Optimized Web Scraping System Using Multithreading Techniques for Corporate Information Extraction

Francisco Hermo, Ángel Gómez, and Carlos Dafonte

Centro de Investigación CITIC - Laboratorio Interdisciplinar de Aplicaciones de la Inteligencia Artificial (LIA2), Faculty of Computer Science, Universidade da Coruña, 15071 A Coruña, Spain
Correspondence: f.hermo@udc.es; angel.gomez@udc.es; carlos.dafonte@udc.es

DOI: *https://doi.org/10.17979/spu.23.c17*

Abstract: This study aims to automatically identify the various ICT innovations implemented by companies on their websites. To achieve this, web scraping techniques are proposed as the most effective approach. However, the need to process large sets of companies presents challenges in terms of execution time and system performance. To address these issues, our system incorporates multithreading techniques, enabling the concurrent execution of analysis drivers that extract the targeted elements. As a complement to our extraction system, we developed a web application that allows users to visualize the obtained results, organized on a map of Spanish provinces and grouped by the analyzed companies.

1 Introduction

Web scraping is a computational technique for the automated extraction and analysis of data from websites. It operates by programmatically accessing web pages, parsing their underlying HTML structure, and retrieving specific elements or tags of interest (Khder, 2021). Figure 1 provides an overview of the web scraping process, illustrating the main stages from data request sending to extracting, processing, and storing the retrieved information.

Figure 1: Conceptual scheme of web scraping.

Modern web scraping tools have evolved to integrate seamlessly with standard search engines, such as Google, enabling large-scale comparison of multiple websites, detailed content analysis, and systematic data collection for subsequent processing (Lotfi et al., 2022).

Today, corporate websites have significantly evolved from simple informational pages or product catalogs into dynamic platforms that incorporate advanced technologies to enhance user experience. This ever-changing environment, combined with the exponential growth of data generated daily on the Internet, has driven the need for efficient web scraping techniques, essential in most competitive environments (Laender et al., 2002). They prove useful for comparing prices across different providers, analyzing markets to implement new customer acquisition strategies, or identifying technological innovations that can enhance corporate websites.

It is at this point that our project comes into play. We have developed a set of custom web scrapers specifically designed to detect the presence of technological innovations on corporate websites. Our scrapers focus on identifying the existence of social media profiles, links to mobile applications along with their associated information, and the presence of intelligent chatbots.

One of the major challenges in web scraping lies in processing large volumes of data efficiently. To address this, we considered several strategies during the development of our system. One approach involves partitioning the dataset into smaller subsets, enabling the scrapers to be run on reduced input groups and later consolidating the extracted results. Another strategy aims at minimizing workload by targeting only the exact elements to be extracted—but this can become problematic, as frequent changes in website HTML structure can quickly obsolete the scrapers. The solution we ultimately adopted, and which will be discussed in further detail, is the use of multithreading to manage multiple scraping drivers concurrently. This distributes the workload and improves overall performance (Charlotte Will, 2024).

The remainder of this document is organized as follows: Section 2 presents the adopted solution and the preliminary versions developed prior to reaching the current implementation. Section 3 describes the development process of the optimized scraper versions and obtained results. Finally, Section 4 concludes the document and outlines potential directions for future work and improvements.

2 Multithreading solution and results

To begin, it is important to provide some context regarding what multithreading is and how it works. Multithreading is a programming technique that allows multiple threads—independent units of execution within a single process—to run concurrently (Carver and Tai, 2005). By enabling tasks to be executed in parallel, multithreading improves the efficiency of resource utilization, particularly in applications that involve intensive input/output operations or the processing of large data volumes, like in the web scraping. This concurrent execution does not imply that each thread always runs simultaneously on separate processor cores; rather, the operating system scheduler manages the allocation of CPU time among threads, creating the effect of parallelism. When effectively implemented, multithreading can significantly reduce execution time, enhance responsiveness, and optimize the performance of complex systems such as large-scale web scraping frameworks.

For the web scraping processes implemented in our system, we rely on the *Beautiful Soup* library (Richardson, 2020) and the tools provided by *Selenium* (SeleniumHQ, 2022), both in Python (George, 2021). The basic structure of the different scrapers, as we can see in figure 2, consists of providing them with a dataset containing information about the companies to be analyzed, using the URL of their websites, parsing the HTML structure to identify the key elements of interest, and storing the extracted results in a JSON file for subsequent use.

