



A SURVEY ON WEB CRAWLER

Jaira Dubey, Divakar Singh

Barkatullah University, Bhopal, Madhya Pradesh, India
jairadubey87@gmail.com, divakar_singh@rediffmail.com

Abstract— In today scenario World Wide Web is flooded with huge amount of information. Finding useful information from the Web is quite challenging task. There are many search engines available in the market that will solve our purpose. However among all, selecting proper search engine with highly effective web crawler is quite necessary. There are many challenges in the design of high performances web crawler like it must be able to download pages at high rate, store it into the database efficiently and also crawl page rapidly. In this paper we present taxonomy of WebCrawler, various challenges and solutions of WebCrawler and various crawling algorithms.

Index Terms— Crawler, Database, Search Engine, World Wide Web

I. INTRODUCTION

With the explosive growth of information sources available on the World Wide Web, it has become necessary to use automated tools for finding the desired information resources, and for tracking and analyzing their usage patterns. For instance if a user wished to locate information on the Web then either had to know the precise address of the documents he sought or had to navigate patiently from link to link in hopes of finding his destination.

These factors give rise to the necessity of creating server-side and client-side intelligent systems that can effectively mine for knowledge. Search engines will serve our purpose. Search engines consist of two fundamental components - web crawlers, which will find, download, and parse content in the WWW and data miners, which will extract keywords from pages, rank document importance and answer user queries.

Web crawlers (also called as spiders, robots, walkers and wanderers) are programs which traverse through the web searching for the relevant information [1] using algorithms that narrow down the search by finding out the most closer and relevant information.

They are mainly used to create a copy of all the visited pages for later processing by mechanisms, that will index the downloaded pages to provide fast searches and further processing. This process is iterative, as long the results are in close proximity of user's interest.

II. RELATED WORK

A. World Wide Web Wanderer

In late 1993 and early 1994, when the Web was small, limited primarily to research and educational institutions, Matthew Gray implemented the World Wide Web Wanderer [2, 20]. It was written in Perl and was able to indexed pages from around 6000 sites. However, as the size of the Web increased, this crawler faced four major problems: fault tolerance, scale, politeness, and supervision. Among all serious of these problems was fault tolerance. Although the system was basically reliable, the machine running the crawler would occasionally crash and corrupt the database.

B. Lycos Crawler

Another crawler named Lycos [3, 20] was developed that ran on a single machine and used Perl's associative arrays to maintain the set of URLs to crawl. It was capable to index tens of millions of pages; however, the design of this crawler remains undocumented.

C. Internet Archive Crawler

Around 1997, Mike Burner's developed Internet Archive crawler [4, 20] that used multiple machines to crawl the web. Each crawler process was assigned up to 64 sites to crawl and no sites are assigned to more than one crawler. Each crawler process (single-threaded) read a list of seed URLs for its assigned sites from disk into per-site queues, and then used asynchronous I/O instructions to fetch pages from these queues in parallel. Once a page gets downloaded, the crawler extracted all the links contained in it. If a link referred to any site of the page was contained in it, then it was added to the

appropriate site queue; else it was logged to disk. Periodically, these logged “cross-site” URLs was merged by a batch process into the site-specific seed sets, filtering out duplicates one.

D. Google Crawler

The original Google crawler [5, 20] (developed at Stanford) consisted of five functional components running in different processes. A URL server process read URLs from a file and forwarded all to the multiple crawler processes. Each crawler process (single-threaded) that was running on a different machine used asynchronous I/O instructions to fetch data from up to 300 web servers in parallel. Then all the crawlers transmitted downloaded pages to a single Store Server process that compressed the pages and stored them to the disk. The pages were then read back from the disk by an indexer process, which extracted links from the HTML pages and saved them to the different disk file. URL resolver process read the link file; derelativized URLs contained therein, and saved the absolute URLs to disk file that was read by URL server.

E. Mercator Crawler

Mercator was highly scalable and easily extensible crawler. It was written in Java. The first version [6] was non-distributed; a later distributed version [7] partitioned the URL space over the crawlers according to host name, and avoided the potential bottleneck of a centralized URL server.

III. ARCHITECTURE OF WEB CRAWLER

Web crawlers recursively traverse and download web pages (Using GET and POST commands) for search engines to create and maintain the web indices. The need for maintaining the up to-date pages causes a crawler to revisit the websites again and again.

