ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

Semantic Web Crawler using Early Detection and Sentimental Analysis

Nidhi Goyal

M. Tech Student, Dept. of Computer Science., Manav Rachna International University at Faridabad, Haryana, India.

ABSTRACT : As the size of the web is growing apace, it's become necessary to create the seek for content quicker and a lot of accurate data, while with no all search engines, it might be not possible to induce correct results, to beat this downside, computer code referred to as "Web Crawler" is applied that uses numerous styles of algorithms to attain the goal. These algorithms use numerous styles of heuristic functions to extend potency of the crawlers. However, using search square measure over sentimental analysis a number of the simplest path finding algorithms uses a Best-First Search and finds the least-cost path from a given initial node to a goal node, during this work, it is been achieved some of the prevailing net Crawler algorithms using Early Detection and more ways are introduced to be utilized in this domain.

KEYWORDS: Crawler; Semantic Network; Sentimental Analysis; Map Reduce; Spiders; Bots

I. INTRODUCTION

In the course of extracting links, any Web crawler will encounter multiple links to the same document. To avoid downloading and processing a document multiple times, a URL-seen test must be performed on each extracted link before adding it to the URL frontier. (An alternative design would be to instead perform the URL-seen test when the URL is removed from the frontier, but this approach would result in a much larger frontier.) To perform the URL-seen test, we store all of the URLs seen by Mercator in canonical form in a large table called the URL set. Again, there are too many entries for them all to fit in memory, so like the document fingerprint set, the URL set is stored mostly on disk. To save space, we do not store the textual representation of each URL in the URL set, but rather a fixed-sized checksum. Unlike the fingerprints presented to the content-seen test's document fingerprint set, the stream of URLs tested against the URL set has a non-trivial amount of locality. To reduce the number of operations on the backing disk file, we therefore keep an in-memory cache of popular URLs. The intuition for this cache is that links to some URLs are quite common, so caching the popular ones in memory will lead to a high in-memory hit rate. In fact, using an in-memory cache of 2^18 entries and the LRU [Last Recently Used] like clock replacement policy, we achieve an overall hit rate on the in-memory cache of 66.2%, and a hit rate of 9.5% on the table of recently-added URLs, for a net hit rate of 75.7%. Moreover, of the 24.3% of requests that miss in both the cache of popular URLs and the table of recently-added URLs, about 1=3 produce hits on the buffer in our random access file implementation, which also resides in user-space. The net result of all this buffering is that each membership test we perform on the URL set results in an average of 0.16 seek and 0.17 read kernel calls (some fraction of which are served out of the kernel's file system buffers). So, each URL set membership test induces one-sixth as many kernel calls as a membership test on the document fingerprint set. These savings are purely due to the amount of URL locality (i.e., repetition of popular URLs) inherent in the stream of URLs encountered during a crawl.

Basically they hash all of the URLs with a hashing function that guarantees unique hashes for each URL and due to the locality of URLs, it becomes very easy to find URLs. Google even open-sourced their hashing function like cityhash however we will implement the Early Detection in this scheme using Sentimental Analysis for fetching the best accurate results from the cached pages or vide data repositories.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

