Web Crawler Architecture

MARC NAJORK
Microsoft Research, Mountain View, CA, USA

Synonyms

Web crawler; Robot; Spider

Definition

A web crawler is a program that, given one or more seed URLs, downloads the web pages associated with these URLs, extracts any hyperlinks contained in them, and recursively continues to download the web pages identified by these hyperlinks. Web crawlers are an important component of web search engines, where they are used to collect the corpus of web pages indexed by the search engine. Moreover, they are used in many other applications that process large numbers of web pages, such as web data mining, comparison shopping engines, and so on. Despite their conceptual simplicity, implementing high-performance web crawlers poses major engineering challenges due to the scale of the web. In order to crawl a substantial fraction of the "surface web" in a reasonable amount of time, web crawlers must download thousands of pages per second, and are typically distributed over tens or hundreds of computers. Their two main data structures – the "frontier" set of yet-to-be-crawled URLs and the set of discovered URLs – typically do not fit into main memory, so efficient disk-based representations need to be used. Finally, the need to be "polite" to content providers and not to overload any particular web server, and a desire to prioritize the crawl towards high-quality pages and to maintain corpus freshness impose additional engineering challenges.

Historical Background

Web crawlers are almost as old as the web itself. In the spring of 1993, just months after the release of NCSA Mosaic, Matthew Gray [6] wrote the first web crawler, the World Wide Web Wanderer, which was used from 1993 to 1996 to compile statistics about the growth of the web. A year later, David Eichmann [5] wrote the first research paper containing a short description of a web crawler, the RBSE spider. Burner provided the first detailed description of the architecture of a web crawler, namely the original Internet Archive crawler [3]. Brin and Page's seminal paper on the (early) architecture of the Google search engine contained a brief description of the Google crawler, which used a distributed system of page-fetching processes and a central database for coordinating the crawl. Heydon and Najork described Mercator [8,9], a distributed and extensible web crawler that was to become the blueprint for a number of other crawlers. Other distributed crawling systems described in the literature include PolyBot [11], UbiCrawler [1], C-proc [4] and Dominos [7].

Foundations

Conceptually, the algorithm executed by a web crawler is extremely simple: select a URL from a set of candidates, download the associated web pages, extract the URLs (hyperlinks) contained therein, and add those URLs that have not been encountered before to the candidate set. Indeed, it is quite possible to implement a simple functioning web crawler in a few lines of a high-level scripting language such as Perl.

However, building a web-scale web crawler imposes major engineering challenges, all of which are ultimately related to scale. In order to maintain a search engine corpus of say, ten billion web pages, in a reasonable state of freshness, say with pages being refreshed every 4 weeks on average, the crawler must download over 4,000 pages/second. In order to achieve this, the crawler must be distributed over multiple computers, and each crawling machine must pursue multiple downloads in parallel. But if a distributed and highly parallel web crawler were to issue many concurrent requests to a single web server, it would in all likelihood overload and crash that web server. Therefore, web crawlers need to implement *politeness policies* that rate-limit the amount of traffic directed to any particular web server (possibly informed by that server's observed responsiveness). There are many possible politeness policies; one that is particularly easy to implement is to disallow concurrent requests to the same web server; a slightly more sophisticated policy would be to wait for time proportional to the last download time before contacting a given web server again.

In some web crawler designs (e.g., the original Google crawler [2] and PolyBot [11]), the page downloading processes are distributed, while the major data structures – the set of discovered URLs and the set of URLs that have to be downloaded – are maintained by a single machine. This design is conceptually simple, but it does not scale indefinitely; eventually the central data structures become a bottleneck. The alternative is to partition the major data structures over the crawling machines. Ideally, this should be done in such a way as to minimize communication between the

crawlers. One way to achieve this is to assign URLs to crawling machines based on their host name. Partitioning URLs by host name means that the crawl scheduling decisions entailed by the politeness policies can be made locally, without any communication with peer nodes. Moreover, since most hyperlinks refer to pages on the same web server, the majority of links extracted from downloaded web pages is tested against and added to local data structures, not communicated to peer crawlers. Mercator and C-proc adopted this design [9,4].

Once a hyperlink has been extracted from a web page, the crawler needs to test whether this URL has been encountered before, in order to avoid adding multiple instances of the same URL to its set of pending URLs. This requires a data structure that supports set membership test, such as a hash table. Care should be taken that the hash function used is collision-resistant, and that the hash values are large enough (maintaining a set of n URLs requires hash values with $\log_2 n^2$ bits each). If RAM is not an issue, the table can be maintained in memory (and occasionally persisted to disk for fault tolerance); otherwise a disk-based implementation must be used. Implementing fast disk-based set membership tests is extremely hard, due to the physical limitations of hard drives (a single seek operation takes on the order of 10 ms). For a disk-based design that leverages locality properties in the stream of discovered URLs as well as the domain-specific properties of web crawling, see [9]. If the URL space is partitioned according to host names among the web crawlers, the set data structure is partitioned in the same way, with each web crawling machine maintaining only the portion of the set containing its hosts. Consequently, an extracted URL that is not maintained by the crawler that extracted it must be sent to the peer crawler responsible for it.

