СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____

Кафедра _____інформаційних технологій та програмування_____

Пояснювальна записка

до магістерської дипломної роботи

_____магістр_____

(освітньо-кваліфікаційний рівень)

на тему: ___Методи вдосконалення продуктивності PHP через оптимізацію веб-серверної взаємодії_____

Виконав: студент 2 курсу, групи ІСТ-23дм___
126 «Інформаційні системи та технології_____

(шифр і назва спеціальності)

_____Аблашимов В. І._____

(прізвище та ініціали)

Керівник _____Лифар В.О._____

(прізвище та ініціали)

Рецензент_____Меняйленко О.С._____

(прізвище та ініціали)

Київ – 2024 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____
Кафедра _____інформаційних технологій та програмування_____
Освітньо-кваліфікаційний рівень __магістр_____
Спеціальність 126 «Інформаційні системи та технології»_____
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ д.т.н., доц. Захожай О.І.
(підпис)
« ___ » _____2024 р.

ЗАВДАННЯ

на магістерську дипломну роботу студенту
_____Аблашимов Владислав Ігорович_____
(прізвище, ім'я, по батькові)

1.Тема роботи: Методи вдосконалення продуктивності PHP через оптимізацію веб-серверної взаємодії,_____
керівник роботи доцент, д.т.н. Лифар Володимир Олексійович ,
(вчене звання, науковий ступінь, прізвище, ім'я, по батькові)
затверджені наказом університету від «_06_»_12_ 2024року №361/15.15-С_
2. Строк подання студентом роботи: 10 грудня 2024 р._____
3. Вихідні дані до роботи: Матеріали науково-дослідної практики, науково-методична література; дані інтернет-мережі . _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ
4.2 Аналітичний огляд питання (огляд публічних джерел інформації)
4.3 Основна частина, в якої висвітлити методи, які будуть використовуватися для реалізації проекту.
4.4 Практична частина – огляд технологій, які використовуються під час реалізації проекту.
4.4 Висновки
4.5 Перелік використаних джерел

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____
_____

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання __08 листопада 2024р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Одержання завдання на виконання роботи	08.11.2024	
2.	Укладання і погодження з керівником плану і етапів виконання роботи	11.11.2023	
3.	Узагальнення даних літературних джерел	13.11.2023	
4.	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху виконання завдання	15.11.2023	
5.	Аналіз технічних засобів та існуючих систем	18.11.2023	
6.	Реалізація практичної частини завдання	24.11.2023	
7.	Укладання, оформлення та погодження пояснювальної записки з керівником	06.12.2023	
8.	Надання пояснювальної записки на кафедру	10.12.2023	
9.	Підготовка доповіді та презентації	11.12.2023	

Студент _____ Аблашимов В. І._
(підпис) (прізвище та ініціали)

Керівник роботи _____ _Лифар В.О.__
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дипломна робота: стор.68, табл.4, рис.27, джерел 14.

Об'єкт дослідження – вдосконалення продуктивності PHP. Предмет дослідження – методи взаємодії з мовою програмування PHP.

Мета дослідження – знайти оптимальні методи запуску скриптів на PHP для більшої продуктивності. Дослідження спрямоване на вивчення та аналіз існуючих методів веб-серверної взаємодії, проведення тестування та замірів відповідних рішень.

Для досягнення поставленої мети, дипломна робота поділена на такі завдання:

1) Аналіз предметної області;

2) Розробка та налаштування методів веб-серверної взаємодії;

3) Розробити навантажувальні тести;

4) Виявити найкращі методі веб-серверної взаємодії.

КЛЮЧОВІ СЛОВА: PHP, ІНФОРМАЦІЙНА СИСТЕМА, ТЕСТУВАННЯ, БЕЗПЕКА ДАНИХ, ВЕБ СЕРВЕР, PHP, APACHE JMETER

ABSTRACT

Master’s Thesis: Page 68, Table 4, Figure 27, References 14.

Object of Study: Optimizing PHP Performance.

Subject of Study: Methods of Interaction with the PHP Programming Language.

Objective: To identify optimal methods for executing PHP scripts to enhance performance. The study focuses on examining and analyzing existing web server interaction methods, conducting testing, and measuring relevant solutions.

To achieve the set goal, the thesis is divided into the following tasks:

1. Analysis of the subject area;
2. Development and configuration of web server interaction methods;
3. Development of load tests;
4. Identification of the best web server interaction methods.

KEYWORDS: PHP, INFORMATION SYSTEM, TESTING, DATA SECURITY, WEB SERVER, PHP, APACHE JMETER

ЗМІСТ

ВСТУП

PHP використовується на приблизно 75.7% усіх вебсайтів, що застосовують серверну мову програмування, згідно з даними W3Techs станом на жовтень 2024 року. Такий значний відсоток відображає широку популярність PHP для створення динамічних сайтів з великим обсягом контенту, зокрема платформ із високим трафіком, таких як WordPress, Facebook і Wikipedia. Він залишається домінуючим завдяки доступності, широкій бібліотеці інструментів та сумісності з популярними CMS, як-от WordPress та Joomla, які значною мірою використовують PHP для обробки на серверному боці. PHP також використовується для написання api на серверному боці. Ця мова програмування стала популярною через її легкість для вивчення, гарну реалізацію ООП і багате ком'юніті

Підвищення продуктивності веб сервісів на PHP залишається ключовою задачею для забезпечення швидкого доступу до інформації та якісного користувацького досвіду, особливо в умовах високого трафіку. PHP, як одна з найпопулярніших серверних мов програмування, використовується для розробки значної частини сучасних веб сайтів та додатків, що обумовлює необхідність впровадження ефективних методів для оптимізації взаємодії з веб-серверами. Основна мета таких методів — скоротити час виконання запитів, знизити навантаження на серверні ресурси і забезпечити стійку роботу веб сайтів навіть при великих навантаженнях.

З виходом кожної нової версії мова йде до більш строгих та типізованих аналогів, також додаються нові можливості та зменшується швидкість обробки, приблизні заміри різних версій:

- **PHP 5.6 vs. PHP 7.0**: Перехід від PHP 5.6 до PHP 7.0 приніс значне зростання продуктивності. У середньому, PHP 7.0 виконував обробку запитів удвічі швидше, ніж PHP 5.6, завдяки значним змінам у механізмі Zend Engine. Веб Сайти, розроблені на популярних фреймворках (наприклад, WordPress), показували зменшення часу відповіді приблизно на 50-60%.
- **PHP 7.0–7.4**: Введення PHP 7 призвело до збільшення продуктивності майже вдвічі порівняно з PHP 5.6. Час відповіді значно зменшився, становлячи в середньому 100–150 мс для аналогічних запитів. Це стало можливим завдяки введенню нового механізму обробки пам’яті Zend Engine 3, який також значно скоротив споживання пам’яті.
- **PHP 8.0**: З виходом PHP 8 було додано JIT-компілятор, який дозволяє ще більше знизити час відповіді для складних запитів, особливо у випадках інтенсивних обчислень. Зокрема, при виконанні складніших операцій, де є багато обчислень, час відповіді може зменшитися до 50–100 мс, хоча для більшості типових запитів поліпшення в порівнянні з PHP 7 є менш помітним.

Згідно з тестами, проведеними на таких фреймворках, як Laravel та Symfony, **PHP 8.1 і 8.2 можуть виконувати до 15–20% більше запитів на секунду** порівняно з PHP 7.4, і до 50% більше, ніж PHP 5.6.

Одним із найефективніших методів оптимізації є впровадження кешування, яке знижує необхідність виконання однакових обчислень для схожих запитів. PHP підтримує декілька видів кешування, таких як **опкеш** (OPCache) для збереження скомпільованих версій скриптів у пам'яті. Це дозволяє суттєво зменшити кількість звернень до диску і скоротити час

виконання запитів, адже PHP-код компілюється лише один раз, а не щоразу при зверненні до скрипту.

PHP, хоч і залишається популярною мовою для розробки веб сайтів але має низку недоліків порівняно з іншими мовами програмування, особливо в сучасному контексті розробки масштабованих та продуктивних систем. **Основні недоліки PHP включають:**

- Низька продуктивність у багатозадачних додатках: PHP створений для обробки одного запиту за раз, що може бути проблемою при розробці високонавантажених багатозадачних додатків. Хоча в PHP 8 з'явився JIT-компілятор, він все ще не забезпечує рівня продуктивності, яку дають мови на зразок Node.js або Golang, які спроектовані для асинхронного програмування і краще справляються з багатопоточністю.
- Відсутність вбудованої асинхронності: PHP не має вбудованої підтримки асинхронних операцій, таких як обробка одночасних запитів до баз даних або паралельна обробка великих обсягів даних. Асинхронна обробка в PHP можлива через додаткові бібліотеки, такі як ReactPHP або Swoole, але це ускладнює розробку.
- Слабка підтримка модульного та контейнеризованого підходу: У PHP відсутні вбудовані можливості для роботи з контейнерами та мікро сервісами, що є популярними підходами в сучасній розробці. В той час як мови, такі як Go і Rust, краще підходять для створення контейнеризованих мікросервісів, PHP потребує значних додаткових налаштувань для ефективного використання в Docker та Kubernetes середовищах.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні відомості

PHP є інтерпретованою мовою, тобто код виконується сервером у режимі реального часу, рядок за рядком. Сучасні версії PHP працюють через проміжний етап — байт код-компіляцію, яку виконує віртуальна машина Zend Engine. Це означає, що PHP-код спочатку компілюється в байт код (внутрішнє представлення), і лише потім виконується.

Як відбувається обробка коду в PHP:

1. **Лексичний аналіз (Tokenizing)**: На цьому етапі PHP-движок розбиває вихідний код на токени (мінімальні одиниці коду, як-от ключові слова, оператори, змінні). Цей процес перетворює текст PHP у серію токенів, які відображають структуру коду.
2. **Синтаксичний аналіз (Parsing)**: Токени об'єднуються в структури на основі синтаксичних правил PHP. Результатом цього є створення *абстрактного синтаксичного дерева* (Abstract Syntax Tree, AST), яке являє собою структуроване подання коду.
3. **Генерація байткоду:** AST перетворюється у внутрішнє представлення, яке називається *байткодом*. Байткод — це проміжний код, зрозумілий віртуальній машині Zend Engine. Байткод є компактнішим і дозволяє швидше виконувати код, оскільки його не потрібно знову розбирати і перевіряти.
4. **Кешування** (опціонально): PHP може кешувати байткод за допомогою *опкешу (Opcache)*. Якщо код не змінюється, кешований байткод зберігається в пам'яті, що дозволяє зекономити час на повторній

компіляції при наступних запитах. Це значно підвищує продуктивність у високонавантажених додатках.

5. **Інтерпретація байткоду:** цей процес виконується віртуальною машиною Zend Engine: Байткод виконується інтерпретатором Zend Engine, який обробляє кожну інструкцію байткоду та виконує необхідні операції, такі як робота із змінними, виклики функцій і робота з пам’яттю.
6. **Очищення пам'яті:** Після відправки відповіді PHP очищує всі виділені ресурси (пам'ять, відкриті файли та підключення). Це частина моделі одноразового виконання коду, що означає: кожен запит починає і завершує своє виконання незалежно, очищуючи використану пам'ять.

