Multiprocessing Approach to Web Scraping

Ralph Angelo Furigay
College of Computer Studies
De La Salle University
Manila, Philippines
ralph_furigay@dlsu.edu.ph

Kim Williame Lee
College of Computer Studies
De La Salle University
Manila, Philippines
kim_leejra@dlsu.edu.ph

***Abstract*—Data collection is one of, if not, the most important aspects of research. It is, however, tedious and time-consuming to manually gather data. One approach to automating the process is through web scraping, which allows the navigation of web pages and extraction of their contents usually faster than the rate of manual data collection. This study explores the implementation of multiprocessing to facilitate the optimization of the web scraping process. The research demonstrates that employing multiprocessing leads to speed-up gains and enhanced efficiency of up to 262% and 421% in link scraping and information extraction processes respectively. However, it is observed that this improvement is subject to diminishing returns, particularly when the number of information scrapers exceeds a specific threshold or depending on the specifications. These results underscore the necessity for optimizing strategies to ensure scalability under varying conditions.**

***Index Terms*—web scraping, parallel programming, multiprocessing, producer-consumer problem, data extraction**

I. Introduction

Web scraping is a technique that makes use of automated tools to programmatically navigate web pages and extract their content [1]. It has become a common approach to efficiently handle data extraction tasks. By using web scraping tools, information, such as email addresses, contact information, product details, prices, and more, can be retrieved and converted into a structured format that can be easily stored, analyzed, or processed further [1]. However, scraping large sites and retrieving voluminous data could be time-intensive tasks wherein bottlenecks could arise in the data collection and processing process. Hence, efficiency is a design consideration in developing these web scraping programs.

This paper explores the use of parallel programming to enhance web scraping pipelines. Specifically, the pipeline scrapes a specified website and fetches the emails and names of the corresponding owner or office under a specified execution time. Parallel programming allows the scraper to distribute tasks across multiple threads or processes, enabling it to process multiple web pages simultaneously. This strategy significantly reduces the overall scraping duration and optimizes resource utilization, leading to more efficient web scraping task execution.

II. Project Implementation

A. Program Input and Output

The email web scraper program is designed to automate the process of gathering email addresses from a particular site within a set time frame. The program's input consists of the URL of the website to be scraped, the scraping time in minutes, and the number of processes to use for parallelization. For this study, the website to be scraped is limited to the website of De La Salle University (https://dlsu.edu.ph). The program outputs two text files: one containing email addresses and their associated information (such as name, office, department, or unit) in CSV format, and a text file containing web scraping statistics, such as the URL, number of pages scraped, number of email addresses found, as well as the start, end, and overall execution time of the web scraping process.

B. Web Scraping

The program utilizes Python libraries such as `requests` [2] for making HTTP requests, `BeautifulSoup` [3] for parsing HTML content, and `csv` for writing the scraped data into a CSV file.

The program starts by requesting the HTML content of the input or base URL and parsing it using `BeautifulSoup`. It filters out non-HTML content to ensure that it only processes web pages. And in doing so, a `try-except` block is used to handle potential exceptions that might occur during the request to access the URL. In particular, it catches scenarios where the request resulted in too many redirects arising from a redirect loop or problematic URLs. With the base URL as the current page, it gets all valid links from the current page and adds them to a list of URLs to visit. Validations are implemented to ensure that only content-rich pages within the `dlsu.edu.ph` domain are considered for further scraping, avoiding unnecessary requests to external or irrelevant sites. URLs redirecting to media content such as PDFs, images, and videos are ignored. This process is further illustrated in Fig. 1.

For each visited page, the program searches for email addresses from the text content and links within a web page. The DLSU website employs an encryption mechanism via Cloudflare to prevent easy scraping by bots. The pipeline utilizes regular expressions and decoding methods to bypass this security measure and decode the email address. It also fetches the associated name or office label of the email but uses a default placeholder, *"No name found"*, if none is found. Additionally, the program ensures that duplicate emails are not added to the scraped data collection. This process is expounded on in Fig. 2.

