IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-2, Issue-4 E-ISSN: 2347-2693

Design and Implementation of Web Crawler

Prashant Dahiwale[1], Ankita Dangre[2*], Puja Kolpyakwar[3],Vishakha Wankhede[4] and Priyanka Akre[5]

[1,2,3,4,5]Dept.of Computer Science and Engineering,,RTMNU,India*

www.ijcaonline.org

Received: 11/03/2014 Revised: 23/03/2014 Accepted: 20/04/201x4 Published: 30/04/2014

Abstract – As the number of Internet users and the number of accessible Web pages grows, it is becoming increasingly difficult for users to find documents that are relevant to their particular needs. The key factors for the success of the World Wide Web are its large size and the lack of a centralized control over its contents. Users must either browse through a large hierarchy of concepts to find the information for which they are looking or submit a query to a publicly available search engine and wade through hundreds of results, most of them irrelevant[5]. Web crawling is the process used by search engines to collect pages from the Web. Web crawlers are one of the most crucial components in search engines and their optimization would have a great effect on improving the searching efficiency. This paper, introduces web crawler that uses a concept of irrelevant pages for improving its crawling performance. [5] Despite their conceptual simplicity, implementing high-performance web crawlers poses major engineering challenges due to the scale of the web. This crawler computes the weights for the pages we come across during the crawling process and hence decide how much a particular page is important to us. Both issues are also the most important source of problems for locating information. The Web is a context in which traditional Information Retrieval methods are challenged, and given the volume of the Web and its speed of change, the coverage of modern search engines is relatively small. Moreover, the distribution of quality is very skewed, and interesting pages are scarce in comparison with the rest of the content.

Index Term— Web Crawler , Seed , Frontier, Page Weight, Threshold Value

I. INTRODUCTION :

Internet is the shared global computing network. The Internet is a global systemof interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to serve several billion users worldwide. It enables global communications between all connected computing devices.[2] It is a network of networks that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless, and optical networking technologies. It provides the platform for web services and the World Wide Web. Web is the totality of web pages stored on web servers. There is a spectacular growth in web-based information sources and services. The Internet carries

an extensive range of information resources and services, such as the inter-linked hypertext documents of the World Wide Web (WWW), the infrastructure to support email, and peer-to-peer networks. It is estimated that, there is approximately doubling of web pages each year. As the Web grows grander and more diverse, search engines also have assumed a central role in the World Wide Web's infrastructure as its scale and impact have escalated. In Internet data are highly unstructured which makes it extremely difficult to search and retrieve valuable information. Search engines define content by keywords.

Corresponding Author: *Ankita Dangre*
Dept.of Computer Science and Engineering,,RTMNU,India

A Web crawler 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.[4] The high rate of change implies that the pages might have already been updated or even deleted.

The number of possible URLs crawled 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.[4] 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. [2]

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. A crawler must carefully choose at each step which pages to visit next.[3]

A. Crawling policy

The behaviour of a web crawler is the outcome of combination of policies:

- 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.

1) 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 a previous study by Steve Lawrence and Lee Giles showed that no search engine indexed more than 16% of the Web in 1999. 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.

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.

a) Restricting followed links

b) URL normalization

c) Path-ascending crawling

d) Focused crawling

2) 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

Freshness: This is a binary measure that indicates whether the local copy is accurate or not.
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:

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.

3) 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.

4) Parallelisation policy:

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.[3]

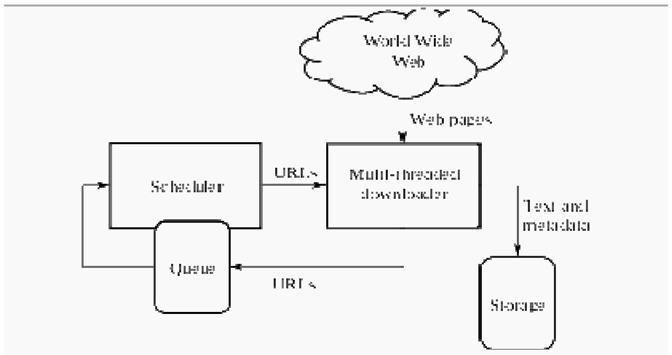
Earlier work was based on how the web crawler works, the process of web crawler and how the sequence of accepting the URL, fetching the page, parsing the page, extracting all the hyperlinks is performed. While performing the following sequence, we are downloading the page we need to verify for the evaluation. Hence, while downloading the page we anyways use up the bandwidth. It will be even more beneficial if we utilize the used bandwidth and get more out of it. 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.[3]

Earlier work was based on how the web crawler works, the process of web crawler and how the sequence of accepting the URL, fetching the page, parsing the page, extracting all the hyperlinks is performed. While performing the following sequence, we are downloading the page we need to verify for the evaluation. Hence, while downloading the page we anyways use up the bandwidth. It will be even more beneficial if we utilize the used bandwidth and get more out of it.

Thus implementing the following method, we use the downloaded pages' bandwidth and get the same bandwidth to get the title, body and the number of outgoing links on that particular page.

II. METHODOLOGY

Architectures[3]



High-level architecture of a standard Web crawler

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.[2] 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.

We define the factors for which we specify the page importance:

weight(page) = weight(URL) + weight(outlinks) + weight(title) + weight(body)
where,

```
1)      if ( search string present in URL)
        {
                weight(URL) returns a predefined weight
        }
        Else
        {
                Return 0
        }
```

This will return the weight assigned for the URL occurrence. If the search string is found in the URL, the page acquires certain importance.

```
2)      if ( search string present in title)
        {
                weight(title) returns a predefined weight
        }
        Else
        {
                Return 0
        }
```

This will return the weight assigned for the title occurrence. If the search string is found in the title, the page acquires certain importance.

```
3)      Occurrence of search string in the body
        {
                weight(body)=occurrence*weight for each occurence
        }
```

This will return the weight assigned for the body occurrence. If the search string is found in the body, the page acquires certain importance. When the search string occurs certain number of times in the body, the occurrence is noted and the page importance is calculated using the occurrence count.

```
4)      Number of hyperlinks on the page
        {
        weight(outlinks)=occurrence*weight for each occurence
        }
```

This will return the weight assigned for the out-links occurrence. The number of links linking to the other page has also been assigned some importance.

