A PROJECT REPORT

ON

WEB CRAWLING USING THREADS

Operating System and System Programming Lab

(15B17CI472)

Submitted by:

Kartik Valecha (9920103090)

Submitted to :

Dr. Neeraj Jain

Department of CSE/IT

Jaypee Institute of Information Technology University, Noida

December, 2022

Contents

List of Figures

Chapter 1: Problem Statement

1.1 Introduction

Web crawling is the process of indexing data on web pages by using a program or automated script. These automated scripts or programs are known by multiple names, including web crawler, spider, spider bot, and often shortened to crawler.

Web crawlers copy pages for processing by a search engine, which indexes the downloaded pages so that users can search more efficiently. The goal of a crawler is to learn what webpages are about. This enables users to retrieve any information on one or more pages when it's needed.

Web Crawler is a bot that downloads the content from the internet and indexes it. The main purpose of this bot is to learn about the different web pages on the internet. This kind of bots is mostly operated by search engines. By applying the search algorithms to the data collected by the web crawlers, search engines can provide the relevant links as a response for the request requested by the user. In this article, let's discuss how the web crawler is implemented.

1.2 Motivation

Web crawler when done sequentially is very slow and requires a lot of bandwidth. Since we have learnt threads and multiprocessing we thought that implementing these techniques would help improve web crawling in the following ways:

- Reduce the total time
- Increase the number of sites to crawl in a given amount of time
- Reduce the total bandwidth of the user

Chapter 2: Objective and Future Scope

2.1 Objective

To compare sequential web crawlers and web crawlers using threads to find out if using threads in web crawlers can help save time.
Our main objective is

- Show the implementation of Web Crawling
- To reduce the total time using threading in web crawlers
- To increase the number of sites that can be visited by the crawler in a given amount of time

2.2 Future Scope

This project can be used to find data/items from all the pages of a website saving a lot of time. This project also has various applications in search engine optimization.
Search engine optimization (SEO) is the process of improving a website to increase its visibility when people search for products or services. If a website has errors that make it difficult to crawl, or it can't be crawled, its search engine results page (SERP) rankings will be lower or it won't show up in organic search results. This is why it's important to ensure webpages don't have broken links or other errors and to allow web crawler bots to access websites and not block them.
Likewise, pages that aren't crawled regularly won't reflect any updated changes that may otherwise increase SEO. Regular crawling and ensuring that pages are updated can help improve SEO, especially for time-sensitive content.

Chapter 3: Detailed description of the project

3.1 Methodology

- **get_html()** - This function returns the html of the given url.
- **extract_link()** - This takes a beautiful soup object as a parameter and returns url of the other pages linked to this page.
- **extract_content()** - This takes a beautiful soup object as a parameter and appends product id product name and price into data.
- **crawl()** - This function is the main backbone of the code which controls the flow of crawling either through threads or sequential.

3.2 Brief description of algorithm

Firstly, we initialize a queue that stores all the unvisited links. This algorithm uses 5 threads that work simultaneously to crawl the websites. The crawler starts. A thread get the html of the first URL using get_html which is then converted to a beautiful soup onject using html.parser. Then links our extracted information from the beautiful soup object. These links are then appended in the queue which are then further analysed for data

