# PARALLELIZATION OF WEB CRAWLER WITH MULTITHREADING AND NATURAL LANGUAGE PROCESSING

Ippili Akarsh[1], Ravi Maithrey Regulagedda[2]

[1-2]*School of Computer Science and Engineering, Vellore Institute of Technology*

---------------------------------------------------------------------***---------------------------------------------------------------------

**Abstract -** *A web crawler, as its name suggests, crawls the web, looking for information it is tasked with searching for. While it is trivial to create a simple crawler, the main challenge lies in making one which offers improvements in both crawling speed and result accuracy, rather than a tradeoff between the two. In this paper, we present a model architecture for such a crawler, leveraging the use of multi-threading parallelization techniques and natural language processing to achieve optimum performance. At the end, we present our results on a sample working of the same proposed model.*

***Keywords:*** **Web Crawler, Multithreading, Priority Queue, Cosine similarity, Tf-idf**

## 1. INTRODUCTION

The Internet is a huge storehouse of data. While this is a great thing, its main issue lies in its size. When one requires to get any information from the internet, we turn to search engines. These search engines in turn depend on web crawlers which run in the background constantly updating their databases in order to fetch the best results.

At its most basic a web crawler is a program which visits webpages, and then visits the webpages which are linked in the first page and so on either until a set number of pages are crawled and the information retrieved, or a certain query is matched with the information contained in the webpage. To do this, they must download the data from the webpage and scrape it, which presents certain challenges.

A web crawler should be able to visit the maximum number of pages, in minimum time and also provide results which are relevant to the topic being queried for. To tackle this, a web crawler which runs in parallel, via multi-threading and with each crawling process on a different logical core can be developed. To further our efforts, we can also use natural language processing in order to ensure that the results are relevant to the query being processed. This will enable us to have a crawler which is both optimal in performance and accurate in results.

### 1.1 Proposed Work

The web crawler has to perform its crawling once a query is passed to the search engine. A root page is present either by default or provided by the user. From this root page, the crawling for the query goes ahead. Each link and webpage visited further on has to be then searched for the query in turn until a positive match is made. This positive match itself can be evaluated in many ways, one of which we aim to propose in this paper.

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 a priority to the new URLs discovered during the crawling process, as the same URL can be found by two different crawling processes. We propose a crawling architecture which aims to do this via parallelization using multi-threading and improving search using natural processing.

## 2. LITERATURE REVIEW

The existing web crawlers use data structures to hold frontier sets in local address space. This space could be used to run more crawler threads for faster operation. All crawler threads fetch the URL to crawl from the centralized frontier. A proposed method [1] is to use a mutex principle to achieve a somewhat parallel-type execution. The approach to utilize the waiting time on mutual exclusion lock in an efficient manner has been discussed in detail.

The next step is to look at a three-phase crawler [2]. The first site locating stage finds the most relevant site for a given topic, the second in-site exploring stage uncovers searchable forms from the site and then the third stage applies naïve base classification to rank the result. This would result in a more efficient crawler.

The paper by Vandana Shrivastava [3] gives us additional info about web crawler structure and best practices when working with them, including but not limited to reverse searching websites and avoiding visiting useless pages for a more accurate result.

Next, we take a look at a modification of the three-phase parser to mine in Deep Web Interfaces. [4] This type of crawler, dubbed SmartCrawler v2 is a focused crawler consisting of two stages: efficient site locating and balanced in-site exploring. SmartCrawlerV2 performs site-based locating by reversely searching the known deep web sites for center pages, which can effectively find many data sources for sparse domains. By ranking collected sites and by focusing the crawling on a topic, SmartCrawlerV2 achieves more accurate results.

Since the focus of this paper is to ultimately design a parallel crawler, a look at [5] tells us of the common problems and challenges faced in the development of them. There are a few common design issues with parallel crawlers. They are scalability, network load dispersion, and network load reduction.

As the size of the Web grows, it becomes more difficult to retrieve the whole or a significant portion of the Web using a single process [6]. We then look a method to run multiple processes to cover more of the web to make our work more efficient and to generate better results.

We then procced to look at some common considerations to keep in mind when developing a web crawler [7]. The main things to be aware of are Web Archiving, Vertical Search Engines, Web Data Mining, Web Monitoring, Detection of malicious websites, Web site/application testing, Copyright violation detection, and detection of illegal activities.