II. RELATED WORK

This section provides an overview of how the whole system of a search engine works. The major functions of the search engine crawling, indexing and searching are also covered in detail in the later sections. Before a search engine can tell you where a file or document is, it must be found. To find information on the hundreds of millions of Web pages that exist, a typical search engine employs special software robots, called spiders, to build lists of the words found on Web sites. When a spider is building its lists, the process is called Web crawling. A Web crawler is a program, which automatically traverses the web by downloading documents and following links from page to page. They are mainly used by web search engines to gather data for indexing. Other possible applications include page validation, structural analysis and visualization, update notification, mirroring and personal web assistants/agents etc. Web crawlers are also known as spiders, robots, worms etc. Crawlers are automated programs that follow the links found on the web pages. There is a URL Server that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the store server. The store server then compresses and stores the web pages into a repository. Every web page has an associated ID number called a doc ID, which is assigned whenever a new URL is parsed out of a web page. The indexer and the sorter perform the indexing function. The indexer performs a number of functions. It reads the repository, un-compresses the documents, and parses them. Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of "barrels", creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link. The URL Resolver reads the anchors file and converts relative URLs into absolute URLs and in turn into doc IDs. It puts the anchor text into the forward index, associated with the doc ID that the anchor points to. It also generates a database of links, which are pairs of doc IDs. The links database is used to compute Page Ranks for all the documents. The sorter takes the barrels, which are sorted by doc ID and resorts them by word ID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of word IDs and offsets into the inverted index. A program called Dump Lexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. A lexicon lists all the terms occurring in the index along with some term-level statistics (e.g., total number of documents in which a term occurs) that are used by the ranking algorithms The searcher is run by a web server and uses the lexicon built by Dump Lexicon together with the inverted index and the Page Ranks to answer queries.

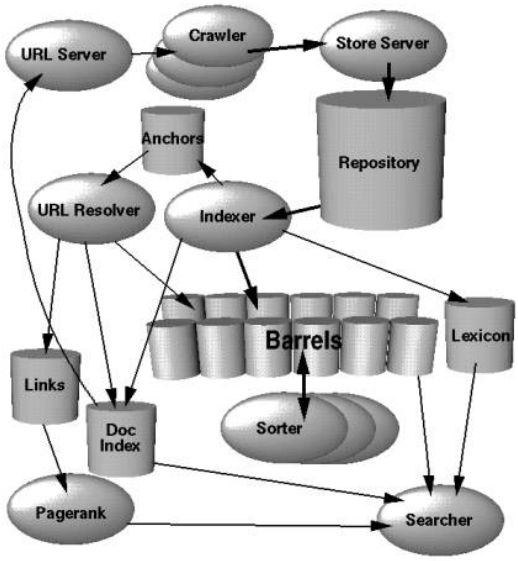

Figure 1: High Level Search Engine Architecture (Brin and Page, 1998)

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

A. How The Web Crawlers Works :-

Web crawlers are an essential component to search engines; running a web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers, which are all beyond the control of the system. Web crawling speed is governed not only by the speed of one's own Internet connection, but also by the speed of the sites that are to be crawled. Especially if one is a crawling site from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel. Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. Following is the process by which Web crawlers work:

1. Download the Web page.
2. Parse through the downloaded page and retrieve all the links.
3. For each link retrieved, repeat the process.

The Web crawler can be used for crawling through a whole site on the Inter-/Intranet. You specify a start-URL and the Crawler follows all links found in that HTML page. This usually leads to more links, which will be followed again, and so on. A site can be seen as a tree-structure, the root is the start-URL; all links in that root-HTML-page are direct sons of the root. Subsequent links are then sons of the previous sons.

A single URL Server serves lists of URLs to a number of crawlers. Web crawler starts by parsing a specified web page, noting any hypertext links on that page that point to other web pages. They then parse those pages for new links, and so on, recursively. Webcrawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve web pages at a fast enough pace. A crawler resides on a single machine. The crawler simply sends HTTP requests for documents to other machines on the Internet, just as a web browser does when the user clicks on links. All the crawler really does is to automate the process of following links. Web crawling can be regarded as processing items in a queue. When the crawler visits a web page, it extracts links to other web pages. So the crawler puts these URLs at the end of a queue, and continues crawling to a URL that it removes from the front of the queue. (Garcia-Molina 2001)

