УДК 004.67 **DOI:** 10.31673/2412-9070.2024.032327
В. М. ДАНИЛЬЧЕНКО[1], ст. викладач, ORCID: 0009-0004-6839-2132,
С. І. ОТРОХ[2], доктор техн. наук, професор, ORCID: 0000-0001-9008-0902,
В. П. КЛЮЧУК[2], студент, ORCID: 0009-0000-1801-8573,
О. В. САРАФАННІКОВ[2], аспірант, ORCID: 0000-0002-8373-4629,
[1] Державний університет інформаційно-комунікаційних технологій, Київ
[2] Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ

ПРИСКОРЕННЯ ЗБОРУ ТА АНАЛІЗУ ДАНИХ ЗА ДОПОМОГОЮ ІНСТРУМЕНТІВ АСИНХРОННОГО ПРОГРАМУВАННЯ У WEB SCRAPING

Підкреслено важливість та актуальність використання технологій web scraping для збору важливої інформації. Досліджено можливості застосування асинхронних інструментів для швидкого та дієвого збору даних із сайтів великого масштабу. Подано докладний опис процесу розроблення та порівняння швидкодії програмних алгоритмів із використанням мови програмування Python та бібліотек «Requests», «Asyncio», «Aiohttp» та «BeautifulSoup». Запропоновано високошвидкісний метод збору інформації, який можна використовувати в різноманітних випадках: починаючи зі збору новин і завершуючи нагромадженням даних для моделі штучного інтелекту.
Ключові слова: асинхронне програмування; Web scraping; Python; Requests; Asyncio; Aiohttp; BeautifulSoup.

ВСТУП

Постановка проблеми. Збір даних відіграє одну з ключових ролей у сучасному світі. Важливість та кількість інформації збільшується з геометричною прогресією, що робить технології для збору ще актуальнішими. Здобута інформація не лише допомагає в розвитку різноманітних технологій, а й впливає на світові тренди, ухвалення вирішальних рішень у сферах економіки, медицини, освіти тощо.

У бізнес-процесах збір даних є найважливішим аспектом створення конкуренції і відшукування та залучення клієнтів. Більшість компаній, окрім збирання інформації про свою аудиторію, намагаються знаходити та створювати бази даних потенційних клієнтів із різноманітних джерел. Для таких компаній одним з основних потоків важливої інформації є вебсайти конкурентів.

Проте щоденний приріст даних в інтернеті не може не відчуватися, адже зі збільшенням обсягу даних збільшується і кількість інтернет-сторінок. Однак паралельно з цим, потреба у збереженні темпу збирання нікуди не зникла, навпаки, потрібно виходити на нові швидкості, щоб встигати за світовим прогресом. Багато сучасних галузей напряму залежні від великої кількості різноманітних даних, які мають бути завжди оновленими та релевантними.

Аналіз останніх досліджень і публікацій. Протягом останніх років світ технологій пережив неймовірний розвиток. Це зумовило масовий потік різноманітної інформації. У статті [1] автор надав дані, посилаючись на дослідження компанії IBM, що кожного дня людство створює приблизно 2,5 квінтильйона байтів даних. Очікується, що ця цифра й надалі буде зростати згідно з геометричною прогресією. Дуже багато галузей потребують рішень, які могли б упоратися з такою величезної вибіркою даних, з метою подальшого аналізу. У статтях [2; 3] наведено сфери, які застосовують інструменти web scraping для своїх цілей. До таких галузей належать сфера нерухомості, електронна комерція, юридична сфера тощо. Також автори приділили увагу важливості використання інструментів збору даних у галузі штучного інтелекту.

Людина не може власноруч обробити такі величезні масиви даних, тож web scraping є доцільним саме в наш час, коли інформація — це головний ресурс, а швидкість збирання цього ресурсу визначає успішність певної сфери.

Метою статті є розроблення удосконаленого алгоритму для прискореного збору даних та порівняння його швидкодії з найпоширенішим базовим методом. Розроблений метод може відкрити великі можливості для збору даних у різноманітних галузях, де щоденний збір значного обсягу інформації стає нагальною потребою.

