A New Model for Web Crawling

Carlos Castillo
Ricardo Baeza-Yates

Computer Science Department
University of Chile
Blanco Encalada 2120, Santiago
Chile.
E-mail: {ccastill,rbaeza}@dcc.uchile.cl
Phone: (56 2) 689-2736

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.

Wordcount: 5,000

1 Introduction

On the dawn of the Web, finding information was done mainly by scanning through lists of links collected and ordered by humans according to some criteria. Web search engines were unnecessary with Web pages counting only by thousands. Nowadays, they are counted by thousands of millions and almost every internet user have to use a search engine when he has a specific information, navigation, or transaction need.

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.

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

In section 2 we present previous work on this subject. In section 3 we show the typical crawler model, that has been used since the first search engines. We argue (and show experimental validation) that the typical model is based on some assumptions that are no longer valid, and that it does not account for the last advances, specially on the area of Web page ranking. This has been recognized by newer search engines such as Google [18].

In section 4 we introduce a new model of crawling. This model considers different pieces of information before taking a decision on what pages to visit next, according to some parameters that can be adjusted. In section 5 a software architecture that implements this model is outlined, and it is shown that it has a low level of dependencies and information coupling and hence is very appropriate for parallelization on multiple machines. Section 6 presents our conclusions.

2 Previous Work

As Web crawlers are mostly used by search engines, their inner components are usually well guarded secrets and only benchmarks [31] are revealed. Some exceptions (in cronological order) follow:

An early work on the RBSE Spider [17], when the Web was estimated in 100.000 pages, shows a crawler in which a "spider" manages a database of URLs to visit, and spawns some process of another program, "mite", which fetch pages. This spider implements a series of graph walking algorithms, such as breadth-first or limited depth-first search.

The crawler of the Internet Archive [5] uses a series of queues to avoid overloading target sites, assigning all pages from a given site to a single crawling process that loops between the sites and wait if needed to let sites "rest".

The Mercator crawler [20] consists of an "URL Frontier" that gives URLs to a number of "Protocol Modules" that fetch pages and pass them to "Processing Modules". The URL Frontier consist of a number of queues, as in the Internet Archive. Mercator uses several modular components, as SPHINX [25] does, and that inspires our modular architecture presented later.

The Google Web crawler, presented in [28], depicts a crawling architecture consisting of an "URL Server" which gives URLs to a number of "Crawlers", that fetch pages and send them to a "Store Server". In [12], it is shown that ordering URLs by Pagerank [29], the algorithm based on citation metrics that Google uses, leads to crawl "interesting" pages first.

The parallel crawler introduced in [7], has multiple crawling processes or "C-Procs", which exchange messages from time to time. In the proposed static setting, every site is assigned a unique C-Proc based on a hash function of the site name. A "coordinator" is an optional part of the architecture, used when a central, dynamic assignment of URLs to crawl is necessary. The high level crawling architecture of typical crawlers is depicted on Figure 1.

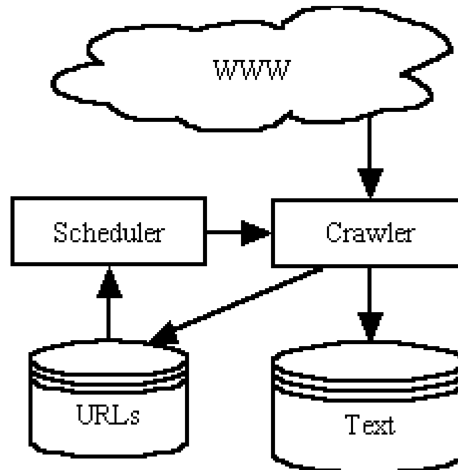

Figure 1: Typical high-level crawler architecture.

These papers mention the same problems during the crawling process: duplicate content (mirrors [6], URL aliases, session-ids in URLs), crawler traps and DNS lookups as a bottleneck. Other hardware and operating systems issues are not minor due to the intensive nature of the process. For a recent survey on the topic of search engines, see [1]; for background information on search engines, see [3].

