Web crawler

A **Web crawler** is a computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion.

Other terms for Web crawlers are *ants*, *automatic indexers*, *bots*,[1] *Web spiders*,[2] *Web robots*,[2] or—especially in the FOAF community—*Web scutters*.[3]

This process is called *Web crawling* or *spidering*. Many sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating HTML code. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for sending spam).

A Web crawler is one type of bot, or software agent. In general, it 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.

The large volume implies that 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 implies that the pages might have already been updated or even deleted.

The number of possible crawlable URLs 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."[4] A crawler must carefully choose at each step which pages to visit next.

The behavior of a Web crawler is the outcome of a combination of policies:[5]

- a *selection policy* that states which pages to download,
- a *re-visit policy* that 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 2005 study showed that large-scale search engines index no more than 40%-70% of the indexable Web;[6] a previous study by Dr. Steve Lawrence and Lee Giles showed that no search engine indexed more than 16% of the Web in 1999.[7] As a crawler always downloads just a fraction of the Web pages, it is highly desirable that the downloaded fraction contains 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.[8] 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.[9]

Najork and Wiener performed an actual crawl on 328 million pages, using breadth-first ordering.[10] 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).[11] 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 omniscent visit) provide very poor progressive approximations.[12][13]

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.[14] 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.[15] 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.

Focused crawling

Main article: Focused crawler

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 Menczer[16][17] and by Chakrabarti *et al.*[18]

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[19] in the first web crawler of the early days of the Web. Diligenti *et al.* [20] 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.

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 a rewrite engine to simplify its URLs.

URL normalization

Main article: 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.[21]

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.[22] 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.

Many path-ascending crawlers are also known as Web harvesting software, because they're used to "harvest" or collect all the content — perhaps the collection of photos in a gallery — from a specific page or host.

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.[23]

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 lo-

cal 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.[24]

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:[25]

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.

(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.[26] 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".[24] 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,[26] while Ipeirotis *et al.* show how to use statistical tools to discover parameters that affect this distribution.[27] 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.[28] 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.[29] 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 Ask Jeeves, MSN and Yahoo 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 connections was 60 seconds.[30] 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,[25] and the WIRE crawler uses 15 seconds as the default.[31] 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 10*t* seconds before downloading the next page.[32] Dill *et al.* use 1 second.[33]

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.[34]

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."[35]

