VFAST Transactions on Software Engineering
http://vfast.org/journals/index.php/VTSE@ 2020 ISSN(e): 2309-3978;ISSN(p): 2411-6246
Volume 8, Number 1, January-December, 2020

pp. 37-42

DESIGNING A PAGERANK BASED PROTOTYPE SEARCH ENGINE

Ayaz Alavi, Muhammad Hanif Durad*
Department of Computer and Information Sciences
Pakistan Institute of Engineering and Applied Sciences
ayaz.alavi@gmail.com, *hanif@pieas.edu.pk

Abstract: In this paper the challenges faced during building a prototype of a search engine have been discussed. Each component of a search engine has been described in details. Architectures for each module have been proposed that is most appropriate according to our research on search engine technology and their implementation details have also been provided. . Several algorithms had been designed and implemented for each of the component but pre-built technologies were also used like Pagerank. As Google's Pagerank [1][2], algorithm is also embedded in our search engine (although in a small scale), thus its explanation is also provided along with the implementation details.

1. Introduction. Our search engine has four major components mentioned below:

Web Crawler: It is an automated program responsible for getting web documents.

Indexer: It is a program that converts raw web documents into searchable format.

Searcher: It is a program that returns a user, set of documents when provided with the query through an interface.

Ranker: It is a program responsible for computing page rank for each page.

Our search engine relies on a crawler module to provide the grist for its operation. Crawlers are small programs that browse the web on search engine behalf just like an internet user would follow links to reach different pages. These programs are given a starting set of URLs, whose pages they retrieve from the web server. The starting set of urls picked by the search engines are usually web directories like dmoz and yahoo. The crawler extracts urls appearing in the result pages and feed those urls back to itself through some predefined strategy. The retrieved pages are saved in repository where it can be indexed by an indexer module.

The Indexer module extracts all the word from the page, and records the URL where each word occurred. For that purpose prebuilt libraries can be used that are designed to do that job for e.g. *Java Lucene project* which is an open source program designed to index web documents.

1.1. Search Engine Architectures

Two architectures of search engine have been designed and implemented.

1st Architecture: This architecture has two programs serving as crawler module, and one program working as an indexer. It has user interface which allows users

to enter the query. It is shown in Figure 1. Because of several shortcomings in each module and absence of ranker module it was extended to the new architecture.

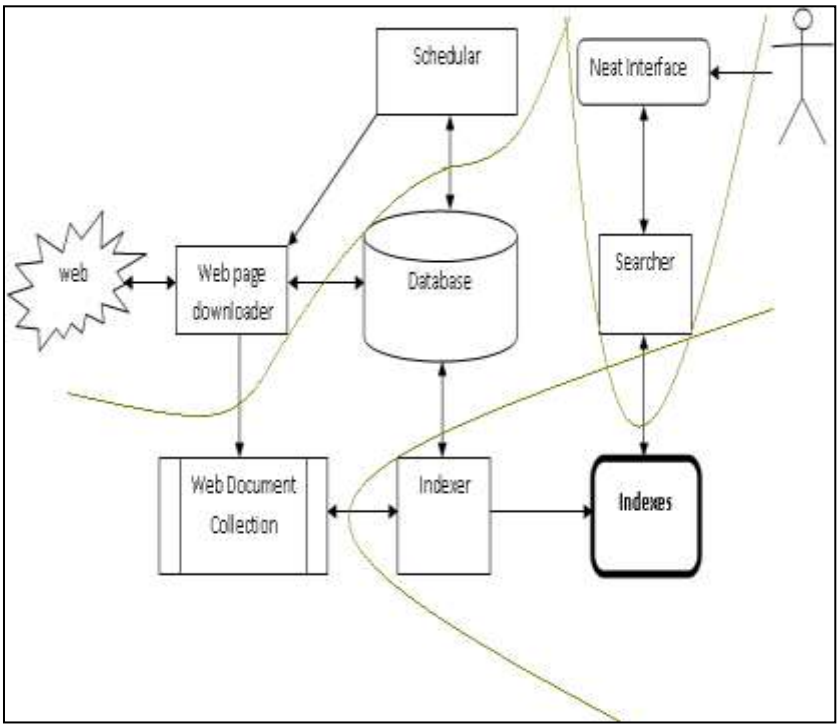

Figure 1. 1st Architecture

2nd Architecture: In this extended architecture Crawlers and Indexer are enhanced with new features. Searching algorithm was designed taking Google and Yahoo searching algorithm as an example. It is shown in Figure 2.