Multi-threaded web crawler [1] [8] is an efficient way to crawl the web in this era of big data. It can improve the utilization rate of the CPU. Different from single-thread, multi-thread can run other threads in parallel while waiting for a process to improve the utilization efficiency of CPU. It can improve the utilization of network bandwidth. When multiple threads are processing at the same time, the network resources will be maximized.

We proceed to look at a few ways to optimize crawlers [9] by focusing on a specific domain. 1) Full Distribution: DSCrawler (Domain Specific Crawler) is distributed over multiple crawling machines (each for a specific domain) for better performance. Crawling machines download web pages independently without communication between them. 2) Scalability: Due to the fully distributed architecture of DSCrawler, its performance can be scaled by adding extra machines depending on the bases of the increase of domains, thus manage to handle the rapidly growing Web. 3) Load Balancing: The URLs to be crawled are distributed by the URL Distributor to the particular Domain Specific Queues, thus distributing the crawling to different crawlers which leads the balancing of the crawling load. 4) Reliability: Multiple, independently working crawlers increases the reliability of the whole system, as failure of the single crawl worker will not affect the function remaining crawl workers.

We then look at RCrawler [10], which is specific web crawler library in the R programming language. This is a multithreaded, flexible, and powerful web crawler that provides a suite of useful functions for web crawling and web scraping.

This paper [11] talks about deploying a web crawler in a client-server model for an increase in crawling performance. This other paper [12] also talks about improving crawler performance but in a manner as to improve the data collection aspect of the crawler. In order to achieve this, they propose a distributed focused crawler which has crawler scheduling and site ordering.

In order to scale the crawler as the size of the data it is crawling over increases, we need a scalable platform. This paper [13] proposes that crawlers be deployed in a cloud platform to provide the easy scalability required.

Now search engines do not depend on a single but on multiple web crawlers that run in parallel to complete the target. While functioning in parallel, crawlers still face many challenging difficulties such as overlapping, quality and network.[14] In distributed crawling, multiple processes are used to crawl and download pages from the Web.

The paper by Mangala et. al. [15] tells us about how one can design a crawler to give good response to the query, how high quality data can be retrieved and how to perform hidden content mining.

## 3. BACKGROUND

A parallel crawler gives optimal results when we need to search data in the internet today. Due to the vast amount data being present, a crawler which runs only a single process can no longer offer any meaningful results in proper times. Coupled with the fact that crawlers have the added task of delivering accurate results and not just random walks, running these in parallel is the safest bet moving forward. Therefore, the architecture being presented in this paper uses a parallel, multi-threaded model.

The best crawlers also have highly optimized architectures to give the best performance. The basic architecture of a crawler is shown below.

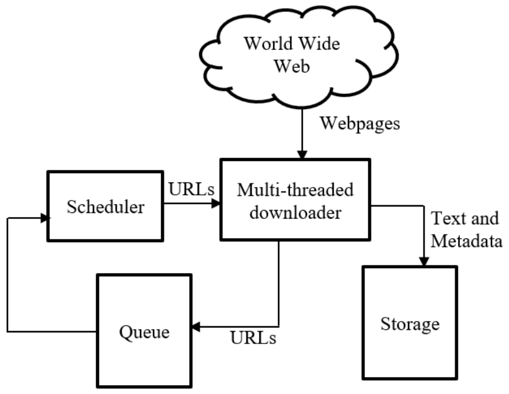

**Fig -1**. Architecture of a basic parallel crawler

In general, it is easy to build a slow crawler. The main challenge is to build a high-performance system which is both robust and efficient. In order to tackle that problem, we propose an architecture as described below

## 4. PROPOSED SYSTEM

We propose an architecture in this paper which utilizes the multithreaded web crawler in a way that behaves as a group of processes together working under the same cluster. This architecture is able to achieve remarkable fetching and parsing speeds in comparison to its sequential counterparts where all these operations are likely to happen in a form of sequence in which individual components perform their parts and pass the information to the next component.

The architecture being proposed can be described as follows, with the following steps in the crawling procedure to increase performance and to take advantage of multi-threading.

1.The user is responsible for giving a target URL to the application which is then communicated to the different segments of the program at the same time.

2. One of the communications happens to indicate the crawler regarding the type of the content and the related information regarding what to fetch for the given URL and the other communication happens for the query parser which is responsible for analysing the exact information from the URL given by the user.

3. Based on the related information first by the crawler with the help of the multiple seed pages which is again a multithreaded process in order to speed up the function and the parts of the information available due to the effort of Query parser multiple pages are downloaded at the same time with the redundancy checking in place.