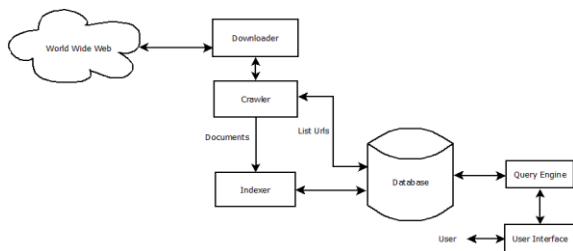


Figure 1 Architecture of Web Crawler

In general, it starts with a list of URLs to visit, known as seed URLs. As the Crawler traverses these URLs, it identifies all hyperlinks in the page and adds them to the list of URLs to be visited, called the crawl frontier. URLs from the crawl frontier are visited one by one and searching of the input pattern is done whenever text content is extracted from the page source of the web page.

IV. TAXONOMY OF WEB CRAWLER

With an increasing number of parties interested in crawling the World Wide Web, for a variety of reasons, a number of different crawl types have emerged. The development team at Internet Archive has highlighted three distinct variations:

- Broad Crawling
- Focused crawling
- Continuous crawling

Broad and focused crawls are in many ways similar, the primary difference being that broad crawls emphasize capturing a large scope, whereas a focused crawler crawls web pages related to a particular topic quickly without having to explore every web page.

Both approaches use a snapshot strategy, which involves crawling the scope once and once only i.e. No information from past crawls is used in new ones, except some changes to the configuration made by the operator, to avoid crawler traps etc. [15, 16]

The snapshot strategy (sometimes referred to as periodic crawling) is useful for large-scale crawls in that it minimizes the amount of state information that needs to be stored at any one time. The crawler need only store a fingerprint of the URIs, once a resource has been collected. This makes it possible to crawl a fairly large scope using a snapshot approach.

However, this does not do a good job of capturing changes in resources. Large crawls take time, meaning that there is a significant gap between revisits. Even if crawled within a reasonable amount of time, a snapshot crawl will fail to detect unchanged documents, leading to unnecessarily duplicates. Snapshot crawling is therefore primarily of use for large scale crawling, i.e. either crawling a large number of websites or trying to crawl each website completely (leading to very 'deep' crawls) or both.

Continuous crawling requires the crawler to revisit the same resources at certain intervals. This means that the crawler must retain detailed state information and, via some intelligent control, reschedule resources that have already been processed. Here, an incremental strategy is used in continuous crawling.

An incremental strategy maintains a record of each resource's history; this is used in turn to determine its ordering in a priority queue of resources waiting to be fetched. Using adaptive revisiting techniques, it is quite possible to capture the changes made in online resources within its scope far more accurately, in turn allowing incremental crawler to revisit each page more often.

Also, revisits are not bound by crawling cycles (as with the snapshot approach), since at any time any page can come up for revisiting, rather than only once. However, because of additional overhead, and the need to revisit

resources, an incremental strategy is not able to cope with as broad scope as a snapshot crawl might. By this incremental update, the crawler refreshes existing pages and replaces “less-important” pages with new and “more-important” pages. To conclude, the choice between an incremental or snapshot strategy can be described as choosing between space and time completeness. Naturally, we would wish to capture both well and with no hardware limitations we might do so, but in light of limited bandwidth and storage, difficult decisions must to be made [17,18].

V. SIMPLE WEB CRAWLER PROCESS

Web-crawling robots, or spiders, have a certain enigma among Internet users. We all use search engines like Yahoo, MSN and Google to find resources on the Internet, and these engines internally use spiders or crawlers to gather the information they present to us. Spiders or crawlers are network applications which traverse the Web, for accumulating statistics about the content found. Simple crawling process has following steps:

- Create a queue of URLs to be searched beginning with one or more known URLs.
- Pull a URL out of the queue and fetch the Hypertext Markup Language (HTML) page which can be found at that location.
- Scan the HTML page looking for new-found hyperlinks.
- Add the URLs for any hyperlinks found to the URL queue.
- If there are URLs left in the queue, then go to step 2.