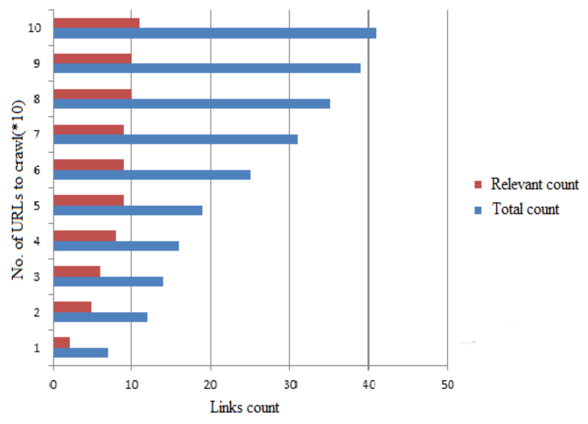
Giving importance to each component of the parsed page, we have assigned weight to each component and hence acquired the page importance in totality. As we get the page weight, we will compare it with the threshold frequency implicitly provided to the algorithm. Depending on the result of comparison, the links are either added to the output or they may be discarded.

Thus, we get the search more focused to the search string eliminating the least important topic.

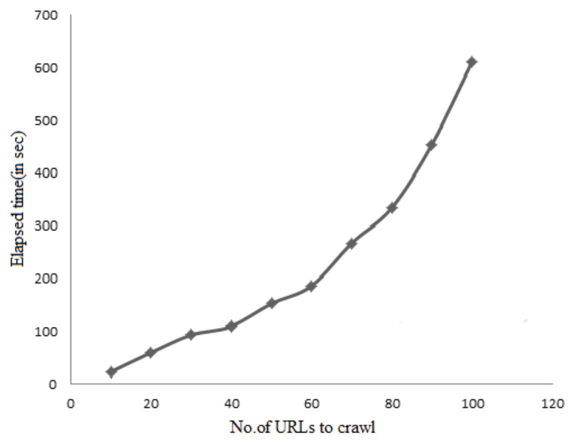
Proposed Algorithm (Pseudo Copde):

```
1. Start
2. Initialize frontier with seed URL.
3. While (frontier is not empty)
{
        Pick URL from frontier
        Fetch page
        Parse page
        calculate weight(page).
}
4. if(weight(page) > (threshold_value))
{
        Add to output.
}
5. Stop
```

III. Result & Discussion



- As the number of URL to crawl increases,the links count also increases.
- When the total links increases, the relevant links don't increase with same speed.
- As we go far away from the seed URL,the frequency of finding relevant links decreases.



- As the number of URLs to crawl increases,the elapsed time also increases.
- As we go far away from seed, the time for finding relevant links also increases respectively.

.

IV. Conclusion

Hence by using the concept of Page Weight, we scan web pages as well as compute the weight of page and hence we can increase efficiency of web crawler as output set of URL generated by this way will always be of better importance than what traditional web crawler is generating.

V. Scope for Further Research

As we parse the page, we have only extracted the hyperlinks on the page. We can proceed the work by extracting the images, videos and other non textual content and hence carry out the further process.

VI .Acknowledgment

We would like to express my special thanks of gratitude to our Guide Prof. Rahul Sathawane as well as our Co-guide Prof. Prashant Dahiwale who gave us the golden opportunity to do this wonderful project on the topic "Design and Implementation of Web Crawler", which also helped us in doing a lot of Research and we came to know about so many new things we are really thankful to them.

VII References

[1] Prashant Dahiwale, Anil Mokhade, M.M. Raghuwanshi, Intelligent Web Crawlers, ICWET, ACM New York, NY, USA, pp. 613-617, 2010.
[2] Brian Pinkerton, Finding what people want: Experiences with the Web Crawler, Proceedings of first World Wide Web conference, Geneva, Switzerland, 1994
[3] Gautam Pant, Padmini Srinivasan, Filippo Menczer, Crawling the Web, pp. 153-178, Mark Levene, Alexandra Poulovassilis (Ed.), Web Dynamics: Adapting to Change in Content, Size, Topology and Use, Springer-Verlag, Berlin, Germany, November 2004.
[4] Christopher Olston, Marc Najork, Web Crawler Architecture, Journal Foundations and Trends in Information Retrieval archive, Volume 4 Issue 3, pp. 175-246, March 2010.
[5] B. Pinkerton, "Finding what people want: Experiences with the WebCrawler," in Proceedings of the 2nd International World Wide Web Conference ,1994.
[6] en.wikipedia.org/wiki/

AUTHORS PROFILE

Name: Prof. Prashant Dahiwale
E-Mail:prashant.dahiwale@gmail.com
Institute:RGCER,Nagpur

Name: Ankita Dangre
E-Mail:ankitavd92@gmail.com
Institute:RGCER,Nagpur

Name: Puja Kolpyakwar
E-Mail:puja.kolpyakwar@gmail.com
Institute:RGCER,Nagpur

Name: Vishakha Wankhede
E-Mail:wankhede.vishakha@gmail.com
Institute:RGCER,Nagpur

Name: Priyanka Akre
E-Mail:priyankaakre779@gmail.com
Institute:RGCER,Nagpur