DOI 10.31891/2307-5732-2023-327-5-243-248
УДК 004.

СУШИНСЬКИЙ ОРЕСТ
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0000-0002-2661-6458
e-mail: orest.sushynskyi@e-u.edu.ua
КОЦУН ВОЛОДИМИР
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0000-0003-2363-8157
e-mail: volodumur.kotsun@e-u.edu.ua
СКЛЯРЕНКО ОЛЕНА
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0000-0001-6555-1223
e-mail: olena.skliarenko@e-u.edu.ua
ЛИТВИНЕНКО ЛЕОНІД
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0000-0002-0828-383X
e-mail: leonid.lytvynenko@e-u.edu.ua

ВИКОРИСТАННЯ МІКРОСЕРВІСНОГО ПІДХОДУ В ПРОЦЕСІ ВЕБ-СКРАПІНГУ ВЕЛИКИХ ОБСЯГІВ ДАНИХ ДЛЯ ВЕБ-САЙТІВ ІЗ ДИНАМІЧНИМ ВМІСТОМ

Проблема веб-скрапінгу виникає у зв'язку зі зростанням складності веб-сторінок, які використовують динамічний вміст, згенерований за допомогою JavaScript. Це ускладнює процес збору даних, оскільки стандартні методи HTTP-запитів не можуть отримати повний вміст сторінки. Мікросервісна архітектура може бути використана для вирішення цієї проблеми, оскільки дозволяє розподілити завдання між невеликими, незалежними сервісами. Аналіз досліджень та публікацій показує, що методи сканування веб-сторінок, які зазвичай використовуються, можуть займати багато часу при скануванні великих обсягів даних. Для вирішення цієї проблеми використовуються різні підходи, такі як швидкий двигун XPath селектора. Метою статті є дослідження особливостей використання мікросервісного підходу в процесі веб-скрапінгу та розгляд основних переваг мікросервісної архітектури. У статті будуть досліджені особливості використання різних підходів у доступі до елементів веб-сайту, зокрема увага буде приділена методам CSS селектори, Regex та XPath. Дослідження показало, що мікросервісна архітектура може покращити продуктивність системи, але може призвести до збільшення часу виконання завдань. Вимірювання показників ефективності показали, що метод Regex має найменше використання CPU і пам'яті порівняно з іншими методами, а метод XPath забезпечує вищу надійність та стійкість.

Ключові слова: мікросервіс, веб-скрапінг, дані

SUSHYNSKYI OREST , KOTSUN VOLODYMYR, SKLIARENKO OLENA, LYTVYNENKO LEONID
Private higher education institution “European University”

USING A MICROSERVICE APPROACH IN THE PROCESS OF WEB SCRAPING OF LARGE VOLUMES OF DATA FOR WEBSITES WITH DYNAMIC CONTENT

One of the main challenges of web scraping is handling dynamic content. Modern websites often use technologies such as AJAX and JavaScript to dynamically update content without reloading the page. The problem of web scraping arises from the increasing complexity of web pages that use dynamic content generated by JavaScript. This complicates the data collection process, as standard HTTP request methods cannot retrieve the full content of the page. Microservice architecture can solve this problem because it allows tasks to be distributed among small, independent services. Research and publications analysis shows that commonly used web scraping techniques can be time-consuming when scanning large amounts of data. Various approaches are used to solve this problem, such as the fast XPath selector engine. The average reliability and resilience of XPath is 96% of successful requests and increases to 98% when using microservices. XPath provides higher reliability and resilience than other methods. The CSS Selector method is the smallest in terms of bandwidth usage compared to other methods. Using microservice processing methods can provide higher reliability and resilience when parsing large amounts of data, but will require an increase in execution time. The article aims to study the features of the microservice approach in the process of web scraping and consider the main advantages of microservice architecture. The article will explore the peculiarities of using different approaches in accessing website elements, in particular, attention will be paid to the methods of CSS selectors, Regex, and XPath. The study found that microservice architecture can improve system performance but can lead to longer turnaround times. Performance measurements have shown that the Regex method has the lowest CPU and memory usage compared to other methods, and the XPath method provides higher reliability and resilience.