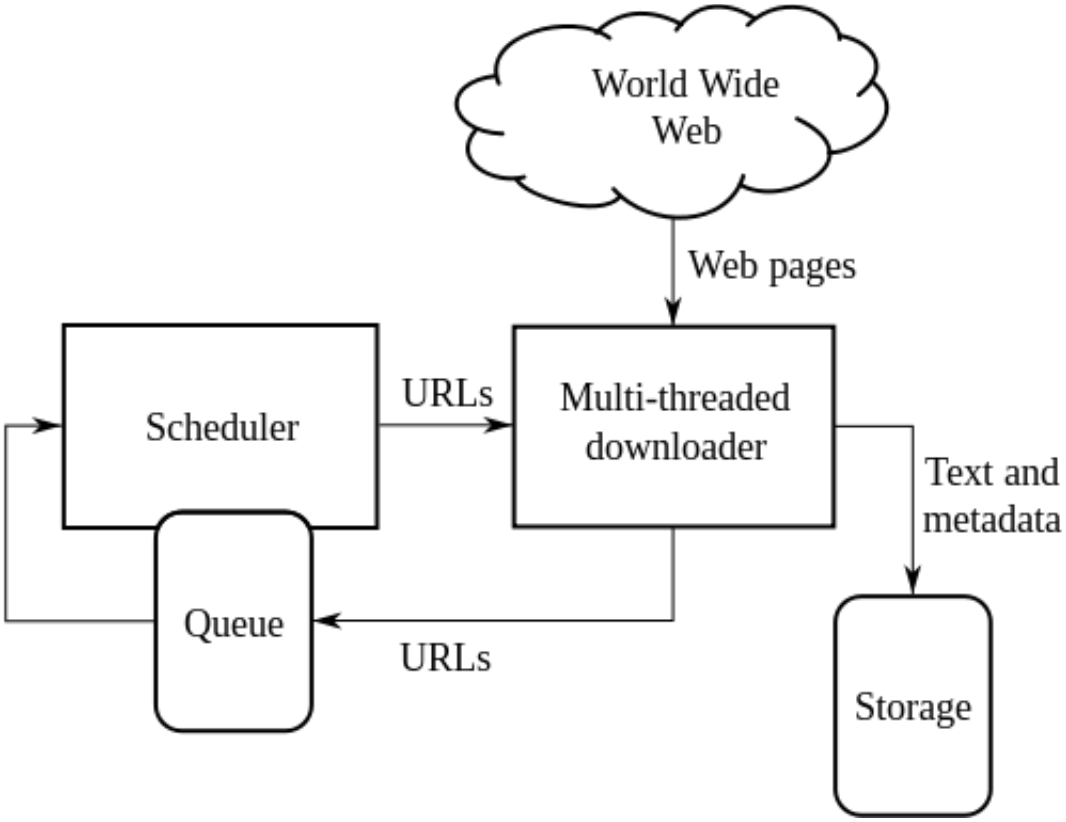
Parallelisation policy

Main article: Distributed web crawling

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:[36]

> 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.

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.

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 therefore some administrators use tools such as CrawlTrack[37] or SEO Crawlytics[38] 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.

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:

- **Yahoo! Slurp** is the name of the Yahoo Search crawler.
- **Bingbot** is the name of Microsoft's Bing webcrawler. It replaced **Msnbot**.
- **FAST Crawler**[39] is a distributed crawler, used by Fast Search & Transfer, and a general description of its architecture is available.[*citation needed*]
- **Googlebot**[35] 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.

- **PolyBot**[36] 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**[40] 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.
- **WebCrawler**[19] 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.
- **World Wide Web Worm**[41] 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.
- **WebFountain**[4] 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**[42] 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 "subscrip-

tions" 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.

In addition to the specific crawler architectures listed above, there are general crawler architectures published by Cho[43] and Chakrabarti.[44]

Open-source crawlers

- **Abot** is a C# web crawler built for speed and flexibility, Apache License 2.0 License (free for commercial and personal use)
- **Aspseek** is a crawler, indexer and a search engine written in C++ and licensed under the GPL
- **DataparkSearch** is a crawler and search engine released under the GNU General Public License.
- **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.
- **ICDL Crawler** is a cross-platform web crawler written in C++ and intended to crawl Web sites based on Web-site Parse Templates using computer's free CPU resources only.
- **mnoGoSearch** is a crawler, indexer and a search engine written in C and licensed under the GPL (Linux machines only)
- **Nutch** is a crawler written in Java and released under an Apache License.

It can be used in conjunction with the Lucene text-indexing package.
- **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. Easy to install it became popular for small MySQL-driven web-sites on shared hosting.
- the **tkWWW Robot**, a crawler based on the tkWWW web browser (licensed under GPL).
- **YaCy**, a free distributed search engine, built on principles of peer-to-peer networks (licensed under GPL).
- **Scrapy**, an open source webcrawler framework, written in python (licensed under BSD).
- **Seeks**, a free distributed search engine (licensed under Affero General Public License).

Crawling the Deep Web

A vast amount of Web pages lie in the deep or invisible Web.[45] 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[46] 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 `<a href="URL">`-shaped URLs. 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.

Crawling Web 2.0 Applications

- Sheeraj Shah provides insight into Crawling Ajax-driven Web 2.0 Applications.
- Interested readers might wish to read AJAXSearch: Crawling, Indexing and Searching Web 2.0 Applications.
- Making AJAX Applications Crawlable, from Google Code. It defines an agreement between web servers and search engine crawlers that allows for dynamically created content to be visible to crawlers. Google currently supports this agreement.[47]
- **Relevant research on crawling Web 2.0 Applications:**

1. Model based crawling of Rich Internet Applications - Software Security Research Group - University of Ottawa and IBM Research Labs

2. Crawljax is an open source Java tool for automatically crawling and testing modern (Ajax) web applications.

3. SPCI Project - Delft University of Technology

See also

- Distributed web crawling
- Focused crawler
- Search Engine Indexing – the step after crawling
- Spambot
- Spider trap
- Spidering Hacks – an O'Reilly book focused on spider-like programming
- Web archiving
- Website mirroring software