Те що PHP інтерпретована мова сприяє швидкому налаштуванню та тестуванню веб-додатків. З моменту свого створення PHP розвивався як мова для інтеграції з HTML, а його синтаксис і можливості оптимізовані для обробки HTTP-запитів, роботи з базами даних, сесій, файлів і різних веб-сервісів. PHP особливо популярний серед малих і середніх проектів завдяки простоті та відкритому коду. Його застосовують для розробки:

- динамічних веб-сайтів,
- систем керування контентом (CMS),
- платформ електронної комерції,
- мікросервісів і API,
- додатків, що працюють із базами даних.

За даними багатьох досліджень, PHP використовується на понад 75% веб-сайтів, зокрема таких великих ресурсів, як Facebook, Wikipedia, і

WordPress. PHP також активно підтримується спільнотою, що дозволяє йому зберігати актуальність та ефективність.

Переваги PHP як інтерпретованої мови

- Гнучкість та зручність для розробників: Внесення змін у код можна перевірити миттєво, без компіляції, що прискорює розробку та налаштування.
- Модульність і швидкість розгортання: PHP швидко інтегрується в будь-яке серверне середовище і має розгалужену підтримку бібліотек та фреймворків (Laravel, Symfony, CodeIgniter).

Недоліки інтерпретованої архітектури:

- Продуктивність: Інтерпретовані мови, як правило, повільніші за компільовані, особливо в високонавантажених застосунках.
- Безпека: Простота PHP може призводити до вразливостей у коді, якщо не дотримуватися кращих практик.

PHP з часом став підтримувати такі важливі технології, як об’єктно-орієнтоване програмування (ООП), асинхронність і багатопоточність (через Swoole та інші інструменти), що розширило його можливості. Крім того, оптимізації у версіях PHP 7 та 8 значно збільшили його продуктивність, знижуючи навантаження на сервери і покращуючи час відповіді для великих додатків.

Таким чином, PHP залишається вагомим інструментом у сфері розробки серверної частини, і аналіз методів його оптимізації є важливим кроком для підвищення ефективності веб-сервісів, побудованих на ньому.

1.2 Основні методи використання

1.2.1 Apache HTTP Server з mod_php

Модуль mod_php для Apache, дозволяє безпосередньо інтегрувати PHP-інтерпретатор в Apache HTTP Server. Він завантажує PHP як частину процесу Apache, що забезпечує швидке виконання PHP-коду, оскільки PHP працює в одному потоці з Apache і не потребує додаткового проксування через протокол (як у випадку з FastCGI).

Це один з найстаріших способів запуску PHP. Модуль mod_php дозволяє Apache працювати з PHP без необхідності додаткових програм або серверів. При використанні mod_php, PHP запускається як частина Apache-процесу, що дозволяє швидко обробляти HTTP-запити. Однак цей метод менш ефективний з точки зору пам'яті, оскільки кожен процес Apache завантажує PHP, що може бути надмірним для високонавантажених сайтів. Цей підхід має свої переваги та недоліки, тому використовується сьогодні рідше, поступаючись іншим методам (наприклад, PHP-FPM).

Переваги:

- Легкість налаштування: Встановлення та налаштування mod_php часто простіша, ніж налаштування PHP-FPM або інших рішень.
- Швидкий доступ до PHP-коду: Оскільки PHP працює в одному процесі з Apache, цей метод має низькі накладні витрати на комунікацію між сервером і PHP-інтерпретатором.

- Стабільність для невеликих проектів: Для проектів з невеликою кількістю користувачів або де продуктивність не є критичною, mod_php забезпечує стабільну і надійну роботу.
- Вища продуктивність для динамічних сайтів: Оскільки PHP працює в межах того ж процесу Apache, це дозволяє швидше обробляти запити, ніж запуск окремого процесу CGI для кожного запиту.
- Підтримка модулів Apache: Всі модулі Apache, такі як mod_rewrite, mod_ssl, можуть використовуватись без додаткових налаштувань.
- Можливість налаштування на рівні директорії: Ви можете використовувати .htaccess файли для налаштування серверу і конфігурацій PHP без перезавантаження Apache.
- Підтримка сесій і кешування: Легко інтегруються механізми кешування та збереження сесій, оскільки PHP працює в одному процесі з Apache.

Недоліки:

- Високі вимоги до пам’яті: Кожен процес Apache включає PHP, що призводить до значного використання пам’яті, особливо при високій кількості одночасних запитів.
- Мульти-потоковість: Apache з mod_php працює краще в префорк-режимі (де кожен запит обробляється окремим процесом), а не в більш ефективному воркер-режимі (де процеси можуть обробляти багато запитів одночасно). Це робить його менш ефективним для великих сайтів.
- Сумісність з іншими веб-серверами: Якщо використовується mod_php, Apache повинен обробляти всі PHP-запити. Інші веб-сервери, як Nginx,

не можуть легко отримувати PHP-відповіді, оскільки Nginx не підтримує mod_php.

1.2.2 Apache з FastCGI (mod_fcgid)

Альтернативний підхід в Apache - використання FastCGI через модуль mod_fcgid. FastCGI дозволяє запускати PHP як окремий процес поза Apache. Це знижує навантаження на пам'ять і робить масштабування більш гнучким, оскільки PHP працює в окремих процесах. Але цей варіант вимагає додаткових налаштувань і іноді може бути трохи складнішим у конфігурації.

Модуль mod_fcgid для Apache HTTP Server, дозволяє використовувати протокол FastCGI для обробки PHP-запитів. Цей підхід забезпечує більш ефективну обробку запитів у порівнянні з традиційним mod_php. FastCGI — це протокол, який дозволяє веб-серверам взаємодіяти з мовами програмування через інтерфейси, що зберігаються в пам'яті. Цей підхід зменшує накладні витрати, пов'язані з запуском процесів для кожного запиту, оскільки PHP може залишатися запущеним у пам’яті, обробляючи кілька запитів.

Переваги:

- Покращена продуктивність: Оскільки FastCGI дозволяє підтримувати постійно запущені процеси PHP, зменшується затримка між запитами.
- Краща керованість ресурсами: Модуль може обробляти великі обсяги запитів з меншими витратами пам’яті у порівнянні з mod_php, де кожен запит створює новий процес.

- Масштабованість: Завдяки можливості налаштування пулів процесів PHP-FPM, mod_fcgid забезпечує гнучкість у масштабуванні веб-додатків.

Недоліки:

- Складніша конфігурація: Налаштування mod_fcgid і PHP-FPM може бути складнішим, ніж просте налаштування mod_php.
- Потенційні проблеми з безпекою: Необхідно правильно налаштувати права доступу до процесів FastCGI, щоб уникнути можливих вразливостей.
- Затримки при високих навантаженнях: Якщо кількість запитів перевищує можливості обробки PHP-FPM, можуть виникнути затримки.

Використання mod_fcgid mod_fcgid підходить для великих веб-додатків, які потребують високої продуктивності та масштабованості. Його переваги у використанні процесів PHP-FPM роблять його гарним вибіром для сучасних веб-сайтів, що мають значну кількість одночасних користувачів.

1.2.3. Nginx з PHP-FPM

Конфігурація Nginx з PHP-FPM є стандартом для обробки PHP-запитів на багатьох веб-серверах завдяки високій продуктивності, гнучкості та ефективному використанню ресурсів. PHP-FPM (FastCGI Process Manager) — це розширення для PHP, яке покращує обробку FastCGI-запитів, додає налаштування для пулу процесів і надає інші можливості для керування навантаженням. PHP-FPM дозволяє масштабувати PHP-процеси відповідно до навантаження, розділяючи PHP і веб-сервер, що підвищує продуктивність і стабільність. PHP-FPM є більш ефективною альтернативою до стандартного

FastCGI, яка дозволяє запитам оброблятися пулом процесів PHP, що зменшує час відгуку і дає змогу керувати багатопотоковістю.

Переваги Nginx з PHP-FPM

- Висока продуктивність: Nginx з PHP-FPM здатні обробляти велику кількість одночасних з'єднань з низькими затримками, що робить цю конфігурацію ідеальною для масштабованих додатків.
- Гнучкість у налаштуваннях: PHP-FPM дозволяє точно налаштувати пул процесів (наприклад, кількість процесів, режим відновлення тощо), що дозволяє оптимізувати споживання ресурсів.
- Ефективне керування пам'яттю: Nginx працює як зворотний проксі для PHP-FPM, завдяки чому віддача статичного контенту (картинок, файлів CSS, JavaScript) відбувається швидше, залишаючи ресурси для обробки лише PHP-запитів.

Недоліки

- Складність конфігурації: Для оптимальної продуктивності потрібно уважно налаштувати параметри як Nginx, так і PHP-FPM, що може вимагати певних знань та досвіду.
- Обмежена сумісність з деякими розширеннями: Деякі специфічні PHP-розширення можуть працювати не так, як очікувалося, через архітектурні особливості FastCGI та Nginx.

1.2.4. Caddy Server з FastCGI

Caddy Server — це сучасний веб-сервер з підтримкою автоматичного SSL/TLS, простого налаштування і зручного керування HTTP/2 та HTTP/3. Використання Caddy з FastCGI забезпечує продуктивний спосіб для обробки PHP-запитів. Caddy Server часто розглядають як альтернативу Nginx завдяки автоматизованим сертифікатам SSL і підтримці різних налаштувань без складних конфігурацій. Caddy, завдяки зрозумілій структурі

конфігураційного файлу Caddyfile, значно спрощує розгортання веб-сайтів і додатків. Простий синтаксис дозволяє легко налаштовувати кореневі директорії для сайтів, встановлювати правила маршрутизації, налаштовувати редиректи, і, що важливо для FastCGI, вказувати адресу до PHP-FPM.

Переваги

- Автоматизація SSL: Caddy автоматично генерує та оновлює SSL-сертифікати, забезпечуючи захищене з'єднання.
- Простота конфігурації: Завдяки зрозумілій синтаксичній структурі конфігурації, Caddy є зручним для налаштування.
- Підтримка HTTP/2 і HTTP/3: Caddy Server підтримує сучасні версії HTTP, що покращує продуктивність і знижує затримки при передачі.

Недоліки

- Обмежена гнучкість для великих конфігурацій: У порівнянні з Nginx або Apache, Caddy може мати обмежені можливості для складних веб-систем.
- Менша спільнота: Caddy менш популярний, і тому для нього є менше документації та підтримки.

1.3 Сучасні методи використання

Окрім традиційних способів запуску PHP через веб-сервери, існують також сучасні рішення для запуску PHP у високопродуктивних середовищах з використанням альтернативних підходів, таких як Swoole, FrankenPHP, RoadRunner, AMPHP та інші. Кожен з них розширює можливості PHP, забезпечуючи низькорівневий доступ до процесів, багатопоточність та

асинхронну обробку запитів, що робить їх ідеальними для створення мікросервісів та інших додатків з високими вимогами до продуктивності.

1.3.1 Swoole

Swoole — це високопродуктивне асинхронне розширення для PHP, яке створене для забезпечення конкурентності та низької затримки в додатках на основі PHP. Завдяки підтримці корутин і багатопоточності, Swoole перетворює PHP із традиційно синхронної мови у потужний інструмент для розробки веб-сервісів, які можуть обробляти велику кількість одночасних запитів і з'єднань.