ОСНОВНА ЧАСТИНА

Нині існує кілька мов програмування, які містять ефективні бібліотеки для збору і оброблення даних. У цій роботі було використано Python і такі бібліотеки як «Requests», «Asyncio», «Aiohttp» та «BeautifulSoup».

Бібліотека «Requests» — це інструмент, який дає можливість здійснювати HTTP-запити до вебресурсів. Головними його перевагами є простота та зручність взаємодії з вебсайтами. У сфері web scraping ця бібліотека набула широкого застосування, адже майже кожен web scraper використовує саме її. Requests дає змогу для відправлення запитів до вебсайтів із відповідними елементами HTTP-протоколу, такими як cookies, headers тощо [4]. Після чого користувач отримує увесь контент HTML-сторінки, який потім можна аналізувати та витягувати з нього корисну інформацію.

BeautifulSoup — це бібліотека, призначена для відбору конкретних даних із HTML- і XML-сторінок. Вона є наступним етапом у програмах збору даних, адже містить багато зручних інструментів для аналізу вебсторінок. Такий інструмент надає інтуїтивний інтерфейс для легкої взаємодії з HTML та XML. Зі свого боку користувач може без проблем витягувати дані, зокрема посилання, таблиці, тексти, фото тощо [5].

Aiohttp — асинхронний інструмент для здійснення HTTP-запитів. Для сфери web scraping він надає безліч можливостей, що уможливлює швидке та ефективне оброблення чималої кількості запитів паралельно. Aiohttp дає змогу застосовувати асинхронний підхід [6], який може підвищити продуктивність збору даних із вебресурсів, що й продемонстровано в даній статті. Також ключовим аспектом є те, що ця бібліотека, як і Requests, надає користувачу право додавати різні елементи HTTP-протоколу та проксі.

Asyncio — це модуль у Python, який дає можливості для побудови асинхронних програм, систем, застосунків тощо. Ця бібліотека надає корутини – функції в Python, які містять ключове слово «async» перед ключовим декларуванням «def» [7]. Актуальність використання цієї бібліотеки є очевидною, адже вона дозволяє виконувати інші операції, не очікуючи на завершення попередніх.

Стандартний алгоритм на основі requests

Написання алгоритму розпочинається з дослідження елементів HTTP-протоколу цільового сайту. Найлегшим способом дістати цю інформацію є звертання до вкладки «Network» у панелі «Inspect». Після чого здійснюється перезавантаження сторінки за допомогою комбінації клавіш CTRL+F5. Далі з'являються всі здійснені запити, серед яких є головний, що містить повну необхідну інформацію.

Наступним етапом написання є створення GET-запиту для отримання потрібної сторінки. Це виконується за допомогою методу requests.get(), яка як параметри приймає headers та cookies, отримані з попереднього кроку.

Після цього збирається вся необхідна інформація за допомогою бібліотеки «BeautifulSoup» для подальшого оброблення. Лістинг головних функцій даного алгоритму на мові Python:

```
from bs4 import BeautifulSoup
import requests

### Створює GET-запит та вертає сторінку
def perform_http_request(destination_link, site_headers, site_cookies):
return requests.get(url=destination_link, headers=site_headers, cookies=site_cookies)

### Перетворює сторінку на об'єкт, з якого можна дістати будь-яку інформацію
def get_bs_object(response):
return BeautifulSoup(response.text, "html.parser")

### Створює цикл, який запускає функцію на збір page_count сторінок
def gather_data(destination_link, site_headers, site_cookies, page_count):
for page in range(1, page_count):
get_page_data(page, destination_link, site_headers, site_cookies)

### Отримує посилання на вакансії зі сторінки під номером page
def get_page_data(page, destination_link, site_headers, site_cookies):
res = perform_http_request(f"{destination_link}?page={page}", site_headers, site_cookies)
soup = get_bs_object(res)
detail_page_links = []
for link in response_text.find_all("a", class_="job-list-item__link", href=True):
detail_page_links.append(f"{destination_link}{link['href']}")
for index, detail_page_link in enumerate(detail_page_links):
get_details(detail_page_link, index, page)
```

Слід зауважити, що функція get_page_data вкінці викликає get_details для отримання сторінки вакансії і збору конкретної інформації з виведенням результату у файл.

