TARANTULA
A Scalable and Extensible Web Spider

Anshul Saxena[1], Keshav Dubey[1], Sanjay K. Dhurandher[2] and Issac Woungang[3]
[1] *Division of Computer Engineering, Netaji Subhas Institute of Technology, University of Delhi, Delhi, India*

[2] *CAITFS, Division of Information Technology, Netaji Subhas Institute of Technology, University of Delhi, Delhi, India*

[3] *Department of Computer Science, Ryerson University, Toronto, Canada*

Keywords: Crawler, Compressed Tries, Crawling Strategies, Distributed Processing.

Abstract: Web crawlers today suffer from poor navigation techniques which reduce their scalability while crawling the World Wide Web (WWW). In this paper we present a web crawler named Tarantula that is scalable and fully configurable. The work on Tarantula project was started with the aim of making a simple, elegant and yet an efficient Web Crawler offering better crawling strategies while walking through the WWW. This paper also presents a comparison with the Heritrix (Mohr et al.) crawler. The structure of the crawler facilitates new navigation techniques which can be used with existing techniques to give improved crawl results. Tarantula has a pluggable, extensible architecture that further facilitates customization by the user.

1 INTRODUCTION

The World Wide Web (WWW) or the web can be viewed as a huge distributed database across several million of hosts over the Internet where data entities are stored as web pages on web servers. The data on the Web is varied, mostly unstructured and not catalogued and their logical relationships are represented by hyperlinks. According to Netcraft survey in April 2009, the number of hostnames has increased by ten times to 232,000,000 than what it was in 1995. On the first look, implementation of a web crawling system appears to be trivial. However, due to the enormous size of the web, its high rate of expansion, its variedness and non uniformity, making a web crawler capable of following links and downloading web pages as it moves from one website to another is a complex task.

A good crawler system can be judged on the basis of two important parameters. The first parameter is the crawling strategy used by the crawler. While there are many existing crawling strategies (Hafri and Djeraba, 2004) each one with its merits and demerits, choice of crawling strategy used is a key factor in deciding the scalability of the web crawler. The other important parameter is the performance of the crawler within the allotted resources. Limitations of primary and secondary memory and network bandwidth are the key bottlenecks in the performance of a crawler. Also, the size of the Internet is in hundreds of Terabytes. Hence, it is imperative to have a design that extends to handle these factors effectively. This paper focuses on the design and architectural details of Tarantula that incorporates all the above mentioned features.

The remaining paper is organized as follows. In section 2, we discuss about the previous work done in this area. Our motivation for this project is presented in section 3. Section 4 describes briefly the architecture of Tarantula. Section 5 explains the algorithms and the hashing techniques used in the various modules. We present our results in section 6 and then finally conclude this work in section 7 and give the future work that can be carried out in this dimension in section 8.

2 RELATED WORK

When the Internet first came into existence, there were very few web pages on the web hence a web crawler was not necessary at that time. But with the internet revolution in the last decade, the number of

Saxena A., Dubey K., Dhurandher S. and Woungang I. (2009).
TARANTULA - A Scalable and Extensible Web Spider.
In *Proceedings of the International Conference on Knowledge Management and Information Sharing*, pages 167-172
DOI: 10.5220/0002302001670172

web pages online grew exponentially, and thus the need for a search engine to index these pages became indispensable. Web spiders were used by these search engines to scale up the web and hence they became popular. Since then, the web spiders have been crawling the web on a regular basis.

A popular web crawler is the UbiCrawler (Boldi et al., 2004). It is made up of several web agents that scan their own share of the web by autonomously coordinating their behaviour. Each agent performs its task by running several threads, each dedicated to scan a single host using a Breadth-First visit thus ensuring that politeness is maintained as different threads visit different hosts at the same time. However, breadth-first navigation technique being a top-down approach results in web crawler missing out on possible detection of new URLs that can be obtained from already discovered URLs. For example, given a URL say www.example.com/pics/page1.html, there might be possible existence of URLs like www.example.com/pics/ and www.example.com/pics/page2.html, etc. Discovering such URLs from the existing URLs increases the scalability of the web crawler.

