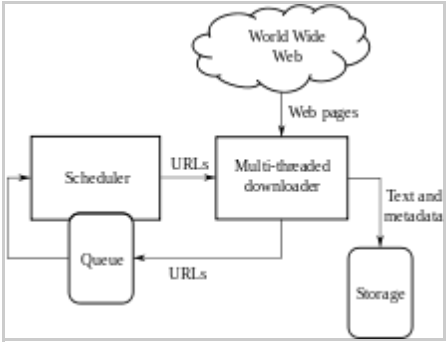
Web crawler

From Wikipedia, the free encyclopedia

A **Web crawler**, sometimes called a **spider**, is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing (*web spidering*).



Architecture of a Web crawler

Web search engines and some other sites use Web crawling or spidering software to update their web content or indices of others sites' web content. Web crawlers can copy all the pages they visit for later processing by a search engine which indexes the downloaded pages so the users can search much more efficiently.

Crawlers consume resources on the systems they visit and often visit sites without approval. Issues of schedule, load, and "politeness" come into play when large collections of pages are accessed. Mechanisms exist for public sites not wishing to be crawled to make this known to the crawling agent. For instance, including a robots.txt file can request bots to index only parts of a website, or nothing at all.

As the number of pages on the internet is extremely large, even the largest crawlers fall short of making a complete index. For that reason search engines were bad at giving relevant search results in the early years of the World Wide Web, before the year 2000. This is improved greatly by modern search engines; nowadays very good results are given instantly.

Crawlers can validate hyperlinks and HTML code. They can also be used for web scraping (see also data-driven programming).

Contents

Nomenclature

A Web crawler may also be called a *Web spider*,[1] an *ant*, an *automatic indexer*,[2] or (in the FOAF software context) a *Web scutter*.[3]

Overview

A Web crawler starts with a list of URLs to visit, called the *seeds*. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the *crawl frontier*. URLs from the frontier are recursively visited according to a set of policies. If the crawler is performing archiving of websites it copies and saves the information as it goes. The archives are usually stored in such a way they can be viewed, read and navigated as they were on the live web, but are preserved as ‘snapshots'.[4]

The archive is known as the repository and is designed to store and manage the collection of web pages. The repository only stores HTML pages and these pages are stored as distinct files. A repository is similar to any other system that stores data, like a modern day database. The only difference is that a repository does not need all the functionality offered by a database system. The repository stores the most recent version of the web page retrieved by the crawler.[5]

The large volume implies the crawler can only download a limited number of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change can imply the pages might have already been updated or even deleted.

The number of possible URLs crawled being generated by server-side software has also made it difficult for web crawlers to avoid retrieving duplicate content. Endless combinations of HTTP GET (URL-based) parameters exist, of which only a small selection will actually return unique content. For example, a simple online photo gallery may offer three options to users, as specified through HTTP GET parameters in the URL. If there exist four ways to sort images, three choices of thumbnail size, two file formats, and an option to disable user-provided content, then the same set of content can be accessed with 48

different URLs, all of which may be linked on the site. This mathematical combination creates a problem for crawlers, as they must sort through endless combinations of relatively minor scripted changes in order to retrieve unique content.

As Edwards *et al.* noted, "Given that the bandwidth for conducting crawls is neither infinite nor free, it is becoming essential to crawl the Web in not only a scalable, but efficient way, if some reasonable measure of quality or freshness is to be maintained."[6] A crawler must carefully choose at each step which pages to visit next.

Crawling policy

The behavior of a Web crawler is the outcome of a combination of policies:[7]

- a *selection policy* which states the pages to download,
- a *re-visit policy* which states when to check for changes to the pages,
- a *politeness policy* that states how to avoid overloading Web sites, and
- a *parallelization policy* that states how to coordinate distributed web crawlers.

Selection policy

Given the current size of the Web, even large search engines cover only a portion of the publicly available part. A 2009 study showed even large-scale search engines index no more than 40-70% of the indexable Web;[8] a previous study by Steve Lawrence and Lee Giles showed that no search engine indexed more than 16% of the Web in 1999.[9] As a crawler always downloads just a fraction of the Web pages, it is highly desirable for the downloaded fraction to contain the most relevant pages and not just a random sample of the Web.

This requires a metric of importance for prioritizing Web pages. The importance of a page is a function of its intrinsic quality, its popularity in terms of links or visits, and even of its URL (the latter is the case of vertical search engines restricted to a single top-level domain, or search engines restricted to a fixed Web site). Designing a good selection policy has an added difficulty: it must work with partial information, as the complete set of Web pages is not known during crawling.

Cho *et al.* made the first study on policies for crawling scheduling. Their data set was a 180,000-pages crawl from the `stanford.edu` domain, in which a crawling simulation was done with different strategies.[10] The ordering metrics tested were breadth-first, backlink count and partial Pagerank calculations. One of the conclusions was that if the crawler wants to download pages with high Pagerank early during the crawling process, then the partial Pagerank strategy is the better, followed by breadth-first and backlink-count. However, these results are for just a single domain. Cho also wrote his Ph.D. dissertation at Stanford on web crawling.[11]

Najork and Wiener performed an actual crawl on 328 million pages, using breadth-first ordering.[12] They found that a breadth-first crawl captures pages with high Pagerank early in the crawl (but they did not compare this strategy against other strategies). The explanation given by the authors for this result is that "the most important pages have many links to them from numerous hosts, and those links will be found early, regardless of on which host or page the crawl originates."