Основні можливості Swoole

- Асинхронність. Swoole підтримує асинхронний запуск PHP-коду, дозволяючи виконувати паралельні завдання без блокування, що підвищує ефективність обробки запитів.
- Корутини. Swoole дозволяють писати асинхронний код у зрозумілому для синхронного програмування стилі, значно спрощуючи розробку.
- Вбудований сервер. Swoole має вбудований HTTP-сервер і TCP-сервер, що дозволяє обходитися без традиційних серверів, як Apache чи Nginx. Це спрощує архітектуру і надає контроль над обробкою запитів.
- Підтримка WebSocket. Swoole дозволяє легко створювати реальні програми, як-от чати, трекери для реального часу або ігрові сервери.
- Багатопоточність і паралелізм. Swoole підтримує багатопоточне виконання, яке дозволяє розробникам розподіляти завдання між потоками і зменшувати навантаження на кожен окремий процес. Така багатопоточність корисна для завдань на зразок роботи з базами даних, взаємодії з API або інших ресурсомістких операцій, де необхідна максимальна продуктивність.
- Збереження стану між запитами. Зазвичай PHP працює в режимі "запит-відповідь", де кожен запит викликає новий процес, який не зберігає стан. Завдяки Swoole стан можна зберігати між запитами, що

дозволяє уникнути витрат на ініціалізацію і налаштування кожного разу.

- Підтримка таймерів і відкладених завдань. Swoole дозволяє легко створювати завдання, які виконуються з затримкою або запускаються за розкладом, що є зручним для автоматизації завдань у додатках.

Переваги

- Підвищення продуктивності: Swoole може обробляти велику кількість запитів одночасно завдяки асинхронності та корутинам, що знижує затримки й підвищує швидкість виконання.
- Зниження використання ресурсів: Замість створення нового процесу для кожного запиту, як це робить традиційний PHP, Swoole працює з перманентними процесами і потоками, що значно зменшує витрати на пам’ять і процесор.
- Сучасна архітектура: Swoole підходить для мікросервісів, серверів для API, серверів реального часу і багатьох інших варіантів використання, де необхідні висока доступність і масштабованість.

Недоліки

- Потреба у зміні підходу: Swoole вимагає від розробників розуміння асинхронного програмування та корутин, що може бути викликом для тих, хто звик до класичного PHP.
- Сумісність з існуючими бібліотеками: Не всі PHP-бібліотеки підтримують асинхронність, що може обмежувати можливість використання Swoole з деякими інструментами.
- Залежність від середовища виконання: Оскільки Swoole є розширенням PHP, воно потребує окремої інсталяції на сервері й підходить переважно для власних серверів або кастомних хостингів, але не завжди доступне у традиційному хостингу для PHP.

1.3.2 FrankenPHP

FrankenPHP — це інноваційний підхід до запуску PHP для високонавантажених веб-додатків, який поєднує традиційні можливості PHP з асинхронними та подієво-орієнтованими функціями, що дозволяє досягти високої продуктивності. Розроблений для роботи з сучасними фреймворками як Symfony та Laravel, так і з іншими фреймворками. FrankenPHP інтегрує PHP безпосередньо з веб-сервером, завдяки чому оптимізується обробка HTTP-запитів і зменшується затримка при завантаженні.

Основні можливості FrankenPHP

- Асинхронність та подієвий підхід. На відміну від стандартного PHP, який є синхронним, FrankenPHP підтримує асинхронну обробку запитів, що дозволяє виконувати паралельні завдання та зменшувати загальний час обробки. Завдяки асинхронній обробці FrankenPHP особливо добре підходить для реального часу, де додатки потребують швидкої реакції на велику кількість запитів.
- Нативна підтримка HTTP/2 і WebSocket. FrankenPHP підтримує новітні протоколи, такі як HTTP/2 та WebSocket, що дозволяє створювати інтерактивні веб-додатки, наприклад, чати, панелі адміністрування в реальному часі та інші сервіси, де важлива швидкість передачі даних.Це забезпечує стабільність з’єднань і швидшу доставку даних між сервером та клієнтом, що є критично важливим для сучасних веб-додатків.
- Оптимізована продуктивність для Symfony та Laravel. FrankenPHP особливо добре інтегрується з фреймворками, забезпечуючи більш

швидкий запуск сервісів, завдяки ефективній обробці запитів та збереженню контексту між сесіями. Оскільки Symfony та Laravel є популярними фреймворками для PHP, FrankenPHP може бути цікавим вибором для великих проєктів, що використовують ці фреймворки.

- Мінімізація накладних витрат на кожен запит. Оскільки FrankenPHP працює на основі подій і асинхронних процесів, він знижує витрати на запуск окремих PHP-процесів для кожного HTTP-запиту. Це дозволяє обробляти більше запитів за менший час порівняно з традиційними налаштуваннями на Apache або Nginx з PHP-FPM.

Переваги:

- Підвищена продуктивність завдяки асинхронному виконанню.
- Зменшення навантаження на сервер за рахунок об’єднання з сервером HTTP.
- Підтримка сучасних протоколів і можливостей реального часу.

Недоліки

- Не всі PHP-додатки можуть бути легко адаптовані до асинхронного підходу.
- Для реалізації потрібні певні знання у налаштуванні сервера, а також можливість розгортання на власних серверах.

FrankenPHP є перспективною технологією для розробників PHP, які хочуть поєднати асинхронні можливості з ефективністю PHP для сучасних веб-додатків з високим навантаженням і вимогами до швидкодії.

1.3.3 RoadRunner

RoadRunner — це високопродуктивний PHP-додаток серверного типу, який поєднує можливості PHP із сучасною багатопотоковою архітектурою, керованою за допомогою Go. RoadRunner був розроблений компанією Spiral Scout для забезпечення високої продуктивності та підвищеної стійкості серверних PHP-додатків, особливо для веб-додатків з високим навантаженням, мікросервісів та сервісів реального часу.

Основні можливості RoadRunner

- Використання багатопоточності та паралельної обробки. RoadRunner побудований на базі Go, що забезпечує високу продуктивність, асинхронну обробку запитів та багатопоточність. На відміну від традиційного підходу PHP-FPM, RoadRunner зберігає інтерпретатор PHP у пам'яті між запитами, що значно скорочує час виконання.
- Підтримка HTTP/2 та WebSocket. Завдяки вбудованій підтримці сучасних протоколів, таких як HTTP/2 та WebSocket, RoadRunner забезпечує стабільне підключення в реальному часі, яке ідеально підходить для створення інтерактивних веб-додатків (чати, онлайн-панелі, ігрові сервери). Це дозволяє уникнути необхідності в зовнішніх проксі або окремих серверів для підтримки WebSocket.
- Можливості для мікросервісів. RoadRunner підтримує асинхронні можливості через інтеграцію з gRPC та інші сучасні протоколи, що робить його особливо ефективним для побудови мікросервісної архітектури. Його функції дозволяють легко інтегрувати PHP-додатки з іншими сервісами, керованими Go.
- Легке конфігурування та масштабування. Додаток підтримує налаштування за допомогою конфігураційних файлів і командної строки, а також має вбудований балансувальник навантаження. RoadRunner може бути налаштований для горизонтального

масштабування, що дозволяє обробляти велику кількість запитів, ідеально підходить для розгортання великих і складних додатків.

- Інтеграція з Laravel та Symfony. RoadRunner підтримує інтеграцію з сучасними фреймворками, надаючи нові можливості для прискорення додатків. Підтримує повторне використання контейнера додатку та зберігання сервісів між запитами, що дозволяє скоротити накладні витрати на запуск.

Переваги

- Підвищена швидкість: Завдяки Go та утриманню інтерпретатора PHP у пам'яті, RoadRunner забезпечує швидшу обробку запитів порівняно з традиційними підходами (PHP-FPM).
- Низьке споживання ресурсів: Менші накладні витрати, пов’язані з кожним запитом, що знижує споживання CPU та пам'яті.
- Гнучкість: Підтримка різноманітних протоколів (HTTP/2, WebSocket, gRPC) робить RoadRunner надзвичайно гнучким для інтеграцій.

Недоліки

- Складність налаштування: Порівняно з PHP-FPM, RoadRunner потребує більшої обізнаності у конфігуруванні та налаштуванні багатопотокових програм.
- Підтримка асинхронного коду: RoadRunner вимагає деяких змін у коді додатка, особливо для роботи з асинхронними та багатопоточними процесами.
- Потребує окремого сервера Go: Для роботи RoadRunner необхідно встановити Go, що може бути складністю для тих, хто звик працювати лише з PHP.

Таким чином, RoadRunner є потужним інструментом для оптимізації PHP-додатків, які потребують високої продуктивності, паралельної обробки та підтримки протоколів реального часу. Це робить його чудовим вибором для сучасних додатків, де важлива швидкість і масштабованість.

1.4 Проблематика та постановка завдань дослідження

Розглянувши основні методи застосування PHP для серверної обробки, можна зробити висновок, що вибір методу залежить від специфіки проєкту, вимог до продуктивності, а також доступних ресурсів. Традиційні підходи, такі як Apache з модулем mod_php, забезпечують простоту інтеграції та налаштування, що підходить для проектів з помірним навантаженням. Однак із зростанням обсягу запитів, цей метод може виявитися обмеженим з точки зору масштабованості та продуктивності. Використання FastCGI, зокрема з PHP-FPM, стало більш оптимальним рішенням для великих проектів, де важлива швидкість та стабільність роботи при високому навантаженні.

Сучасні методи запуску PHP скриптів, такі як RoadRunner, Swoole, та Caddy з FastCGI, відкривають нові можливості для покращення продуктивності та гнучкості веб-додатків. Використання асинхронних технологій і багатопоточності дозволяє досягти значно вищих показників швидкості обробки запитів та зменшити затримки. Зокрема, Swoole та RoadRunner виділяються своєю здатністю підтримувати паралельне виконання запитів, що робить їх придатними для застосунків, що вимагають високої пропускної здатності, наприклад для систем реального часу. У той час як ці методи вимагають більшої обізнаності у налаштуванні, вони забезпечують сучасні додатки необхідною гнучкістю та продуктивністю для роботи в умовах високих навантажень.

В умовах стрімкого розвитку веб-технологій і зростання кількості високонавантажених систем, постає важлива задача — забезпечити оптимальну продуктивність серверних застосунків. Важливо, щоб серверні

технології, зокрема PHP, відповідали на виклики сучасного світу, де швидкість обробки даних, масштабованість та надійність є ключовими факторами успіху.

1.4.1 Проблематика дослідження

Попри популярність PHP у розробці веб-додатків, він має певні недоліки порівняно з новими високопродуктивними рішеннями, такими як Go, Node.js або Rust. Наприклад, інтерпретація PHP-коду впливає на швидкість обробки запитів, особливо в середовищах з високим навантаженням, де кожна секунда затримки може призводити до втрати користувачів або зниження ефективності системи. Класичні методи використання PHP (Apache з mod_php) більше не задовольняють сучасні вимоги до продуктивності й безпеки.

З розвитком нових технологій виникає необхідність вивчити альтернативні підходи та серверні конфігурації для PHP, що спрямовані на підвищення продуктивності, такі як Swoole, RoadRunner та інші. Ці рішення можуть забезпечити:

- Паралельне виконання запитів та асинхронну обробку, що значно знижує час відгуку сервера.
- Зменшення використання ресурсів завдяки ефективному управлінню потоками та процесами.
- Підтримку сучасних протоколів (наприклад, HTTP/2), що полегшує масштабування додатків.