Heritrix is the web crawler developed by Internet Archive's, an open-source corporation. Heritrix provides a number of storing and scheduling strategies to crawl the seed list. Each of its crawler process can be assigned up to 64 sites to crawl, and it is ensured that no site is assigned to more than one crawler. The crawler process reads a list of seed URLs for its assigned sites from disk into the queues, and then uses asynchronous I/O to fetch pages from these queues in parallel. After the page is downloaded, the crawler performs link extraction on it and if a link refers to the site of the page it was contained in, it is added to the appropriate site queue; otherwise it is logged to disk. Periodically, a batch process performs merging of these logged “cross-site” URLs into the site-specific seed sets and thus filtering out duplicates in the process and the process is repeated till exhaustion of URL in the list.

KSpider (Koht-arsa and Sanguanpong, 2002), is a scalable, cluster based web spider. It uses a URL compression scheme that stores the URLs in a balanced AVL tree. The compressed URLs are stored in memory rather than on hard disk because storing the URLs in memory improves the performance of the crawler. Common prefix among URLs is used to reduce the size of the URLs by storing the common prefixes once and reusing them for many URLs. However, the structure of AVL restricts the number of children to two and increases the height of the tree even though it is balanced.

Tarantula capitalizes over KSpider by making optimal use of the common prefixes among URLs by using a slight modified version of compressed tries. These data structure are broad and therefore can have more than two children thereby decreasing the height of the tree. Also, unlike KSpider, Tarantula stores all the URLs belonging to the same host in the same compressed trie. This is useful in restricting the depth of crawling a hostname and also provides easy mechanism for ensuring politeness of the crawler system. Applying compression algorithms to URLs and then expanding them again leads to a lot of CPU usage and time expenditure. It is therefore advantageous to compress the URLs based on common prefixes. Though the compression is not as high, the speed of the crawler is greatly enhanced.

3 MOTIVATION

One of the initial motivations for this work was to develop a crawling system which is able to scale a greater degree of the web. The crawling system should be able to process a large number of URLs from far and wide thus trying to cover the entire breadth of the Internet. This prompted us to come up with unique crawling strategies, which when combined with the page ranking and refreshing crawling schemes, gives excellent results.

We also wanted that the design used data structures that reduce the amount of I/O needed and CPU processing performed to pursue the newly extracted URLs for downloading the web pages.

By looking at some URLs, it is possible to detect the existence of newer URLs that might be valid but have not yet been discovered by the crawler possibly because of broken web links. Therefore, mining on the URLs discovered was yet another motivation behind development of Tarantula web crawler.