Abiteboul designed a crawling strategy based on an algorithm called OPIC (On-line Page Importance Computation).[13] In OPIC, each page is given an initial sum of "cash" that is distributed equally among the pages it points to. It is similar to a Pagerank computation, but it is faster and is only done in one step. An OPIC-driven crawler downloads first the pages in the crawling frontier with higher amounts of "cash". Experiments were carried in a 100,000-pages synthetic graph with a power-law distribution of in-links. However, there was no comparison with other strategies nor experiments in the real Web.

Boldi *et al.* used simulation on subsets of the Web of 40 million pages from the `.it` domain and 100 million pages from the WebBase crawl, testing breadth-first against depth-first, random ordering and an omniscient strategy. The comparison was based on how well PageRank computed on a partial crawl approximates the true PageRank value. Surprisingly, some visits that accumulate PageRank very quickly (most notably, breadth-first and the omniscient visit) provide very poor progressive approximations.[14][15]

Baeza-Yates *et al.* used simulation on two subsets of the Web of 3 million pages from the `.gr` and `.cl` domain, testing several crawling strategies.[16] They showed that both the OPIC strategy and a strategy that uses the length of the per-site queues are better than breadth-first crawling, and that it is also very effective to use a previous crawl, when it is available, to guide the current one.

Daneshpajouh *et al.* designed a community based algorithm for discovering good seeds.[17] Their method crawls web pages with high PageRank from different communities in less iteration in comparison with crawl starting from random seeds. One can extract good seed from a previously-crawled-Web graph using this new method. Using these seeds a new crawl can be very effective.

Restricting followed links

A crawler may only want to seek out HTML pages and avoid all other MIME types. In order to request only HTML resources, a crawler may make an HTTP HEAD request to determine a Web resource's MIME type before requesting the entire resource with a GET request. To avoid making numerous HEAD requests, a crawler may examine the URL and only request a resource if the URL ends with certain characters such as .html, .htm, .asp, .aspx, .php, .jsp, .jspx or a slash. This strategy may cause numerous HTML Web resources to be unintentionally skipped.

Some crawlers may also avoid requesting any resources that have a "?" in them

(are dynamically produced) in order to avoid spider traps that may cause the crawler to download an infinite number of URLs from a Web site. This strategy is unreliable if the site uses URL rewriting to simplify its URLs.

URL normalization

Crawlers usually perform some type of URL normalization in order to avoid crawling the same resource more than once. The term *URL normalization*, also called *URL canonicalization*, refers to the process of modifying and standardizing a URL in a consistent manner. There are several types of normalization that may be performed including conversion of URLs to lowercase, removal of "." and ".." segments, and adding trailing slashes to the non-empty path component.[18]

Path-ascending crawling

Some crawlers intend to download as many resources as possible from a particular web site. So *path-ascending crawler* was introduced that would ascend to every path in each URL that it intends to crawl.[19] For example, when given a seed URL of http://llama.org/hamster/monkey/page.html, it will attempt to crawl /hamster/monkey/, /hamster/, and /. Cothey found that a path-ascending crawler was very effective in finding isolated resources, or resources for which no inbound link would have been found in regular crawling.

Focused crawling

The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. Web crawlers that attempt to download pages that are similar to each other are called **focused crawler** or **topical crawlers**. The concepts of topical and focused crawling were first introduced by Filippo Menczer[20][21] and by Soumen Chakrabarti *et al.*[22]

The main problem in focused crawling is that in the context of a Web crawler, we would like to be able to predict the similarity of the text of a given page to the query before actually downloading the page. A possible predictor is the anchor text of links; this was the approach taken by Pinkerton[23] in the first web crawler of the early days of the Web. Diligenti *et al.*[24] propose using the complete content of the pages already visited to infer the similarity between the driving query and the pages that have not been visited yet. The performance of a focused crawling depends mostly on the richness of links in the specific topic being searched, and a focused crawling usually relies on a general Web search engine for providing starting points.

Academic-focused crawler

An example of the focused crawlers are academic crawlers, which crawls free-access academic related documents, such as the *citeseerxbot*, which is the

crawler of CiteSeer^X search engine. Other academic search engines are Google Scholar and Microsoft Academic Search etc. Because most academic papers are published in PDF formats, such kind of crawler is particularly interested in crawling PDF, PostScript files, Microsoft Word including their zipped formats. Because of this, general open source crawlers, such as Heritrix, must be customized to filter out other MIME types, or a middleware is used to extract these documents out and import them to the focused crawl database and repository.[25] Identifying whether these documents are academic or not is challenging and can add a significant overhead to the crawling process, so this is performed as a post crawling process using machine learning or regular expression algorithms. These academic documents are usually obtained from home pages of faculties and students or from publication page of research institutes. Because academic documents takes only a small fraction in the entire web pages, a good seed selection are important in boosting the efficiencies of these web crawlers.[26] Other academic crawlers may download plain text and HTML files, that contains metadata of academic papers, such as titles, papers, and abstracts. This increases the overall number of papers, but a significant fraction may not provide free PDF downloads.

Re-visit policy

The Web has a very dynamic nature, and crawling a fraction of the Web can take weeks or months. By the time a Web crawler has finished its crawl, many events could have happened, including creations, updates, and deletions.