Результат роботи алгоритму, який зібрав увесь вміст вакансій і відібрав лише категорії, зображено на рис. 1.

```
----------------------------------------------------------------
Page number: 1
----------------------------------------------------------------
Vacancy number: 1
Категорія: Sales
----------------------------------------------------------------
Vacancy number: 2
Категорія: SQL
----------------------------------------------------------------
Vacancy number: 3
Категорія: Lead Generation
----------------------------------------------------------------
Vacancy number: 4
Категорія: Social Media, Head Chief
----------------------------------------------------------------
Vacancy number: 5
Категорія: Sales
----------------------------------------------------------------
Vacancy number: 6
Категорія: JavaScript
----------------------------------------------------------------
Vacancy number: 7
Категорія: Python
----------------------------------------------------------------
Vacancy number: 8
Категорія: Support
----------------------------------------------------------------
```

Рис. 1. Результат роботи алгоритму

Як випливає з рис. 1, виконується дуже багато дій, адже з однієї сторінки можна отримати список 15-ти і більше сторінок із вакансіями, які також потрібно проаналізувати. Це все створює проблему нагромадження, адже буквально 20 викликів функції get_page_data змусить скрапер зібрати мінімум 300 сторінок. У пункті порівняння ефективності подано результати швидкості цього методу.

Покращений алгоритм на основі Asyncio та Aiohttp

Даний алгоритм, як і попередній, потребує елементи HTTP-протоколу цільового сайту для уникнення блокування з боку сервера. Після чого він може розпочати збір даних. Проте, оскільки це покращений метод з використанням асинхронного підходу, його реалізація має досить помітну різницю.

Передусім потрібно запустити основний асинхронний код за допомогою функції asyncio.run(gather_data()), де gather_data, яка є вхідним параметром і також має бути асинхронною. Вигляд функції зображено у лістингу:

```
async def gather_data(link, headers, cookies, pages):
    await with aiohttp.ClientSession() as session:
        tasks = []
        for page in range(1, pages):
            task = asyncio.create_task(get_page_data(page, link, headers, cookies, session))
            tasks.append(task)
    await asyncio.gather(*tasks)
```

Якщо говорити детальніше, у реалізації маємо елемент створення асинхронного менеджера контексту для ClientSession. Це потрібно для гарантування звільнення ресурсів, зокрема з'єднання HTTP після виконання блоку коду.

Наступним важливим моментом є створення асинхронної задачі для функції get_page_data. Після чого утворюється список усіх задач, які запускаються за допомогою await asyncio.gather. Це забезпечує паралельне виконання get_page_data для всіх сторінок, які потрібно обробляти.

Схожу будову має й функція get_page_data, яка вже із зібраної сторінки дістає посилання на сторінки вакансій. Вміст продемонстровано у лістингу:

```
async def get_page_data (page, link, headers, cookies, session):
    url = f"{link}?page={page}"
    async with session.get(url=url, headers=headers, cookies=cookies) as response:
        response_text = BeautifulSoup(await response.text(), "html.parser")
        detail_page_links = []
        for link in response_text.find_all("a", class_="job-list-item__link", href=True):
            detail_page_links.append(f"{destination_link}{link['href']}")
        tasks = []
        for i, link in enumerate(detail_page_links):
            task = asyncio.create_task(get_details(session, i, link, headers, cookies))
            tasks.append(task)
    await asyncio.gather(*tasks)
```

Спочатку формується URL для HTTP-запиту, після чого за допомогою асинхронного контекстного менеджера отримується повне позначення та дані. Слід зауважити, що програма зберігає ту саму сесію, яку було створено у функції gather_data. Наступним кроком BeautifulSoup відокремлює основний текст та надає метод find_all для збирання всіх посилань сторінок детальної інформації про вакансію.

Наприкінці знову створюється і виконується список завдань за допомогою asyncio.gather.

