Analysis of Searching Algorithms in Web Crawling

Pushkar Jha[1] Riddhi Patel[2] Aditya K. Sinha[3]
[3]Principal Technical Officer
[1, 2,3]Department of Computer Engineering
[1,2]Gujarat Technological University, Ahmedabad, Gujarat, India
[3] CDAC-ACTS, Pune, India

Abstract— In today's online scenario finding the appropriate content in minimum time is most important. The number of web pages and users is increasing into millions and trillions around the world. As the Internet expanded, the concern of storing such expanded data or information also came into existence. This leads to the maintaining of large databases critical for the effective and relevant information retrieval. Database has to be analyzed on a continuous basis as it contains dynamic information and database needs to be updated periodically over the internet. To make searching easier for users, web search engines came into existence. Even in forensic terms, the forensic data including social network such as twitter is over several Petabytes. As the evidence is large scale data, an investigator takes long time to search critical clue which is related to a crime. So, high speed analysis from large scale forensic data is necessary. So this paper basically focuses on the faster searching methods for the effective carving of data in limited time.

Keywords: Searching, online, databases, forensics, high speed.

I. INTRODUCTION

According to Internet World Stats survey [1], as on March 31, 2012, 1.407 billion people use the Internet. The vast expansion of the internet is getting more and more day by day. With the vast expansion of internet, it also leads to storage of this vast expanded internet data getting bigger and bigger. Even if someone simply fires a query in the search engine and waits for the response and results get generated in fewer seconds but the repository of the data stored gets bigger as the search engine crawls through each and every link encountered and all this data needs to be regularly updated as the information is dynamic so these all scenarios lead to a huge database and their needs a faster mechanism to index out the relevant data in an efficient manner. Finding out the relevant data from such huge repository of information is very vital in terms of information retrieval as well as in forensic terms. It has been seen that upto 93% of all information never leaves the digital domain. There are many activities such as chats and social networking that are specific to digital and are even unimaginable outside of the virtual realm. Most such activities leave definite traces, allowing investigators to obtain essential evidence, solve criminal cases and prevent crimes.

II. LITERATURE SURVEY

A. Basic Web Search Engine

The plentiful content of the World-Wide Web is useful to millions. Some simply browse the Web through entry points such as Yahoo, MSN etc. But many information seekers use a search engine to begin their Web activity [2]. In this case, users submit a query, typically a list of keywords, and receive a list of Web pages that may be relevant, typically pages that contain the keywords. By Search Engine in relation to the Web, we are usually referring to the actual search form that searches through-databases of HTML documents.

There are basically *3 types* of search engines:

- Those that are powered by robots (called crawlers, ants or spiders)
- Those that are powered by human submissions
- And those that are a hybrid of the two.

The main steps in any search engine are

1) *Gathering also called "Crawling":*

Every engine relies on a crawler module to provide the grist for its operation. This operation is performed by special software; called "Crawlers" Crawlers are small programs that `browse' the Web on the search engine's behalf, similarly to how a human user would follow links to reach different pages. The programs are given a starting set of URLs, whose pages they retrieve from the Web. The crawlers extract URLs appearing in the retrieved pages, and give this information to the crawler control module. This module determines what links to visit next, and feeds the links to visit back to the crawlers.

2) *Maintaining Database/Repository:*

All the data of the search engine is stored in a database. All the searching is performed through that database and it needs to be updated frequently. During a crawling process, and after completing crawling process, search engines must store all the new useful pages that they have retrieved from the Web.

3) *Indexing:*

Once the pages are stored in the repository, the next job of search engine is to make a index of stored data. The indexer module extracts all the words from each page, and records the URL where each word occurred. The result is a generally very large "lookup table" that can provide all the URLs that point to pages where a given word occurs. The table is of course limited to the pages that were covered in the crawling process. As mentioned earlier, text indexing of the Web poses special difficulties, due to its size, and its rapid rate of change. In addition to these quantitative challenges, the Web calls for some special, less common kinds of indexes. For example, the indexing module may also create a structure index, which reflects the links between pages.

4) *Querying:*

This sections deals with the user queries. The query engine module is responsible for receiving and filling search requests from users. The engine relies heavily on the indexes, and sometimes on the page repository. Because of the Web's size, and the fact that users typically only enter one or two keywords, result sets are usually very large.

5) *Ranking:*

Since the user query results in a large number of results, it is the job of the search engine to display the most appropriate results to the user. To do this efficient searching, the ranking of the results are performed. The ranking module therefore has the task of sorting the results such that results near the top are the most likely ones to be what the user is looking for. Once the ranking is done by the Ranking component, the final results are displayed to the user. This is how any search engine works.

B. Web Crawlers

One of the most essential jobs of any search engine is gathering of web pages, also called, "Crawling"**.** This crawling procedure is performed by special software called, "Crawlers" or "Spiders"

A web-crawler is a program/software or automated script which browses the World Wide Web in a methodical, automated manner.