Keywords: microservice, web scraping, data

Постановка проблеми у загальному вигляді та її зв’язок із важливими науковими чи практичними завданнями

Веб-скрапінг є важливим інструментом для збору великих обсягів даних з різноманітних веб-сайтів. Ці дані можуть використовуватись для аналітики, досліджень ринку, моніторингу конкурентів та багатьох інших цілей. Однак, зі зростанням складності веб-сторінок, що використовують динамічний вміст, згенерований за допомогою JavaScript, традиційні методи веб-скрапінгу стають менш ефективними. Це створює потребу в нових підходах та інструментах, які здатні ефективно обробляти такі динамічні веб-сторінки.

Мікросервісний підхід до розробки програмного забезпечення набув значної популярності завдяки своїй гнучкості, масштабованості та надійності. Цей підхід передбачає розбиття великої монолітної програми

на набір невеликих, незалежних сервісів, кожен з яких виконує окрему функцію. Використання мікросервісної архітектури для веб-скрапінгу дозволяє ефективно вирішувати проблеми, пов'язані з динамічним вмістом, та забезпечує можливість обробки великих обсягів даних.

Однією з основних проблем веб-скрапінгу є обробка динамічного вмісту. Сучасні веб-сайти часто використовують технології, такі як AJAX та JavaScript, для динамічного оновлення контенту без перезавантаження сторінки. Це ускладнює процес збору даних, оскільки стандартні методи HTTP-запитів не можуть отримати повний вміст сторінки. Для вирішення цієї проблеми використовуються інструменти, які можуть взаємодіяти з браузером та виконувати JavaScript, такі як Selenium та Puppeteer. Ці інструменти дозволяють завантажувати сторінки так само, як це робить користувач, та отримувати динамічний вміст.

Ще однією важливою проблемою є масштабованість. Збір великих обсягів даних з багатьох веб-сайтів може вимагати значних обчислювальних ресурсів та часу. Традиційні монолітні підходи до веб-скрапінгу можуть бути недостатньо ефективними для обробки таких обсягів даних. Для вирішення цієї проблеми використовуються розподілені системи та мікросервісна архітектура, які дозволяють розподілити навантаження між кількома сервісами та забезпечити паралельне виконання завдань.

Аналіз досліджень та публікацій

Сьогодні збір даних став необхідністю, особливо тому, що багато джерел даних в Інтернеті можна використовувати для різних потреб. Існують різні методи сканування веб-сторінок, які зазвичай використовуються. Обсяг даних, розкиданих в Інтернеті, займе досить багато часу, якщо сканування веб-сайту проводитиметься у великих масштабах [1].

Запит даних з форматів представлення, таких як HTML, для таких цілей, як вилучення інформації, вимагає розгляду деревовидних структур, а також врахування просторових зв'язків між розкладеними елементами. Основне обґрунтування полягає в тому, що часто візуалізація деревоподібних структур є дуже складною і оновлюється частіше, ніж результуюча структура макета [2].

Швидкий двигун XPath селектора, який виконує запити до деревовидної структури над стислими індексами XML представлено в роботі [3]. В цій роботі досліджуються причини, чому XPath такий швидкий. Показано, що деревовидні структури можна використовувати як загальну структуру для детальної оптимізації XML-запитів. Авторами було визначено «релевантні вузли» запиту як ті вузли, яких мінімальний автомат повинен торкнутися, щоб відповісти на запит. Це поняття дозволяє пропускати багато піддерев під час виконання, а за допомогою певних індексів дерева дозволяє навіть пропускати внутрішні вузли дерева.