In the initial implementation of the system, a Selenium driver was launched for each company to be analyzed, performing the search and extraction before being closed. One advantage that can improve the scraper's performance and execution time, and which we tested in this iteration, is the use of Selenium drivers in headless mode. In this configuration, the browser window is not displayed, and all operations are executed in the background without visual rendering. However, we encountered a limitation with this headless mode drivers: in certain

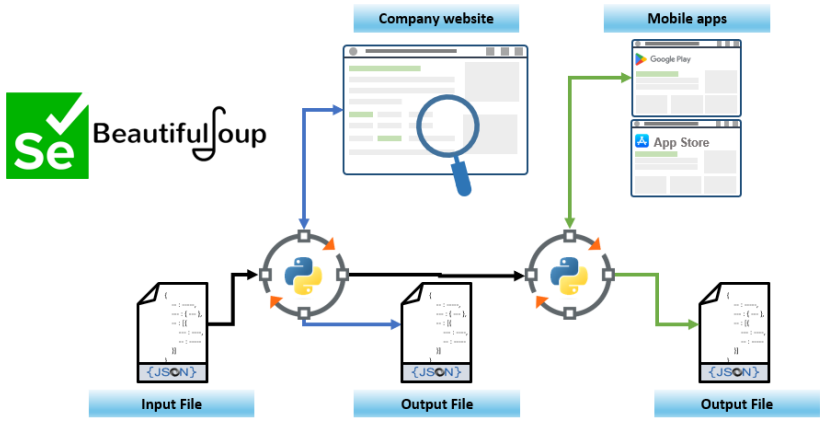

Figure 2: Scraping system diagram.

cases, web pages failed to load correctly, which led to incomplete results being extracted.

For this version, the execution times obtained were excessively high, as will be shown later. At this stage, we began developing a new version that incorporated multithreading in order to take advantage of its benefits and reduce execution time (Sharma et al., 2021). In this updated system, we implemented threads so that each one managed a Selenium driver in parallel. We maintained the option of running drivers with a visible window to ensure reliable results, while the writing of the extracted data into the JSON file was performed once all extractions had been completed.

The results obtained in this version showed improvements compared to the initial implementation, but they were still not fully optimal. The main challenge we needed to address in order to further reduce execution time was the use of headless drivers while ensuring reliable output. To tackle this, we developed a more robust version of the system that incorporated Selenium's built-in waiting functions and leveraged Beautiful Soup more extensively for processing the HTML content, even when the driver's interaction with the website was shorter than in previous iterations. Finally, we implemented a writing queue to store the extracted results progressively instead of saving all the data only at the end of the process.

From a technical perspective, this hybrid approach provided a balance between Selenium's ability to dynamically interact with complex, JavaScript-driven websites and Beautiful Soup's efficiency in parsing and analyzing the underlying HTML. By delegating dynamic rendering to Selenium while offloading most of the structural analysis to Beautiful Soup, the system achieved greater robustness and efficiency, particularly under headless execution.

In Table 1, the execution times obtained for the different versions of the scrapers are presented. For all the executions we used the same dataset of 500 companies. As previously mentioned, the implemented improvements consistently resulted in a substantial reduction in execution time. In the case of the chatbot scraper, no measurements are available for the initial stage of the project, as it was developed at a later phase when we expanded the models to target additional technological innovations.

Scraper	Times V1 (sec)	Times V2 (sec)	Times V3 (sec)
Social media	324005.38	4255,03	690.15
Mobile apps	324944.05	2246.87	929,09
Chat-bots	-	2669.08	703.27

Table 1: Execution times of the different versions

3 Development of the optimazed version

In this section, we provide a detailed analysis of the structure and improvements introduced in the latest version of the developed scrapers, explaining the rationale behind their implementation and how they contributed to enhancing our results (Figure 3).

3.1 Basic structure

The structure adopted for all the scrapers consists of several key components: specific functions responsible for locating and extracting the targeted elements, a function dedicated to managing the saving queue and writing the results, a management function that handles the threads and invokes the extraction functions, and a main execution block.

The purpose of the extraction functions is straightforward: to access the companies' websites using Selenium drivers, retrieve the HTML content with Beautiful Soup, and search for the tags or elements of interest. Once extracted, the data are passed to the function responsible for managing the saving queue.

3.2 Queue function

The queue management function is responsible for managing the writing of extracted data to disk in a controlled and reliable manner. The process begins by opening a temporary JSON file, where results are written sequentially as they are retrieved from the queue. Each new element is serialized into JSON format and appended to the file, while a counter keeps track of the total number of saved items. To ensure thread safety, an optional locking mechanism is employed, updating the saving status without concurrency conflicts. The function continues processing until it receives a termination signal from the queue, at which point the temporary file is closed and atomically replaced with the final output file. This design prevents data loss in case of system interruptions, reduces memory overhead by avoiding in-memory accumulation of results, and guarantees the integrity of the final dataset. Additionally, event signaling is used to notify the system when the saving process has been successfully completed.