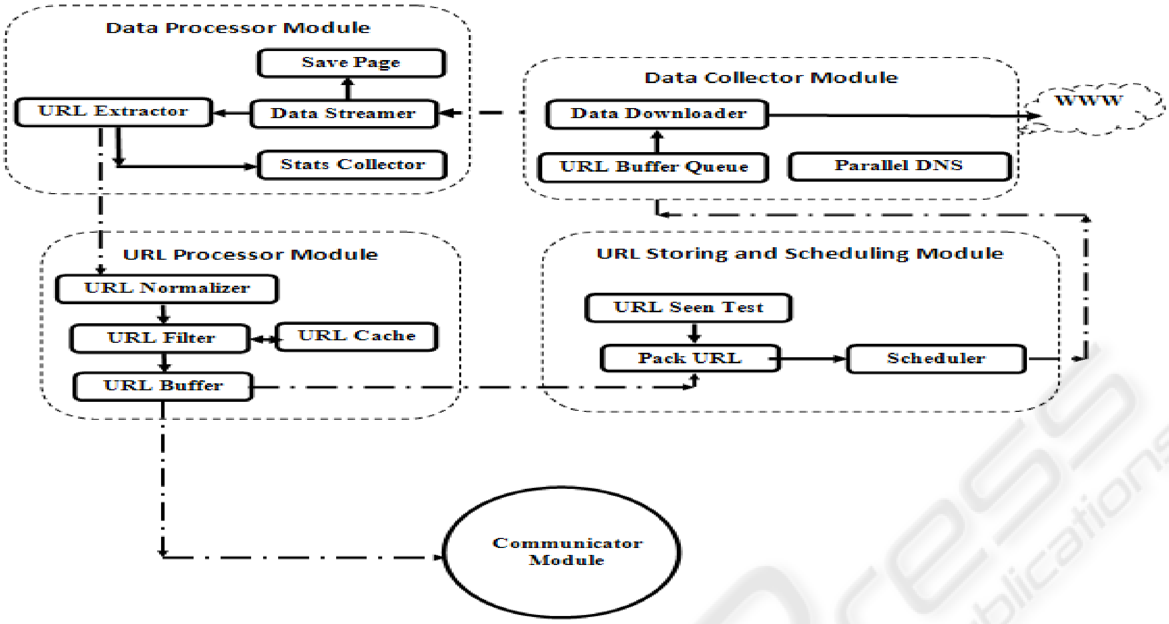

Figure 1: Tarantula Architecture.

4 ARCHITECTURE

The Figure 1 depicts the architecture of the proposed Tarantula web crawler. The *URL Storage and scheduler* module sends a list of URLs to the *Data Collector* module for downloading the corresponding web pages. It consists of *URL Buffer Queue* which stores a queue of URLs that are to be fetched by the crawler. One by one the *Data Downloader* sub module fetches the web pages from the Internet and sends the downloaded data to the *Data Processing* module. During the process of downloading the web page, the crawler system maintains a DNS repository as well as a DNS cache to speed up the download rate of the crawler by avoiding latency caused by redundant DNS resolving queries.

The web page downloaded by the *Data Collector* module is sent to the *Data Processing* module. Here, the web page is stored as a data stream variable and is processed by the *URL Extractor* to retrieve fresh web links from the web page and the *Stats Collector* sub module processes it to collect statistical data about the web page. Tarantula can be configured to collect different types of statistical data from the web pages. Once processed, the web page is handed over to a free “save thread” from the thread pool that store the web page at an appropriate location on the hard disk. The list of freshly extracted URLs is then sent to the *URL Processing* module.

The *URL Processing* module firstly consists of a *URL Normalizer* sub module which converts the URL into their canonical form. This is necessary to avoid different URLs pointing to the same web page from being scheduled for downloading. While there is no universally accepted canonical form, depending upon the focus of the crawler system, appropriate normalization techniques can be applied. The *URL Filter* reads the output from the *URL Normalizer* and filters out unwanted or already downloaded URLs. Here again different filters based on keywords, file type, etc. can be used depending upon the requirement. The filtered and normalized URLs are now ready to be scheduled for downloading. The URLs to be downloaded by another crawler thread or by a crawler thread on another terminal is decided by the *Communicator* module. This module is responsible for unbiased distribution of URLs to every crawler thread on every terminal. Using a hash technique, the URL list is distributed amongst crawler threads running on different terminals on a high speed crawler LAN. Those URLs that are to be scheduled for downloading by the current crawler thread is then sent to the *URL Storage and Scheduling* module.

The *URL Storing and Scheduling* module consists of a *URL Seen Test* sub module which checks if the URL has been fetched by the crawler system or not. Those URLs that have been fetched in the past and need no refreshment are removed while the rest are forwarded to the URL Packing module where the URLs are stored in memory as compressed tries (Maly, 1976). Compressed tries apart from reducing the size of the URL offer other advantages as discussed later. From the compressed tries, the Scheduler sub module picks out the URLs in the order in which they are to be downloaded by the Data collector module.