Формулювання цілей статті

Метою статті є дослідження особливостей використання мікросервісного підходу в процесі веб-скрапінгу великих обсягів даних з веб-сайтів із динамічним вмістом. У статті буде розглянуто основні переваги мікросервісної архітектури, описано архітектуру мікросервісного рішення для веб-скрапінгу. У статті буде досліджено особливості використання різних підходів у доступі до елеметів веб-сайту, зокрема увага буде приділена наступним методам: CSS селектори, Regex та XPath у контексті оптимізації продуктивності мікросервісних систем у процесі веб-скрапінгу.

Викладення основного матеріалу дослідження

Веб-скрапінг є потужним інструментом для збору даних, але його ефективне використання вимагає врахування багатьох технічних, етичних та правових аспектів. Використання сучасних інструментів та підходів, таких як мікросервісна архітектура, дозволяє вирішувати складні завдання веб-скрапінгу та забезпечувати високу продуктивність та надійність систем.

Мікросервісна архітектура є сучасним підходом до розробки програмного забезпечення, який передбачає розбиття великої монолітної програми на набір невеликих, незалежних сервісів. Кожен мікросервіс виконує окрему функцію та взаємодіє з іншими сервісами через чітко визначені інтерфейси, зазвичай за допомогою HTTP або повідомлень. Такий підхід дозволяє значно підвищити гнучкість, масштабованість та надійність програмних систем.

Однією з основних переваг мікросервісної архітектури є її здатність до масштабування. Оскільки кожен мікросервіс є незалежним, його можна масштабувати окремо від інших сервісів. Це дозволяє ефективно розподілити навантаження та забезпечити високу продуктивність системи. Наприклад, якщо один з мікросервісів відповідає за обробку великого обсягу даних, його можна розгорнути в кількох екземплярах для паралельного виконання завдань.

Реалізація мікросервісного підходу починається з розробки архітектури системи, яка враховує розділення функціональності на окремі сервіси. Кожен мікросервіс повинен виконувати чітко визначену задачу, наприклад, завантаження веб-сторінок, парсинг HTML-коду, збереження даних та обробка помилок. Важливо забезпечити чітко визначені інтерфейси для взаємодії між мікросервісами, що дозволить їм легко обмінюватися даними та координувати свої дії.

Для завантаження веб-сторінок з динамічним вмістом часто використовуються інструменти, такі як Selenium або Puppeteer, які дозволяють взаємодіяти з браузером та виконувати JavaScript. Цей мікросервіс може бути реалізований як окремий контейнер, який отримує URL-адреси сторінок, завантажує їх та передає HTML-код наступному мікросервісу для парсингу. Наприклад, можна написати простий скрипт на Python з використанням бібліотеки Selenium, який буде запускати браузер, завантажувати сторінку та зберігати її вміст.

Парсинг HTML-коду може бути реалізований за допомогою таких бібліотек, як Cheerio. Цей мікросервіс отримує HTML-код від сервісу завантаження сторінок, аналізує його та витягує необхідні дані. Важливо забезпечити гнучкість парсера, щоб він міг адаптуватися до змін у структурі веб-сторінок. Наприклад, можна використовувати CSS-селектори, Regex та XPath для визначення елементів, які потрібно витягнути. Результати парсингу передаються наступному мікросервісу для збереження даних.

Рис.1. Архітектура системи для реалізації мікросервісного підходу в процесі веб-скрапінгу великих обсягів даних з веб-сайтів із динамічним вмістом

Збереження даних може бути реалізовано за допомогою різних баз даних, залежно від вимог до зберігання та обробки інформації. Наприклад, для структурованих даних можна використовувати реляційні бази даних, такі як PostgreSQL. Цей мікросервіс отримує дані від парсера та зберігає їх у базі даних, забезпечуючи можливість швидкого пошуку та аналізу.

