which URLs were transferred first.

Distributed crawlers exchange URLs in batches. If the crawler is distributed, then it has to send 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 that is mostly given by the context switches, not for the (relatively small) size of the URLs [CGM02]. This means that how the URLs are ordered *locally* 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 steady state 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

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
8: $\Gamma^+(p) \leftarrow$ pages pointed by p
9: **for** each $p' \in \Gamma^+(p)$ **do**
10: **if** $p' \notin V \wedge p' \notin Q$ **then**
11: $Q = Q \cup \{p'\}$
12: **end if**
13: **end for**
14: **end while**

[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, then probably the page can be reached from several links.

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 3.3.
- The two related problems that 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 3.4.

3.3 Separating short-term from long-term scheduling

We intend to deal with long-term scheduling and short-term scheduling separatelly. To be able to do this, we must prove that both problems can be separated, namely, we must check if the intrinsic quality of a Web

page or a Web server is related to the bandwidth available to download that page. If that were the case, then we would not be able to select the most important pages first and later re-order pages to use the bandwidth effectively, because while choosing the important pages we would be affecting the network transfer speed.

We designed and ran the following experiment to validate this hypothesis. We took one thousand Chilean site names at random from the approximately 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 successfully measure 750 of them, as the others were down during a substantial fraction of the observed period, or did not answer our request with an actual Web page. In the analysis, we consider only Web sites that 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 r between the number of in-links and the speed ($r = -0.001$), and between the number of in-links and the latency ($r = -0.069$). 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 3.4 shows a scatter plot of connection speed and number of in-links.

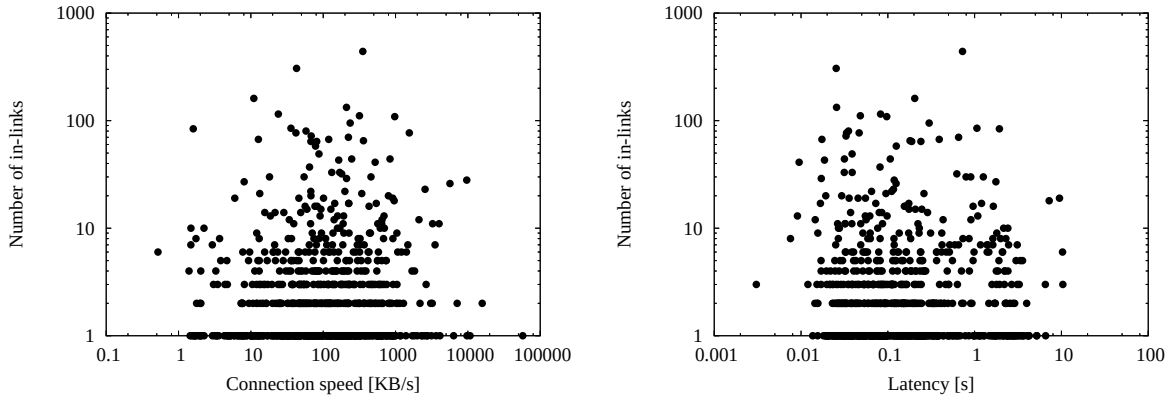


Figure 3.4: Scatter plot of connection speed (in Kilobytes per second) and latency (in seconds) versus popularity (measured as the number of in-links). In our experiments, we found no significant correlation between these variables. The results are averages of the measures obtained by connecting to 750 Web sites sequentially every 6 hours during a 2-weeks period.

