

Implementing an effective Web Crawler

Shalin Shah



Implementing an effective Web Crawler

Introduction:

Web crawler (also known as a Web spider or Web robot) is a program or automated script which browses the World Wide Web in a methodical and automated manner.

This process is called Web crawling or **spidering**. Many legitimate 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, which 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 codes. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for 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.

Why do we need a web crawler?

Following are some reasons to use a web crawler:

- To maintain mirror sites for popular Web sites.
- To test web pages and links for valid syntax and structure.
- To monitor sites to see when their structure or contents change.
- To search for copyright infringements.
- To build a special-purpose index—for example, one that has some understanding of the content stored in multimedia files on the Web.

How does a Web Crawler Work?

A typical web crawler starts by parsing a specified web page: noting any hypertext links on that page that point to other web pages. The Crawler then parses those pages for new links, and so on, recursively. A crawler is a software or script or automated program which 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.

This is the basic concept behind implementing web crawler, but implementing this concept is not merely a bunch of programming. The next section describes the difficulties involved in implementing an efficient web crawler.

Difficulties in implementing efficient web crawler:

There are two important characteristics of the Web that generate a scenario in which Web crawling is very difficult:

1. Large volume of Web pages.
2. Rate of change on web pages.

A large volume of web page implies that web crawler can only download a fraction of the web pages and hence it is very essential that web crawler should be intelligent enough to prioritize download.

Another problem with today's dynamic world is that web pages on the internet change very frequently, as a result, by the time the crawler is downloading the last page from a site, the page may change or a new page has been placed/updated to the site.

Solutions – Right strategies:

The difficulties in implementing efficient web crawler clearly state that bandwidth for conducting crawls is neither infinite nor free. So, it is becoming essential to crawl the web in not only a scalable, but efficient way, if some reasonable amount of quality or freshness of web pages is to be maintained. This ensues that a crawler must carefully choose at each step which pages to visit next.

Thus the implementer of a web crawler must define its behavior.

Defining the behavior of a Web crawler is the outcome of a combination of below mentioned strategies:

- Selecting the better algorithm to decide which page to download.
- Strategizing how to re-visit pages to check for updates.
- Strategizing how to avoid overloading websites.

• Selecting the right algorithm:

Given the current size of the web, it is essential that the crawler program should crawl on a fraction of the web. Even large search engines in today's dynamic world crawls fraction of web pages from web. But, a crawler should observe that the fraction of pages crawled must be most relevant pages, and not just random pages.

While selecting the search algorithm for the web crawler an implementer should keep in mind that algorithm must make sure that web pages are chosen depending upon their importance. The importance of a web page lies in its popularity in terms of links or visits, or even its URL.

• Algorithm types:

- **Path-ascending crawling:**

We intend the crawler to download as many resources as possible from a particular Web site. That way a crawler would ascend to every path in each URL that it intends to crawl. For example, when given a seed URL of `http://foo.org/a/b/page.html`, it will attempt to crawl `/a/b/`, `/a/`, and `/`.

The advantage with Path-ascending crawler is that they are very effective in finding isolated resources, or resources for which no inbound link would have been found in regular crawling.

- **Focused crawling:**

The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. In this strategy we can intend web crawler to download pages that are similar to each other, thus it will be called focused crawler or topical crawler.

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; to resolve this problem proposed solution would be to use 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.

- **How to Re-visit web pages:**

The optimum method to re-visit the web and maintain average freshness high of web page is to ignore the pages that change too often.

The approaches could be:

- Re-visiting all pages in the collection with the same frequency, regardless of their rates of change.
- Re-visiting more often the pages that change more frequently.

(In both cases, the repeated crawling order of pages can be done either at random or with a fixed order.)

The re-visiting methods 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.

- **How to avoid overloading websites:**

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.

The use of Web crawler is useful for a number of tasks, but comes with a price for the general community. 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.
- Personal crawlers that, if deployed by too many users, can disrupt networks and Web servers.

To resolve this problem we can use robots exclusion protocol, also known as the robots.txt protocol.

The robots exclusion standard or robots.txt protocol is a convention to prevent cooperating web spiders and other web robots from accessing all or part of a website. We can specify the top level directory of web site in a file called robots.txt and this will prevent the access of that directory to crawler.