The collection must be kept up to date in relation to the objects or Web pages it represent; this problem is addressed in [8, 13, 11] and requires knowing how Web pages change over time [9, 4, 16].

Finally, with the exponential growth of available Web pages available, Web crawlers cannot visit every existent page, and the covering of most search engines is very low [19, 30], so usually some ordering of the Web pages prior to crawling is used to focus on some "important" pages [12, 15, 27].

A related topic is to study how Web robots "behave" [33] and how they "misbehave" by using a lot of bandwitdh and/or CPU from the servers visited [24].

3 Problems of the Typical Crawling Model

The typical crawling model comes from the early days of the World Wide Web and was first used on large scale by Webcrawler [23]. The operation of it is given by:

1. Start with a queue Q and some starting URLs $u_1, ..., u_n \in Q$.
2. Extract some URL $u \in Q$ according to some criteria and do an asynchronous network fetch for u.

3. Process u to extract the text and add it to the local collection that will be indexed. During this process, find new URLs that are pushed into Q.

4. Go back to (2) unless Q is empty.

This model emphasizes on the words "crawler" and "spider", and those words suggests walking through a directed graph. That is very far from what is really going on, because it is just automatic browsing which does not need to follow Web links: in most cases a breadth-first approach is favored or something entirely different that has no direct topological representation on the Web graph.

Some important observations follow. The first is that this 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 usually will be very inefficient to do one document at a time, unless you can strike an exact balance between the incoming stream of documents and the processing speed of the index [32], and in that case it is common that the index construction becomes the bottleneck. In many cases, the index is not updated continuously but completely at the same time[1]; in this case it is not important which URL was transferred first, just that all the required URL were transferred.

A more strong argument that shows that each page does not need to be available immediatly to the index is that if the crawler is distributed, then it will have to sent the results back, and for better performance, it will have to send many URLs at a time, because as shown in [7], the overhead of an URL exchange is given by the context switches, not for the (relatively small) size of the URLs. This means that how *locally* the URLs are ordered is not important for the *global* order.

Another observation is that if some URL ordering will be done and if this ordering is not based on text-similarity to a query, then in permanent regime usually a page that we have just seen is a very unlikely candidate to be downloaded in the near future. This means that "good" pages are seen early in the crawling process, and hence good URLs to visit are not a scarce resource. 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 very interesting. This is obviously true if Pagerank [29] is used, because it models a link-biased random walk through the Web. In that case, pages that are visited very often are important, also if backlink count or the hubs and authorities score [22] is used.

Notice also that when connectivity metrics are used on the Web (such as the ones mentioned before), only a very small number of URLs have significant scores, and most pages have only very small values; mostly because the distribution of links on the Web is very skewed, with a few very popular pages (in even fewer sites) [2].

Finally, we have noticed that 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 contradictory, the software must decide between, 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 bandwitdh usage and be polite with servers (probably accounting some

[1] To the best of our knowledge, this is the case for most large search engines.

stationary variations of the Web servers during the day or the week [14]) and long-term efficiency (ordering the set to favor some important pages).

Next we present experimental results that validate some of our observations. We present evidence that the intrinsical quality of a Web page is not related to the bandwidth available to download that page. If that were the case, then the two problems will be tightly coupled and no separation could be possible.

We designed and run the following experiment to validate this hypothesis. We took two different Web page samples from the TodoCL [34] Web search engine, one at random and the other one from the search engine access log[2]. The first sample can be considered to represent "normal" pages, and the second sample "important" pages.

We observed that actual access times and effective transfer speed were very similar between these samples. For comprehensiveness, we include here some of the (slight) differences between them:

1. The biased sample has lower latency time, probably because they are more popular and so they are stored on disk pages that are cached in memory (or a proxy cache), or because Web server process is never swapped to disk.

2. The random sample has slightly higher speeds, probably because as they are less popular, they use less of their available bandwidth.