From the search engine's point of view, there is a cost associated with not detecting an event, and thus having an outdated copy of a resource. The most-used cost functions are freshness and age.[27]

Freshness: This is a binary measure that indicates whether the local copy is accurate or not. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

Age: This is a measure that indicates how outdated the local copy is. The age of a page p in the repository, at time t is defined as:

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases}$$

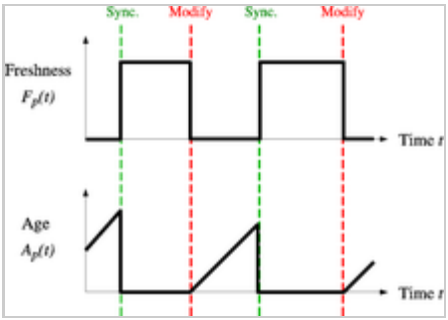
Coffman *et al.* worked with a definition of the objective of a Web crawler that is equivalent to freshness, but use a different wording: they propose that a crawler must minimize the fraction of time pages remain outdated. They also noted that the problem of Web crawling can be modeled as a multiple-queue, single-server polling system, on which the Web crawler is the server and the Web sites are the queues. Page modifications are the arrival of the customers,

and switch-over times are the interval between page accesses to a single Web site. Under this model, mean waiting time for a customer in the polling system is equivalent to the average age for the Web crawler.[28]

The objective of the crawler is to keep the average freshness of pages in its collection as high as possible, or to keep the average age of pages as low as possible. These objectives are not equivalent: in the first case, the crawler is just concerned with how many pages are out-dated, while in the second case, the crawler is concerned with how old the local copies of pages are.

Two simple re-visiting policies were studied by Cho and Garcia-Molina:[29]

- Uniform policy: This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.
- Proportional policy: This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.



Evolution of Freshness and Age in a web crawler

In both cases, the repeated crawling order of pages can be done either in a random or a fixed order.

Cho and Garcia-Molina proved the surprising result that, in terms of average freshness, the uniform policy outperforms the proportional policy in both a simulated Web and a real Web crawl. Intuitively, the reasoning is that, as web crawlers have a limit to how many pages they can crawl in a given time frame, (1) they will allocate too many new crawls to rapidly changing pages at the expense of less frequently updating pages, and (2) the freshness of rapidly changing pages lasts for shorter period than that of less frequently changing pages. In other words, a proportional policy allocates more resources to crawling frequently updating pages, but experiences less overall freshness time from them.

To improve freshness, the crawler should penalize the elements that change too often.[30] The optimal re-visiting policy is neither the uniform policy nor the proportional policy. The optimal method for keeping average freshness high includes ignoring the pages that change too often, and the optimal for keeping average age low is to use access frequencies that monotonically (and sub-linearly) increase with the rate of change of each page. In both cases, the optimal is closer to the uniform policy than to the proportional policy: as Coffman *et al.* note, "in order to minimize the expected obsolescence time, the accesses to any particular page should be kept as evenly spaced as possible".[28] Explicit formulas for the re-visit policy are not attainable in general, but they are obtained numerically, as they depend on the distribution of page changes. Cho and Garcia-Molina show that the exponential distribution is a good fit for describing page changes,[30] while Ipeirotis *et al.* show how to

use statistical tools to discover parameters that affect this distribution.[31] Note that the re-visiting policies considered here regard all pages as homogeneous in terms of quality ("all pages on the Web are worth the same"), something that is not a realistic scenario, so further information about the Web page quality should be included to achieve a better crawling policy.

Politeness policy

Crawlers can retrieve data much quicker and in greater depth than human searchers, so they can have a crippling impact on the performance of a site. Needless to say, if a single crawler is performing multiple requests per second and/or downloading large files, a server would have a hard time keeping up with requests from multiple crawlers.

As noted by Koster, the use of Web crawlers is useful for a number of tasks, but comes with a price for the general community.[32] The costs of using Web crawlers include:

- network resources, as crawlers require considerable bandwidth and operate with a high degree of parallelism during a long period of time;
- server overload, especially if the frequency of accesses to a given server is too high;
- poorly written crawlers, which can crash servers or routers, or which download pages they cannot handle; and
- personal crawlers that, if deployed by too many users, can disrupt networks and Web servers.

A partial solution to these problems is the robots exclusion protocol, also known as the robots.txt protocol that is a standard for administrators to indicate which parts of their Web servers should not be accessed by crawlers.[33] This standard does not include a suggestion for the interval of visits to the same server, even though this interval is the most effective way of avoiding server overload. Recently commercial search engines like Google, Ask Jeeves, MSN and Yahoo! Search are able to use an extra "Crawl-delay:" parameter in the robots.txt file to indicate the number of seconds to delay between requests.

The first proposed interval between successive pageloads was 60 seconds.[34] However, if pages were downloaded at this rate from a website with more than 100,000 pages over a perfect connection with zero latency and infinite bandwidth, it would take more than 2 months to download only that entire Web site; also, only a fraction of the resources from that Web server would be used. This does not seem acceptable.

Cho uses 10 seconds as an interval for accesses,[29] and the WIRE crawler uses 15 seconds as the default.[35] The MercatorWeb crawler follows an adaptive politeness policy: if it took t seconds to download a document from a given server, the crawler waits for $10t$ seconds before downloading the next page.[36] Dill *et al.* use 1 second.[37]

For those using Web crawlers for research purposes, a more detailed cost-benefit analysis is needed and ethical considerations should be taken into account when deciding where to crawl and how fast to crawl.[38]

Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. It is worth noticing that even when being very polite, and taking all the safeguards to avoid overloading Web servers, some complaints from Web server administrators are received. Brin and Page note that: "... running a crawler which connects to more than half a million servers (...) generates a fair amount of e-mail and phone calls. Because of the vast number of people coming on line, there are always those who do not know what a crawler is, because this is the first one they have seen."[39]