Before we discuss the working of crawlers, it is worth to explain some of the basic terminology that is related with crawlers.

1) *Seed Page:*

By crawling, we mean to traverse the Web by recursively following links from a starting URL or a set of starting URLs. This starting URL set is the entry point though which any crawler starts searching procedure. This set of starting URL is known as "Seed Page". The selection of a good seed is the most important factor in any crawling process.

2) *Frontier (Processing Queue):*

The crawling method starts with a given URL (seed), extracting links from it and adding them to an un-visited list of URLs. This list of un-visited links or URLs is known as, *"Frontier"***.** Each time, a URL is picked from the frontier by the Crawler Scheduler**.** This frontier is implemented by using Queue, Priority Queue Data structures. The maintenance of the Frontier is also a major functionality of any Crawler.

3) *Parser:*

Once a page has been fetched, we need to parse its content to extract information that will feed and possibly guide the future path of the crawler. Parsing may imply simple hyperlink/URL extraction or it may involve the more complex process of tidying up the HTML content in order to analyze the HTML tag tree. The job of any parser is to parse the fetched web page to extract list of new URLs from it and return the new un-visited URLs to the Frontier.

Common web crawler implements method composed from following steps:

1. Acquire URL of processed web document from processing queue
2. Download web document
3. Parse document's content to extract set of URL links to other resources and update processing queue
4. Store web document for further processing

Due to the enormous size of the Web, crawlers often run on multiple machines and download pages in parallel. This parallelization is often necessary in order to download a large number of pages in a reasonable amount of time. Clearly these parallel crawlers should be coordinated properly, so that different crawlers do not visit the same Web site multiple times, and the adopted crawling policy should be strictly enforced. The coordination can incur significant communication overhead, limiting the number of simultaneous crawlers.

III. RELATED WORK

A. String Matching Algorithms

The string matching algorithm is used to find the occurrences of a pattern P of length m in a large text T. Many techniques and algorithms have been designed to solve this problem [3]. These algorithms have many applications including Information retrieval, database query, and DNA and protein sequence analysis. Therefore the efficiency of string matching has great impact. Basically a string matching algorithm uses the pattern to scan the text. The size of the text taken from file is equal to length of pattern. It aligns the left ends of the pattern and text. Then it checks if the pattern occurs in the text and shifts the pattern to right. It repeats the same procedure again and again till the right end of the pattern goes to right end of the text. Two types of matching are currently in use. Exact string matching technique [4] used to find all occurrence of pattern in the text. Approximate string matching is the technique of finding approximate matches of a pattern in a text.

*B. Brute Force Algorithm [5]***.**

The most obvious approach to string matching problem is Brute-Force algorithm, which is also called "naïve algorithm". The principles of brute force string matching are quite simple. This problem involves searching for a *pattern* (substring) in a string of *text*.

The basic procedure:

1. Align the pattern at beginning of the text
2. Moving from left to right, compare each character of the pattern to the corresponding character in the text until
   - All characters are found to match (successful search); or
   - A mismatch is detected
3. While pattern is not found and the text is not yet exhausted, realign the pattern one position to the right and repeat Step 2.

The result is either the index in the text of the first occurrence of the pattern, or indices of all occurrences. This algorithm is unacceptably slow.

Brute force string matching can be very ineffective, but it can also be very handy in some cases, just like the sequential search.

C. Rabin Karp Algorithm [5]:

Michael O. Rabin and Richard M. Karp came up with the idea of hashing the pattern and to check it against a hashed sub-string from the text in 1987. In general the idea seems quite simple, the only thing is that we need a hash function that gives different hashes for different sub-strings. Such hash function, for instance, may use the ASCII codes for every character, but we must be careful for multi-lingual support.

The Rabin-Karp string searching algorithm calculates a *hash value* for the pattern, and for each M-character subsequence of text to be compared.

– If the hash values are unequal, the algorithm will calculate the hash value for next M-character sequence.

– If the hash values are equal, the algorithm will do a Brute Force comparison between the pattern and the M-character sequence.

– In this way, there is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

The hash function may vary depending on many things, so it may consist of ASCII char to number converting, but it can be also anything else. The only thing we need is to convert a string (pattern) into some hash that is faster to compare. Let's say we have the string "hello world", and let's assume that its hash is hash ('hello world') = 12345. So if hash ('he') = 1 we can say that the pattern "he" is contained in the text "hello world". Thus on every step we take from the text a sub-string with the length of m, where m is the pattern length. Thus we hash this sub-string and we can directly compare it to the hashed pattern.

D. Knuth-Morris-Pratt Algorithm [6]:

Knuth, Morris and Pratt discovered first linear time string-matching algorithm by skipping lots of useless comparisons is more effective than brute force string matching. The Knuth-Morris-Pratt (KMP) string searching algorithm differs from the brute-force algorithm by keeping track of information gained from previous comparisons.