Варто зазначити, що get_details також є асинхронною функцією, яка збирає дані про категорію вакансії зі сторінки. Насправді, реалізація цього методу не зупиняється лише на одному виді інформації, адже там відбувається оброблення вже окремої сторінки, де можна дістати і більш цікаві дані, проте для прикладу було виведено лише категорію. Загалом, функція виконується за схожим сценарієм як і в попередньому лістингу.

Підхід цього алгоритму добре вкладається у збір даних великого маштабу, адже всі ці функції дають змогу програмі «не чекати» завершення кожного етапу, щоб перейти на наступний [8]. Це уможливлює ефективно розпоряджатися часом та ресурсами системи.

Результати роботи алгоритму унаочнюють рис. 2, рис. 3.

```
---------------------------------------------
Page number: 6
Page number: 8
Page number: 2
Page number: 10
Page number: 20
Page number: 7
Page number: 12
Page number: 19
Page number: 16
Page number: 17
Page number: 9
Page number: 4
Page number: 5
Page number: 11
Page number: 18
Page number: 3
Page number: 13
Page number: 14
Page number: 15
Page number: 1
---------------------------------------------
```

Рис. 2. Зібрані сторінки

```
---------------------------------------------
Vacancy number: 106
Категорія: Marketing
---------------------------------------------
Vacancy number: 106
Категорія: QA Automation
---------------------------------------------
Vacancy number: 76
Категорія: (Other)
---------------------------------------------
Vacancy number: 139
Категорія: Marketing
---------------------------------------------
Vacancy number: 140
Категорія: Recruiter
---------------------------------------------
Vacancy number: 166
Категорія: Project Manager
---------------------------------------------
Vacancy number: 137
Категорія: Lead Generation
---------------------------------------------
```

Рис. 3. Зібрані вакансії

На рис. 2 і рис. 3 спостерігаємо головну характеристику роботи алгоритму — сторінки та вакансії зібрані не за порядком. Це демонструє найважливішу фразу «не чекати завершення», адже протягом роботи було запущено багато паралельних задач, які залежать від часу виконання зовнішнього запиту, котрий зазвичай варіюється залежно від різних чинників. У нашому разі час відповіді від кожної сторінки або оброблення вхідних даних сайту може впливати на порядок подання елементів. Якщо важливо зберігати послідовність, варто використовувати базовий метод і тоді результат буде мати вигляд, як на рис. 1.

Порівняння ефективності базового та покращеного алгоритмів

Для оцінювання роботи алгоритмів було збережено час, коли завершила роботу функція відбору категорій вакансій. Загалом ця функція спрацювала 300 разів та зібрала стільки ж сторінок із деталями вакансії. Ще алгоритми зібрали 20 сторінок із усіма посиланнями на зазначені раніше матеріали. Після чого було побудовано графік, зображений на рис. 4.

Розглядаючи графік, можна помітити величезну різницю у швидкості. Тоді, коли синхронний скрапер збирає 1,91 сторінки за 1 с, асинхронний підхід дає можливість прискорити роботу до 43,17 сторінок за 1 с. Також до уваги потрібно взяти те, що це не лише збирання сторінок, а й відбір потрібної інформації, що навантажує алгоритм. Загалом цей метод дає перевагу у швидкодії в межах 20-25 разів.

Також важливо зауважити, що такі алгоритми мають лінійну залежність. Ці знання можуть бути корисними для оцінювання можливого часу на збір і оброблення відомого обсягу сторінок.

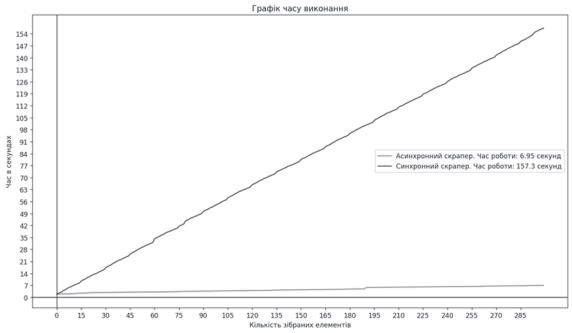

Рис. 4. Графік швидкодії алгоритмів

ВИСНОВКИ

У статті було наведено вдосконалений алгоритм збору даних, який використовує інструменти асинхронного програмування. Для порівняння було подано базовий синхронний алгоритм. На основі часу та зібраних сторінок із даними, було наведено графік із різницею у швидкості двох алгоритмів, де асинхронний продемонстрував велику перевагу над синхронним, а саме в 20-25 разів.