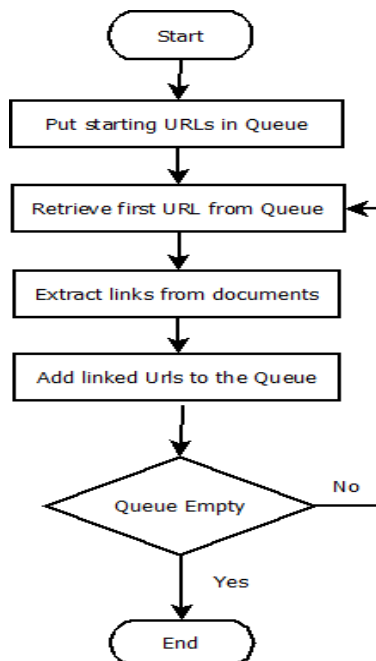


Figure 2 Web Crawler Process

VI. CHALLENGES OF WEB CRAWLING

Given the enormous size and the change rate of the Web, there are many issues [21] arise in design of high performance web crawler; some of them are as following:

• **What pages should the crawler download?** - Crawler cannot download all pages available on the Web. Even the most comprehensive search engine currently only indexes a small fraction of the entire Web [8]. Given this fact, it is important for the crawler to carefully select the pages and to visit “important” pages first by prioritizing the URLs in the queue properly.

• **How should the crawler refresh pages?** - Once crawler downloaded a significant number of pages from web, it started revisiting the downloaded pages for detecting changes and refreshing [10] the downloaded collection. As web pages are changing at very different rates, the crawler needs to decide carefully what page to revisit and what page to skip, as this decision may significantly impact the “freshness” of the downloaded collection.

• **How should the load on the visited Web sites be minimized?** - Crawler consumes resources belonging to other organizations [9] for collecting pages from the Web. For instance, when crawler downloads the page p on site S, then the site S needs to retrieve page p from its file system that is consuming disk and CPU resource. It might possible that after this retrieval the page needs to be transferred through the network, which is resource consumption, shared by the multiple organizations. The high performance crawler should minimize its impact on these resources. Otherwise, the administrators of the Website or any particular network may complain and sometimes completely block access by the crawler.

• **How should the crawling process be parallelized?** - Because of the enormous size of the Web, a crawler needs to run on multiple machines and download pages in parallel. This parallelization is quite necessary for downloading a large number of pages in a reasonable amount of time. These parallel crawlers must be coordinated properly for ensuring that different crawlers do not visit the same Website or page multiple times, and the adopted crawling policy must be strictly enforced.

VII. WEB CRAWLING ALGORITHMS

Breadth-First Crawling - The idea of breadth-first indexing is to retrieve all the pages around the starting point before following links further away from the start. It is the most common way through which crawlers or robots follow links. If crawler is indexing several hosts,

then this approach distributes the load quickly. It becomes also easier for robot writers to implement parallel processing for this system.

Andy-Yo-Et-Al [11] proposed a distributed BFS for numerous branches using Poisson random graphs and achieved high scalability through a set of clever memory and communication optimizations.

Depth-First Crawling - Depth-first indexing follows all the links from the first link on the starting page, and then follows the first link on the second page, and so on. Once the first link on each page has indexed, it goes on to the second and subsequent links, and then follows them. Some unsophisticated robots or spiders use this method, as it might be easier to code.

Page Rank Crawling - Page Rank algorithm [19] was described by Lawrence Page and Sergey Brin in several publications and given as:

$$PR(p) = (1-d) + d(PR(T_1)/L(T_1) + \dots + PR(T_n)/L(T_n))$$

Where,

$PR(p)$ is the Page Rank of page p,

$PR(T_i)$ is the Page Rank of pages T_i that is linking to page p

$L(T_i)$ is the number of outbound links present on page T_i

d is a damping factor sets between 0 and 1

From above expression, we see that Page Rank does not rank web sites as a whole, but also determined for each page individually. Further, the Page Rank of page p is recursively defined by the Page Ranks of those pages which link to page p.

Genetic Algorithm - Genetic algorithm is a simulation technique that uses a formal approach to simulate above situation and finally come up with an approximate solution to a problem. Its process is defined as:

- Starts with some random or predefined initial guesses
- Searches for those keywords
- Selects "acceptable" results from your search results and mark down some keywords from it
- Repeat this until the results are approximately what we are looking for
- Stop if after searching over and over several we are not getting good results

[18] shows the genetic algorithm is best suited when the user has literally no or less time to spend in searching a huge database and also very efficient in multimedia results. While almost all conventional methods search from a single point, Genetic Algorithms always operates

on a whole population. This contributes much to the robustness of genetic algorithms.