2. Web Crawler. Web crawlers implemented in our search engine evolved from very basic crawling model to heuristics based multiple modules web crawler with several functionalities embedded to provide grist of a real web crawler.

2.1. Design Issues: Web crawlers need to deploy technique in order to fetch URL to be downloaded. For this purpose either breadth first approach is used or some heuristics are used. Web crawlers also need to extract urls from the document in order to continue its execution. For this purpose either same program can used to extract urls as well as download them continuously in the same process or different programs can be used to handle network transfers and for URL extraction. Web crawlers also need to refresh downloaded documents in order to maintain local collection up-to-date [3]. For this purpose either document can be updated at the same frequency or at the different frequency. Web crawlers also can be classified according to how they are running. If crawler is running continuously then we call it steady crawler and if crawler is running periodically (say once every month) then we call it batch-mode crawler. Web crawlers also need to decide the way of updating local document collection. For this purpose either shadowing is deployed or In-place technique is deployed. In shadowing new set of pages are collected from the web and stored in a separate space from the current collection. After all new pages are collected and processed; the current collection is instantaneously replaced by this new collection. In In-place strategy crawler collection of documents is immediately available to the user.

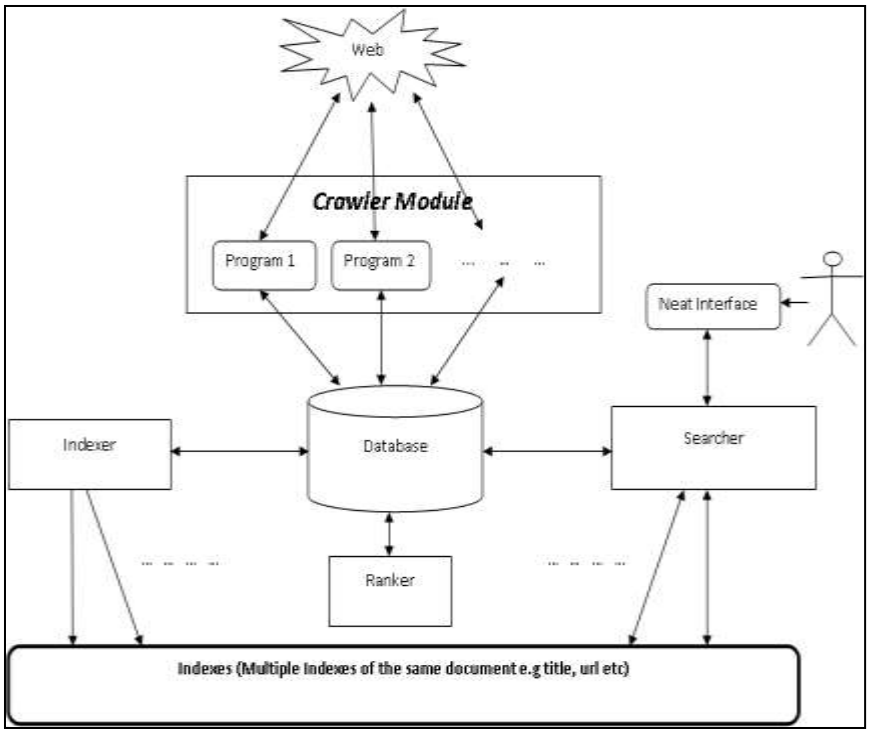

Figure 2. 2nd Architecture

2.2. Crawlers Designs

Our initial crawler was breadth first in which we were storing urls in a text file and keeps on extracting them in a FIFO order. We were using same program to handle network transfers from the web server and for the document manipulation. In this crawler we did not deployed document updates. From this basic crawling model, we discovered new objective of a web crawler i.e. to divide the crawling tasks into different subtasks that will be carried out efficiently by specialized modules that will eventually be implemented as computer programs. We achieved this separation of tasks with 2-Module crawler architecture and 4-Module crawler architecture.