Parallelization policy

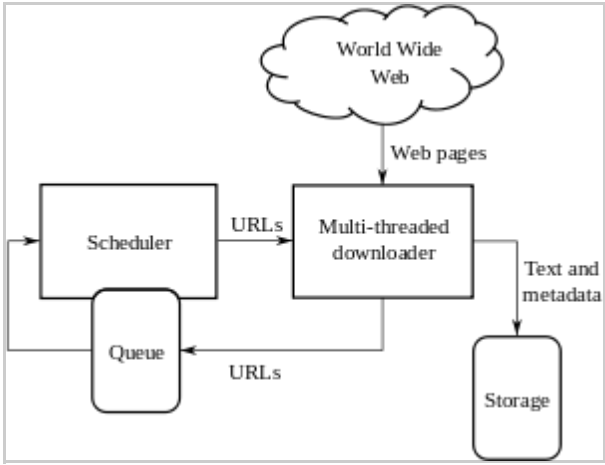
A parallel crawler is a crawler that runs multiple processes in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page. To avoid downloading the same page more than once, the crawling system requires a policy for assigning the new URLs discovered during the crawling process, as the same URL can be found by two different crawling processes.

Architectures

A crawler must not only have a good crawling strategy, as noted in the previous sections, but it should also have a highly optimized architecture.

Shkapenyuk and Suel noted that:[40]

> While it is fairly easy to build a slow crawler that downloads a few pages per second for a short period of time, building a high-performance system that can download hundreds of millions of pages over several weeks presents a number of challenges in system design, I/O and network efficiency, and robustness and manageability.



High-level architecture of a standard Web crawler

Web crawlers are a central part of search engines, and details on their algorithms and architecture are kept as business secrets. When crawler designs are published, there is often an important lack of detail that prevents others from reproducing the work. There are also emerging concerns about "search engine spamming", which prevent major search engines from publishing their ranking algorithms.

Security

While most of the website owners are keen to have their pages indexed as broadly as possible to have strong presence in search engines, web crawling can also have unintended consequences and lead to a compromise or data breach if search engine indexes resources that shouldn't be publicly available or pages revealing potentially vulnerable versions of software.

Apart from standard web application security recommendations website owners can reduce their exposure to opportunistic hacking by only allowing search engines to index the public parts of their websites (with robots.txt) and explicitly blocking them from indexing transactional parts (login pages, private pages, etc.).

Crawler identification

Web crawlers typically identify themselves to a Web server by using the User-agent field of an HTTP request. Web site administrators typically examine their Web servers' log and use the user agent field to determine which crawlers have visited the web server and how often. The user agent field may include a URL where the Web site administrator may find out more information about the crawler. Examining Web server log is tedious task, and therefore some administrators use tools to identify, track and verify Web crawlers. Spambots and other malicious Web crawlers are unlikely to place identifying information in the user agent field, or they may mask their identity as a browser or other well-known crawler.

It is important for Web crawlers to identify themselves so that Web site administrators can contact the owner if needed. In some cases, crawlers may be accidentally trapped in a crawler trap or they may be overloading a Web server with requests, and the owner needs to stop the crawler. Identification is also useful for administrators that are interested in knowing when they may expect their Web pages to be indexed by a particular search engine.

Crawling the deep web

A vast amount of web pages lie in the deep or invisible web.[41] These pages are typically only accessible by submitting queries to a database, and regular crawlers are unable to find these pages if there are no links that point to them. Google's Sitemaps protocol and mod oai[42] are intended to allow discovery of

these deep-Web resources.

Deep web crawling also multiplies the number of web links to be crawled. Some crawlers only take some of the URLs in `<a href="URL">` form. In some cases, such as the Googlebot, Web crawling is done on all text contained inside the hypertext content, tags, or text.

Strategic approaches may be taken to target deep Web content. With a technique called screen scraping, specialized software may be customized to automatically and repeatedly query a given Web form with the intention of aggregating the resulting data. Such software can be used to span multiple Web forms across multiple Websites. Data extracted from the results of one Web form submission can be taken and applied as input to another Web form thus establishing continuity across the Deep Web in a way not possible with traditional web crawlers.[43]

Pages built on AJAX are among those causing problems to web crawlers. Google has proposed a format of AJAX calls that their bot can recognize and index.[44]

Web crawler bias

A recent study based on a large scale analysis of robots.txt files showed that certain web crawlers were preferred over others, with Googlebot being the most preferred web crawler.[45]

Visual vs programmatic crawlers

There are a number of "visual web scraper/crawler" products available on the web which will crawl pages and structure data into columns and rows based on the users requirements. One of the main difference between a classic and a visual crawler is the level of programming ability required to set up a crawler. The latest generation of "visual scrapers" like Diffbot (https://www.diffbot.com),[46] outwithub,[47] and import.io[48] remove the majority of the programming skill needed to be able to program and start a crawl to scrape web data.

The visual scraping/crawling methodology relies on the user "teaching" a piece of crawler technology, which then follows patterns in semi-structured data sources. The dominant method for teaching a visual crawler is by highlighting data in a browser and training columns and rows. While the technology is not new, for example it was the basis of Needlebase which has been bought by Google (as part of a larger acquisition of ITA Labs[49]), there is continued growth and investment in this area by investors and end-users.[50]

Examples

The following is a list of published crawler architectures for general-purpose

crawlers (excluding focused web crawlers), with a brief description that includes the names given to the different components and outstanding features:

- *Bingbot* is the name of Microsoft's Bing webcrawler. It replaced *Msnbot*.
- *FAST Crawler*[51] is a distributed crawler.
- *Googlebot*[39] is described in some detail, but the reference is only about an early version of its architecture, which was based in C++ and Python. The crawler was integrated with the indexing process, because text parsing was done for full-text indexing and also for URL extraction. There is a URL server that sends lists of URLs to be fetched by several crawling processes. During parsing, the URLs found were passed to a URL server that checked if the URL have been previously seen. If not, the URL was added to the queue of the URL server.
- *GM Crawl* is a crawler highly scalable usable in SaaS mode[52]
- *PolyBot*[40] is a distributed crawler written in C++ and Python, which is composed of a "crawl manager", one or more "downloaders" and one or more "DNS resolvers". Collected URLs are added to a queue on disk, and processed later to search for seen URLs in batch mode. The politeness policy considers both third and second level domains (e.g.: www.example.com and www2.example.com are third level domains) because third level domains are usually hosted by the same Web server.
- *RBSE*[53] was the first published web crawler. It was based on two programs: the first program, "`spider`" maintains a queue in a relational database, and the second program "`mite`", is a modified `www` ASCII browser that downloads the pages from the Web.
- Swiftbot is Swiftype's web crawler, designed specifically for indexing a single or small, defined group of web sites to create a highly customized search engine. It enables unique features such as real-time indexing that are unavailable to other enterprise search providers.[54]
- *WebCrawler*[23] was used to build the first publicly available full-text index of a subset of the Web. It was based on lib-WWW to download pages, and another program to parse and order URLs for breadth-first exploration of the Web graph. It also included a real-time crawler that followed links based on the similarity of the anchor text with the provided query.
- *WebFountain*[6] is a distributed, modular crawler similar to Mercator but written in C++. It features a "controller" machine that coordinates a series of "ant" machines. After repeatedly downloading pages, a change rate is inferred for each page and a non-linear programming method must be used to solve the equation system for maximizing freshness. The authors recommend to use this crawling order in the early stages of the crawl, and then switch to a uniform crawling order, in which all pages are being visited with the same frequency.
- *WebRACE*[55] is a crawling and caching module implemented in Java, and used as a part of a more generic system called eRACE. The system receives requests from users for downloading web pages, so the crawler acts in part as a smart proxy server. The system also handles requests for "subscriptions" to Web pages that must be monitored: when the pages change, they must be downloaded by the crawler and the subscriber must be notified. The most outstanding feature of WebRACE is that, while most

crawlers start with a set of "seed" URLs, WebRACE is continuously receiving new starting URLs to crawl from.
- *World Wide Web Worm*[56] was a crawler used to build a simple index of document titles and URLs. The index could be searched by using the `grep` Unix command.
- *Yahoo! Slurp* was the name of the Yahoo! Search crawler until Yahoo! contracted with Microsoft to use Bingbot instead.

In addition to the specific crawler architectures listed above, there are general crawler architectures published by Cho[57] and Chakrabarti.[58]

Open-source crawlers

- Frontera is web crawling framework implementing crawl frontier component and providing scalability primitives for web crawler applications.
- *GNU Wget* is a command-line-operated crawler written in C and released under the GPL. It is typically used to mirror Web and FTP sites.
- *GRUB* is an open source distributed search crawler that Wikia Search used to crawl the web.
- *Heritrix* is the Internet Archive's archival-quality crawler, designed for archiving periodic snapshots of a large portion of the Web. It was written in Java.
- *ht://Dig* includes a Web crawler in its indexing engine.
- *HTTrack* uses a Web crawler to create a mirror of a web site for off-line viewing. It is written in C and released under the GPL.
- *mnoGoSearch* is a crawler, indexer and a search engine written in C and licensed under the GPL (*NIX machines only)
- *Apache Nutch* is a highly extensible and scalable web crawler written in Java and released under an Apache License. It is based on Apache Hadoop and can be used with Apache Solr or Elasticsearch.
- *Open Search Server* is a search engine and web crawler software release under the GPL.
- *PHP-Crawler* is a simple PHP and MySQL based crawler released under the BSD License.
- *Scrapy*, an open source webcrawler framework, written in python (licensed under BSD).
- *Seeks*, a free distributed search engine (licensed under AGPL).
- *Sphinx (search engine)*, a free search crawler, written in c++.
- StormCrawler, a collection of resources for building low-latency, scalable web crawlers on Apache Storm (Apache License).
- *tkWWW Robot*, a crawler based on the tkWWW web browser (licensed under GPL).
- *Xapian*, a search crawler engine, written in c++.
- *YaCy*, a free distributed search engine, built on principles of peer-to-peer networks (licensed under GPL).
- *Octoparse*, a free client-side Windows web crawler written in .NET.

See also

- Automatic indexing
- Gnutella crawler
- Web archiving
- Webgraph
- Website mirroring software
- Search Engine Scraping