HITS Algorithm -This algorithm put forward by Kleinberg is previous to Page rank algorithms which uses scores to calculate the relevance [13]. This method retrieves a set of results for a search and calculates the authority and hub score within that set of results. Because of these reasons this method is not often used [12]. Joel C. Miller et al [14] proposed a modification on adjacency matrix input to HITS algorithm which gave intuitive results.

VIII. CONCLUSION

Web crawlers are a central part of search engines. We have described researches related to web crawler. We also presented the architecture of crawler along with simple crawling process.

Furthermore, this paper also discussed the issues addressed by Crawler and various search algorithms.

REFERENCES

- [1] S. Pavalam, M. Jawahar, F. Akorli, S. Raja, "Web Crawler in Mobile Systems," IJMLC, vol. 2, pp. 531-534.
- [2] M. Gray, "Internet Growth and Statistics: Credits and Background," available at: <http://www.mit.edu/people/mkgray/net/background.html>.
- [3] M. Mauldin, "Lycos: Design Choices in an Internet Search Service," IEEE Expert, vol. 12, pp. 8-11, 1997.
- [4] M. Burner, "Crawling towards Eternity: Building an Archive of the World Wide Web," Web Techniques Magazine, vol. 2, pp. 37-40, 1997.
- [5] S. Brin, L. Page. "The Anatomy of a Large-scale Hyper textual Web Search Engine", International World Wide Web Conference, pp. 107-117, 1998.
- [6] A. Heydon, M. Najork, "Mercator: A Scalable, Extensible Web Crawler, "World Wide Web, vol. 2, pp. 219-229, 1999.
- [7] M. Najork, A. Heydon, "High-performance Web Crawling," Technical report, Compaq SRC Research Report 173, 2001.
- [8] S. Lawrence, C. Giles, "Accessibility of Information on the Web", Nature, vol. 400, pp. 107-109, 1999.
- [9] M. Koster, "Robots in the Web: Threat or Treat?," ConneXions, vol. 4, 1995.
- [10] L. Page, S. Brin, R. Motwani, T. Winograd, "The Pagerank Citation Ranking: Bringing Order to the Web," Technical report, Computer Science Department, Stanford University, 1998.

- [11] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, A. CatalyÅurek, "A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L," ACM 2005.
- [12] A. Signorini, "A Survey of Ranking Algorithms," "available at: <http://www.divms.uiowa.edu/~assignori/phd/report/asurvey-of-ranking-algorithms.pdf> 29/9/2011
- [13] J. Kleinberg, "Hubs, Authorities, and Communities," ACM computing survey, 1998.
- [14] J. Miller, G. Rae, F. Schaefer, "Modifications of Kleinberg's HITS Algorithm Using Matrix Exponentiation and Web Log Records," SIGIR'01, ACM 2001.
- [15] M. Najork, A. Heydon, "High-Performance Web Crawling," available at: <ftp://gatekeeper.research.compaq.com/pub/DEC/SRC/researchreports/SRC173.pdf>.
- [16] B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Wolff, "A Brief History of the Internet," available at: www.isoc.org/internet/history.
- [17] C. Dyreson, H. Lin, Y. Wang, "Managing Versions of Web Documents in a Transaction-time Web Server," In Proceedings of the World-Wide Web Conference.
- [18] Heydon A., Najork M., "Mercator: A Scalable, Extensible Web Crawler", World Wide Web, vol. 2, pp. 219-229, 1999.
- [19] S. Pavalam, M. Jawahar, F. Akorli, S. Raja, "A Survey of Web Crawler Algorithms," IJCSI, vol. 8, Issue 6, No 1, November 2011
- [20] Olston and M. Najork, "Web Crawling", Foundations and Trends in Information Retrieval, vol. 4, No. 3, pp. 175–246, 2010
- [21] Dr Rajender Nath and Khyati., " Web Crawlers: Taxonomy, Issues & Challenges", International Journal of Advanced Research in Computer Science and Software Engineering 3(4), Vol. 3, Issue 4, pp. 944-948, April – 2013