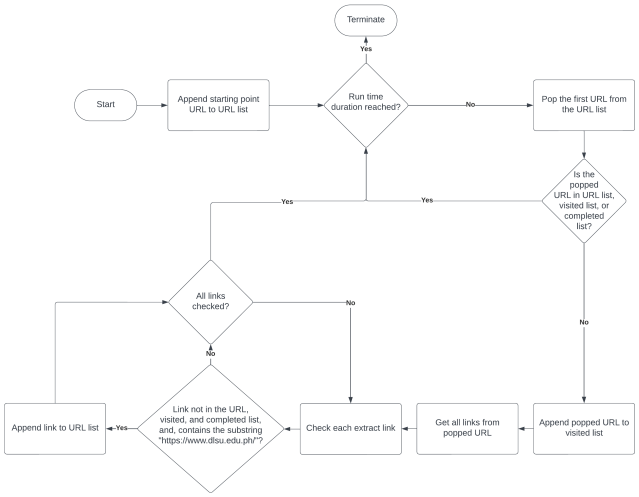

Fig. 1. Link Scraping Flowchart

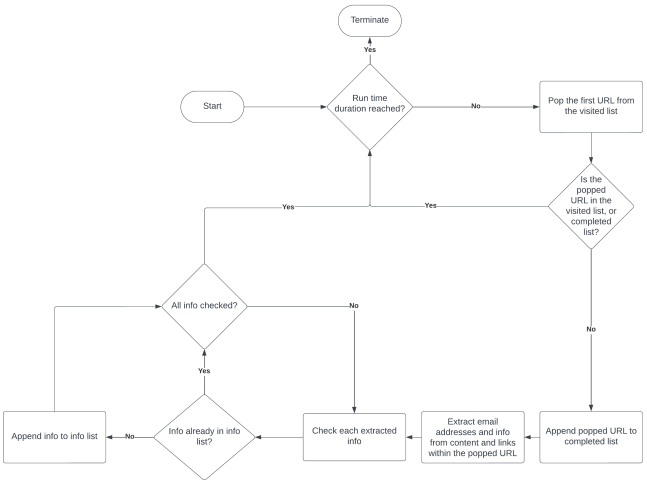

Fig. 2. Information Scraping Flowchart

Multiple instances of `LinkScraper` and `InfoScraper` processes are created based on the input. Then, they are started to perform parallel scraping tasks. The scraping process is time-bound; hence, it will automatically stop after the specified duration. Once the scraping is complete, the program produces the required output files.

The program is run on a system with an 8-core, 16-thread Ryzen 7 5700X processor, and 32 GB of 3,600 MHz DDR4 memory.

C. Multiprocessing Implementation

The program applies multiprocessing by dedicating separate processes for link scraping and information logging. Custom classes for both link scraping and information scraping were created, namely `LinkScraper` and `InfoScraper`, which inherit from the `Process` class in the `multiprocessing` [4] module. `LinkScraper` requires to be initialized with an ID, a URL starting point, a shared list of URLs to be scraped, a shared list of URLs that have been link scraped, a shared list of URLs that have been both link scraped and info scraped, the time when the program was executed, and the specified duration of the program run time. With some similarity, `InfoScraper` requires the same arguments except for the URL starting point, and the shared list of URLs to be link scraped, but with the addition of a shared information list. In essence, a producer-consumer relationship can be observed in the multiprocessing implementation wherein `LinkScraper` processes produce links where information could be taken while the `InfoScraper` processes go through each of the scraped links to fetch information.

Since web scraping is a deep dive into web pages for links and information, it is important to consider data management when multiple processes are involved so that tasks are allocated accordingly and are not executed multiple times. Several issues could arise when information is not made known to separate processes such as task overlapping and repetitive logging of information and URLs which could lead to the infinite execution of the program and inconsistent resulting data. To prevent these possible outcomes, a `Manager` object is created to allow data sharing among the spawned processes and to minimize or eliminate the repeated execution of tasks. The `Manager` object is used to create shared lists, namely `url_list`, `visited_list`, and `completed_list`, and a dictionary with the name `info_list`. `url_list` is used to store URLs that are to be scraped, `visited_list` for storing URLs that have been scraped for links, `completed_list` for storing URLs that have been scraped for both links and information, while `info_list` is used to record the email address and their associated name or office. In addition to shared lists and the dictionary, logic checks were put in place, specifically on the lists that handle information and history of visited web pages, to ensure that no information is duplicated and that already visited web pages are not visited again. Applying these techniques not only minimizes the risk of inconsistent or invalid resulting data, it also ensures that resources are not wasted with repetitive tasks.

III. Result

In this study, the performance of a parallel programming approach to web scraping email addresses and their associated information from a specific website was evaluated. The parallelization strategy employed a producer-consumer model using multiprocessing, with the `LinkScraper` Process responsible for extracting links from the DLSU web pages and the `InfoScraper` Process processing these links to retrieve the required data. The number of processes for both `LinkScraper` and `InfoScraper` were varied to investigate their impact on the efficiency of the scraping process.

As seen in the results in Table I, increasing the number of processes for both `LinkScraper` and `InfoScraper` leads to improved performance in terms of the number of pages

scraped and email addresses retrieved. This observation suggests that parallelization effectively harnesses computational resources to expedite the scraping process and enhance data collection efficiency. Consequently, these findings align with the expected behavior of parallel computing, where parallel execution allows for concurrent processing as compared to sequential execution [5]. As such, more tasks can run and produce output within a limited time frame.

However, beyond a specific threshold, the rate of improvement diminishes. For instance, as seen in Table II, when the number of `InfoScraper` processes increases from 4 to 6, the increase in email addresses retrieved is not always as significant as it was from 2 to 4 processes. This observation suggests that there may be inherent bottlenecks or inefficiencies in the parallelization strategy, particularly at higher process counts. It could also indicate that the execution time plays a huge part in determining the number of email addresses fetched given a configuration since the rate of page scraped shows an increase as the number of `LinkScraper` processes also increases possibly leading to the scraping of web pages that have better email address availability.