For completeness, we also used the data gathered during this experiment to measure the correlation

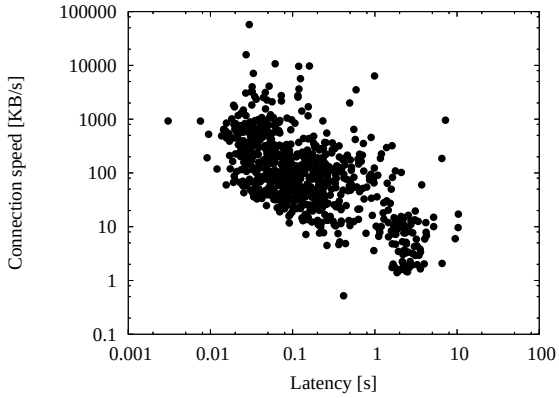


Figure 3.5: Scatter plot of connection speed versus latency. Web sites with low connection speeds tend to have also low latency.

between the connection speed and latency ($r = -0.645$), which is high, as shown in Figure 3.5. In the graph, we can see that Web sites with higher bandwidths tend to have low 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. The daily and weekly periodicity in Web server response time observed by Liu [Liu98] has to be considered for a more accurate prediction: Diligenti *et al.* [**?**] maintain several observed values for predicting connection speed, and group the observations by time of the day to account for the periodicity in Web server response time.

3.4 Combining page freshness and quality

A search engine's crawler is designed to create a collection of pages that is useful for the search engine's index. To be useful, the index should balance comprehensiveness and quality. These two goals compete, 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 that 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 that 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 given 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 in the index should *accurately represent* a real object in 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 may increase 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 + Summary of text ("snippet") extracted using natural language processing techniques or simply by taking a few words from the beginning of the text
- URL + Index of text near links to that page + Index of the full text + Full text.

There are other possibilities, involving indexing just portions of the page using the HTML markup as a guide, i.e., indexing only titles, metadata and/or page headings.

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 ever having been actually downloaded: the index for these pages is built using the URL and a few words that appeared in 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.

There is no reason why this should be a binary variable. A selective index, which indexes partially certain pages and completely other pages can be a good solution for saving disk space in the future, especially if the distance between storage capacity and the amount of information available on the Web further increases.

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 **??** (page **??**).

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:

$$V(p) = \text{IntrinsicQuality}(p)^{\alpha} \times \text{RepresentationalQuality}(p)^{\beta} \times \text{Freshness}(p)^{\gamma} \tag{3.3}$$

The parameters α, β and γ are adjustable by the crawler's owner, 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 and rate of change are very skewed and we usually will be working with the logarithm of the ranking function for the intrinsic quality.

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:

$$V(I) = \sum_{i=1}^{n} V(p_i) \tag{3.4}$$

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.

A good coverage, i.e., indexing a large fraction of the available objects, certainly increases the value of an index, but only if the variables we have cited: intrinsic quality, representational quality and freshness are considered. Coverage also depends on freshness, as new pages are usually found only on changed pages.

The proposed model covers many particular cases that differ on the relative importance of the measures described above. In Figure 3.6, different types of crawlers are classified in a taxonomy based on the proposed three factors.

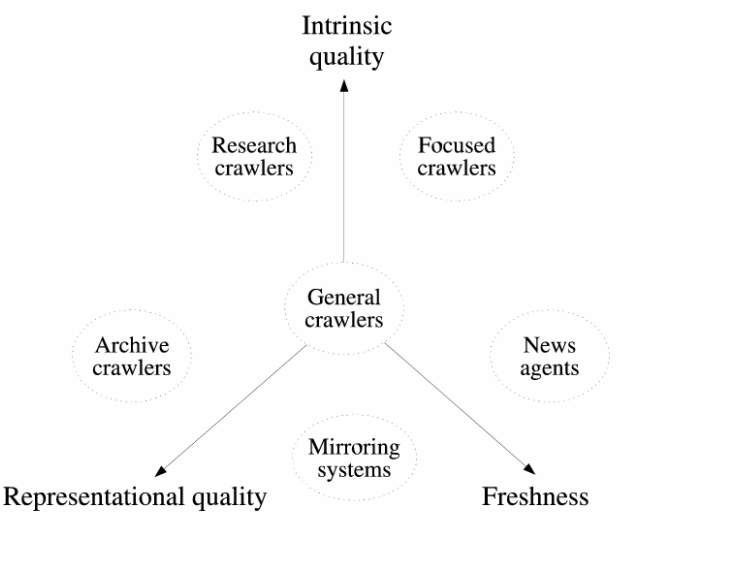


Figure 3.6: Different types of Web crawlers can be classified in our framework, based on the relative importance given to freshness, representational quality and intrinsic quality.