Для управління контейнерами та оркестрації мікросервісів доцільно використовувати інструменти, такі як Kubernetes. Kubernetes, автоматизує розгортання, масштабування та управління контейнерами, що значно спрощує адміністрування складних систем. За допомогою Kubernetes можна легко налаштувати автоматичне масштабування мікросервісів, балансування навантаження та відновлення після збоїв.

Як було зазначено вище метою статі було визначення методів необхідних для реалізації процесу парсингу із використанням мікросервісного підходу. А саме використання методів таких як CSS селектори, Regex та XPath у контексті оптимізації продуктивності мікросервісних систем у процесі веб-скрапінгу.

- XPath (XML Path Language) — це мова запитів для вибору частин (вузлів) XML-документа. Окрім вибору, XPath також можна використовувати для обчислення таких значень, як рядки, числа та логічні значення у файлі XML та HTML . Консорціум World Wide Web (W3C) навіть встановив стандарти для використання XPath.
- CSS Selector — це метод пошуку HTML-елементів на веб-сторінках і вилучення з них даних. Селектор CSS оголошується як частина стилю розмітки, який застосовується для відповідності тегам і атрибутам у розмітці.
- Регулярний вираз (Regex) — це мовна конструкція для зіставлення тексту на основі певних шаблонів, особливо для складних випадків. Регулярне вираз також використовується для відповідності певним шаблонам символів у наборі рядків. Regex має два види символів, а саме звичайні символи та метасимволи.

У ході проведених досліджень було взято за мету визначення ряду показників що характеризують ефективність процесу парсингу.

1. Час виконання (Execution Time). Загальний час виконання: Час, який витрачається на повний цикл парсингу, від початку до кінця. Час на запит: Час, витрачений на отримання HTML-коду веб-сторінки. Час на обробку: Час, витрачений на парсинг та обробку отриманих даних.

2. Пропускна здатність (Throughput). Кількість запитів в секунду (RPS): Кількість запитів, які система може обробити за одну секунду. Кількість оброблених сторінок: Кількість веб-сторінок, які система може обробити за певний період часу.

3. Використання ресурсів (Resource Utilization). Центральний процесор (CPU) та пам'ять (RAM): Вимірювання використання CPU та RAM під час процесу парсингу. Використання диску: Оцінка використання дискового простору для зберігання даних.

4. Надійність та стійкість (Reliability and Robustness). Кількість помилок: Кількість помилок, які виникають під час парсингу. Відсоток успішних запитів: Відсоток запитів, які були успішно виконані без помилок.

5. Масштабованість (Scalability). Масштабованість системи: Здатність системи ефективно обробляти збільшене навантаження, наприклад, збільшення кількості паралельних запитів.

6. Затримки (Latency). Затримка відповіді: Час, який проходить від моменту відправлення запиту до отримання відповіді.

Нижче наведено фрагменти коду для вимірювання продуктивності із використанням Python.

```python
import time
from bs4 import BeautifulSoup
import requests

start_time = time.time()

response = requests.get('https://example.com')
html = response.text
soup = BeautifulSoup(html, 'html.parser')
data = soup.find_all('p')

end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")
```

Рис. 2.Вимірювання часу виконання

```python
import time
import requests

def fetch_page(url):
    response = requests.get(url)
    return response.status_code

urls = ['https://example.com'] * 100  # Список URL для тестування
start_time = time.time()

for url in urls:
    fetch_page(url)

end_time = time.time()
total_time = end_time - start_time
throughput = len(urls) / total_time
print(f"Throughput: {throughput} requests per second")
```

Рис.3. Вимірювання пропускної здатності

Для вимірювання використання CPU та RAM було використано бібліотеку psutil.

```python
import psutil
import time

start_cpu = psutil.cpu_percent(interval=None)
start_memory = psutil.virtual_memory().percent

# Виконання парсингу
time.sleep(2)  # Імітація процесу парсингу

end_cpu = psutil.cpu_percent(interval=None)
end_memory = psutil.virtual_memory().percent

cpu_usage = end_cpu - start_cpu
memory_usage = end_memory - start_memory

print(f"CPU Usage: {cpu_usage}%")
print(f"Memory Usage: {memory_usage}%")
```