Застосування такого алгоритму може мати значний вплив на різні сфери, зокрема економіка, бізнес, машинне навчання тощо, адже він дає велику перевагу у швидкодії, що економить ресурси та час.

Список використаної літератури

*1. **Data** usage trends [Електронний ресурс] // Utilities One. URL:*
https://utilitiesone.com/data-usage-trends-are-people-using-more-or-less-data-over-time (дата звернення: 10.03.2024).
*2. **Sheremeta A.** Web scraping: importance, techniques, and applications in 2024 [Електронний ресурс] // DataForest. URL:*
https://dataforest.ai/blog/what-is-web-scraping-and-how-can-it-benefit-your-business (дата звернення: 10.03.2024)
*3. **What** are the practical uses and advantages of web scraping? [Електронний ресурс] // WebHarvy. URL:*
https://www.webharvy.com/articles/web-scraper-use-cases.html (дата звернення: 10.03.2024).
*4. **Requests** documentation [Електронний ресурс] // Requests: HTTP for Humans™. URL:*
https://requests.readthedocs.io/en/latest/ (дата звернення: 10.03.2024).
*5. **Beautiful** Soup Documentation [Електронний ресурс] // Beautiful Soup. URL:*
https://beautiful-soup-4.readthedocs.io/en/latest/ (дата звернення: 10.03.2024).
*6. **Welcome** to AIOHTTP [Електронний ресурс] // Aiohttp documentation. URL:*
https://docs.aiohttp.org/en/stable/ (дата звернення: 10.03.2024).
*7. **Asyncio** – Asynchronous I/O [Електронний ресурс] // Python documentation. URL:*
https://docs.python.org/3/library/asyncio.html (дата звернення: 10.03.2024).
*8. **Асинхронне** програмування: що це таке та його особливості [Електронний ресурс] // FoxmindEd. URL:*
https://foxminded.ua/asynkhronne-prohramuvannia/ (дата звернення: 10.03.2024).

V. Danylchenko S. Otrokh, V. Kliuchuk, O. Sarafannikov

ACCELERATION OF DATA COLLECTION AND ANALYSIS USING ASYNCHRONOUS PROGRAMMING TOOLS IN WEB SCRAPING

This article discusses the importance and relevance of using web scraping technologies to effectively collect a significant amount of information in various fields. The potential of using asynchronous tools for fast and productive data retrieval from large-scale web resources has been studied. The article analyzes in detail the possibilities of using asynchronous tools in the context of web scraping, considering their advantages compared to synchronous approaches. Special attention is paid to the use of the Requests libraries, which provide tools for the standard linear approach, and Asyncio, Aiohttp, which help implement the asynchronous approach. After that, the article conducts a comparative analysis of their performance in the scenario of data collection from the website. The development process using the Python programming language is described in detail, and code is presented that illustrates each stage of the execution of the synchronous and asynchronous algorithms in combination with the libraries presented. The authors of the article consider asynchronous web scraping as a powerful tool for creating fast and efficient data collection mechanisms that can be used to train models and analyze large volumes of information. The article discusses the importance of further development of this method in order to ensure high speed of data collection and improve their applicability in various areas. It demonstrates the practical advantages of asynchronous web scraping, and also indicates the prospects for improving this method to improve the collection and processing of information on a scale that goes beyond standard methods. Further research may consider aspects of automating and expanding the capabilities of asynchronous web scraping, as well as the impact of this approach on the development of other areas of information technology. Taking these aspects into account will contribute to the further evolution and optimization of web scraping technologies for a wide range of applications.

Keywords: asynchronous programming; Web scraping; Python; Requests; Asyncio; Aiohttp; BeautifulSoup.