This entire cycle is repeated by every crawler thread on every terminal on the high speed LAN.

5 ALGORITHM

The algorithm conceived for the Tarantula makes it the most unique feature of the entire implementation. This spans over the new scheduling methodologies and provides an innovative URL compression technique.

5.1 Data Structure

The URLs filtered out from the URL Seen Test are packed into a data structure of the type compressed tries. The URLs are hashed on their host name as per two equations: *equation 1* and *equation 2* as shown below that distribute the URLs between different crawler terminals, and between the different threads of a terminal into corresponding compressed trie in which the URL is to be packed in.

hash(URL) = (Σ Sum of ASCII values of even characters of hostname) % Total number of Threads per Crawler System (1)

hash(URL) = (Σ Sum of ASCII values of odd characters of hostname) % Total number of Threads per Crawler System (2)

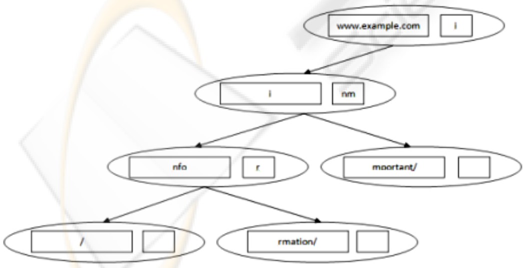

Figure 2: Compressed Trie structure.

The compressed trie node as shown in Figure 2 consists of a variable storing a portion of the URL and a list of characters storing the first character of the portion of the URL represented by every child node. The order of the characters in the list decides the order of find of the URLs by the crawler thread.

Since the trie stores the URLs based on their common parts, it offers high compressions for big websites having long URLs with large common parts. This data structure has the advantage that URLs of the same host names are stored in the same sub trie. Hence only one node (may be more) stores the host name while all the URLs with the same host name share the sub trie and as we go down the sub trie, the URL represented by the current node is used as prefix by the child node to represent a new URL.

The crawler follows a bottom up approach by starting with the URL represented by the leaf node and moves upwards to retrieve new URLs. It is also possible for the crawler to retrieve URLs that have not yet been seen by the crawler but are implicitly understood. To see how, consider a sample trie as shown in Figure 3.

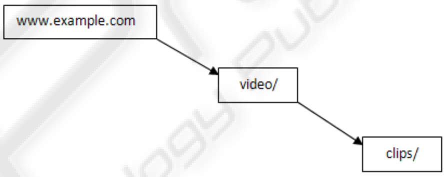

Figure 3: Retrieval of new URLs.

If the crawler got a URL www.example.com/video/clips/, it is stored in the compressed tries as shown in Figure 3. The crawler follows a bottom-up approach to retrieve the URLs from the compressed tries. In this case, it starts with the leaf node "clips/" and identifies the URL www.example/com/video/clips/. After scheduling this URL, it moves on to a node at a higher level "video/" representing the URL www.example.com/video/. Finally, we get www.example.com/ as the last URL. Hence from one URL, the crawler is implicitly able to find out two new URLs which it has not seen before. Also, if the crawler had discovered a new URL www.example.com/pics1/Page1.html, then there might be a possibility of existence of other URLs such as www.example.com/pics1/Page2.html or www.exampe.com/pics2/Page1.html. The bottom-up approach imposed by the above data structure makes it possible to mine the discovered URLs and help discovering possible new URLs.

5.2 Scheduling Techniques

Two custom scheduling techniques have been devised which are used in conjuncture with the URL prioritization and ranking schemes for better crawl results. These have been discussed below.