1.4.2 Завдання дослідження:

1. Провести аналіз сучасних методів веб-серверної взаємодії для PHP, порівняти їх продуктивність та визначити оптимальні сценарії використання.
2. Дослідити, як кожен із методів (FastCGI, PHP-FPM, RoadRunner, Swoole тощо) впливає на швидкість обробки запитів і стабільність роботи при великих навантаженнях.
3. Провести тестування й заміри часу відгуку для різних конфігурацій PHP, щоб визначити ефективність кожного рішення в реальних умовах.
4. Надати рекомендації щодо вибору та налаштування серверного середовища залежно від вимог продуктивності та надійності.

Таким чином, це дослідження спрямоване на забезпечення обґрунтованого вибору інструментів для розробників, що використовують PHP, надаючи їм можливість створювати більш швидкі, стабільні та масштабовані веб-додатки.

РОЗДІЛ 2. РОЗРОБКА ТА НАЛАШТУВАННЯ МЕТОДІВ ВЗАЄМОДІЇ

Для налаштування середовищ дослідження буде використовуватися Docker. Docker — це платформа контейнеризації, яка дозволяє створювати, розгортати та запускати програми у вигляді контейнерів. Контейнери є відокремленими середовищами, що забезпечують однакові умови для запуску програмного коду, незалежно від операційної системи або специфіки конфігурації на комп'ютері, на якому він запускається.

Перевагами Docker у дослідженні є:

- Уніфіковане середовище тестування: Завдяки контейнерам Docker усі необхідні сервіси та бібліотеки можуть бути розгорнуті з однаковими налаштуваннями, що зменшує ризик появи помилок через розбіжності в оточенні.
- Швидке розгортання різних конфігурацій: За допомогою Docker можна швидко створювати нові контейнери для різних серверних налаштувань та методів запуску PHP.
- Простота в управлінні залежностями: Docker дозволяє ізолювати залежності кожної конфігурації, забезпечуючи відсутність конфліктів між ними, що є важливим при тестуванні різних версій серверів або бібліотек.

Завдяки контейнеризації, Docker дозволяє максимально стандартизувати і автоматизувати процес налаштування і запуску середовищ для дослідження продуктивності PHP, що забезпечує надійність і відтворюваність результатів експериментів. Також було використано docker compose, для полегшення налаштування та взаємодії між контейнерами.

```
[PHP]
memory_limit = 256M
max_execution_time = 60
error_log = /dev/stderr
display_errors = On
display_startup_errors = On
log_errors = On
error_reporting = E_ALL
[opcache]
opcache.enable = 1
opcache.memory_consumption = 256
opcache.interned_strings_buffer = 16
opcache.max_accelerated_files = 16229
```

Рис 2.1 Конфігурація php.ini

2.1 Apache з mod_php

Для створення докер образу було використаю офіційний образ php:8.3-apache який містить і PHP і Apache із вже налаштованим mod_php, що забезпечує швидкий початок роботи з Docker, а також було додатково проведено налаштування під поточні потреби.

```
FROM php:8.3-apache

RUN apt-get update && apt-get install -y \
    libpq-dev \
    && docker-php-ext-configure pdo_pgsql \
    && docker-php-ext-install pdo pdo_pgsql opcache \
    && rm -rf /var/lib/apt/lists/*

RUN a2enmod php8.3

COPY ./server/apache2.conf /etc/apache2/sites-available/000-default.conf
COPY ./src /var/www/html/

CMD ["apache2-foreground"]
```

Рис 2.2 Docker image для apache2 і mod_php

Також для коректної обробки запитів потрібно додати налаштування для апачу.

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    # Для роботи з PHP через mod_php
    <IfModule mod_php.c>
        AddHandler application/x-httpd-php .php
    </IfModule>

    # Налаштовуємо доступ до каталогу
    <Directory /var/www/html>
        Options +ExecCGI
        AllowOverride All
        Require all granted
    </Directory>

    # Логи
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Рис 2.3 Налаштування apache2.conf

2.2 Apache з PHP-FPM через FastCGI

Для створення докер образу було використано офіційний образ php:8.3-apache містить PHP і Apache2, а також додатково інсталюється та вмикається модуль fcgi та інші розширення.

```
FROM php:8.3-fpm
RUN apt-get update && apt-get install -y \
    apache2 \
    libapache2-mod-fcgid \
    && apt-get clean

RUN a2enmod actions alias proxy_fcgi fcgid
RUN mkdir -p /run/php && chown -R www-data:www-data /run/php
COPY php-fpm.conf /usr/local/etc/php-fpm.d/www.conf
EXPOSE 80

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

Рис 2.4 Docker image для apache2 і fcgi

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    <Directory /var/www/html>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    <FilesMatch \.php$>
        SetHandler "proxy:unix:/run/php/php8.3-fpm.sock|fcgi://localhost/"
    </FilesMatch>
</VirtualHost>
```

Рис 2.5 Налаштування apache2.conf

2.3 Nginx з PHP-FPM

Для створення налаштування були використані офіційні докер контейнери php, postgresql та nginx.Також було додано налаштування php,

пула менеджера процессів php-fpm та веб-серверу nginx, це базові налаштування які в подальшому можуть змінюватися.

```
FROM php:8.3-fpm

RUN apt-get update && apt-get install -y \
    && docker-php-ext-install pdo pdo_mysql opcache

RUN apt-get clean && rm -rf /var/lib/apt/lists/*

COPY ./php/www.conf /usr/local/etc/php-fpm.d/www.conf
COPY ./php/php.ini /usr/local/etc/php/conf.d/custom.ini

WORKDIR /var/www/html
COPY ./src/index.php /var/www/html

EXPOSE 9000

CMD ["php-fpm"]
```

Рис 2.6 Docker image для php-fpm

```
server {
    listen 80;
    server_name localhost;

    set $ROOTPATH /var/www/html;
    root  $ROOTPATH;

    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    charset UTF-8;
    index index.html index.php;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }

    location ~* (\.php$|\.htaccess$|\.git|\.env) {
        deny all;
    }
}
```

Рис. 2.7 Налаштування nginx

```
[www]
user = www-data
group = www-data

listen = 9000
listen.owner = www-data
listen.group = www-data
listen.mode = 0660

pm = dynamic
pm.max_children = 10
pm.start_servers = 2
pm.min_spare_servers = 1
pm.max_spare_servers = 3

php_admin_value[error_log] = /dev/stderr
php_flag[display_errors] = on
php_admin_flag[log_errors] = on
php_admin_value[error_reporting] = E_ALL
php_admin_value[display_startup_errors] = on
```

Рис. 2.8 Налаштування php-fpm

2.4 Caddy Server з FastCGI

Докерфайл для образу php-fpm використовуємо як у прикладі вище, додаткових змін не потрібно робити, відрізняється лише сервер який приймає запити та направляє їх до менеджеру процесів php-fpm. Також використаємо ті ж самі налаштування для php.

```
:80 {
    root * /var/www/html
    php_fastcgi php:9000
    file_server
}
```

Рис. 2.9 Налаштування caddy server

2.5 Swoole

Для створення налаштування були використані офіційні контейнери php, postgresql та nginx, також додатково були встановлені пакети для коректного запуску swoole. В даному випадку nginx використовується як reverse-proxy для прийняття запитів на домені і пере направлені їх до серверу swoole.

```php
<?php

use Swoole\Http\Server;

$server = new Server("0.0.0.0", 9501);

$server->on("request", function ($request, $response) {
    //some code to execute
});

$server->start();
```

Рис. 2.10 Запуск swoole серверу

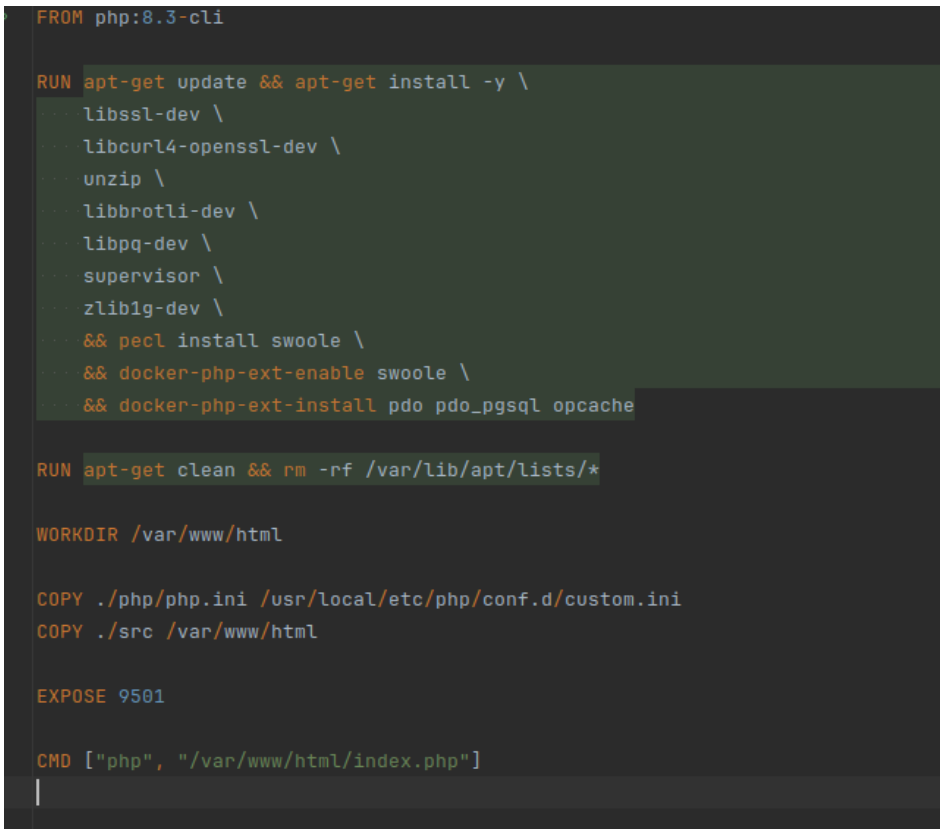
```
server {
    listen 80;

    server_name localhost;

    location / {
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://swoole:9501;
    }

    location ~* (\.php$|\.htaccess$|\.git|\.env) {
        deny all;
    }
    error_log /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
}
```

Рис. 2.11 Налаштування nginx як reverse-proxy



```
FROM php:8.3-cli

RUN apt-get update && apt-get install -y \
    libssl-dev \
    libcurl4-openssl-dev \
    unzip \
    libbrotli-dev \
    libpq-dev \
    supervisor \
    zlib1g-dev \
    && pecl install swoole \
    && docker-php-ext-enable swoole \
    && docker-php-ext-install pdo pdo_pgsql opcache

RUN apt-get clean && rm -rf /var/lib/apt/lists/*

WORKDIR /var/www/html

COPY ./php/php.ini /usr/local/etc/php/conf.d/custom.ini
COPY ./src /var/www/html

EXPOSE 9501

CMD ["php", "/var/www/html/index.php"]
```