B. Meta-Search Engine :-

A meta-search engine is the kind of search engine that does not have its own database of Web pages. It sends search terms to the databases maintained by other search engines and gives users the results that come from all the search engines queried. Fewer metasearchers allow you to delve into the largest, most useful search engine databases. They tend to return results from smaller and/or free search engines and miscellaneous free directories, often small and highly commercial. The mechanism and algorithms that meta-search engines employ are quite different. The simplest meta-search engines just pass the queries to other direct search engines. The results are then simply displayed in different newly opened browser windows as if several different queries were posed. Some improved meta-search engines organize the query results in one screen in different frames, or in one frame but in a sequential order. Some more sophisticated meta-search engines permit users to choose their favorite direct search engines in the query input process, while using filters and other algorithms to process the returned query results before displaying them to the users. Problems often arise in the query-input process though. Meta-Search engines are useful if the user is looking for a unique term or phrase; or if he (she) simply wants to run a couple of keywords. Some meta-search engines simply pass search terms along to the underlying direct search engine, and if a search contains more than one or two words or very complex logic, most of them will be lost. It will only make sense to the few search engines that supports such logic. Following are some of the powerful meta-search engines with some direct search engines like AltaVista and Yahoo. (Liu, 1999)

C. Improvements of Web Search Engines:-

There are different ways to improve the performance of web search engines. Generally speaking, there are three main directions:

1. Improving user interface on query input
2. Using Filtering towards the query results
3. Solving algorithms in web page spying and collecting, indexing, and output

Method 3 is the fundamental solution for any direct search engines to deal with the problems of unequal accessing, out-of-date information, and low metadata using, as well as information coverage. It is also very important to look at the

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

user interface issues that methods 1 and 2 are dealing with — How to handle user queries effectively and present the results efficiently

D. Crawling Techniques

Focused Crawling: A general purpose Web crawler gathers as many pages as it can from a particular set of URL's. Where as a focused crawler is designed to only gather documents on a specific topic, thus reducing the amount of network traffic and downloads. The goal of the focused crawler is to selectively seek out pages that are relevant to a pre-defined set of topics. The topics are specified not using keywords, but using exemplary documents. Rather than collecting and indexing all accessible web documents to be able to answer all possible ad-hoc queries, a focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the web. This leads to significant savings in hardware and network resources, and helps keep the 7 crawl more up-to-date. The focused crawler has three main components: a classifier, which makes relevance judgments on pages crawled to decide on link expansion, a distiller which determines a measure of centrality of crawled pages to determine visit priorities, and a crawler with dynamically reconfigurable priority controls which is governed by the classifier and distiller. The most crucial evaluation of focused crawling is to measure the harvest ratio, which is rate at which relevant pages are acquired and irrelevant pages are effectively filtered off from the crawl. This harvest ratio must be high, otherwise the focused crawler would spend a lot of time merely eliminating irrelevant pages, and it may be better to use an ordinary crawler instead (Baldi, 2003).

Distributed Crawling: Web sites also often have restricted areas that crawlers should not crawl. To address these concerns, many Web sites adopted the Robot protocol, which establishes guidelines that crawlers should follow. Over time, the protocol has become the unwritten law of the Internet for Web crawlers. The Robot protocol specifies that Web sites wishing to restrict certain areas or pages from crawling have a file called robots.txt placed at the root of the Web site. The ethical crawlers will then skip the disallowed areas. Following is an example robots.txt file and an explanation of its format:

```
# robots.txt for http://somehost.com/
User-agent: *
Disallow: /cgi-bin/
Disallow: /registration #
Disallow robots on registration page
Disallow: /login
```

The first line of the sample file has a comment on it, as denoted by the use of a hash (#) character. Crawlers reading robots.txt files should ignore any comments. 8 The third line of the sample file specifies the User-agent to which the Disallow rules following it apply. User-agent is a term used for the programs that access a Web site. Each browser has a unique User-agent value that it sends along with each request to a Web server. However, typically Web sites want to disallow all robots (or User-agents) access to certain areas, so they use a value of asterisk (*) for the User-agent. This specifies that all User-agents be disallowed for the rules that follow it. The lines following the User-agent lines are called disallow statements. The disallow statements define the Web site paths that crawlers are not allowed to access. For example, the first disallow statement in the sample file tells crawlers not to crawl any links that begin with "/cgi-bin/". Thus, the following URLs are both off limits to crawlers according to that line. http://somehost.com/cgi-bin/ http://somehost.com/cgi-bin/register (Searching Indexing Robots and Robots.txt 2002).