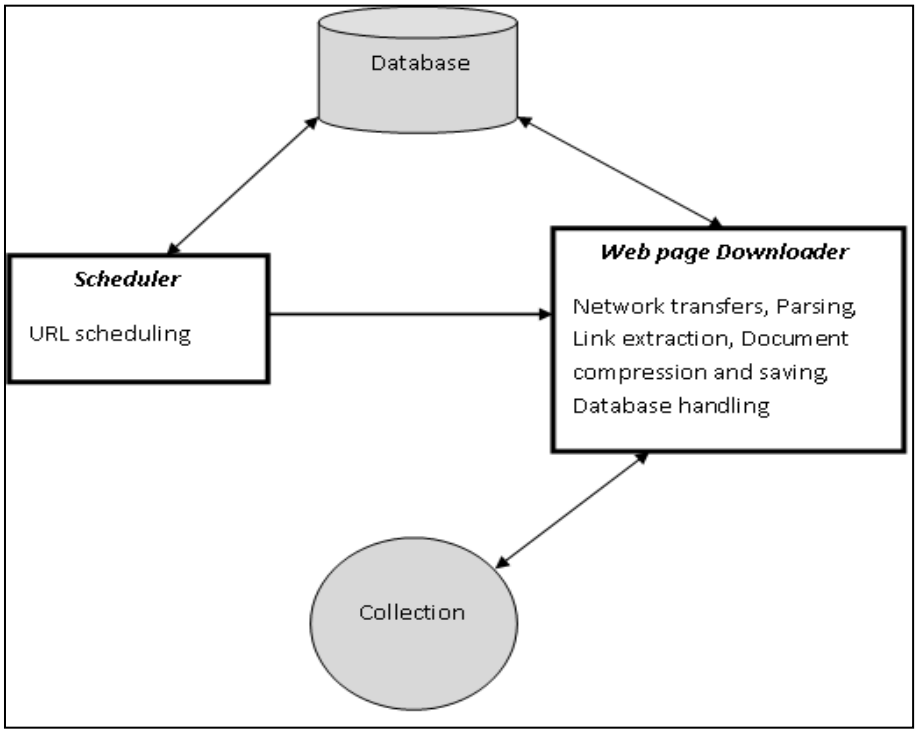

Figure 3. A software architecture with two modules, URL ordering is done by scheduler and download and parsed by web page downloader

In two module crawler we have following processes:

Scheduler: This program is scheduling urls for web page downloader.

Web Page Downloader: This program is handling Document downloading, parsing, link extraction and document saving.

2-Module crawler is storing urls in a database and keeps on extracting them in an alphabetical order and we are using same programs to handle network transfers from the web server and for the document manipulation. In this