Рис 2.12 Docker image для swoole

2.7 RoadRunner

Для створення докер образу були використані офіційні контейнери php, postgresql та nginx в якості reverse proxy, також додатково був використан офіційний docker image для запуску roadrunner. Також потрібно встановити composer, менеджер управління залежностями для php, і пакети для створення серверу обробки запитів

```php
require __DIR__ . '/vendor/autoload.php';

$worker = RoadRunner\Worker::create();
$psr7 = new RoadRunner\Http\PSR7Worker(
    $worker,
    new Psr17Factory(),
    new Psr17Factory(),
    new Psr17Factory()
);

while ($req = $psr7->waitRequest()) {
    try {
        $psr7->respond(handleRequest($req));
    } catch (Throwable $e) {
        $psr7->getWorker()->error((string)$e);
    }
}
```

Рис. 2.13 Запуск roadrunner серверу

```dockerfile
FROM ghcr.io/roadrunner-server/roadrunner:latest AS roadrunner
FROM php:8.3-cli

COPY --from=roadrunner /usr/bin/rr /usr/local/bin/rr

RUN apt-get update && apt-get install -y \
    libpq-dev \
    && docker-php-ext-install pdo pdo_pgsql opcache
RUN apt-get install -y git

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /var/www/html
COPY ./src /var/www/html
COPY ./php/php.ini /usr/local/etc/php/conf.d/custom.ini

RUN composer install --ignore-platform-reqs --no-scripts

EXPOSE 8080

CMD ["rr", "serve", "-c", "/var/www/html/roadrunner.yaml"]
```

Рис. 2.14 Docker image roadrunner та php

Рис. 2.15 Конфігурація composer.json файлу

2.4 Висновки розділу

В даному розділі, було створено основні налаштування для обраних методів веб-серверної взаємодії, а також проаналізовано наявні рішення та кращі практики використання. На цьому етапі можна починати тестувати методи веб-серверної взаємодії , та імплементувати тестування обраних варіантів.

РОЗДІЛ 3. ДОСЛІЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ ВЗАЄМОДІЇ

Для виявлення найбільш продуктивного методу будуть використані наступні тести:

- Обробка простого PHP-скрипта.
- Тест асинхронності - наприклад завантаження результатів з двох різних таблиць бази даних.
- Обробка великого файла (Large Payload Handling). Надіслати великий JSON-файл (розмір від 5 до 50 МБ) для обробки та збереження.
- Запуск коду з використанням фреймворка

Метрики:

- Час відгуку (response time).
- Кількість запитів за секунду (requests per second, RPS).
- Кількість відмов через перевантаження (failed requests)

Для створення навантаження було обрано Apache jmeter, він дозволяє гнучко вказувати кількість активних користувачів, і скільки запитів буде відправлятися 1 користувачем в секунду, а також є можливість побудови графіків залежності. Також встановлено обмеження на час виконання запиту на рівні 5 секунд, все що вище буде вважатися помилкою.

3.1 Обробка простого скрипту

Для того щоб зрозуміти з якою швидкістю оброблюється код в тому чи іншому випадку, було розроблено скрипт який в циклі обробляє числа.

```php
<?php
$array = range(0, 1000000);
$result = 0;
$start = microtime(true);
foreach ($array as $value) {
    $result = sqrt($value) + sqrt($result);
}
$executionTime = microtime(true) - $start;

echo json_encode([
    'executionTime' => $executionTime,
], JSON_THROW_ON_ERROR);
```

Рис. 3.1 Вихідний код простого скрипта

Для початкового тесту було задано максимум 500 користувачів, які починаються з 1 на початку тесту і їх кількість збільшується до 500 протягом 300 секунд, після цього навантаження тримається 1 хвилину і тест завершується.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**166290**	**14094**	**8.48%**	**3017.43**	**26**	**52157**	**2679.00**	**3001.00**	**3204.95**	**3457.00**	**410.55**	**97.14**	**46.41**
caddyHTTP Request	42002	0	0.00%	1559.68	27	3551	2679.00	3001.00	3204.95	3457.00	114.09	24.63	12.92
roadrunnerHTTP Request	38426	0	0.00%	1800.85	28	3998	3031.00	3727.00	3790.00	3877.00	104.41	24.15	11.83
nginxHTTP Request	39267	0	0.00%	1739.40	26	4114	2872.00	3337.00	3453.00	3680.00	106.64	28.17	12.08
swooleHTTP Request	38305	6551	17.10%	1816.64	27	5072	2851.50	5003.00	5004.00	5009.00	103.47	23.92	11.72
modphpHTTP Request	4172	3803	91.16%	27411.98	245	51812	27752.00	48206.80	49322.00	50553.56	10.34	2.69	1.12
fcgiHTTP Request	4118	3740	90.82%	27879.86	240	52157	27307.00	47998.20	49027.35	50792.10	10.17	2.65	1.10

Рис. 3.2 Загальні результати тесту

Отриманні дані після першого тесту показують що використання апач програє всім іншим методам веб-серверної взаємодії:

Метод запуску	Кількість запитів	Відмови (%)	Середній час обробки (мс)	Максимальний час (мс)	Транзакцій/сек
Caddy	42,002	0.00%	1,559.68	3,457	114.09
Nginx	39,267	0.00%	1,739.40	3,680	106.64
RoadRunner	38,426	0.00%	1,800.85	3,877	104.41
Swoole	38,305	17.10%	1,816.64	5,009	103.47
mod_php	4,172	91.16%	27,411.98	50,553	10.34
FastCGI	4,118	90.82%	27,879.86	50,792	10.17

Таблиця 3.1 Порівняльні дані тесту

Тому розглянемо графік залежності кількості запитів до часу відгуку окремо для апач та інших тестуємих методів. Також потрібно додати що для апач неможливо додати обмеження на рівні сервера таймаут очікування відповіді, тому час відповіді на графіки більше ніж для інших.

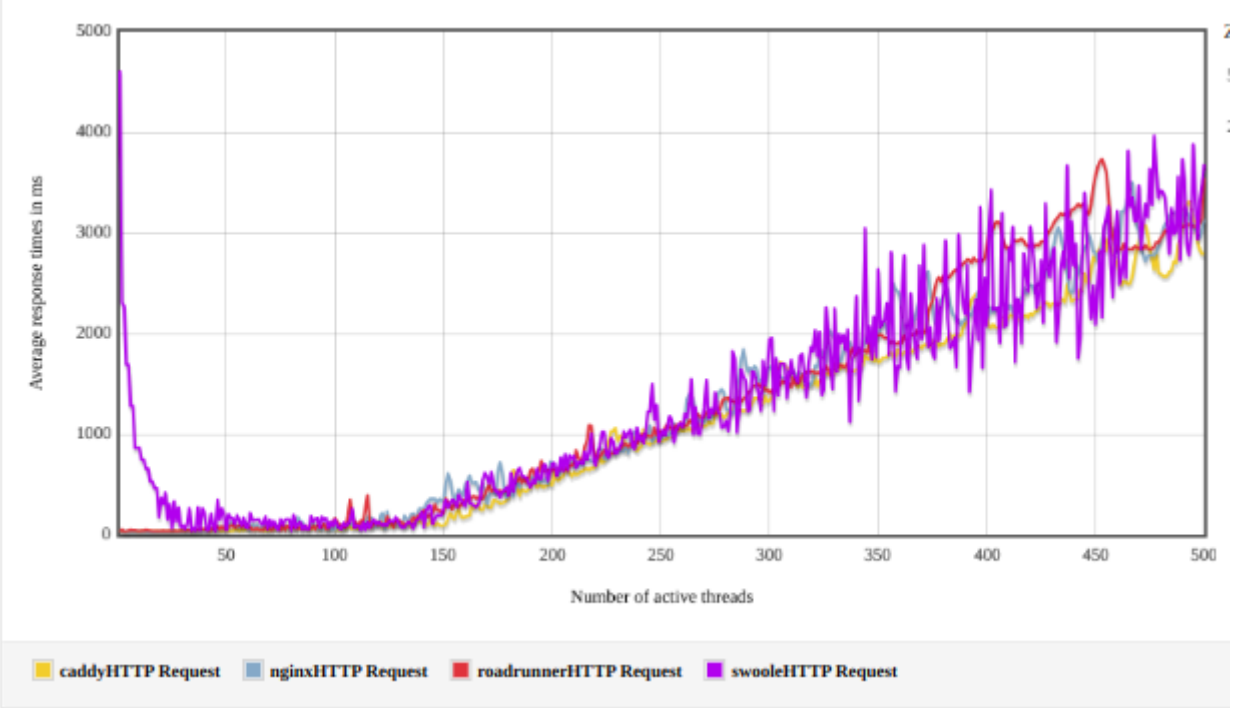


Рис. 3.3 Графік залежності кількості одночасних користувачів до часу відгуку

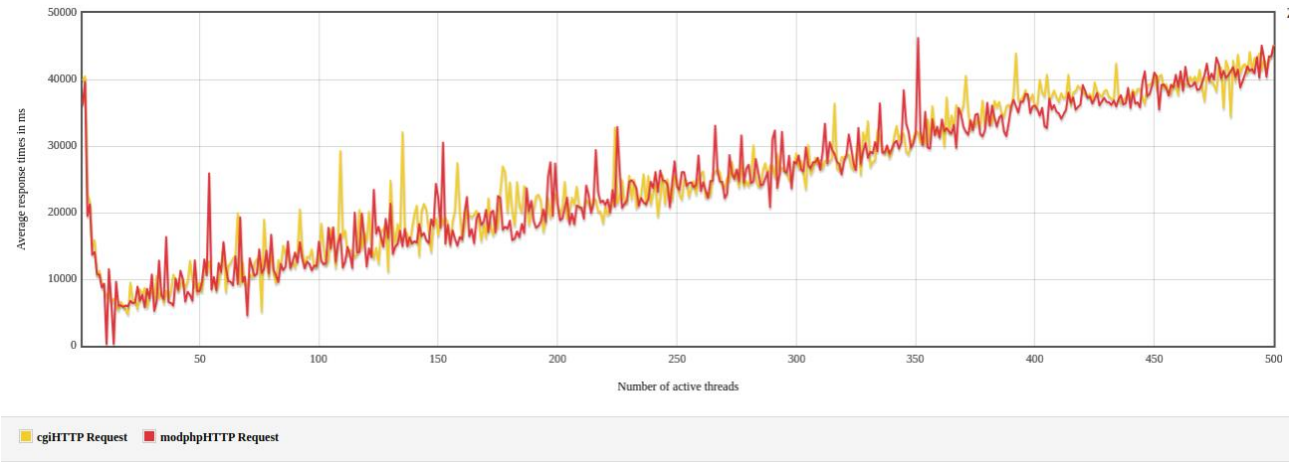


Рис. 3.4 Графік залежності кількості одночасних користувачів до часу відгуку з використанням apache2

Caddy та Nginx показали найкращі результати завдяки ефективному розподілу запитів, кешуванню та асинхронній архітектурі. RoadRunner та Swoole демонструють потенціал, але вимагають ретельного налаштування для стабільної роботи, mod_php та FastCGI не підходять для сучасних навантажених систем через стару архітектуру та низьку ефективність. Тож для подальших тестів було прийнято рішення виключити використання apache2.