- Website Parse Template
- Web scraping
- Webgraph

References

1. Kobayashi, M. and Takeda, K. (2000). "Information retrieval on the web". *ACM Computing Surveys* (ACM Press) **32** (2): 144–173. doi:10.1145/358923.358934.
2. ^ *a* *b* Spetka, Scott. "The TkWWW Robot: Beyond Browsing". NCSA. Archived from the original on 3 September 2004. Retrieved 21 November 2010.
3. See definition of scutter on FOAF Project's wiki
4. ^ *a* *b* Edwards, J., McCurley, K. S., and Tomlin, J. A. (2001). "An adaptive model for optimizing performance of an incremental web crawler". *In Proceedings of the Tenth Conference on World Wide Web* (Hong Kong: Elsevier Science): 106–113. doi:10.1145/371920.371960.
5. Castillo, Carlos (2004). *Effective Web Crawling* (Ph.D. thesis). University of Chile. Retrieved 2010-08-03.
6. Gulli, A.; Signorini, A. (2005). "The indexable web is more than 11.5 billion pages". *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM Press.. pp. 902–903. doi:10.1145/1062745.1062789.
7. Lawrence, Steve; C. Lee Giles (1999-07-08). "Accessibility of information on the web". *Nature* **400** (6740): 107. Bibcode 1999Natur.400..107L. doi:10.1038/21987. PMID 10428673.
8. Cho, J.; Garcia-Molina, H.; Page, L. (1998-04). "Efficient Crawling Through URL Ordering". *Seventh International World-Wide Web Conference*. Brisbane, Australia. Retrieved 2009-03-23.
9. Cho, Junghoo, "Crawling the Web: Discovery and Maintenance of a Large-Scale Web Data", Ph.D. dissertation, Department of Computer Science, Stanford University, November 2001
10. Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quali-

ty pages. In Proceedings of the Tenth Conference on World Wide Web, pages 114–118, Hong Kong, May 2001. Elsevier Science.

11. Abiteboul, Serge; Mihai Preda, Gregory Cobena (2003). "Adaptive on-line page importance computation". *Proceedings of the 12th international conference on World Wide Web*. Budapest, Hungary: ACM. pp. 280–290. doi:10.1145/775152.775192. ISBN 1-58113-680-3. Retrieved 2009-03-22.
12. Boldi, Paolo; Bruno Codenotti, Massimo Santini, Sebastiano Vigna (2004). "UbiCrawler: a scalable fully distributed Web crawler". *Software: Practice and Experience* **34** (8): 711–726. doi:10.1002/spe.587. Retrieved 2009-03-23.
13. Boldi, Paolo; Massimo Santini, Sebastiano Vigna (2004). "Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations". *Algorithms and Models for the Web-Graph*. pp. 168–180. Retrieved 2009-03-23.
14. Baeza-Yates, R., Castillo, C., Marin, M. and Rodriguez, A. (2005). Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering. In Proceedings of the Industrial and Practical Experience track of the 14th conference on World Wide Web, pages 864–872, Chiba, Japan. ACM Press.
15. Shervin Daneshpajouh, Mojtaba Mohammadi Nasiri, Mohammad Ghodsi, A Fast Community Based Algorithm for Generating Crawler Seeds Set, In proceeding of 4th International Conference on Web Information Systems and Technologies (WEBIST-2008), Funchal, Portugal, May 2008.
16. Menczer, F. (1997). ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery. In D. Fisher, ed., Machine Learning: Proceedings of the 14th International Conference (ICML97). Morgan Kaufmann
17. Menczer, F. and Belew, R.K. (1998). Adaptive Information Agents in Distributed Textual Environments. In K. Sycara and M. Wooldridge (eds.) Proc. 2nd Intl. Conf. on Autonomous Agents (Agents '98). ACM Press
18. Chakrabarti, S., van den Berg, M., and Dom, B. (1999). Focused crawling: a new approach to topic-specific web resource discovery. Computer Networks, 31(11–16):1623–1640.
19. ^ *a b* Pinkerton, B. (1994). Finding what people want: Experiences with the