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 an 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.

3.5 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 that will be carried efficiently by specialized modules.

A separation of tasks can be achieved with two modules, as shown in Figure 3.7. The *scheduler* calculates scores and assigns pages to several *downloader* modules that transfer pages through the network, parse their contents, extract new links and maintain the link structure.

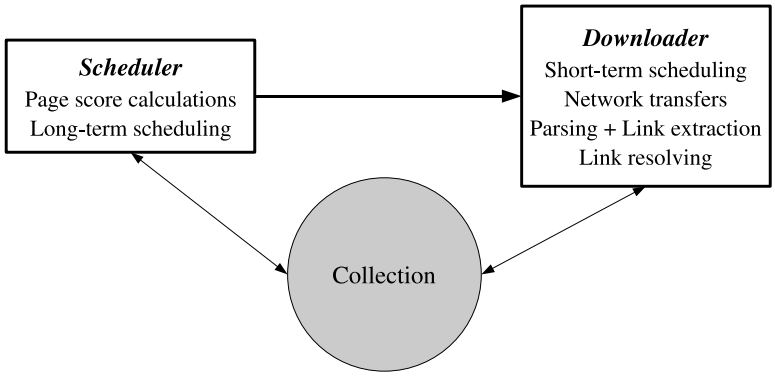


Figure 3.7: A software architecture with two modules. A "batch" of pages is generated by the *scheduler*, and downloaded and parsed by the *downloader*. Under this scheme, the scheduler requires read access to the collection, and the downloader read and write access.

There are some problems with this two-module architecture. One problem is that for the scheduler to work on the Web graph, during the calculations, the Web graph cannot change. So, the the process of modifying the Web graph should be as fast as possible, but parsing the pages can be slow and this could mean that we have to "lock" the Web graph during a long time. What can be done to overcome this is to parse all pages and accumulate links, and then add all the links found to the collection.

Another issue is that we could have different, optimized hardware architectures for the tasks of downloading and storing pages and for the task of parsing pages. Parsing pages can be expensive in terms of processing, while downloading pages requires mostly high network connectivity and fast disks. Moreover, if the network downloads must be carried with high parallelism, then each downloading task should be very lightweight. To solve these issues we divide the tasks of downloading, parsing and keeping the link structure, as shown in Figure 3.8. The following module names are used through the thesis:

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 3.9 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.

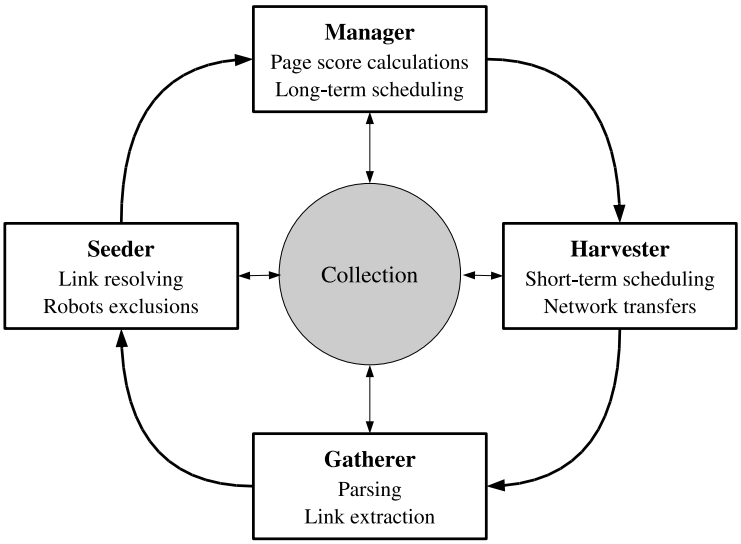


Figure 3.8: The proposed software architecture has a manager, that generates batches of URLs to be downloaded by the harvester. The pages then go to a gatherer that parses them and send the discovered URLs to a seeder.