4.Several workers are then deployed in order to pass the information in different websites and hyperlinks provided in them to get the suitable content. This can be easily achieved via multi-threading in order to maximise performance by running one worker in one logical core.

5. The information is then passed for the storage into a storage bucket which is a central repository for all the information passed during the process

6.The user then is responsible for getting the information from the storage bucket where the information is stored and several queries can be performed on the information to get the suitable data according to the requirement.

The architecture proposed in the section above is shown below.

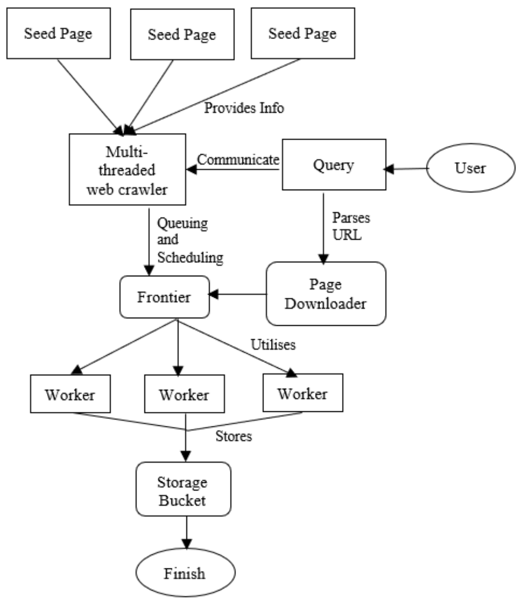

**Fig -2**. Architecture of the proposed parallel multi-threaded crawler.

### 4.1 Structure of the multi-threaded crawler

As can be seen from the architecture above, the main part of this system is the crawler itself. The crawler is structured as follows –

1. A root URL is given by the user along with their query.
2. This URL is scraped for links contained within it.
3. The text of the webpages in each link is extracted.
4. Each text is converted into its TF-IDF [16] features.
5. Using these TF-IDF features of the webpages and the original query, cosine similarity of each is calculated.
6. The cosine similarity value is used as a priority to order each link in a priority queue, thus achieving a best-first search model.

As described in J. Ramos [16], the TF-IDF features for the word t in the document d from the document set D is calculated as follows -

$$tf\ idf(t,d,D) = tf(t,d).idf(t,D)$$

where,

$$tf(t,d) = log\big(1 + freq(t,d)\big)$$
$$idf(t,D) = log\left(\frac{N}{count(d \in D: t \in d)}\right)$$

Once we have our TF IDF score of the query and the document as a whole, we perform cosine similarity calculation on it as shown.

$$cosine\ sim(Query, doc) = \frac{tf - idf_{Query} \cdot tf - idf_{doc}}{|tf - idf_{Query}| * |tf - idf_{doc}|}$$

This cosine similarity index helps us identify how close the document/webpage is to the original query put forward by the user. Therefore, when the results of the query are presented to the user, they are done so in a priority queue, where the priority is the cosine similarity index of each webpage/document. Thus, we are also able to achieve a best first search.

## 5. RESULTS

For the purposes of testing this proposed architecture, we deployed this in code written in python, though any programming language can be used to do the same. The architecture was deployed in a physical machine with the following specifications.

**Hardware Specification**
CPU: Intel Core i5-10210U
Sockets: 1
Cores: 4
Logical Processors: 8
Base Speed: 2.10 GHz RAM
Memory: 8 GB
RAM: SODIMM 2667 MHz

**Software Environment**
OS: Ubuntu 20.04.02 LTS
OS Type: 64-bit
Kernel: Linux
Kernel version: 5.8.0-53-generic

The results are collated in the form of a table showing the mean execution times in ms, showing the number of threads assigned to each worker, and the number of pages set as the limit to scrape.