3.2 Обробка даних з бази даних

Для цього тесту було створено скрипт, який виконує 3 запити до бази даних. База даних була використана postgres, кількість одночасних користувачів була знижена до 200, які виконували 1 запит на секунду. Обсяг оперативної пам'яті також було збільшено до 512МБ. Базовий код виглядає наступним чином.

```php
<?php

$dsn = 'pgsql:host=db;port=5432;dbname=diplom;user=diplom;password=diplom';
$user = 'diplom';
$password = 'diplom';

try {
    $start = microtime(true);
    $pdo = new PDO($dsn, $user, $password);

    $queries = [
        'users' => "SELECT count(*) FROM users where created_at between '2020-01-01' and '2025-01-31'",
        'lists' => 'SELECT count(*) FROM lists group by user_id',
        'files' => "select count(*) from files where path like '%2%' group by model_type",
    ];

    $results = [];
    foreach ($queries as $query) {
        $results[] = $pdo->query($query)->fetchAll(PDO::FETCH_ASSOC);
    }

    $end = microtime(true);
    $executionTime = microtime(true) - $start;
    echo json_encode([
        'time' => $executionTime,
    ]);
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
```

Рис. 3.5 Скрипт запитів до бази даних

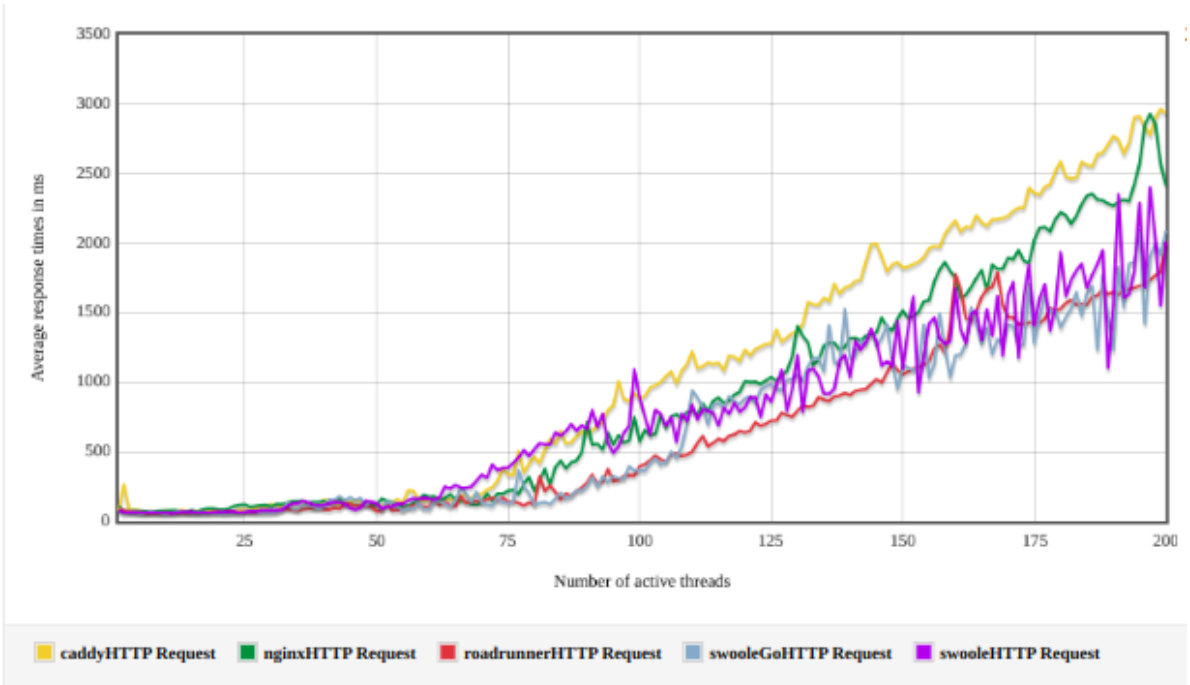
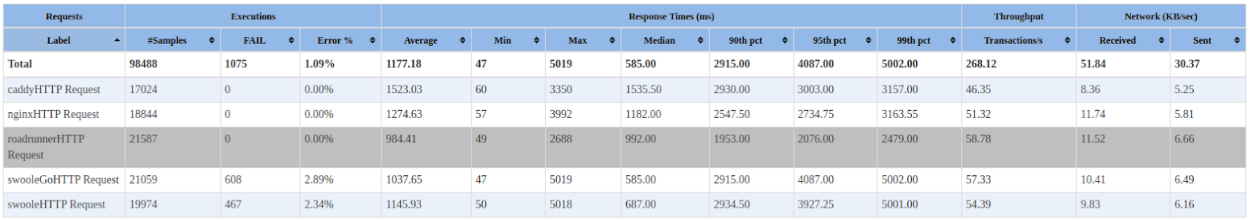


Рис. 3.6 Графік залежності кількості одночасних користувачів до часу відгуку для асинхроного тесту



Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	98488	1075	1.09%	1177.18	47	5019	585.00	2915.00	4087.00	5002.00	268.12	51.84	30.37
caddyHTTP Request	17024	0	0.00%	1523.03	60	3350	1535.50	2930.00	3003.00	3157.00	46.35	8.36	5.25
nginxHTTP Request	18844	0	0.00%	1274.63	57	3992	1182.00	2547.50	2734.75	3163.55	51.32	11.74	5.81
roadrunnerHTTP Request	21587	0	0.00%	984.41	49	2688	992.00	1953.00	2076.00	2479.00	58.78	11.52	6.66
swooleGoHTTP Request	21059	608	2.89%	1037.65	47	5019	585.00	2915.00	4087.00	5002.00	57.33	10.41	6.49
swooleHTTP Request	19974	467	2.34%	1145.93	50	5018	687.00	2934.50	3927.25	5001.00	54.39	9.83	6.16

Рис. 3.7 Загальна таблиця результатів для асинхронного тесту

Метод запуску	Кількість запитів	Відмови (%)	Середній час обробки (мс)	Максимальний час (мс)	Транзакцій/сек
Caddy	17,024	0.00%	1523.03	3157.00	46.35
Nginx	18,844	0.00%	1274.63	3163.55	51.32
RoadRunner	21,587	0.00%	984.41	2479.00	58.78
Swoole	21,059	2.89%	1037.65	5002.00	57.33
Swoole coroutine	19,974	2.34%	1145.93	5001.00	54.39

Таблиця 3.2 Загальна таблиця результатів для асинхронного тесту

RoadRunner продемонстрував найкращий результат із найменшим середнім часом відповіді (984.41 мс) і найвищою кількістю оброблених запитів (21,587 запитів/сек). Swoole і Swoole з використанням coroutine також показали високі результати, але мали певний відсоток помилок (2.89% і 2.34% відповідно). Nginx і Caddy мають стабільність (0% помилок), але їхня продуктивність нижча, ніж у RoadRunner, особливо за кількістю запитів і часом відповіді.

3.3 Обробка великого файла (Large Payload Handling)

Для цього тесту був взятий файл з розширенням csv розміром у 13.3mb. Сам скрипт бере завантажений файл, зчитує його рядок за рядком та переводе текст у верхній регістр.

```php
<?php

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!isset($_FILES['file']) || $_FILES['file']['error'] !== UPLOAD_ERR_OK) {
        http_response_code(400);
        echo json_encode(['status' => 'error', 'message' => 'File upload error']);
        exit;
    }

    $filePath = $_FILES['file']['tmp_name'];

    if (($handle = fopen($filePath, 'r')) !== false) {
        while (($row = fgetcsv($handle, 1000, ',')) !== false) {
            array_map('strtoupper', $row);
        }
        fclose($handle);
    } else {
        http_response_code(400);
        echo json_encode(['status' => 'error', 'message' => 'Failed to open file']);
    }

    echo json_encode(['status' => 'success']);
}
```

Рис. 3.8 Скрипт обробки великого файлу

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**6880**	**1111**	**16.15%**	**2861.66**	**549**	**11745**	**2688.00**	**5010.00**	**5148.95**	**6684.41**	**18.14**	**3.41**	**235984.09**
swooleHTTP Request	1811	217	11.98%	2665.98	562	11745	2443.00	5030.80	5525.60	8045.56	4.78	0.85	62163.23
roadrunnerHTTP Request	1652	217	13.14%	3014.56	582	7749	3040.00	5218.70	5679.10	6567.23	4.36	0.82	56705.94
nginxHTTP Request	1743	181	10.38%	2811.91	549	6087	2775.00	5025.60	5095.00	5283.36	4.60	1.06	59819.31
caddyHTTP Request	1674	496	29.63%	2974.25	563	7240	2856.50	5007.00	5010.00	5017.00	4.41	0.67	57418.22

Рис. 3.9 Загальна таблиця результатів для обробки файлу

Метод запуску	Кількість запитів	Відмови (%)	Середній час обробки (мс)	Максимальний час (мс)	Транзакцій/сек
Caddy	1669	16.06%	2974.03	8382	4.4
Nginx	1751	12.79%	2796.39	7656	4.62
RoadRunner	1689	14.33%	2930.06	9845	4.46
Swoole	1744	15.60%	2805.20	23167	4.6

Таблиця 3.3 Загальна таблиця результатів для обробки файлу

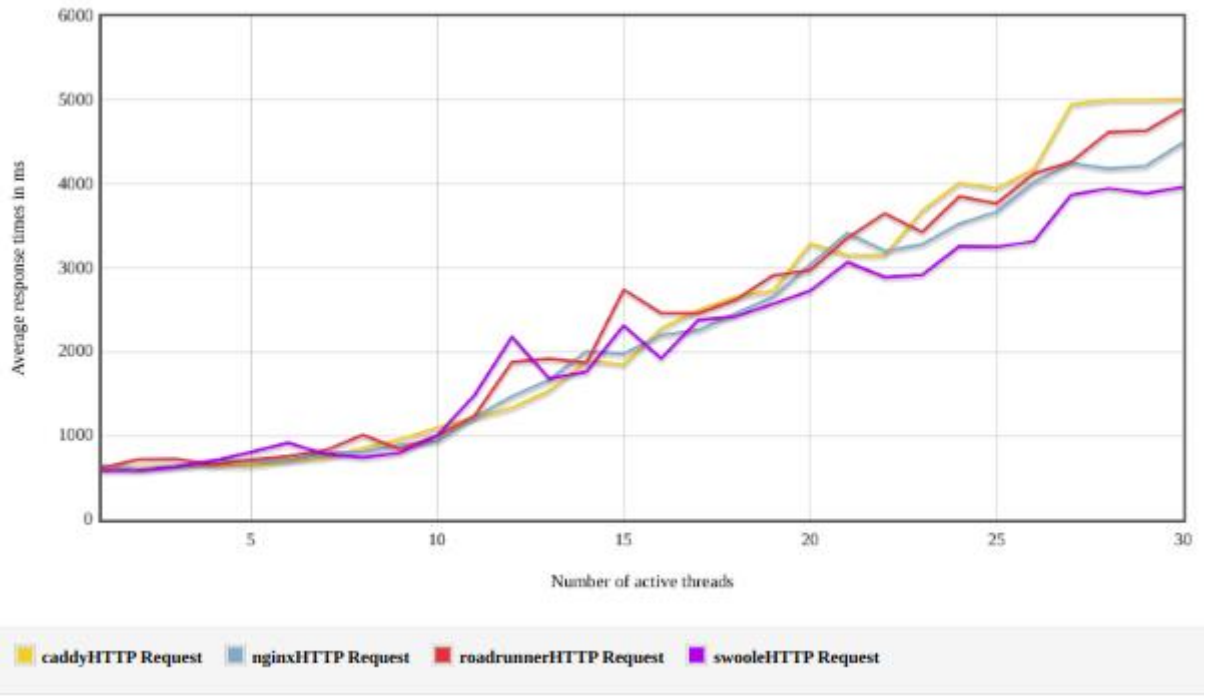


Рис. 3.10 Графік залежності кількості одночасних користувачів до часу відгуку для обробки великого файлу

Серед протестованих серверів **Nginx** показав найкращі результати за стабільністю та продуктивністю. Він продемонстрував найнижчий відсоток помилок (10.38%), другий найшвидший середній час обробки запитів

(2811.91 мс) і найкращий максимальний час обробки (6087 мс). Сервер **Swoole** показав найменший середній час обробки (2665.98 мс), але водночас характеризується найвищим максимальним часом виконання (11745 мс), що може свідчити про високе навантаження на CPU. **Roadrunner** демонструє середню продуктивність: час обробки запитів стабільний, але середній час (3014.56 мс) і максимальний час (7749 мс) поступаються конкурентам. **Caddy** хоча і забезпечує хорошу стабільність часу виконання, має найвищий відсоток помилок (29.63%), що суттєво знижує його придатність для використання у критичних виробничих середовищах.