– A failure function (*f*) is computed that indicates how much of the last comparison can be reused if it fais.

– Specifically, *f* is defined to be the longest prefix of the pattern P[0,..,j] that is also a suffix of P[1,..,j]

The only thing is to use the information gathered during the comparisons of the pattern and a possible match. To do that first we have to preprocess the pattern in order to get possible positions for next matches. Thus after we start to find a possible match in case of a mismatch we'll know exactly where we should jump in order to skip unusual comparisons. However in case of repeating character in the pattern if we have a mismatch after that character a possible match must begin from this repeating character. Finally if there is more than one repeating character in the text the "next" table will consist show their position. After we have this table of possible "next" positions we can start exploring the text for our pattern.

E. Boyer Moore Algorithm [5]:

Boyer-Moore is an algorithm that improves the performance of pattern searching into a text by considering some observations. First of all this algorithm starts comparing the pattern from the leftmost part of text and moves it to the right. Unlike other string searching algorithms though, Boyer-Moore compares the pattern against a possible match from right to left. The main idea of Boyer-Moore in order to improve the performance is some observations of the pattern. In the terminology of this algorithm they are called good-suffix and bad-character shifts.

1) *Good-suffix Shifts:*

If t is the longest suffix of P that matches T in the current position, then P can be shifted so that the previous occurrence of t in P matches T. In fact, it can be required that the character before the previous occurrence of t be different from the character before the occurrence of t as a suffix. If no such previous occurrence of t exists, then the following cases apply:

1. Find the smallest shift that matches a prefix of the pattern to a suffix of t in the text
2. If there's no such match, shift the pattern by n (the length of P)

2) *Bad Character Shifts:*

Beside the good-suffix shifts the Boyer-Moore algorithm makes use of the so called bad-character shifts. In case of a mismatch we can skip comparisons in case the character in the text doesn't happen to appear in the pattern.

F. Common Issues of Existing Algorithms

Some of the common issues identified from the existing algorithms listed below

– The case sensitiveness is not considered
– Character wise searching is performed.
– Special characters and spaces are not eliminated.
– Pre-processed text is not formatted.

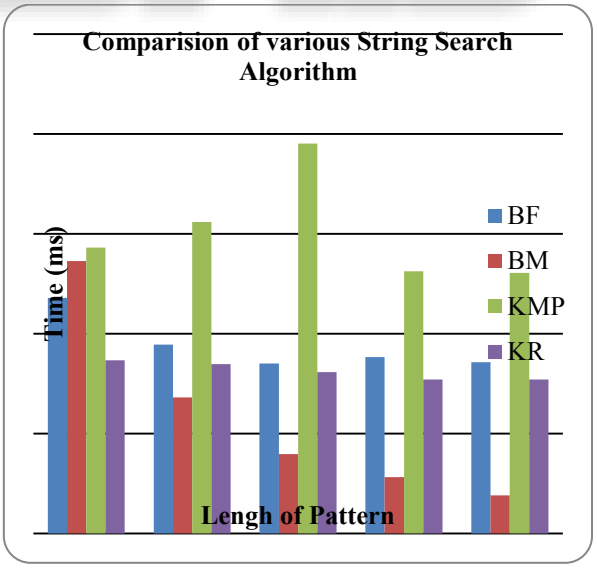

Figure 3. Bar chart showing comparison of various string search algorithms

To overcome these issues and to speed up the searching a new algorithm has to be considered which can eliminate the spaces, special characters and stop words while searching. Pre-processing the text helps to minimize the size of search data. The reduced data is then stored in a data base where the searching actually done.

IV. CONCLUSION

To facilitate fast retrieval, some of the existing string searching algorithms have been discussed and have been compared. From the chart shown above, it can be observed that Boyer Moore stands best for the searching purpose as it takes less time when the pattern length and the text length increases. But still there are some drawbacks regarding the algorithms as discussed. So to increase the speed and efficiency there needs to be better proposed mechanism which can consider some big words or data and can be implemented in future.

REFERENCES

[1] World Internet Usage and population statistics available at- << http://www.internetworldstats.com/stats.htm >>
[2] Arasu, Cho, Paepcke: "Searching the Web**",** Computer Science Department, Stanford University.
[3] Aqel & Ibrahiem.M, (2007)"Multiple skip multiple pattern matching Algorithm", IAENG international journal
[4] Pan , Lee, (2008) "An approximate String matching based on Exact String matching with constant pattern length", Department of computer science and Information Engineering, National Chi Nan university.
[5] http://www.stoimen.com/blog/2012/03/27/computer algorithms-brute-force-string-matching/
[6] Kunth, D.E, Morris J.H, Pratt V.R (1977), "Fast pattern matching in strings", Journal on computing.