Once it has been determined that a URL has not been previously discovered, it is added to the *frontier set* containing the URLs that have yet to be downloaded. The frontier set is generally too large to be maintained in main memory (given that the average URL is about 100 characters long and the crawling system might maintain a frontier of ten billion URLs). The frontier could be implemented by a simple disk-based FIFO queue, but such a design would make it hard to enforce the politeness policies, and also to prioritize certain URLs (say URLs referring to fast-changing news web sites) over other URLs. URL prioritization could be achieved by using a priority queue implemented as a heap data structure, but a disk-based heap would be far too expensive, since adding and removing a URL would require multiple seek operations. The Mercator design uses a frontier data structure that has two stages: a front-end that supports prioritization of individual URLs and a back-end that enforces politeness policies; both the front-end and the back-end are composed of a number of parallel FIFO queues [9]. If the URL space is partitioned according to host names among the web crawlers, the frontier data structure is partitioned along the same lines.

In the simplest case, the frontier data structure is just a collection of URLs. However, in many settings it is desirable to attach some attributes to each URL, such as the time when it was discovered, or (in the scenario of continuous crawling) the time of last download and a checksum or sketch of the document. Such historical information makes it easy to determine whether the document has changed in a meaningful way, and to adjust its crawl priority.

In general, URLs should be crawled in such a way as to maximize the utility of the crawled corpus. Factors that influence the utility are the aggregate quality of the pages, the demand for certain pages and topics, and the freshness of the individual pages. All these factors should be considered when deciding on the crawl priority of a page: a high-quality, highly-demanded and fast-changing page (such as the front page of an online newspaper) should be recrawled frequently, while high-quality but slow-changing and fast-changing but low-quality pages should receive a lower priority. The priority of newly discovered pages cannot be based on historical information about the page itself, but it is possible to make educated guesses based on per-site statistics. Page quality is hard to quantify; popular proxies include link-based measures such as PageRank and behavioral measures such as page or site visits (obtained from web beacons or toolbar data).

In addition to these major data structures, most web-scale web crawlers also maintain some auxiliary data structures, such as caches for DNS lookup results. Again, these data structures may be partitioned across the crawling machines.

Key Applications

Web crawlers are a key component of web search engines, where they are used to collect the pages that are to be indexed. Crawlers have many applications beyond general search, for example in web data mining (e.g., Attributor, a service that mines the web for

copyright violations, or ShopWiki, a price comparison service).

Future Directions

Commercial search engines are global companies serving a global audience, and as such they maintain data centers around the world. In order to collect the corpora for these geographically distributed data centers, one could crawl the entire web from one data center and then replicate the crawled pages (or the derived data structures) to the other data centers; one could perform independent crawls at each data center and thus serve different indices to different geographies; or one could perform a single geographically-distributed crawl, where crawlers in a given data center crawl web servers that are (topologically) close-by, and then propagate the crawled pages to their peer data centers. The third solution is the most elegant one, but it has not been explored in the research literature, and it is not clear if existing designs for distributed crawlers would scale to a geographically distributed setting.

URL to Code

Heritrix is a distributed, extensible, web-scale crawler written in Java and distributed as open source by the Internet Archive. It can be found at http://crawler.archive.org/

Cross-references

- Focused Web Crawling
- Incremental Crawling
- Indexing the Web
- Web Harvesting
- Web Page Quality Metrics

Recommended Reading

1. Boldi P., Codenotti B., Santini M., and Vigna S. UbiCrawler: a scalable fully distributed web crawler. Software Pract. Exper., 34(8):711–726, 2004.
2. Brin S. and Page L. The anatomy of a large-scale hypertextual search engine. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 107–117.
3. Burner M. Crawling towards eternity: building an archive of the World Wide Web. Web Tech. Mag., 2(5):37–40, 1997.
4. Cho J. and Garcia-Molina H. Parallel crawlers. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 124–135.
5. Eichmann D. The RBSE Spider – Balancing effective search against web load. In Proc. 3rd Int. World Wide Web Conference, 1994.
6. Gray M. Internet Growth and Statistics: Credits and background. http://www.mit.edu/people/mkgray/net/background.html
7. Hafri Y. and Djeraba C. High performance crawling system. In Proc. 6th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2004, pp. 299–306.
8. Heydon A. and Najork M. Mercator: a scalable, extensible web crawler. World Wide Web, 2(4):219–229, December 1999.
9. Najork M. and Heydon A. High-performance web crawling. Compaq SRC Research Report 173, September 2001.
10. Raghavan S. and Garcia-Molina H. Crawling the hidden web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 129–138.
11. Shkapenyuk V. and Suel T. Design and Implementation of a high-performance distributed web crawler. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 357–368.