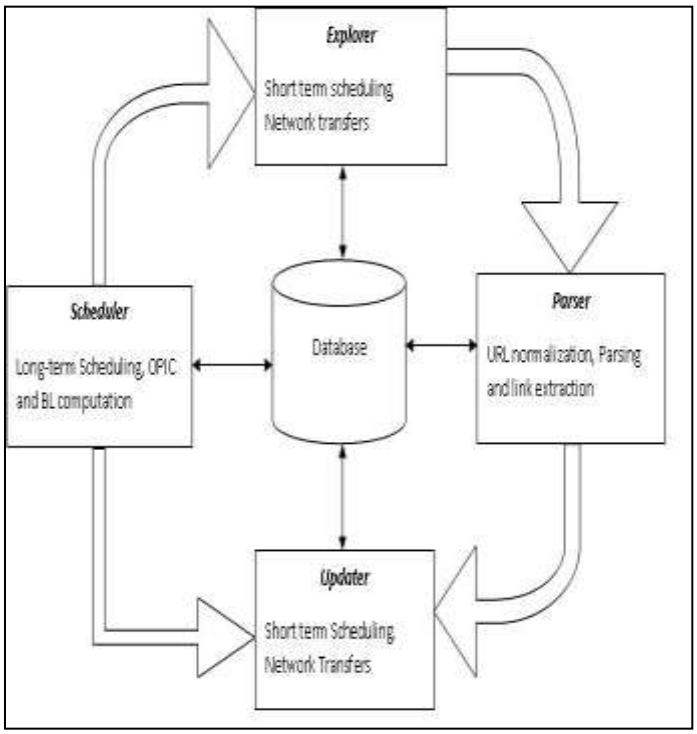

Figure 4. 4-modules architecture with separate parser and downloader modules, also updater module is present to update already download documents

crawler we did not deployed document updates. This crawler is steady and it is using In-place document updates. It is shown in Figure 3.

In four module crawler we had following modules.

Explorer: It handles network transfers for the new pages found by the crawler.

Updater: It handles document updates for the pages already downloaded by the explorer module.

Parser: It handles parsing, URL normalization and link extraction.

Scheduler: It handles long term scheduling of the URL, OPIC and BL scheduling algorithms. The 4-module crawler is storing urls in a database and keeps on extracting them using modified OPIC strategy. Separate modules are written for handling network transfers and for the document manipulation. Document update was deployed taking research of Junghoo and hector Garcia [3] under consideration i.e. we are updating pages neither using completely FIFO strategy nor using completely variable strategy but somewhere in-between these strategies but closer to fixed strategy which is optimal. This crawler is steady and it is using In-place document updates. It is shown in Figure 4.

2.3. Implementation Details. The programming language used was php for all of the applications. Mysql is used for the database development. Several software packages were used including SIMPLE HTML DOM, Xpdf Parser [4] etc. Also several prebuilt classes were used including DomainTld Parser, TimeCount etc.

All Meta data about web documents are stored inside mysql database. Database contains all necessary information to be used in our crawler programs. Currently database has 13 tables serving our search engine. It contains different types of information including Update Information, URL Information, URL Importance, Robot

Exclusion Protocol, Document contents, Meta Data about Page Contents, Anchors, Page Title and Descriptor Meta tag, and Index Status.Following issues are handled by our web crawler,

Fetching: Fetching of document from the web server and handling of error 404 and 999 handled by our crawler

Robot Exclusion Protocol: Our crawler obeyed this protocol by including separate Robot Class that is provided with URL and it returns whether URL is allowed to be crawled or not. ***[2]***

Parsing: Our crawlers are parsing documents in order to extract useful information from it. It includes urls, title, description etc.

Normalization: Our crawlers are normalizing [2] urls using several rules.

Php limitation: php limitations have been overcome by our crawlers using different strategies.

3. Indexer. Indexer is a program that converts web documents into a format that is searchable. The first stage for indexing web pages is to extract a standard logical view from the documents. The most used logical view for documents in search engines is the "bag of words" model, in which each document is seen only as an unordered set of words. In modern Web search engines, this view is extended with extra information concerning word frequencies and text formatting attributes, as well as meta-information about Web pages including embedded descriptions and explicit keywords in the HTML markup.

In our web search engine indexing operation is applied on a set of documents consisting of only html and pdf files stored either in either a file system or inside database. Thus there are two approaches implemented for Indexing purpose.

File system approach: In this technique set of documents are retrieved from the local hard disk. Index types are defined according to the content type with master index indexing all types of documents.

Database approach: In this technique set of documents are retrieved from the database. Index types are defined according to the field type with several threads running concurrently for indexing each field of the document. Our search engine is using inverted indexing scheme, in order to index documents. This algorithm is used by the search engines. In this algorithm each distinctive term is attached to the set of documents that contain that term. This index consists of two parts: a vocabulary V, containing all distinct terms in the document set, and for each distinct term an inverted list of postings. Each posting stores the ID of the document that contains that term and other pieces of information about that term in the document.

