2. Кітораги В. О. Програмний асистент для підтримки роботи тестувальників : кваліфікаційна робота. Черкаси : ЧДБК, 2026.
3. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.
4. Wang J., Huang Y., Chen C. та ін. Software Testing with Large Language Model: Survey, Landscape, and Vision. arXiv:2307.07221. 2023. URL: https://arxiv.org/abs/2307.07221 (дата звернення: 17.03.2026).
5. Hnatushenko V. V., Pavlenko I. V. Використання генеративного штучного інтелекту в тестуванні програмного забезпечення. Системні технології. 2024. Вип. 2(151). С. 10–20.

УДК 004.4

ОПТИМІЗАЦІЯ ПРОЦЕСУ АВТОМАТИЗОВАНОГО WEB-ПАРСИНГУ ЗА ДОПОМОГОЮ БАГАТОПОТОКОВОСТІ

Прудиус В.М.
agyshii@gmail.com
Черкаський державний технологічний університет
Метелап В.В.
м. Черкаси, Україна

Сучасний етап розвитку інформаційних технологій характеризується експоненційним зростанням обсягів даних у мережі Інтернет. Для аналізу ринку, машинного навчання, агрегації новин та інших науково-дослідних завдань критично важливим є інструментарій автоматизованого збору інформації, відомий як web-парсинг (web scraping). Традиційний однопотоковий підхід до реалізації подібних алгоритмів має суттєвий недолік – він характеризується значними часовими витратами та низьким коефіцієнтом корисної дії апаратного забезпечення. Це пов'язано із проблемою синхронного блокування (I/O Bound операції), при якому програма, відправивши мережевий запит, зупиняє своє виконання і більшу частину часу просто очікує на відповідь від цільового сервера та завантаження HTML-документа, тоді як ресурси центрального процесора (CPU) залишаються абсолютно незадіяними [1].

Ефективним та раціональним вирішенням цієї проблеми є впровадження архітектури багатопотоковості (Multithreading) або паралельних обчислень. Багатопотоковий парсинг базується на принципі конкурентного виконання завдань, що дозволяє ініціювати десятки або сотні мережевих запитів одночасно у межах одного процесу. Поки один потік очікує на відповідь від сервера, операційна система перемикає контекст на інший потік, який у цей час може здійснювати парсинг вже отриманого DOM-дерева або відправляти новий запит.

Оптимізація процесу за допомогою пулу потоків (Thread Pool) дозволяє масштабувати швидкість збору даних практично лінійно до певної межі, яка визначається пропускною здатністю мережевого каналу. Порівняльну ефективність застосування різних підходів до автоматизованого збору інформації на масиві з тисячі web-сторінок наведено у таблиці (табл. 1). Як видно з даних, застосування багатопотоковості зменшує час виконання завдання майже на порядок.

Таблиця 1 – Порівняння швидкодії та ресурсоємності методів web-парсингу

№	Метод обробки даних	Час виконання (1000 сторінок)	Завантаження CPU	Споживання ОЗП
1	Однопотоковий (Синхронний)	320 с	До 5%	50 МБ
2	Багатопотоковий (10 потоків)	38 с	15-20%	120 МБ
3	Багатопотоковий (50 потоків)	12 с	40-50%	350 МБ
4	Асинхронний (Event Loop)	9 с	25-30%	90 МБ

Проте, практична реалізація багатопотокового парсингу зіштовхується із серйозними перешкодами. Висока інтенсивність мережевих запитів з однієї IP-адреси миттєво ідентифікується системами кіберзахисту цільового ресурсу (наприклад, Web Application Firewalls – WAF, Cloudflare або AWS Shield) як потенційна DDoS-атака або нелегітимна активність бота [2]. Це призводить до

появи CAPTCHA, тимчасового або перманентного блокування доступу. Для нівелювання цих ризиків розробникам доводиться проектувати складну розподілену архітектуру із залученням проміжних вузлів (рис. 1).

Варто зазначити, що для успішного уникнення блокувань недостатньо лише змінити IP-адресу. Системи антифрод-захисту аналізують патерни поведінки та частоту звернень. Тому критично важливим є впровадження механізмів штучних затримок (Throttling). Математично, безпечний час затримки між запитами, який гарантує неперевищення лімітів цільового сервера, можна розрахувати за формулою (1):

$$T_{безпеч.} = (N_{потоків} \times T_{відповіді}) / K_{допуск}, \quad (1)$$

де:

$T_{безпеч.}$ – необхідний інтервал між зверненнями (у секундах);

$N_{потоків}$ – кількість одночасно активних робочих потоків;

$T_{відповіді}$ – середній час очікування відповіді від сервера;

$K_{допуск}$ – емпіричний коефіцієнт толерантності сервера до навантаження (зазвичай від 0.5 до 0.8).

Рисунок 1 – Архітектурна схема взаємодії багатопотокового парсера із цільовим сервером через ротацію проксі-пулу

Отже, проектування оптимізованого багатопотокового парсера вимагає комплексного дотримання наступних інженерних практик:

- використання обмежених пулів потоків на базі черг завдань (Task Queues) замість неконтрольованого створення нових;
- обов'язкова та випадкова ротація HTTP-заголовків (User-Agent, Accept-Language) та проксі-серверів для кожного нового з'єднання;
- вирішення проблем синхронізації (Race Conditions) за допомогою потокобезпечних структур даних для збереження зібраної інформації без втрати її цілісності;
- впровадження системи експоненційної затримки (Exponential Backoff) для коректної обробки мережевих помилок (коди 429, 503) та таймаутів.

Список використаних джерел

1. Мітчелл Р. Скрапінг web-сайтів із використанням Python. Київ: Видавництво, 2021. 300 с.
2. Вилучено з https://liga.science/conferences.html ; 2020 ГО Молодіжна наукова ліга.

УДК 004.738.5

РОЗРОБКА ФРОНТЕНДУ ЗА ДОПОМОГОЮ РІЗНИХ ФРЕЙМВОРКІВ: ПЕРЕВАГИ, НЕДОЛІКИ ТА ЗАВДАННЯ

Стеценко Я. І.
yanastetforcomp@gmail.com
Черкаський державний фаховий бізнес-коледж
Подорошко Д. І.
м. Черкаси, Україна

Сучасна розробка фронтенду є фундаментальною складовою створення прикладних рішень, що відповідають високим вимогам користувачів щодо швидкодії, масштабованості та безпеки. Еволюція вебтехнологій призвела до домінування односторінкових додатків (SPA), які забезпечують миттєву реакцію