**Table -1.** Comparison of execution times

| **workers/ work** | **1** | **2** | **4** | **8** | **16** | **32** | **64** |
|---|---|---|---|---|---|---|---|
| **100** | **2.18** | **2.06** | **1.71** | **1.69** | **1.5** | **2.01** | **2.02** |
| **500** | **14.62** | **6.5** | **5.49** | **4.58** | **5.49** | **3.71** | **4.63** |
| **1000** | **27.3** | **15.7** | **7.49** | **5.98** | **7.710** | **8.84** | **12.1** |
| **2500** | **50.62** | **32.5** | **20.9** | **16.0** | **16.3** | **19.32** | **28.7** |

A visualization of the results is shown

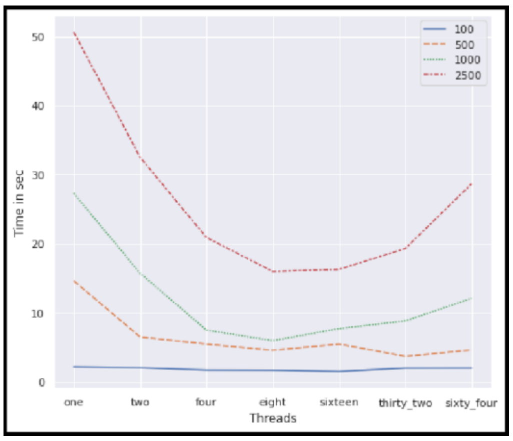

**Fig -3.** Performance of different execution times under different workloads

## 6. CONCLUSION

As seen from the results above, the crawler was able to crawl several thousand websites and give out the data with just 8 threads. In a machine better suited for scraping, with a higher number of logical processors, we would achieve an even higher performance. This type of crawler architecture overcomes the need for waiting for each page to load before moving on the next one, as we use a type of best first search, which guarantees the most appropriate results first.

## REFERENCES

[1] Kartik Kumar Perisetla, "Mutual Exclusion Principle for Multithreaded Web Crawlers" *International Journal of Advanced Computer Science and Applications* (IJACSA), 3(9), 2012.
[2] Zade, Pranali, and S. W. Mohod. "An Efficient Method for Deep Web Crawler based on Accuracy." *International Journal on Future Revolution in Computer Science & Communication Engineering* 4.4 (2018): 393-399.
[3] Shrivastava, Vandana. "A methodical study of web crawler." Vandana Shrivastava, *Journal of Engineering Research and Application* 8.11 (2018): 01-08.
[4] Pooja, Dr Gundeep Tanwar. "Smart Three Phase Crawler for Mining Deep Web Interfaces." *International Journal on Future Revolution in Computer Science & Communication Engineering* 4.4 (2018): 853-858.
[5] Komal, Dr. Ashutosh Dixit, "Design Issues in Web Crawlers and Review of Parallel Crawlers", *International Journal of Science and Research* (IJSR), Volume 5 Issue 6, June 2016, 61 – 64
[6] Amudha, S., and M. Phil. "Web crawler for mining web data." *International Research Journal of Engineering and Technology* 3 (2017): 128-136.
[7] Chatterjee, Soumick, and Asoke Nath. "Auto-Explore the Web–Web Crawler." *International Journal of Innovative Research in Computer and Communication Engineering* 5.4 (2017).
[8] Sun, Guang, Huanxin Xiang, and Shuanghu Li. "On multi-thread crawler optimization for scalable text

searching." *Journal on Big Data* 1.2 (2019): 89.
[9] Tyagi, Nidhi, and Deepti Gupta. "A novel architecture for domain specific parallel crawler." *Department of Computer Engineering, Shobhit University. Meerut, India* (2010).
[10] Khalil, Salim, and Mohamed Fakir. "RCrawler: An R package for parallel web crawling and scraping." *SoftwareX* 6 (2017): 98-106.
[11] Kausar, Md Abu, V. S. Dhaka, and Sanjeev Kumar Singh. "Design of web crawler for the client-server technology." *Indian Journal of Science and Technology* 8.36 (2015): 1-7.
[12] Gunawan, Dani, Amalia Amalia, and Atras Najwan. "Improving data collection on article clustering by using distributed focused crawler." *Data Science: Journal of Computing and Applied Informatics* 1.1 (2017): 1-12.
[13] ElAraby, M. E., et al. "Elastic Web crawler service-oriented architecture over cloud computing." *Arabian Journal for Science and Engineering* 43.12 (2018): 8111-8126.
[14] Kausar, Md Abu, V. S. Dhaka, and Sanjeev Kumar Singh. "Web crawler: a review." *International Journal of Computer Applications* 63.2 (2013).
[15] Mangla, Sweety, and Geetanjali Gandhi. "Study of Crawlers and Indexing Techniques in Hidden Web." *International Journal of Computer Science and Mobile Computing* 4.4 (2015): 598-606.
[16] Ramos, Juan. "Using tf-idf to determine word relevance in document queries." *Proceedings of the first instructional conference on machine learning*. Vol. 242. No. 1. 2003.