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 that are passed to the seeder.

4. Efficient URL manipulation The URLs found are processed by a "seeder" module, which searches for new URLs that have not been seen before. This module also checks for URLs that should not be crawled because of the `robots.txt` exclusion protocol, described in Section **??** (page **??**). The module maintains a data structure describing Web links.

The pattern of read and write accesses to the data structures is designed to improve the scalability of the crawler as, for instance, the pages can be downloaded and parsed while the Web graph is analyzed, and the analysis only must stop while the seeder is running.

The programs and data structures in Figure 3.9 are explained in detail in Chapter **??**.

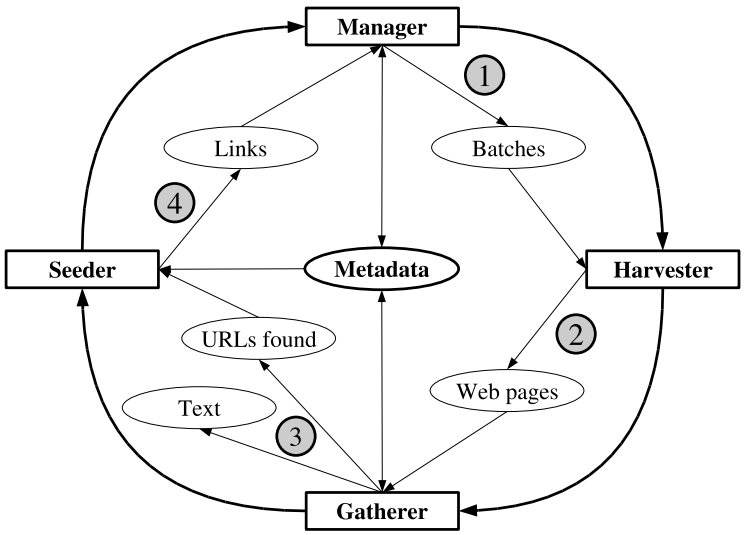


Figure 3.9: The main data structures and the operation steps of the crawler: (1) the manager generates a batch of URLs, (2) the harvester downloads the pages, (3) the gatherer parses the pages to extract text and links, (4) the seeder checks for new URLs and maintains the link structure.

3.6 Conclusions

Web crawling is not only a trivial graph traversal problem. It involves several issues that arise from the distributed nature of the Web. First, Web crawlers must share resources with other agents, mostly with humans, and cannot monopolize Web sites' time –indeed, a Web crawler should try to minimize its impact on Web sites. Second, Web crawlers have to deal with an information repository which contains many objects of varying quality, including objects with very low quality created to lure the Web crawler and deceive ranking schemes.

We consider the problem of Web crawling as a process of discovering relevant objects, and one of the main problems is that a Web crawler always works with partial information, because it must infer the properties of the unknown pages based on the portion of the Web actually downloaded. In this context, the Web crawler requires access to as most information as possible about the Web pages.

While the model implies that all the portions of the search engine should know all the properties of the Web pages, the architecture introduced in this chapter is an attempt of separating these properties into smaller units (text, link graph, etc.) for better scalability. This architecture is implemented in the WIRE crawler and details on the implementation of the WIRE crawler is explained in Chapter **??**.

Benchmarking this architecture requires a framework that allows comparisons in different settings of network, processor, memory and disk, and during this thesis we did not carry any benchmark of this type. However, the findings about scheduling strategies, stop criteria and Web characteristics presented in the

following chapters are mostly independent of the chosen architecture.

The proposed model considers the processes of indexing and crawling the Web as a whole because it the context of today's Web, it is impossible to download all of the Web pages, furthermore, in Chapter **??** we argue that the number of Web pages is infinite, so the fraction of the Web that a crawler downloads should represent the most important pages. The next chapter studies algorithms for directing the crawler towards important pages early in the crawl.

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. IOS Press, 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.

[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.

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