Again programming language used was php for all the applications. Mysql is used for the database development. Total approximately 1500 lines of code have been written for indexer module during this project. Several software packages were used including SIMPLE HTML DOM, Xpdf Parser etc. Also several prebuilt classes were used including TimeCount etc.

4. Searcher. Searcher module is the one responsible for serving our users. This module takes user query as an input through a user interface. This process is depending upon how indexes are distributed among computers. There are two methods of partitioning inverted index mentioned below [3-4]

Global inverted file: When using global inverted file, vocabulary is divided into several parts. Each computer is assigned a part of the vocabulary and all of its occurrences. Whenever the query is received, it is transmitted to the computers holding query terms. Afterwards, results from several computers are merged together. Thus in this technique load balancing is not easy.

Local inverted file: In local inverted file technique, document identifiers are divided, but each computers gets full vocabulary. A query is broadcasted to all computers, obtaining good load balance. This is the architecture used in the main search engines today, because building a global index is very hard for document collection as large as the web.

Our indexer can deploy local inverted file technique by creating an index of each field in a separate computer thus index of document titles will be created in a separate computer while index of document body will be created on a separate computer. Thus our searcher needs to transmit query to all of the computers and merge returned results later on.

In order to create more meaningful and relevant SERP we designed a searching algorithm taking Google and Yahoo searching algorithms as an example. Our searching algorithm is considering following fields in order to assign each document a score.

Document Title: It is a text between title tag in html document.

Document Anchors: It is a text between anchor tags.

Document URL: This parameter is included in our searching algorithm because URL's domain and path sometimes contain important keywords.
Pagerank: It is a score assigned to each document using link analysis.
Description Meta Tag: This tag gives short description of the page.
Document body: Since authors of the page might manually insert keywords in the document body in order to rank well thus it is used as the least effecting factor in our searching algorithm. We are parsing document bodies for HTML and pdf files.

5. **Ranker.** Including Pagerank in our searching algorithm was because of two reasons:
    a. Web is very large with great variation in the amount, quality and the type of information present in the web pages. Thus many pages that contain the search terms may be of poor quality or not relevant.
    b. Many web pages are not sufficiently self descriptive, so on the page factors for calculating its relevancy may not work well. Moreover, web pages are frequently manipulated by adding misleading terms so they are ranked higher by a search engine.

Pagerank is link analysis algorithm and it is implemented in our Ranker module. Pagerank importance is determined by votes in the form of links from other pages on the web. It is computed using following formula [5-6].

` $$\mathbf{r(i) = (1 - d) / n + d * \sum r(j) / N(j)}$$

It is implemented in php and is using "Only forward-links Pagerank computation algorithm".

Conclusion. In this paper several designs for each component were proposed. It resulted in the development of different components of a web search engine which were integrated to provide full software. This software is developed from the scratch, so it is better for other developers to extend it by including features that are valuable in the search engine market or improve its algorithms.

Architecture in this work is on a medium scale i.e. several components are interacting with each other to give desired result. It can easily be extended to the market scale by using distributed computing. Several algorithms are designed and implemented in this thesis for each of the component but pre-built technologies were also used like Pagerank.

Open source software's were also used and modified to give desired effect. Software is also able to index set of web documents, supported web documents are html and pdf. This software can also search index taking different fields of the document under consideration and can provide most relevant results to the user.

REFERENCES

[1]. Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.
[2]. Castillo, C. (2004). *Effective web crawler* (Doctoral dissertation, PhD thesis, Chile University).
[3]. Pant, G., Srinivasan, P., & Menczer, F. (2004). Crawling the web. In *Web dynamics* (pp. 153-177). Springer, Berlin, Heidelberg.
[4]. Cho, J., & Garcia-Molina, H. (1999). *The evolution of the web and implications for an incremental crawler*. Stanford.
[5]. Xpdf 3.02, Glyph & Cog, LLC. (2007), , Released under the open source GPL licence (2007), http://www.foolabs.com/xpdf/
[6]. PHP Simple Html Dom, Sourceforge, Manual, (2008), [Online] Available: http://simplehtmldo m .sourceforge.net/