- Product id
- Product Name
- Product Price

Chapter 4: Code

4.1 Code

Chapter 5: Result

5.1 Output

SEQUENTIAL CRAWLER

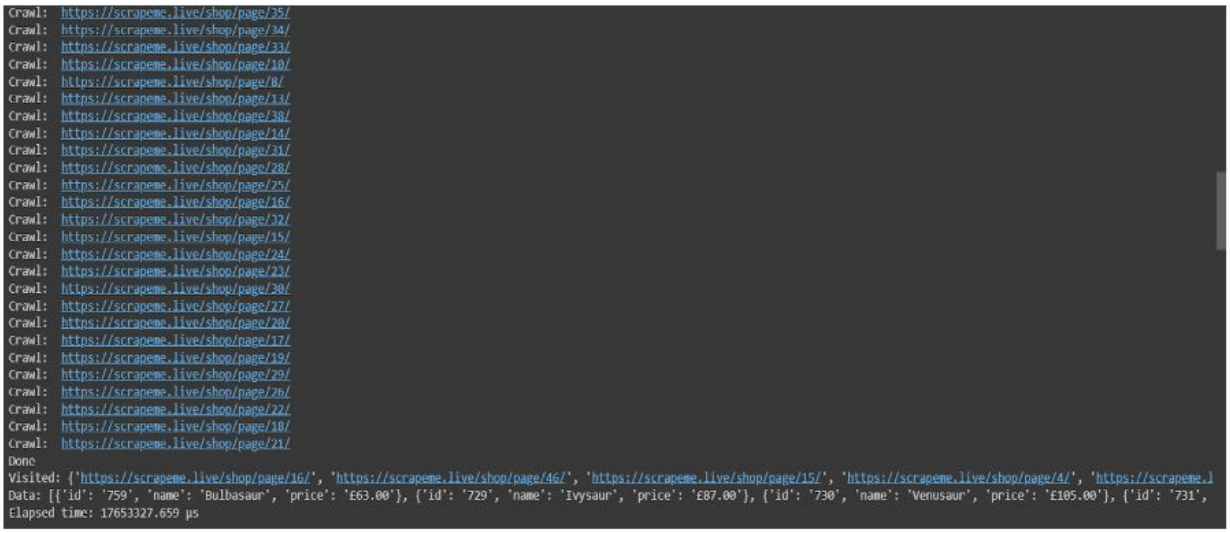

Figure 5.1: SEQUENTIAL CRAWLER

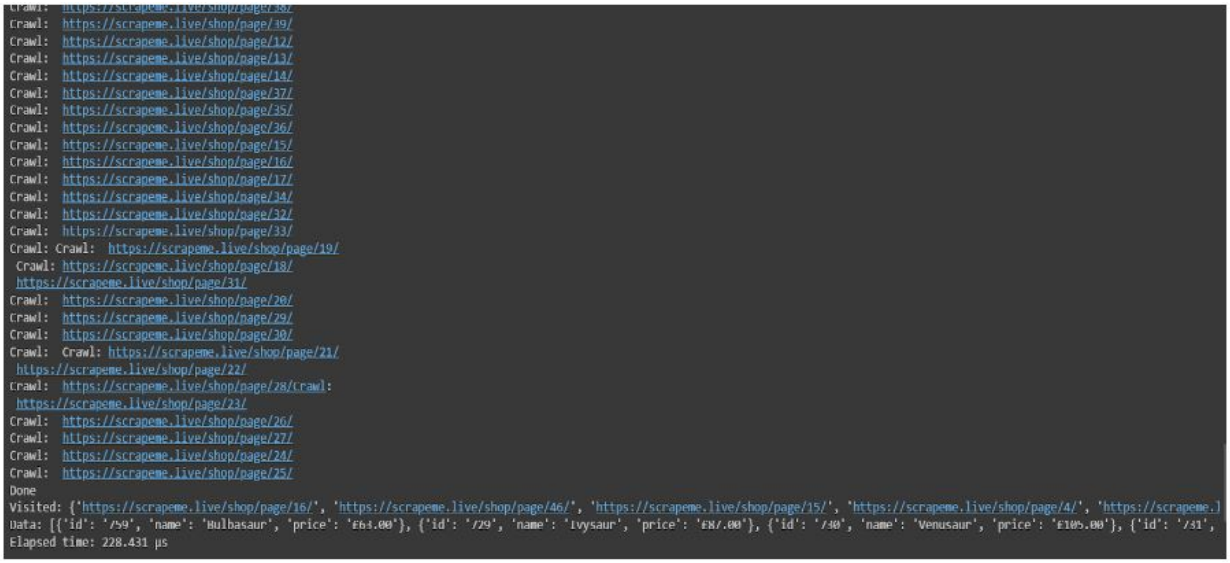

Figure 5.2: WEB CRAWLER THROUGH THREADS

Chapter 6: Conclusion

6.1 Conclusion

- **Site** = https://scrapeme.live/shop/page/1/
- **Total number of pages crawled by both crawlers** = 48 pages
- **Time taken by Sequential Web Crawler** = 17.6533s
- **Time taken by Web Crawler using Threads** = 0.000228s
- **Time saved** = 17.65302s

Chapter 7: References

7.1 Reference

- https://www.geeksforgeeks.org/multithreaded-crawler-in-python
- https://scrapeme.live/shop/
- https://www.zenrows.com/blog/mastering-web-scraping-in-python-crawling-from-scratchhow-to-get-all-the-links-on-the-page