3. The random sample reports more hosts that are not found, probably because less popular servers disappear more often.

4. The selected sample servers use older Web server software, probably because they are older than the average.

The results show that the differences between the "important" pages and the "normal" pages are not enough to consider them in the long-term strategy.

The problem of short-term efficiency is an important and difficult one. It can be stated as follows. There is B (Bytes/Second) bandwidth available 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 $T = B/S$ (Seconds), as shown in Figure 2. 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 more realistic diagram is depicted in Figure 3.

Two extreme solutions are shown in Figure 4. Any crawler will perform between these two. It cannot have too many processes, but it if has too few, then inefficiency is impossible to avoid.

We propose a new model that accounts for these observations at the same time, and leads to a new crawler architecture.

[2]The search engine stores the addresses of the pages that are picked by the user from the search engine result.

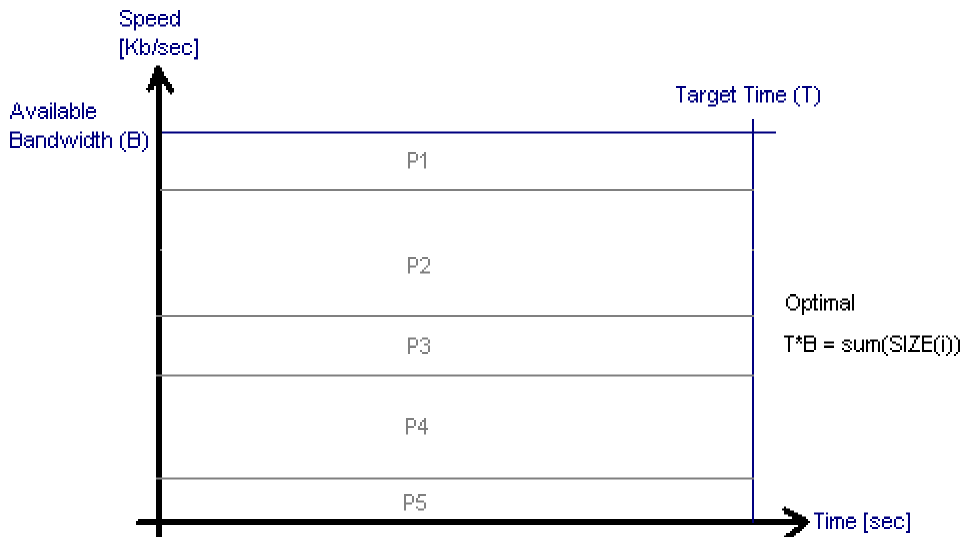

Figure 2: Optimal crawling scenario for five Web pages. The target time $T = B/S$ is achieved.

4 Our 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).

For the first goal, the definition of what will be *interesting* for users is a slippery one, and currently a subject of research. A number of strategies have been proposed [12, 15, 27], usually relying in some *ranking function* that is used to order the list of objects found by the search engine.

If the ranking function cannot be determined at crawling time (this is the more common scenario), then at least an approximation should be used. As stated in [12] we cannot know in advance the interest $f(p)$ that a Web page p will have to users, but we can approximate it using, by example, $f'(p) = Pagerank(p)$ or other function.

For the second goal, there is more consensus about the rate of change of Web pages [16, 4], finding that it is in general between a few months to one year, with the most popular objects having a higher rate of change than the rest.

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).