References

1. Spetka, Scott. "The TkWWW Robot: Beyond Browsing" (https://web.archive.org/web/20040903174942/archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/spetka/spetka.html). NCSA. Archived from the original (http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/spetka/spetka.html) on 3 September 2004. Retrieved 21 November 2010.
2. Kobayashi, M. & Takeda, K. (2000). "Information retrieval on the web" (http://doi.acm.org/10.1145/358923.358934). *ACM Computing Surveys*. ACM Press. **32** (2): 144–173. doi:10.1145/358923.358934 (https://doi.org/10.1145%2F358923.358934).
3. See definition of scutter on FOAF Project's wiki (http://wiki.foaf-project.org/w/Scutter)
4. Masanès, Julien (February 15, 2007). *Web Archiving* (https://nocrus.co.uk/shared-hosting/). Springer. p. 1. ISBN 978-3-54046332-0. Retrieved April 24, 2014.
5. Patil, Yugandhara; Patil, Sonal (2016). "Review of Web Crawlers with Specification and Working" (http://www.ijarcce.com/upload/2016/january-16/IJARCCE%2052.pdf) (PDF). *International Journal of Advanced Research Computer and Communication Engineering*. **5** (1): 4.
6. Edwards, J., McCurley, K. S., and Tomlin, J. A. (2001). "An adaptive model for optimizing performance of an incremental web crawler" (http://www10.org/cdrom/papers/210/index.html). *In Proceedings of the Tenth Conference on World Wide Web*. Hong Kong: Elsevier Science: 106–113. doi:10.1145/371920.371960 (https://doi.org/10.1145%2F371920.371960). ISBN 1581133480.
7. Castillo, Carlos (2004). *Effective Web Crawling* (http://chato.cl/research/crawling_thesis) (Ph.D. thesis). University of Chile. Retrieved 2010-08-03.
8. A. Gulli; A. Signorini (2005). "The indexable web is more than 11.5 billion pages" (http://doi.acm.org/10.1145/1062745.1062789). *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM Press. pp. 902–903. doi:10.1145/1062745.1062789 (https://doi.org/10.1145%2F1062745.1062789).
9. Steve Lawrence; C. Lee Giles (1999-07-08). "Accessibility of information on the web". *Nature*. **400** (6740): 107–9. Bibcode:1999Natur.400..107L (http://adsabs.harvard.edu/abs/1999Natur.400..107L). doi:10.1038/21987 (https://doi.org/10.1038%2F21987). PMID 10428673 (https://www.ncbi.nlm.nih.gov/pubmed/10428673).
10. Cho, J.; Garcia-Molina, H.; Page, L. (April 1998). "Efficient Crawling Through URL Ordering" (http://ilpubs.stanford.edu:8090/347/). *Seventh International World-Wide Web Conference*. Brisbane, Australia. Retrieved 2009-03-23.
11. Cho, Junghoo, "Crawling the Web: Discovery and Maintenance of a Large-Scale Web Data" (http://oak.cs.ucla.edu/~cho/papers/cho-thesis.pdf), Ph.D. dissertation, Department of Computer Science, Stanford University, November 2001
12. Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages (http://www10.org/cdrom/papers/pdf/p208.pdf). In Proceedings of the Tenth Conference on World Wide Web, pages 114–118, Hong Kong, May 2001. Elsevier Science.

13. Serge Abiteboul; Mihai Preda; Gregory Cobena (2003). "Adaptive on-line page importance computation" (http://www2003.org/cdrom/papers/refereed/p007/p7-abiteboul.html). *Proceedings of the 12th international conference on World Wide Web*. Budapest, Hungary: ACM. pp. 280–290. doi:10.1145/775152.775192 (https://doi.org/10.1145%2F775152.775192). ISBN 1-58113-680-3. Retrieved 2009-03-22.
14. Paolo Boldi; Bruno Codenotti; Massimo Santini; Sebastiano Vigna (2004). "UbiCrawler: a scalable fully distributed Web crawler" (http://vigna.dsi.unimi.it/ftp/papers/UbiCrawler.pdf) (PDF). *Software: Practice and Experience*. **34** (8): 711–726. doi:10.1002/spe.587 (https://doi.org/10.1002%2Fspe.587). Retrieved 2009-03-23.
15. Paolo Boldi; Massimo Santini; Sebastiano Vigna (2004). "Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations" (http://vigna.dsi.unimi.it/ftp/papers/ParadoxicalPageRank.pdf) (PDF). *Algorithms and Models for the Web-Graph* (http://springerlink.com/content/g10m122f9hb6). pp. 168–180. Retrieved 2009-03-23.
16. Baeza-Yates, R., Castillo, C., Marin, M. and Rodriguez, A. (2005). Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering (http://www.dcc.uchile.cl/%7Eccastill/papers/baeza05_crawling_country_better_breadth_first_web_page_ordering.pdf). In Proceedings of the Industrial and Practical Experience track of the 14th conference on World Wide Web, pages 864–872, Chiba, Japan. ACM Press.
17. Shervin Daneshpajouh, Mojtaba Mohammadi Nasiri, Mohammad Ghodsi, A Fast Community Based Algorithm for Generating Crawler Seeds Set (http://ce.sharif.edu/~daneshpajouh/publications/A%20Fast%20Community%20Based%20Algorithm%20for%20Generating%20Crawler%20Seeds%20Set.pdf), In proceeding of 4th International Conference on Web Information Systems and Technologies ((http://www.webist.org/)Webist-2008), Funchal, Portugal, May 2008.
18. Pant, Gautam; Srinivasan, Padmini; Menczer, Filippo (2004). "Crawling the Web" (http://dollar.biz.uiowa.edu/~pant/Papers/crawling.pdf) (PDF). In Levene, Mark; Poulovassilis, Alexandra. *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*. Springer. pp. 153–178. ISBN 978-3-540-40676-1.
19. Cothey, Viv (2004). "Web-crawling reliability". *Journal of the American Society for Information Science and Technology*. **55** (14): 1228–1238. doi:10.1002/asi.20078 (https://doi.org/10.1002%2Fasi.20078).
20. Menczer, F. (1997). ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery (http://informatics.indiana.edu/fil/Papers/ICML.ps). In D. Fisher, ed., Machine Learning: Proceedings of the 14th International Conference (ICML97). Morgan Kaufmann
21. Menczer, F. and Belew, R.K. (1998). Adaptive Information Agents in Distributed Textual Environments (http://informatics.indiana.edu/fil/Papers/AA98.ps). In K. Sycara and M. Wooldridge (eds.) Proc. 2nd Intl. Conf. on Autonomous Agents (Agents '98). ACM Press
22. Chakrabarti, S., van den Berg, M., and Dom, B. (1999). Focused crawling: a new approach to topic-specific web resource discovery (https://web.archive.org/web/20040317210216/http://www.fxpal.com/people/vdberg/pubs/www8/www1999f.pdf). Computer Networks, 31(11–16):1623–1640.
23. Pinkerton, B. (1994). Finding what people want: Experiences with the WebCrawler (https://web.archive.org/web/20010904075500/http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/pinkerton/WebCrawler.html). In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

24. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. (2000). Focused crawling using context graphs (http://nautilus.dii.unisi.it/pubblicazioni/files/conference/2000-Diligenti-VLDB.pdf). In Proceedings of 26th International Conference on Very Large Databases (VLDB), pages 527-534, Cairo, Egypt.
25. Jian Wu, Pradeep Teregowda, Madian Khabsa, Stephen Carman, Douglas Jordan, Jose San Pedro Wandelmer, Xin Lu, Prasenjit Mitra, C. Lee Giles, Web crawler middleware for search engine digital libraries: a case study for citeseerX (http://dl.acm.org/citation.cfm?id=2389936.2389949&coll=DL&dl=ACM&CFID=176007707&CFTOKEN=93317127), In proceedings of the twelfth international workshop on Web information and data management Pages 57-64, Maui Hawaii, USA, November 2012.
26. Jian Wu, Pradeep Teregowda, Juan Pablo Fernández Ramírez, Prasenjit Mitra, Shuyi Zheng, C. Lee Giles , The evolution of a crawling strategy for an academic document search engine: whitelists and blacklists (http://dl.acm.org/citation.cfm?id=2380718.2380762), In proceedings of the 3rd Annual ACM Web Science Conference Pages 340-343, Evanston, IL, USA, June 2012.
27. Junghoo Cho; Hector Garcia-Molina (2000). "Synchronizing a database to improve freshness" (http://www.cs.brown.edu/courses/cs227/2002/cache/Cho.pdf) (PDF). *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. Dallas, Texas, United States: ACM. pp. 117–128. doi:10.1145/342009.335391 (https://doi.org/10.1145%2F342009.335391). ISBN 1-58113-217-4. Retrieved 2009-03-23.
28. E. G. Coffman Jr; Zhen Liu; Richard R. Weber (1998). "Optimal robot scheduling for Web search engines". *Journal of Scheduling*. **1** (1): 15–29. doi:10.1002/(SICI)1099-1425(199806)1:1<15::AID-JOS3>3.0.CO;2-K (https://doi.org/10.1002%2F%28SICI%291099-1425%28199806%291%3A1%3C15%3A%3AAID-JOS3%3E3.0.CO%3B2-K).
29. Cho, J. and Garcia-Molina, H. (2003). Effective page refresh policies for web crawlers (http://portal.acm.org/citation.cfm?doid=958942.958945). ACM Transactions on Database Systems, 28(4).
30. Junghoo Cho; Hector Garcia-Molina (2003). "Estimating frequency of change" (http://portal.acm.org/citation.cfm?doid=857166.857170). *ACM Trans. Internet Technol.* **3** (3): 256–290. doi:10.1145/857166.857170 (https://doi.org/10.1145%2F857166.857170). Retrieved 2009-03-22.
31. Ipeirotis, P., Ntoulas, A., Cho, J., Gravano, L. (2005) Modeling and managing content changes in text databases (http://pages.stern.nyu.edu/~panos/publications/icde2005.pdf). In Proceedings of the 21st IEEE International Conference on Data Engineering, pages 606-617, April 2005, Tokyo.
32. Koster, M. (1995). Robots in the web: threat or treat? ConneXions, 9(4).
33. Koster, M. (1996). A standard for robot exclusion (http://www.robotstxt.org/wc/exclusion.html).
34. Koster, M. (1993). Guidelines for robots writers (http://www.robotstxt.org/wc/guidelines.html).
35. Baeza-Yates, R. and Castillo, C. (2002). Balancing volume, quality and freshness in Web crawling (http://www.chato.cl/papers/baeza02balancing.pdf). In Soft Computing Systems – Design, Management and Applications, pages 565–572, Santiago, Chile. IOS Press Amsterdam.
36. Heydon, Allan; Najork, Marc (1999-06-26). "Mercator: A Scalable, Extensible Web Crawler" (https://web.archive.org/web/20060219085958/http://www.cindoc.csic.es/cybermetrics/pdf/68.pdf) (PDF). Archived from the original (http://www.cindoc.csic.es/cybermetrics/pdf/68.pdf) (PDF) on 19 February 2006. Retrieved 2009-03-22.