3.4 Запуск коду з використанням фреймворка

В сучасному світі важко знайти написані системи без використання фреймворку, тому для останнього тесту було обрано один з найпопулярніших фреймворків на php — laravel. Максимальний рівень навантаження був встановлений на рівні 150 одночасних користувачів.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**75719**	**222**	**0.29%**	**775.18**	**28**	**7054**	**234.00**	**727.00**	**952.00**	**1573.99**	**195.75**	**222.25**	**33.64**
caddyHTTP Request	10941	170	1.55%	2080.03	75	7054	2260.00	3775.80	4224.80	5237.16	28.32	31.92	4.87
nginxHTTP Request	10903	52	0.48%	2085.39	74	6657	2050.00	3972.00	4299.80	4900.92	28.20	34.02	4.85
roadrunnerHTTP Request	27175	0	0.00%	234.06	29	1600	309.00	544.00	641.00	915.99	70.32	77.82	12.09
swooleHTTP Request	26700	0	0.00%	256.20	28	2961	234.00	727.00	952.00	1573.99	69.03	78.61	11.86

Рис. 3.11 Загальна таблиця результатів для навантаження з використанням фреймворку

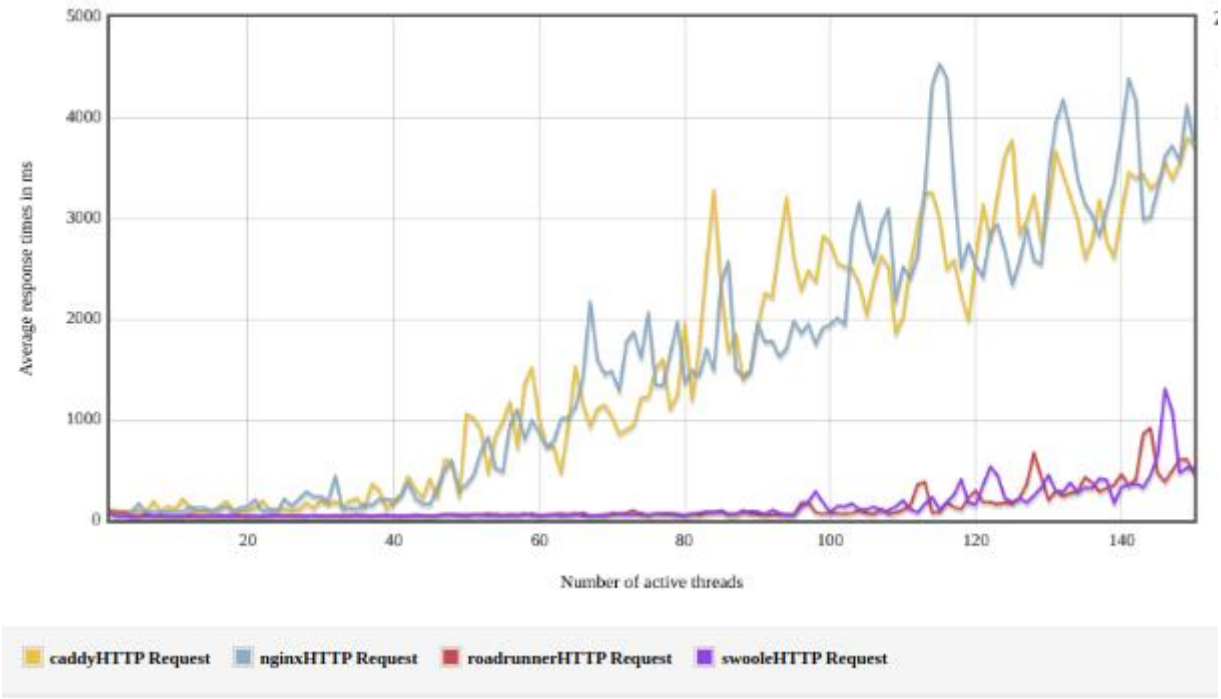


Рис. 3.12 Графік залежності кількості одночасних користувачів до часу відгуку з використанням фреймворку

Метод запуску	Кількість запитів	Відмови (%)	Середній час обробки (мс)	Максимальний час (мс)	Транзакцій/сек
Caddy	10941	1.55%	2080.03	7054	28.32
Nginx	10903	0.48%	2085.39	6657	28.20
RoadRunner	27175	0.00%	234.06	1600	70.32
Swoole	26700	0.00%	256.20	2961	69.03

Таблиця 3.4 Загальна таблиця результатів для запуску фреймворку

RoadRunner продемонстрував найкращий результат із найменшим середнім часом відповіді (234.06 мс) і найвищою кількістю оброблених запитів (27,175 запитів/сек). Swoole також показав високий результат з середнім часом відповіді 256.20 та кількістю оброблених запитів 26,700. Nginx і Caddy мають приблизно однакові показники, але вони значно програють Swoole та RoadRunner.

3.5 Висновки розділу

Аналізуючи результати тестів ми бачимо, що Caddy/Nginx + php-fpm показують гарні результати при запуску скриптів які не вимагають ініціалізації великих застосунків або асинхронних дій. В той самий час Swoole та RoadRunner навпаки, можуть показувати гіршу продуктивність при використанні їх для обробки простих, синхронних операцій через додаткові витрати ресурсів.

Розглянемо більш детально приклад з фреймворком. При використанні Caddy/Nginx + php-fpm кожен запит змушує Laravel знову ініціалізуватися, проганяючи всю бутстрап-логіку, виклики бази даних, кешу тощо. RoadRunner і Swoole працюють не за класичним, а як постійні процеси, тримаючи в пам’яті стан додатку та уникаючи постійного перезапуску Laravel ядра при кожному запиті. Це дає значне зниження латентності та стабільніший час відповіді. Для RoadRunner та Swoole ці інструменти інтегрують PHP-додаток і HTTP-сервер в один постійний процес. Laravel завантажується один раз, а надалі кожен запит обробляється без повторної ініціалізації. Це пояснює майже нульову кількість помилок, меншу затримку та стабільність навіть під високим навантаженням.

Традиційні підходи (Caddy/Nginx + PHP-FPM) значно програють за продуктивністю при високому навантаженні, оскільки постійно відтворюють оточення для кожного запиту. Використання таких рушіїв як RoadRunner чи Swoole дає змогу суттєво знизити час відповіді та кількість помилок за рахунок відсутності повторної ініціалізації додатку при кожному запиті. Але при розробці додатків за допомогою RoadRunner чи Swoole, потрібно змінювати підхід до розробки та мати наувазі те, що застосунок живе більше ніж 1 запит.

Підбиваючи підсумки:

1. **RoadRunner** є найбільш продуктивним та надійним методом для обробки запитів у сценаріях із високим навантаженням та роботою з фреймворком. Це ідеальний вибір для асинхронної обробки запитів.

2. **Swoole** підходить для високопродуктивних задач, але його стабільність залишається проблемою, особливо при обробці великих файлів.

3. **Caddy** та **Nginx** забезпечують високу надійність та помірну продуктивність, що робить їх оптимальними для сценаріїв із вимогами до безпеки та стабільності.

4. **mod_php** та **FastCGI** демонструють низьку продуктивність і не рекомендуються для сучасних високонавантажених застосунків.

ВИСНОВКИ

На основі проведених тестувань були оцінені такі характеристики, як кількість оброблених запитів, відмови, середній та максимальний час обробки, а також продуктивність (транзакцій/сек).

Під час дослідницької роботи, було розглянуто основні проблеми веб-серверної взаємодії з php. Проведено аналіз та пошук оптимального методу взаємодії. Результатом дослідження, є обґрунтовані рекомендації щодо вибору методів веб-серверної взаємодії, які мають допомогти з обранням того чи іншого методу в залежності від вимог . Підсумовуючи загалом для найкращої продуктивності варто розглядати **RoadRunner**, тоді як **Nginx** або **Caddy** підійдуть для менш вимогливих систем.

ПЕРЕЛІК ПОСИЛАНЬ

1. Docker Documentation. (n.d.). *Docker Inc*. Доступно за посиланням: https://docs.docker.com
2. Docker Compose Documentation. (n.d.). *Docker Inc*. Доступно за посиланням: https://docs.docker.com/compose/
3. PHP: Hypertext Preprocessor. (n.d.). *PHP Group*. Доступно за посиланням: https://www.php.net
4. RoadRunner Documentation. (n.d.). *Spiriit Technologies*. Доступно за посиланням: https://roadrunner.dev/docs
5. SWOOLE Documentation. (n.d.). *Swoole Group*. Доступно за посиланням: https://www.swoole.co.uk
6. Frankent PHP Documentation. (n.d.). *Frankent Group*. Доступно за посиланням: https://github.com/frankent/frankent
7. Caddy Documentation. (n.d.). *Caddy Inc*. Доступно за посиланням: https://caddyserver.com/docs
8. NGINX Documentation. (n.d.). *NGINX Inc*. Доступно за посиланням: https://nginx.org/en/docs/
9. Apache HTTP Server Documentation. (n.d.). *Apache Software Foundation*. Доступно за посиланням: https://httpd.apache.org/docs/2.4/
10. Apache JMeter Documentation. (n.d.). *Apache Software Foundation*. Доступно за посиланням: https://jmeter.apache.org/
11. Laravel Documentation. (n.d.). *Laravel*. Доступно за посиланням: https://laravel.com/docs
12. W3C Statistics. (n.d.). *W3C Consortium*. Доступно за посиланням: https://www.w3.org/Statistics/
13. PostgreSQL Documentation. (n.d.). *PostgreSQL Global Development Group*. Доступно за посиланням: https://www.postgresql.org/docs/
14. Додаток до Умов прийому на навчання до закладів вищої освіти України, 2018. – 10 с. – (Міністерство освіти України)