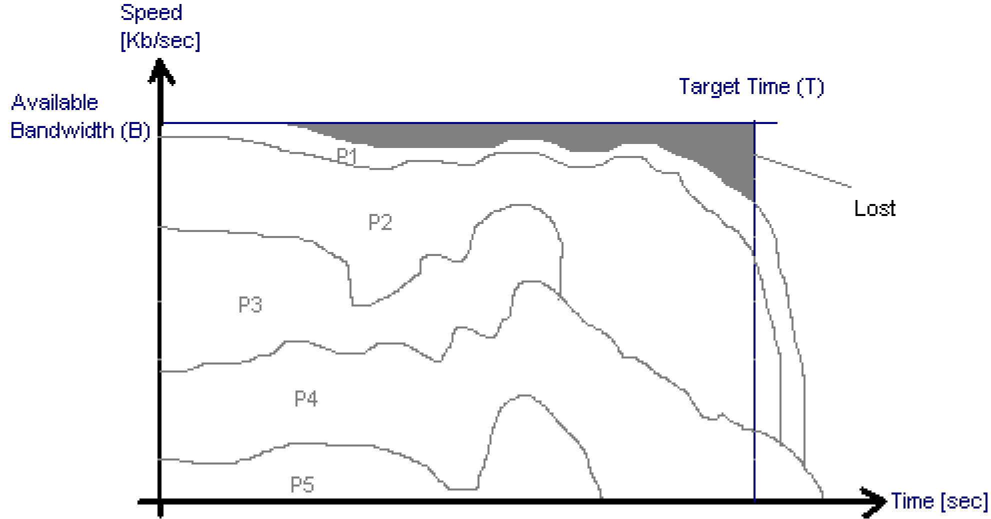

Figure 3: A more realistic crawling scenario. The crawling takes longer than expected, mainly due to variation on download speed of every page.

We propose that the *value* of an index $I = o_1, o_2, ..., o_n$ will be the sum[3] of the values of the objects o_i stored on the index:

$$V(I) = \sum_{i=1}^{n} V(o_i)$$

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

$$V(o_i) = q_i^a \times r_i^b \times p_i^c$$

where:

- 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.

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

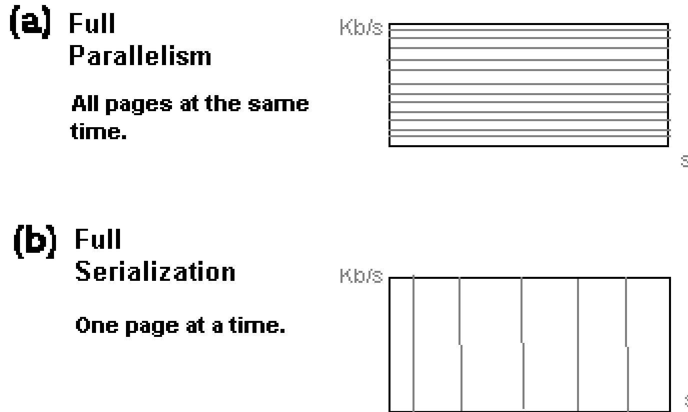

Figure 4: Two extreme solutions. (a) is impractical: it requires a very large number of processes. (b) is impossible, because Web sites are usually slower than the bandwidth available for the crawler.

Other functions could be used, as long as they are increasing in q_i, r_i and p_i and allow to specify the relative importance between these values[4].

The variable q_i can be estimated in many ways [12]:

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

On the other hand, r_i 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 quantity of 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.
- URL + Index of text near links to that page + Index of the full text + Full text.

[4] Notice that the computing effort to calculate this function can be simplified by using $\log V(o_i)$

Rendering time will depend on the format, particularly if it uses compression to give more fine-grained values. For instance, text or images can be compressed except for those objects in which a large r_i is required, because they are accessed frequently by the search engine.

At this moment, Google [18] uses only two values, either $r_i = high$ and the Web page is stored almost completely, or either $r_i = low$ 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 transferred to the search engine: we have 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 r_i can increase, with the cost of more storage space used.

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 [8]. If H_i hours have passed since the last visit, then [4]:

$$p_i = e^{-\lambda_i H_i}$$

The parameter λ is estimated as:

$$\lambda_i \approx \frac{(X_i - 1) - \frac{X_i}{N_i \log(1 - X_i/N_i)}}{S_i T_i}$$

- N_i: number of visits to the page.
- S_i: time since first visit.
- X_i: number of times the page has been found to have changed.
- T_i: total number of hours with no modification, according to the server, added for all visits.