5.2.1 Top Sliding Window Technique

Consider a compress trie consisting of URLs of *n* different websites. The scheduler will maintain a sliding window of size less than or equal to *n*. By top sliding window we mean that the sliding window will contain elements pointing to the left most leaf node of all the child nodes of the compressed trie which have node values as the hostnames of different websites as shown in Figure 4.This will guarantee that the URLs pointed to by elements in the sliding window belong to different web servers. Each compressed trie will have a different sized sliding window and a URL in any sliding window will have different host name from any other URL in any sliding window. Thus the politeness of the crawler is maintained.

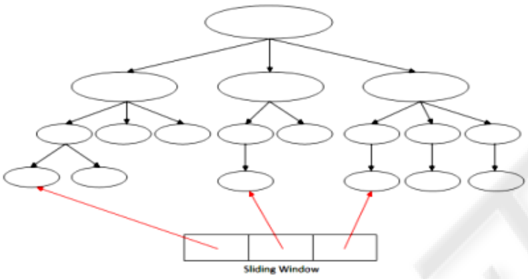

Figure 4: Top Sliding Window Scheduling.

For simplicity, let us assume that level 1 nodes of a compressed trie consists of www.example1.com, www.example2.com, www. example3.com, and www.example4.com as shown in Figure 5. Let us also assume that all these URLs point to different web servers and the crawler maintains a sliding window of size 2. The compiler picks up the left most leaf of the www.example1.com sub trie and the www.example2.com sub trie in the first crawl (represented as dotted border rectangle). In the second crawl, the sliding window moves to the right and contain URLs represented by the left most leaf of www.example3.com and www.example4.com (represented as single border rectangle).

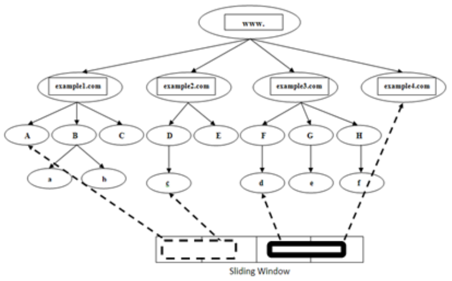

Figure 5: Top Sliding Window Scheduling Example.

5.2.2 Leaf Sliding Window Technique

In this technique, we maintain a sliding window containing pointers to all the leaf nodes of a trie as shown in Figure 6. The size of the window is fixed for all the compressed tries and it moves to and fro from the left most leaf node to the right most leaf node of the trie. The sliding windows are then arranged in a column fashion to form a table of URLs. Row wise scheduling of the URLs in the table so formed is done by the URL Scheduler.

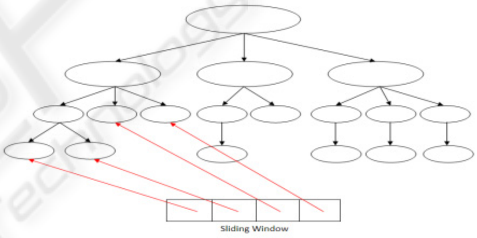

Figure 6: Leaf Sliding Window Scheduling.

In both the above crawling strategies, the number of elements must be large enough to have sufficient interval between the URLs with the same hostname.

6 EXPERIMENTAL RESULTS

The proposed web spider Tarantula is written entirely in C#.net. The operating system of the crawler machine was Windows XP Service Pack 2 (SP2). Only one machine was used. Its processor was the AMD Turion™ 64 X2 TL-60 2.00GHz and had a RAM of 1GB with 320GB SATA hard disk.
The Tarantula was run for four hours on a shared internet connection of 2Mbps. The Heritrix crawler with which the comparison was performed is Internet Archive's open source web crawler project. The number of threads used in Tarantula and Heritrix were 10. Heritrix was run under similar test condition and the results have been compiled as

follows:

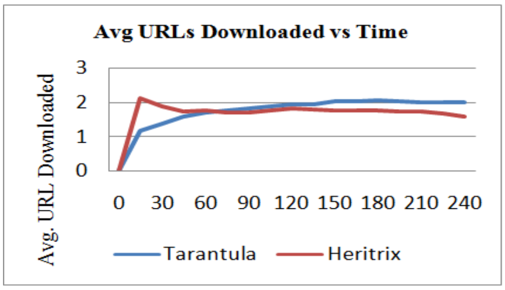

Figure 7: Average Download Rate vs. Time.