WebCrawler. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

20. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. (2000). Focused crawling using context graphs. In Proceedings of 26th International Conference on Very Large Databases (VLDB), pages 527-534, Cairo, Egypt.
21. Pant, Gautam; Srinivasan, Padmini; Menczer, Filippo (2004). "Crawling the Web". 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. Retrieved 2009-03-22.
22. 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.
23. Cho, Junghoo; Hector Garcia-Molina (2000). "Synchronizing a database to improve freshness". *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. ISBN 1-58113-217-4. Retrieved 2009-03-23.
24. ^ *a b* Jr, E. G. Coffman; 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.
25. ^ *a b* Cho, J. and Garcia-Molina, H. (2003). Effective page refresh policies for web crawlers. ACM Transactions on Database Systems, 28(4).
26. ^ *a b* Cho, Junghoo; Hector Garcia-Molina (2003). "Estimating frequency of change". *ACM Trans. Interet Technol.* **3** (3): 256–290. doi:10.1145/857166.857170. Retrieved 2009-03-22.
27. Ipeirotis, P., Ntoulas, A., Cho, J., Gravano, L. (2005) Modeling and managing content changes in text databases. In Proceedings of the 21st IEEE International Conference on Data Engineering, pages 606-617, April 2005, Tokyo.
28. Koster, M. (1995). Robots in the web: threat or treat? ConneXions, 9(4).
29. Koster, M. (1996). A standard for robot exclusion.
30. Koster, M. (1993). Guidelines for robots writers.

31. Baeza-Yates, R. and Castillo, C. (2002). Balancing volume, quality and freshness in Web crawling. In Soft Computing Systems – Design, Management and Applications, pages 565–572, Santiago, Chile. IOS Press Amsterdam.
32. Heydon, Allan; Najork, Marc (1999-06-26) (PDF). *Mercator: A Scalable, Extensible Web Crawler*. Retrieved 2009-03-22.[*dead link*]
33. Dill, S., Kumar, R., Mccurley, K. S., Rajagopalan, S., Sivakumar, D., and Tomkins, A. (2002). Self-similarity in the web. ACM Trans. Inter. Tech., 2(3):205–223.
34. "Web crawling ethics revisited: Cost, privacy and denial of service". *Journal of the American Society for Information Science and Technology*. 2006.
35. ^ *a b* Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 30(1-7):107–117.
36. ^ *a b* Shkapenyuk, V. and Suel, T. (2002). Design and implementation of a high performance distributed web crawler. In Proceedings of the 18th International Conference on Data Engineering (ICDE), pages 357-368, San Jose, California. IEEE CS Press.
37. CrawlTrack
38. SEO Crawlytics
39. Risvik, K. M. and Michelsen, R. (2002). Search Engines and Web Dynamics. Computer Networks, vol. 39, pp. 289–302, June 2002.
40. Eichmann, D. (1994). The RBSE spider: balancing effective search against Web load. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.
41. McBryan, O. A. (1994). GENVL and WWWW: Tools for taming the web. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.
42. Zeinalipour-Yazti, D. and Dikaiakos, M. D. (2002). Design and implementation of a distributed crawler and filtering processor. 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.
43. Cho, Junghoo; Hector Garcia-Molina (2002). "Parallel crawlers". *Proceedings of the 11th international conference on World Wide Web*. Honolulu, Hawaii, USA: ACM. pp. 124–135. doi:10.1145/511446.511464. ISBN 1-58113-449-5. Retrieved 2009-03-23.
44. Chakrabarti, S. (2003). Mining the Web. Morgan Kaufmann Publishers. ISBN 1-55860-754-4
45. Shestakov, Denis (2008). Search Interfaces on the Web: Querying and Characterizing. *TUCS Doctoral Dissertations 104, University of Turku*
46. Nelson, Michael L; Herbert Van de Sompel, Xiaoming Liu, Terry L Harrison, Nathan McFarland (2005-03-24). "mod_oai: An Apache Module for Metadata Harvesting". *Eprint arXiv:cs/0503069*: 3069. arXiv:cs/0503069. Bibcode 2005cs.....3069N.
47. Making AJAX Applications Crawlable: Full Specification

Further reading

- Cho, Junghoo, "Web Crawling Project", UCLA Computer Science Department.
- WIVET is a benchmarking project by OWASP, which aims to measure if a web crawler can identify all the hyperlinks in a target website.