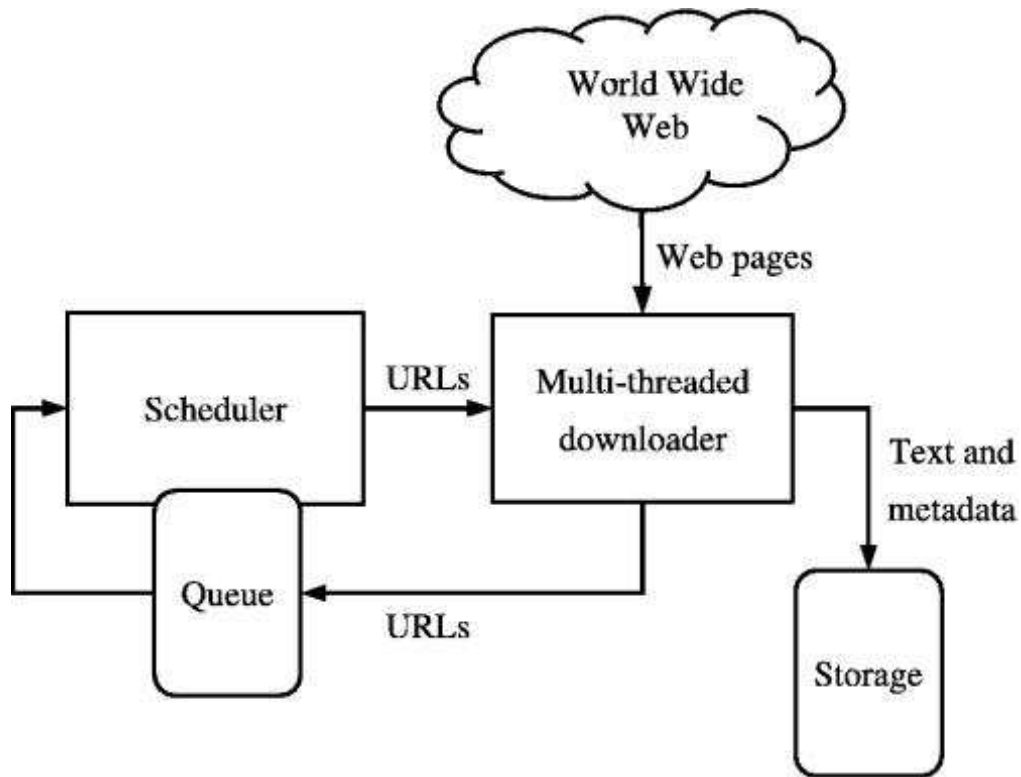
This protocol uses simple substring comparisons to match the patterns defined in robots.txt file. So, while using this robots.txt file we need to make sure that we use final '/' character appended to directory path. Else, files with names starting with that substring will be matched rather than directory.

Example of robots.txt files that tells all crawlers not to enter into four directories of a website:

```
User-agent: *  
Disallow: /cgi-bin/  
Disallow: /images/  
Disallow: /tmp/  
Disallow: /private/
```

Web crawler architectures:

A crawler must have a good crawling strategy, as noted in the previous sections, but it also needs a highly optimized architecture.



High-level architecture of a standard Web crawler

❖ **Pseudo Code for a Web Crawler:**

Here's a pseudo code summary of the algorithm that can be used to implement a web crawler:

Ask user to specify the starting URL on web and file type that crawler should crawl.

Add the URL to the empty list of URLs to search.

```

While not empty ( the list of URLs to search )
{

    Take the first URL in from the list of URLs
    Mark this URL as already searched URL.

    If the URL protocol is not HTTP then
        break;
        go back to while

    If robots.txt file exist on site then
        If file includes "Disallow" statement then
            break;
            go back to while

    Open the URL

    If the opened URL is not HTML file then
        Break;
        Go back to while

    Iterate the HTML file

    While the html text contains another link {

        If robots.txt file exist on URL/site then
            If file includes "Disallow" statement then
                break;
                go back to while

        If the opened URL is HTML file then
            If the URL isn't marked as searched then
                Mark this URL as already searched URL.

        Else if type of file is user requested
            Add to list of files found.

    }
}

```

Conclusion:

Building an effective web crawler to solve your purpose is not a difficult task, but choosing the right strategies and building an effective architecture will lead to implementation of highly intelligent web crawler application.