Рис.4. Вимірювання використання ресурсів

Таблиця 1 відображає середньо апроксимовані дані основних параметрів під час виконання веб-парсингу для кожного методу. Дані були розраховані із використання мікросервісів та без них. Розрахунки були проведені для парсингу великих об'ємів даних. CSS селектор використовує в середньому 60,3% CPU, і спостерігається збільшення в середньому до 75,6% при застосуванні мікросервісів, Regex використовує в середньому 35,4% CPU і збільшення в середньому до 65,2% при застосуванні мікросервісів, тоді як XPath використовує в середньому 55,8% процесора, а при застосуванні мікросервісів спостерігається збільшення в середньому до 75,8%.

За результатами експерименту отримано такі дані: для методу CSS селектор потрібен середній час виконання 542,5 мілісекунд, а час виконання зростає в середньому до 590,4 мілісекунди, при використанні мікросервісів, Regex вимагає середній час виконання 200,3 мілісекунд, а час виконання зростає в середньому

до 453,6 мілісекунди, коли застосовано мікросервіси, тоді як для Regex Xpath середній час виконання становить 353,8 мілісекунд, а час виконання зростає до 550,7 мілісекунди, коли застосовуються мікросервіси.

Таблиця 1

Середньо апроксимовані дані основних параметрів під час виконання веб-парсингу для кожного методу

	CSS-селектори		**Регулярні вирази (Regex)**		**XPath**	
	Із використанням мікросервісів	Без використання мікросервісів	Із використанням мікросервісів	Без використання мікросервісів	Із використанням мікросервісів	Без використання мікросервісів
Час виконання (Execution Time)	590,4 мс	542,5 мс	453,6 мс	200,3 мс	550,7 мс	353,8 мс
Пропускна здатність (Throughput)	70 запитів/сек	90 запитів/сек	75 запитів/сек	120 запитів/сек	35 запитів/сек	75 запитів/сек
Використання ресурсів (Resource Utilization)	CPU - 75,6%, RAM - 550 МБ	CPU - 60,3%, RAM - 540 МБ	CPU - 65,2%, RAM - 400 МБ	CPU - 35,4%, RAM - 200 МБ	CPU - 75,8%, RAM - 600 МБ	CPU - 55%, RAM - 400 МБ
Надійність та стійкість (Reliability and Robustness)	92% успішних запитів, кожен сервіс ізольований і може бути незалежно перезапущений	91% успішних запитів, залежить від стабільності HTML структури	95% успішних запитів, кожен сервіс ізольований і може бути незалежно перезапущений	90% успішних запитів, залежить від стабільності HTML структури	98% успішних запитів, кожен сервіс ізольований і може бути незалежно перезапущений	96% успішних запитів, залежить від стабільності HTML структури
Затримки (Latency)	600 мс	500 мс	450 мс	200 мс	600 мс	300 мс

Також слід зазначити показник надійності та стійкості при використанні мікросервісів. Загалом всі методи показали високі показники. Але середнє значення надійності та стійкості XPath становить 96% успішних запитів і зростає до 98% при використанні мікросервісів.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Виходячи з експериментальних даних, метод Regex має найменше використання CPU і пам’яті порівняно з методами CSS Selector та XPath. Тоді як XPath забезпечує вищу надійсть та стійкість порівняно з іншими методами. Метод CSS Selector є найменшим з точки зору використання пропускної здатності порівняно з іншими методами. Застосування методів мікросервісної обробки може забезпечити вищу надійсть та стійкість у процесі парсингу великого об’єму даних, але потребуватиме зростання часу виконання. Мікросервіси можуть мати більші затримки та використання ресурсів через накладні витрати на комунікацію та обробку великого об'єму даних, вони забезпечують високу надійність та гнучкість у масштабуванні окремих компонентів системи.