Tarantula initially started slowly but picked up and maintained a speed of about 2 documents per second (doc/sec), while the Heritrix had an initial spike of 2.1 doc/sec but slowed down to nearly 1.5 doc/sec. The Figure 7 shows the plot of average download rate versus the time for Tarantula as well as Heritrix.

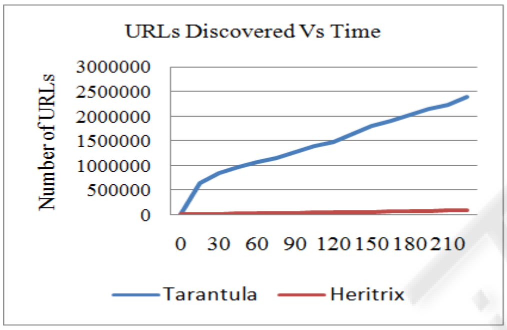

Figure 8: URLs Discovered vs. Time.

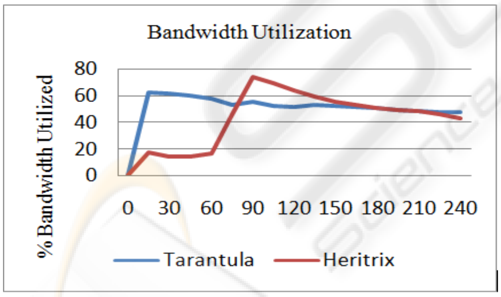

Figure 9: Bandwidth Utilization vs. Time.

The Figure 8 shows the number of URLs discovered with time. From the graph, it is clear that the mining of the URLs and the fast crawling rate have resulted in exceptionally good results for the proposed Tarantula crawler than the Heritrix crawler.

The Figure 9 shows the percentage bandwidth utilization of Tarantula and Heritrix with time. With high and low curves, the utilization of the bandwidth by both the crawlers is nearly same.

7 CONCLUSIONS

In this paper, we have described the architecture and implementation details of Tarantula web crawler, and have presented some preliminary experimental results for the same. We have been successful in building such a system using efficient data structures and scheduling algorithms. We have also used a unique and particularly useful crawling strategy that helps our crawler to make its requests politely without compromising on the speed at which the web pages are downloaded. Based upon the crawling results, our crawling system proved to be much more scalable and faster than the Heritrix web crawler.

8 FUTURE WORKS

Most of the web is hidden behind forms. To retrieve these pages, the web crawler should be equipped to handle deep web. Another way to handle this problem is to divide the web into different categories and design focused crawlers for each of these categories. Tarantula is still a broad crawler which does not perform either deep web crawling or focused crawling. This technique may be considered as a future enhancement to this work.

REFERENCES

Djeraba, C., Hafri, Y., 2004. Dominos: a New Web Crawler's Design. In *ECDL'04, 8th European Conference on Research and Advanced Technologies for Digital Libraries*. Springer Press.

Mohr, G., Stack, M., Ranitovic, I., Avery, D. and Kimpton, M., 2004. An Introduction to Heritrix An open source archival quality web crawler. In *IWAW'04, 4th International Web Archiving Workshop.* Springer Press.

Boldi, P., Codenotti, B., Santini, M., Vigna, S., 2004. UbiCrawler: A Scalable Fully Distributed Web Crawler. In *8th Australian World Wide Web Conference.* John Wiley & Sons Publications.

Koht-arsa, K., Sanguanpong, S., 2002. High Performance Large Scale Web Spider Architecture. In *International Symposium on Communications and Information Technology.* ANREG Publication.

Maly, K., 1976. Compressed Trie. ACM Publications.