37. Dill, S., Kumar, R., Mccurley, K. S., Rajagopalan, S., Sivakumar, D., and Tomkins, A. (2002) Self-similarity in the web (http://www.mccurley.org/papers/fractal.pdf). *ACM Trans. Inter. Tech.* 2(3):205–223.
38. M. Thelwall; D. Stuart (2006). "Web crawling ethics revisited: Cost, privacy and denial of service" (http://www.scit.wlv.ac.uk/%7Ecm1993/papers/Web_Crawling_Ethics_preprint.doc). *Journal of the American Society for Information Science and Technology*. **57** (13): 1771. doi:10.1002/asi.20388 (https://doi.org/10.1002%2Fasi.20388).
39. Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine (http://infolab.stanford.edu/~backrub/google.html). Computer Networks and ISDN Systems, 30(1-7):107–117.
40. Shkapenyuk, V. and Suel, T. (2002). Design and implementation of a high performance distributed web crawler (http://cis.poly.edu/tr/tr-cis-2001-03.pdf). In Proceedings of the 18th International Conference on Data Engineering (ICDE), pages 357-368, San Jose, California. IEEE CS Press.
41. Shestakov, Denis (2008). Search Interfaces on the Web: Querying and Characterizing (https://oa.doria.fi/handle/10024/38506). *TUCS Doctoral Dissertations 104, University of Turku*
42. Michael L Nelson; Herbert Van de Sompel; Xiaoming Liu; Terry L Harrison; Nathan McFarland (2005-03-24). "mod_oai: An Apache Module for Metadata Harvesting". arXiv:cs/0503069 (https://arxiv.org/abs/cs/0503069).
43. Shestakov, Denis; Bhowmick, Sourav S.; Lim, Ee-Peng (2005). "DEQUE: Querying the Deep Web" (http://www.mendeley.com/download/public/1423991/3893295922/dc0f7d824fd2a8fbbc84f6fdf9e4f337d343987d/dl.pdf) (PDF). *Data & Knowledge Engineering*. **52** (3): 273–311.
44. "AJAX crawling: Guide for webmasters and developers" (http://support.google.com/webmasters/bin/answer.py?hl=en&answer=174992). Google. Retrieved March 17, 2013.
45. Sun, Yang. "A COMPREHENSIVE STUDY OF THE REGULATION AND BEHAVIOR OF WEB CRAWLERS" (https://etda.libraries.psu.edu/paper/9230/). Retrieved 11 August 2014.
46. "Web Crawler" (http://www.diffbot.com/products/crawlbot/). *Crawlbot*. Retrieved 2016-02-10.
47. "OutWit Hub - Find, grab and organize all kinds of data and media from online sources" (https://www.outwit.com/products/hub/). Outwit.com. 2014-01-31. Retrieved 2014-03-20.
48. "Create a Crawler – import.io Help Center" (http://support.import.io/knowledgebase/articles/247570-create-a-crawler). Support.import.io. Retrieved 2014-03-20.
49. ITA Labs "ITA Labs Acquisition" (http://semanticweb.com/more-semantics-for-google-ita-software-acquisition-comes-with-needlebase-too_b19329) April 20, 2011 1:28 AM
50. Crunchbase.com March 2014 "Crunch Base profile for import.io" (http://www.crunchbase.com/company/importio)
51. Risvik, K. M. and Michelsen, R. (2002). Search Engines and Web Dynamics (http://citeseer.ist.psu.edu/rd/1549722%2C509701%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/26004/http:zSzzSzwww.idi.ntnu.nozSz%7EalgkonzSzgenereltzSzse-dynamicweb1.pdf/risvik02search.pdf). Computer Networks, vol. 39, pp. 289–302, June 2002.
52. GM Crawl : Identifies and collects data from the internet (http://www.aleph-networks.com/en/bigdata.php?solutionstabs=0) 2014
53. Eichmann, D. (1994). The RBSE spider: balancing effective search against Web load (http://mingo.info-science.uiowa.edu/eichmann/www94/Spider.ps). In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

54. "About Swiftbot - Swiftype" (https://swiftype.com/swiftbot). *Swiftype*.
55. Zeinalipour-Yazti, D. and Dikaiakos, M. D. (2002). Design and implementation of a distributed crawler and filtering processor (http://www.cs.ucr.edu/~csyiazti/downloads/papers/ngits02/ngits02.pdf). In Proceedings of the Fifth Next Generation Information Technologies and Systems (NGITS), volume 2382 of Lecture Notes in Computer Science, pages 58–74, Caesarea, Israel. Springer.
56. McBryan, O. A. (1994). GENVL and WWWW: Tools for taming the web. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.
57. Junghoo Cho; Hector Garcia-Molina (2002). "Parallel crawlers" (http://portal.acm.org/citation.cfm?id=511464). *Proceedings of the 11th international conference on World Wide Web*. Honolulu, Hawaii, USA: ACM. pp. 124–135. doi:10.1145/511446.511464 (https://doi.org/10.1145%2F511446.511464). ISBN 1-58113-449-5. Retrieved 2009-03-23.
58. Chakrabarti, S. (2003). Mining the Web (http://www.cs.berkeley.edu/~soumen/mining-the-web/). Morgan Kaufmann Publishers. ISBN 1-55860-754-4

Further reading

- Cho, Junghoo, "Web Crawling Project" (http://oak.cs.ucla.edu/~cho/research/crawl.html), UCLA Computer Science Department.
- A History of Search Engines (http://www.wiley.com/legacy/compbooks/sonnenreich/history.html), from Wiley
- A tutorial (http://emanueleminotto.github.io/blog/how-to-write-a-crawler) for creating basic crawlers.
- WIVET (http://code.google.com/p/wivet/) is a benchmarking project by OWASP, which aims to measure if a web crawler can identify all the hyperlinks in a target website.
- Shestakov, Denis, "Current Challenges in Web Crawling" (http://www.slideshare.net/denshe/icwe13-tutorial-webcrawling) and "Intelligent Web Crawling" (http://www.slideshare.net/denshe/intelligent-crawling-shestakovwiiat13), slides for tutorials given at ICWE'13 and WI-IAT'13.
- Guide: How to Make a Web Crawler in Under 50 Lines of Code (Python) - Saint (http://saintlad.com/make-a-web-crawler/)

Categories: Search engine software | Web crawlers | Internet search algorithms

---