E. Storing the Web Content

In addition to indexing the web content, the individual pages are also stored in the search engine's database. Due to cheaper disk storage, the storage capacity of search engines is very huge, and often runs into terabytes of data. However, retrieving this data quickly and efficiently requires special distributed and scalable data storage functionality. The amount of data, that a search engine can store, is limited by the amount of data it can retrieve for search results. Google can index and store about 3 billion web documents. This capacity is far more than any other search engine during this time. "Spiders" take a Web page's content and create key search words that enable online users to find pages they're looking for. (Franklin, 2002)

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

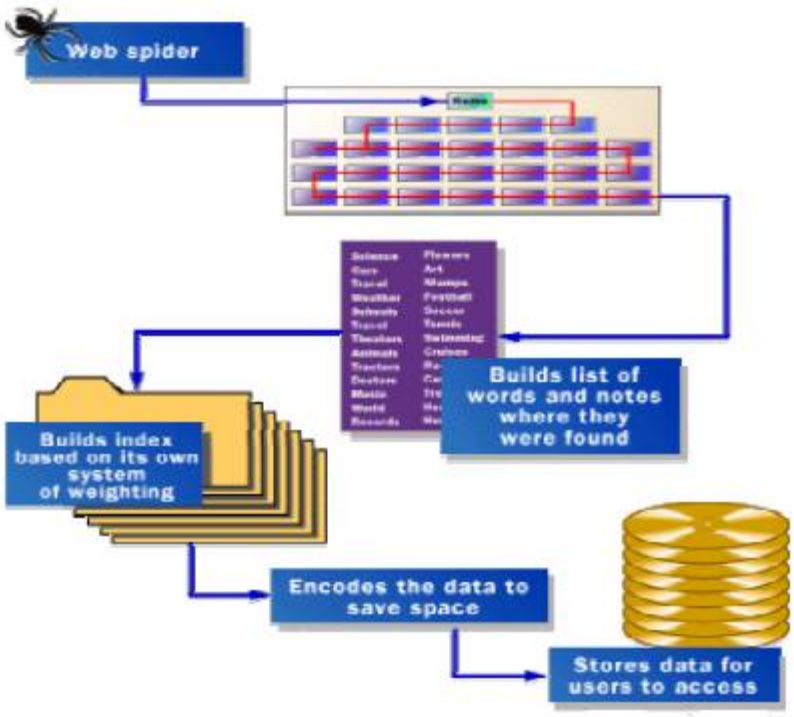

Figure 2 Explaining Web Crawler.

III. PROPOSED ALGORITHM

In the above scheme will be proposed Early Detection algorithm for selecting pages, then it's fairly easy to avoid bot-traps of the infinite loop kind. Here is a summary of how Early Detection algorithm works:

Pseudo Generation as follows :-

1. Get a set of N seed pages.
2. Allocate X amount of credit to each page, such that each page has X/N credit (i.e. equal amount of credit) before crawling has started.
3. Select a page P, where the P has the highest amount of credit (or if all pages have the same amount of credit, then crawl a random page).
4. Crawl page P (let's say that P had 100 credits when it was crawled).
5. Extract all the links from page P (let's say there are 10 of them).
6. Set the credits of P to 0.
7. Take a 10% "tax" and allocate it to a Lambda page.
8. Allocate an equal amount of credits each link found on page P from P's original credit - the tax: so (100 (P credits) - 10 (10% tax))/10 (links) = 9 credits per each link.
9. Repeat from step 3.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