If the server does not give the last-modified time:

$$\lambda_i \approx \frac{-N_i \log(1 - X_i/N_i)}{S_i}$$

Keeping a high p_i, means using more network resources to transfer the object to the search engine. But freshness is not always important, for instance, for a project like the *Internet Archive* [21] or *Cite Seer* [26], $c \approx 0$, because it is not important to update URLs, just getting the pages. But, for an specialized search engine that seeks news, c must be set high, because we are interested in having an accurate copy of the news pages.

The proposed model covers many particular cases. In particular, it covers mirroring systems, focused crawlers and research/archive crawlers, as shown in Figure 5. General, all-purpose search engine crawlers, are at the center of the graph.

In this way, we are proposing a model that considers the index and the Web as a whole, and does not focus into separate problems, generalizing the current search engines.

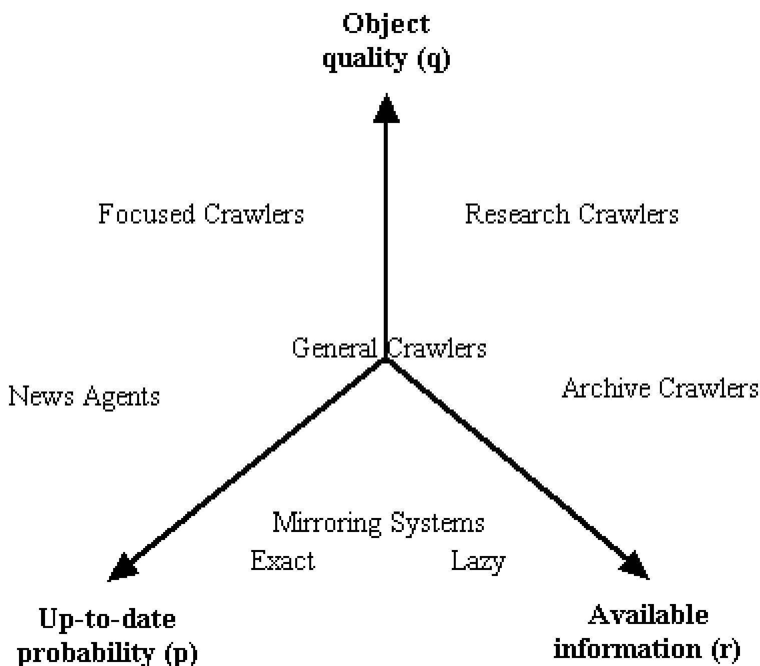

Figure 5: Different crawlers in our model

5 A Software Architecture

We present a software architecture that comes from the model and from the observations discussed previously, 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.

Note that some objects may "starve" in the repository, namely those with $q_i \approx 0$, which are most of the Web (less than 5% of Web pages have a meaningful Pagerank score [2]). The manager can then arbitrarily fix a minimum q_i for every object (a base score) if we are interested in having a good covering of available Web pages.

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 6 shows the whole architecture, which is currently under implementation. Each module is multithreaded. In the case of the manager, the new scores are recalculated in parallel. In the case of the harvester, multiple fetcher threads are used. For the gatherer, multiple parsers can be running simultaneously. Finally, the seeder must check every URL seen and that can also be done in parallel.

We are currently implementing this architecture for a vertical search engine and specialized, site-specific crawlers. We are using Intel hardware with SuSe Linux and the C programming language.