ДОДАТКИ

Додаток А

Docker compose налаштування з php nginx postgresql

```
version: '3.8'

services:
    php:
        build:
            context: .
            dockerfile: Dockerfile
        volumes:
            - ./src:/var/www/html
            - ./php/www.conf:/usr/local/etc/php-fpm.d/www.conf
            - ./../php.ini:/usr/local/etc/php/conf.d/custom.ini
        networks:
            - app-network
    nginx:
        image: nginx:latest
        container_name: nginx_app
        ports:
            - "8081:80"
        volumes:
            - ./src:/var/www/public
            - ./server/nginx.conf:/etc/nginx/conf.d/default.conf
        depends_on:
            - php
        networks:
            - app-network
```

```
    restart: unless-stopped
  db:
    image: postgres:15-alpine
    restart: always
    environment:
      POSTGRES_DB: 'diplom'
      POSTGRES_USER: 'diplom'
      POSTGRES_PASSWORD: 'diplom'
    volumes:
      - ./../pg-db:/var/lib/postgresql/data
    networks:
      - app-network
    ports:
      - "54321:5432"
networks:
  app-network:
```

Додаток Б

Docker compose налаштування php, caddy, postgresql

```
services:

    php:
        build:
            context: .
            dockerfile: Dockerfile
        volumes:
            - ./src:/var/www/html
            - ./php/www.conf:/usr/local/etc/php-fpm.d/www.conf
            - ./../php.ini:/usr/local/etc/php/conf.d/custom.ini
        networks:
            - app-network
    caddy:
        image: caddy:latest
        depends_on:
            - php
        volumes:
            - ./src:/var/www/html
            - ./server/Caddyfile:/etc/caddy/Caddyfile
        ports:
            - "8081:80"
        networks:
            - app-network
    db:
        image: postgres:15-alpine
        restart: always
```

```
    environment:
      POSTGRES_DB: 'diplom'
      POSTGRES_USER: 'diplom'
      POSTGRES_PASSWORD: 'diplom'
    volumes:
      - ./../pg-db:/var/lib/postgresql/data
    networks:
      - app-network
    ports:
      - "54321:5432"
networks:
  app-network:
```

Додаток В

Docker compose налаштування з swoole nginx postgresql

```
version: '3.8'

services:
    swoole:
        build:
            context: .
            dockerfile: Dockerfile
        volumes:
            - ./src:/var/www/html
            - ./../php.ini:/usr/local/etc/php/conf.d/custom.ini
        networks:
            - app-network
        depends_on:
            -   db
    nginx:
        image: nginx:latest
        volumes:
            - ./server/nginx.conf:/etc/nginx/conf.d/default.conf
        ports:
            - "8081:80"
        depends_on:
            - swoole
        networks:
            - app-network
        restart: unless-stopped
```

```
    db:
        image: postgres:15-alpine
        restart: always
        environment:
            POSTGRES_DB: 'diplom'
            POSTGRES_USER: 'diplom'
            POSTGRES_PASSWORD: 'diplom'
        volumes:
            - ./../pg-db:/var/lib/postgresql/data
        networks:
            - app-network
        ports:
            - "54321:5432"
networks:
    app-network:
```

Додаток Г

Docker compose налаштування з roadrunner nginx postgresql

```
version: '3.8'
services:
    roadrunner:
        build:
            context: .
            dockerfile: Dockerfile
        volumes:
            - ./src:/var/www/html
            - /var/www/html/vendor
        expose:
            - "8080"
        networks:
            - app-network
        depends_on:
            -   db
    nginx:
        image: nginx:latest
        volumes:
            - ./server/nginx.conf:/etc/nginx/conf.d/default.conf
        ports:
            - "8081:80"
        depends_on:
            - roadrunner
        networks:
```

```
            - app-network
        restart: unless-stopped
    db:
        image: postgres:15-alpine
        restart: always
        environment:
            POSTGRES_DB: 'diplom'
            POSTGRES_USER: 'diplom'
            POSTGRES_PASSWORD: 'diplom'
        volumes:
            - ./../pg-db:/var/lib/postgresql/data
        networks:
            - app-network
        ports:
            - "54321:5432"
volumes:
    vendor:
networks:
    app-network:
```

Додаток Д

Приклади модіфікованого коду для roadrunner

```
<?php

use Nyholm\Psr7\Factory\Psr17Factory;
use Psr\Http\Message\ResponseInterface;
use Spiral\RoadRunner;
require __DIR__ . '/vendor/autoload.php';
$pdo = new PDO('pgsql:host=db;port=5432;dbname=diplom', 'diplom', 'diplom', [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$factory = new Psr17Factory();
$worker = RoadRunner\Worker::create();
$psr7 = new RoadRunner\Http\PSR7Worker(
    $worker, $factory, $factory, $factory
);
while ($request = $psr7->waitRequest()) {
    try {
        $psr7->respond(handleRequest($pdo));
    } catch (\Throwable $e) {
        $psr7->getWorker()->error((string)$e);
    }
}
function handleRequest(PDO $pdo): ResponseInterface
{
    $start = microtime(true);
```

```
    $factory = new Psr17Factory();
    $queries = [
        'users' => "SELECT count(*) FROM users where created_at between '2020-01-01' and '2025-01-31'",
        'lists' => 'SELECT count(*) FROM lists group by user_id',
        'files' => "select count(*) from files where path like '%2%' group by model_type",
    ];
    $results = [];
    foreach ($queries as $query) {
        $stmt = $pdo->query($query);
        $stmt->fetchAll();
    }
    $executionTime = microtime(true) - $start;
    return $factory->createResponse(200)
        ->withHeader('Content-Type', 'application/json')
        ->withBody(
            $factory->createStream(
                json_encode([
                    'time' => $executionTime,
                ])
            )
        );
}
```

Додаток Е

Приклади модіфікованого коду для swoole

```
<?php

use Swoole\Http\Server;
$server = new Server("0.0.0.0", 9501);
$server->set([
    'worker_num' => 16,
    'max_request' => 1000,
    'dispatch_mode' => 2,
    'task_enable_coroutine' => true
]);
$array = range(0, 1000000);
$server->on("request", function ($request, $response) use ($array) {
    $result = 0;
    $start = microtime(true);
    foreach ($array as $value) {
        $result = sqrt($value) + sqrt($result);
    }
    $executionTime = microtime(true) - $start;
    $response->header('Content-Type', 'application/json');
    $response->end(json_encode([
        'executionTime' => $executionTime,
    ]));
});
$server→start();
```

Coroutine

```
<?php

use Swoole\Http\Server;
use Swoole\Database\PDOConfig;
use Swoole\Database\PDOPool;
$server = new Server("0.0.0.0", 9501);
$server->set([
    'worker_num' => swoole_cpu_num() * 2,
    'max_coroutine' => 10000,
    'open_tcp_nodelay' => true,
    'socket_buffer_size' => 2 * 1024 * 1024,
    'buffer_output_size' => 2 * 1024 * 1024,
]);
$pool = new PDOPool(
    (new PDOConfig())
        ->withDriver('pgsql')
        ->withHost('db')
        ->withPort(5432)
        ->withDbname('diplom')
        ->withUsername('diplom')
        ->withPassword('diplom')
        ->withOptions([
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        ])
);
$server->on("request", function ($request, $response) use ($pool) {
    $start = microtime(true);
    $queries = [
```

```
        'users' => "SELECT count(*) FROM users where created_at between '2020-01-01' and '2025-01-31'",
        'lists' => 'SELECT count(*) FROM lists group by user_id',
        'files' => "select count(*) from files where path like '%2%' group by model_type",
    ];
    try {
        $tasks = [];
        foreach ($queries as $key => $query) {
            $tasks[$key] = go(function () use ($pool, $query) {
                $conn = $pool->get();
                try {
                    return $conn->query($query)->fetch();
                } finally {
                    $pool->put($conn);
                }
            });
        }
        Swoole\Coroutine::join(array_values($tasks));
        $executionTime = microtime(true) - $start;
        $response->header("Content-Type", "application/json");
        $response->end(json_encode(['time' => $executionTime]));
    } catch (Throwable $e) {
        $response->status(500);
        $response->end(json_encode(['status' => 'error', 'message' => $e->getMessage()]));
    }
});
```

Додаток Є

Запуск за допомогою фреймворка

DockerFile

```
FROM php:8.3.8-cli AS dev

ENV COMPOSER_HOME=/tmp/composer
ENV APPLICATION_DIRECTORY=/app
ARG USER_NAME=laravel
ARG USER_ID=1000
ARG GROUP_ID=1000
RUN groupadd -g $GROUP_ID $USER_NAME && \
    useradd -u $USER_ID -g $USER_NAME -m $USER_NAME
RUN apt-get update \
    && apt-get install -y \
    curl \
    libicu-dev \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    libpq-dev \
    libzip-dev \
    libbrotli-dev \
    && mkdir -p /var/www/.npm && chown -R $USER_NAME:$USER_NAME
/var/www/.npm \
```

```
    && pecl install swoole \
    && docker-php-ext-enable swoole \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*
RUN docker-php-ext-configure intl \
    && docker-php-ext-configure pdo_pgsql \
    && docker-php-ext-install exif pcntl bcmath gd intl soap pdo_pgsql zip sockets
COPY --from=composer:2.5.8 /usr/bin/composer /usr/bin/composer
RUN mkdir -p /.npm && chown -R $USER_NAME:$USER_NAME /.npm
USER $USER_NAME
WORKDIR $APPLICATION_DIRECTORY
COPY --chown=$USER_NAME:$USER_NAME ./composer.json ./
./composer.lock ./
COPY --chown=$USER_NAME:$USER_NAME ./package.json ./ ./package-
lock.json ./
COPY ./php.ini /usr/local/etc/php/conf.d/custom.ini
RUN composer install --ignore-platform-reqs --no-scripts
COPY --chown=$USER_NAME:$USER_NAME . ./
COPY ./../php.ini /usr/local/etc/php/conf.d/custom.ini
EXPOSE 8000
RUN chmod +x ./entrypoint.sh
ENTRYPOINT ["/app/entrypoint.sh"]
#CMD ["php", "artisan", "octane:start", "--workers=8", "--host=0.0.0.0", "--
port=8000"]
```

docker-compose.yaml

```
services:

    php:
        build:
```

```
        context: .
        args:
          GIT_TOKEN: ${GIT_TOKEN}
          GIT_DOMAIN: gitlab.quartsoft.com
    restart: always
    env_file:
      - .env
    volumes:
      - .:/app
      - vendor:/app/vendor
      - ./php.ini:/usr/local/etc/php/conf.d/custom.ini
    user: "${UID}:${GID}"
    ports:
      -   "8081:8000"
    networks:
      - app-network
    depends_on:
      - db
#  nginx:
#    image: nginx:latest
#    container_name: nginx_app
#    ports:
#      - "8081:80"
#    volumes:
#      - .:/app
#      - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf
#    depends_on:
#      - php
#    networks:
#      - app-network
```

```
#       restart: unless-stopped
#   caddy:
#       image: caddy:latest
#       depends_on:
#           - php
#       volumes:
#           - ./:/app
#           - ./caddy/Caddyfile:/etc/caddy/Caddyfile
#       ports:
#           - "8081:80"
#       networks:
#           - app-network
    db:
        image: postgres:15-alpine
        restart: always
        environment:
            POSTGRES_DB: 'diplom'
            POSTGRES_USER: 'diplom'
            POSTGRES_PASSWORD: 'diplom'
        volumes:
            - ./../pg-db:/var/lib/postgresql/data
        networks:
            - app-network
        ports:
            - "54321:5432"
volumes:
    postgres-data:
    vendor:
networks:
    app-network:
```