Since the Lambda page continuously collects tax, eventually it will be the page with the largest amount of credit and we'll have to "crawl" it. I say "crawl" in quotes, because we don't actually make an HTTP request for the Lambda page, we just take its credits and distribute them equally to **all** of the pages in our database. Since bot-traps only give internal links credits and they rarely get credit from the outside, they will continually leak credits (from taxation) to the Lambda page. The Lambda page will distribute that credits out to all of the pages in the database evenly and upon each cycle the bot trap page will lose more and more credits, until it has so little credits that it almost never gets crawled again. This will not happen with good pages, because they often get credits from back-links found on other pages. This also results in a dynamic page rank and what you will notice is that any time you take a snapshot of your database, order the pages by the amount of credits they have, then they will most likely be ordered roughly according to their **true page rank**.This only avoid bot traps of the infinite-loop kind and then the results are passed for sentimental analysis.

The proposed sentimental analysis algorithm has following steps:

1.Initialize P (pos) <- nr_popozitii (pos) / nr_total_propozitii

2.Initialize P (neg) <- nr_popozitii (neg) / nr_total_propozitii

Tokenize sentence in words

3.For each class of {pos, neg}:

4.For each word in {phrase} P (word | class) <- nr_apartii (word | class) 1 / nr_cuv (class) + nr_total_cuvinte

5.P (class) <-P (class) * P (word | class)

6.Returns max {P(pos), P(neg)}

IV. RESULTS

Using above scheme and algorithm the crawl test results following statistics:-

Parameter [Single Crawl Test on Intel i3 2.1 GHZ with 4GB RAM over 8 Mbps Line]	**Values**
Download thread count	200
Dns thread number	5
Crawl depth	15
Same server access interval	35s
Initial URL	www.sharetipsinfo.com

Crawl Results: 2 hours, Download page :176804, the size of 6.16GB

Parameter [Parallel Crawling Test for 2 Nodes on Intel i3 2.1 GHZ with 4GB RAM over 8 Mbps Line]	**Values**
Download thread count	200
Dns thread number	5
Crawl depth	15
Same server access interval	35s
Initial URL	www.sharetipsinfo.com

Crawl Results: 3.25 hours, Download page :176804, the size of 6.16GB

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

REFERENCES

1.A. Dasgupta, R. Kumar, and A. Sasturkar, ―De-Duping URLs via Rewrite Rules,‖Proc. 14th ACM SIGKDD Int‘l Conf. Knowledge Discovery and Data Mining,pp. 186-194, 2008.
2.M. Henzinger, ―Finding Near-Duplicate Web Pages: A LargeScale Evaluation of Algorithms,‖Proc. 29th Ann. Int‘l ACM SIGIR. Conf. Research and Development in Information Retrieval,pp. 284-291, 2006.
3.H.S. Koppula, K.P. Leela, A. Agarwal, K.P. Chitrapura, S. Garg, and A. Sasturkar, ―Learning URL Patterns for Webpage DeDuplication,‖Proc. Third ACM Conf. Web Search and Data Mining, pp. 381-390, 2010.
4.J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI‘04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, pages 10–10, December 2004.
5. L. Zhang, B. Liu, S.H. Lim, and E. O‘Brien-Strain, ―Extracting and Ranking Product Features in Opinion Documents,‖ Proc. 23rd Int‘l Conf. Computational Linguistics, pp. 1462-1470, 2010.
6.M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In 34th Annual ACM Symposium on Theory of Computing (May 2002).
7.A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic Clustering of the Web. In 6th International World Wide Web Conference (Apr. 1997), 393-404.
8.U. Manber. Finding similar files in a large file system. In Proc. of the USENIX Winter 1994 Technical Conference (Jan. 1994).
9.N. Heintze. Scalable Document Fingerprinting. In Proc. of the 2nd USENIX Workshop on Electronic Commerce (Nov 1996).
10.T.C. Hoad and J. Zobel. Methods for identifying versioned and plagiarised documents. Journal of the American Society for Information Science and Technology 54(3):203-215, 2003.
11. M. Rabin. Fingerprinting by random polynomials. Report TR-15-81, Center for Research in Computing Technology , Harvard University , 1981.