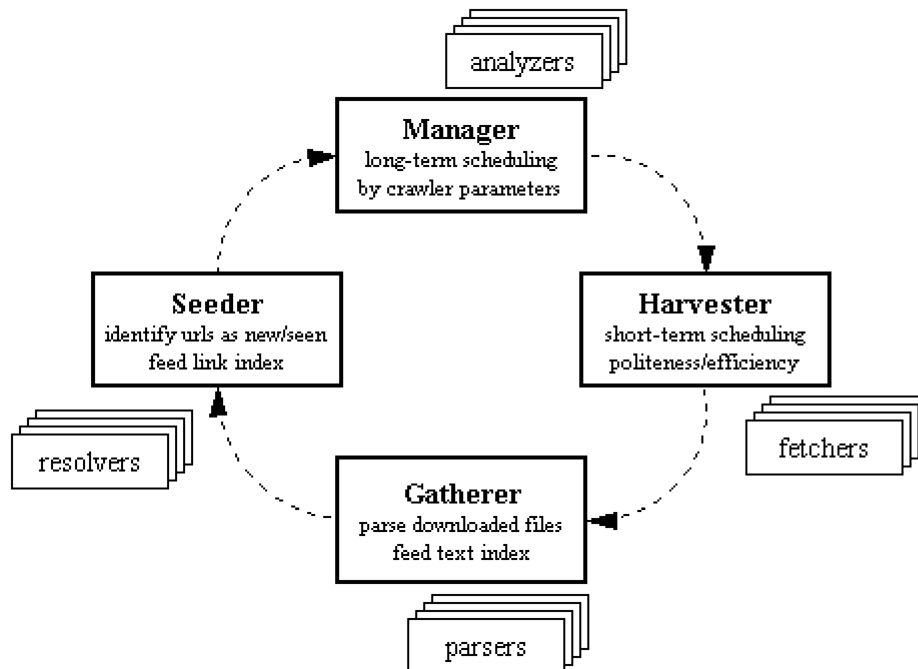

Figure 6: The proposed software architecture has a manager, that generates batchs of URLs to be downloaded by the harvester. The pages then go to a gatherer that parses them and send discovered URLs to a seeder.

6 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. Complete results will be presented in the final version.

Further work is needed to better estimate the intrinsic quality of a page, as current metrics are all heuristics, and they do not necessarily agree, as was shown for pagerank, authorities, and hubs in [2]. A better estimation of this parameter implies performance losses. For example, natural language processing is a perfect tool, but needs more CPU resources and we have presented a framework in which it could be decided to use it selectively only in some of the stored objects. The same is valid for multimedia objects.

References

[1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, S. Raghavan: *Searching The Web*, ACM Transactions on Internet Technology, August 2001

[2] R. Baeza-Yates, C. Castillo: *Relating Web characteristics with link analysis*, IEEE Cs. Press, Proc. of SPIRE 2001, pp. 21-32.

[3] R. Baeza-Yates, B. Ribeiro-Neto: *Modern Information Retrieval*, Addison-Wesley, 1999.

[4] B. Brewington, G. Cybenko, R. Stata, K. Bharat, F. Maghoul: *How dynamic is the Web?*, Proc. WWW9, 2000.

[5] 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/`.

[6] 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.

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

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

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

[10] J. Cho, H. Garcia-Molina: *Measuring frequency of change.* Technical Report, Dept. of Computer Science, Stanford University, 1999.

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

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

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

[14] 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.

[15] 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.

[16] 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.

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

[18] Google Search Engine, `http://www.google.com`.

[19] M. Henzinger, A. Heydon, M. Mitzenmacher, M. Najork, *On near-uniform URL sampling*, Proc. of WWW9, May 2000.

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

[21] *Internet Archive Project*, `http://www.archive.org/`.

[22] J. Kleinberg: *Authoritative sources in a hyperlinked environment*, Proc. 9th Symposium on Discrete Algorithms, 1998.

[23] M. Koster: *The Web Robots Page*, `http://info.webcrawler.com/~mak/projects-/robots/robots.html`.

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

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

[26] *NEC Research Index (citeseer)*, `http://citeseer.nj.nec.com/`.

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

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

[29] L. Page, S. Brin, R. Motwani, T. Winograd: *The Pagerank citation algorithm: bringing order to the Web.* Proc. of WWW7, 1998.

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

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

[32] 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.

[33] 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

[34] TodoCL, http://www.todocl.cl/.