3.3 Multithreading function

The multithreading control function serves as the main orchestration component of the scraping process, coordinating the extraction of information from multiple companies in parallel. It begins by initializing control structures such as queues for results, locks to avoid concurrency issues, and an event signal to manage synchronization between threads. A dedicated saving thread is launched to continuously process the results queue, writing partial outputs to a temporary file and ensuring data persistence throughout execution. The function then creates a pool of Selenium drivers, each managed by a thread, to visit the websites of the companies and perform the extraction of the required elements. For each company, the corresponding thread handles navigation, parsing of the target information, and preparation of a structured result that is pushed into the queue. Robust error handling mechanisms are included to capture timeouts or unexpected failures, guaranteeing that every company is accounted for in the

final dataset, even when the extraction is unsuccessful. Once all threads complete, the function signals the termination of the saving process, waits for synchronization, and finalizes the output file. This design leverages multithreading to significantly reduce execution time, maintain system robustness, and ensure reliable and complete data collection (Sharma and V., 2025).

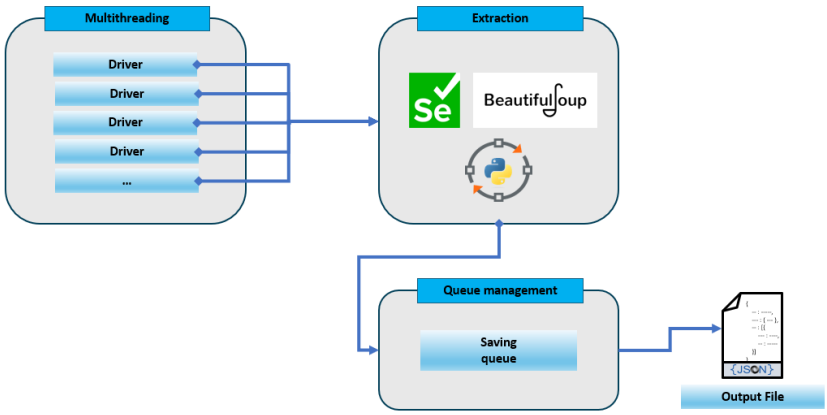

Figure 3: Multithreading solution diagram.

4 Conclusion

The main objective of this study was to optimize the extraction of technological innovation elements that companies implement on their websites. The integration of multithreading into our scrapers, the design of a result management and storage system based on queues, and the effective use of Selenium and BeautifulSoup have constituted the fundamental pillars that enabled us to achieve optimal performance across all developed versions.

As future work, we plan to extend our approach by developing new scrapers capable of detecting and extracting additional types of innovations, while incorporating the improvements consolidated throughout this study. Regarding optimization, our focus will be on refining the filtering mechanisms applied during extraction, aiming to increase the precision of results while preserving the efficient execution times already achieved. As a complement of this, we are going to try new experiments to assess whether multithreading enhancements produce betther results, while also relying on additional performance metrics such as CPU and memory usage.

Acknowledgements

This work is part of the project *"Ciencia e ingeniería de datos para la mejora de la función estadística oficial"* (CIDMEFEO) in the line of investigation *"Línea 8: Utilización de información pública en la web para determinar características de empresas relativas a la Innovación y/o a la Sociedad de la Información"* Ref. 2021N1013002, alongside the *"Instituto Nacional de Estadística"* (INE).

Bibliography

R. Carver and K.-C. Tai. *Modern Multithreading : Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs*. Wiley-Interscience, 2005.

Charlotte Will. How to optimize web scraper performance with multi-threading. *https://parazun.com/how-to-optimize-web-scraper-performance-with-multi-threading/*, 2024.

N. George. *Practical Data Science with Python: Learn Tools and Techniques from Hands-On Examples to Extract Insights from Data*. Packt Publishing, 2021.

M. Khder. Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal Of Advances In Soft Computing & Its Applications.*, 13:145–168, 2021.

A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. Web scraping or web crawling: A brief survey of web data extraction tools. *ACM SIGMOD Record.*, 31:84–93, 2002.

C. Lotfi, S. Srinivasan, M. Ertz, and I. Latrous. Web scraping techniques and applications: A literature review. *SCRS Publications.*, pages 381–394, 2022.

L. Richardson. Beautiful soup, documentación oficial. *https://www.crummy.com/software/BeautifulSoup/bs4/doc/*, 2020.

SeleniumHQ. Selenium, sitio web. *https://www.selenium.dev*, 2022.

A. Sharma, V. Shrivastava, and S. H. Experimental performance analysis of web crawlers using single and multi-threaded web crawling and indexing algorithm for the application of smart web contents. *Materials Today: Proceedings*, 37:1403–1408, 2021.

V. Sharma and A. V. Code optimization techniques for improving execution time in python. *International Journal of Research Publication and Reviews*, 2025.