Література

1. Darmawan, I., Maulana, M. F., Gunawan, R., & Widiyasono, N. (2022). Оцінка продуктивності веб-скрейпінгу за допомогою XPath, селектора CSS, регулярного виразу та HTML DOM із багатопроцесорними технічними програмами. 6(4). https://doi.org/10.30630/joiv.6.4.1525 [Англійською].

2. Oro, E., Ruffolo, M., & Staab, S. (2010). SXPath: розширення XPath для просторових запитів у веб-документах. 4(2). https://doi.org/10.14778/1921071.1921079. [Англійською].

3. Maneth, S., & Nguyen, K.. (2010). Оптимізація всіх запитів XPath. 3(1). https://doi.org/10.14778/1920841.1920954. [Англійською].

4. Rizaldi, T., & Putranto, H. A. (2017). Порівняння методів веб-скрейпінгу за допомогою селектора CSS і селектора Xpath 6(1). https://doi.org/10.34148/TEKNIKA.V6I1.56. [Англійською].

5. Ramanan, P.. (2012). Переписування запитів XPath за допомогою матеріалізованих представлень XPath. 78(4). https://doi.org/10.1016/J.JCSS.2011.12.001. [Англійською].

6. Shanmughaneethi, Ravichandran & Swamynathan, (2011). PXpathV: запобігання вразливостям XPath у веб-додатках. 2(3). https://doi.org/10.5121/IJWSC.2011.2305. [Англійською].

7. Liu, H., & Ma, Y. X.. (2011). Дослідження вилучення веб-даних на основі технології Wrapper і XPath. https://doi.org/10.4028/WWW.SCIENTIFIC.NET/AMR.271-273.706. [Англійською].

8. Björklund, H., Gelade, W., & Martens, W.. (2010). Інкрементна оцінка XPath. 35(4). https://doi.org/10.1145/1862919.1862926. [Англійською].

9. Скляренко О. В., Федік О.І., Колодінська Я.О. Використання Big data: сучасні можливості для бізнесу //Журнал «Економіка і управління» №3,- 2020-С.106-110.

References

1. Darmawan, I., Maulana, M. F., Gunawan, R., & Widiyasono, N.. (2022). Evaluating Web Scraping Performance Using XPath, CSS Selector, Regular Expression, and HTML DOM With Multiprocessing Technical Applications. 6(4). https://doi.org/10.30630/joiv.6.4.1525

2. Oro, E., Ruffolo, M., & Staab, S.. (2010). SXPath: extending XPath towards spatial querying on web documents. 4(2). https://doi.org/10.14778/1921071.1921079

3. Maneth, S., & Nguyen, K.. (2010). XPath whole query optimization. 3(1). https://doi.org/10.14778/1920841.1920954

4. Rizaldi, T., & Putranto, H. A.. (2017). Perbandingan Metode Web Scraping Menggunakan CSS Selector dan Xpath Selector. 6(1). https://doi.org/10.34148/TEKNIKA.V6I1.56

5. Ramanan, P.. (2012). Rewriting XPath queries using materialized XPath views. 78(4). https://doi.org/10.1016/J.JCSS.2011.12.001

6. Shanmughaneethi, Ravichandran, & Swamynathan, (2011). PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications. 2(3). https://doi.org/10.5121/IJWSC.2011.2305

7. Liu, H., & Ma, Y. X.. (2011). Web Data Extraction Research Based on Wrapper and XPath Technology. https://doi.org/10.4028/WWW.SCIENTIFIC.NET/AMR.271-273.706

8. Björklund, H., Gelade, W., & Martens, W.. (2010). Incremental XPath evaluation. 35(4). https://doi.org/10.1145/1862919.1862926.

9. Skliarenko O.V., Fedik O.I., Kolodinska Y.O. Using Big Data: Modern Business Opportunities // Journal "Economics and Management" №3, 2020, pp. 106-110.