According to Amdahl's Law, the speedup of a parallel program is limited by the fraction of the code that cannot be parallelized [5]. In the context of this experiment, the diminishing returns observed at higher process counts indicate that the program is approaching the limits imposed by Amdahl's Law. Moreover, while multiprocessing did lead to significant performance improvements, other aspects of the program, such as interprocess communication and the built-in synchronization mechanisms of the `multiprocessing` module in Python may introduce overhead that restricts the overall scalability.

Interprocess communication (IPC) refers to the methods used by processes to exchange data and synchronize their execution [5]. Efficient communication between these processes is crucial for coordinating their tasks. However, as the number of processes increases, the overhead associated with IPC can become a bottleneck. In the study, the increasing number of `InfoScraper` processes may have exacerbated the communication overhead, leading to the convergence in terms of performance improvement. Synchronizing tasks such as sharing links and coordinating access to shared resources introduced latency and contention, ultimately limiting the scalability of the implemented parallelization solution.

While the results might vary depending on the configuration, it still highlights how some setups might be more efficient compared to others. For example, a configuration of 2 `LinkScraper` processes and 4 `InfoScraper` processes might be the best balance of resource allocation when the execution time is set to 3 or 6 minutes while a setup consisting of 2 `LinkScraper` processes and 6 `InfoScraper` processes might be best when the execution time is set to at least 9 minutes.

Overall, the results underscore the effectiveness of multiprocessing in accelerating web scraping tasks and improving its data collection capabilities. However, with interprocess communication, the communication overhead may affect the scalability of multiprocessing while the specifications might dictate and influence the efficiency of a given configuration.

TABLE I
WEB SCRAPING PERFORMANCE EVALUATION RESULTS

Input			**Output**	
Execution Time (mins)	***LinkScraper Processes***	***InfoScraper Processes***	***Pages Scraped***	***Emails Found***
3	1	1	250	17
3	1	2	317	28
3	2	2	355	28
3	2	4	693	64
3	2	6	694	64
6	1	1	389	28
6	1	2	792	65
6	2	2	854	67
6	2	4	1410	75
6	2	6	1411	75
9	1	1	675	32
9	1	2	1075	70
9	2	2	1232	72
9	2	4	2021	89
9	2	6	2117	167

TABLE II
WEB SCRAPING PERFORMANCE IMPROVEMENTS

Input			**Change from baseline**	
Execution Time (mins)	***LinkScraper Processes***	***InfoScraper Processes***	***Pages Scraped***	***Emails Found***
3	1	1	-	-
3	1	2	+26.80%	+64.71%
3	2	2	+42.00%	+64.71%
3	2	4	+177.20%	+276.47%
3	2	6	+177.60%	+276.47%
6	1	1	-	-
6	1	2	+103.60%	+132.14%
6	2	2	+119.54%	+139.29%
6	2	4	+262.47%	+167.86%
6	2	6	+262.72%	+167.86%
9	1	1	-	-
9	1	2	+59.26%	+118.75%
9	2	2	+82.52%	+125.00%
9	2	4	+199.41%	+178.13%
9	2	6	+213.63%	+421.88%

IV. CONCLUSION

The study shows that distributing tasks to multiple processes does result in a speedup in the web scraping process and improvement in the efficiency of the pipeline. It also highlights the importance of synchronization as a means of ensuring that data is consistent throughout the spawned processes because it allows the program to properly allocate tasks and, consequently, improve the rate of task completion. Without applying multiprocessing, the program would have to execute in a sequential manner which would lessen the rate of how fast each task, in this case link scraping and data collection, is accomplished. Meanwhile, the lack of shared data storage and their corresponding consistency checks could lead to the continuous distribution of tasks that have already been accomplished. This could burden the processes with tasks that do not need to be executed anymore— reducing the program's

overall efficiency while also adding the risk if invalid or dirty resulting output.

There are, however, factors that need to be considered when setting the configuration for a multiprocessing program such as Amdahl's Law and the communication overhead of IPC as these factors might dictate how limited the scalability of the system might be. The specifications must also be reviewed when determining the right balance of processes to set per a given task to maximize the performance without excessive use of resources. Hence, the optimization of parallel strategies is essential to find the balance between performance, scalability, and resource usage, all of which are relevant aspects of data-intensive programs like web scrapers.

REFERENCES

[1] GeeksforGeeks, "What is web scraping and how to use it?," March 2023.
[2] K. Reitz, "requests," May 2023.
[3] L. Richardson, "Beautiful soup documentation," *April*, 2007.
[4] python, "multiprocessing — process-based parallelism."
[5] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. Nashville, TN: John Wiley & Sons, Dec. 2012.