ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Зелінський Володимир Васильович

УДК 004.891

Дослідження методів оптимізації роботи веб-сайтів

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації магістра

Нормоконтроль

_____к.т.н. Мануляк І.
(підпис, дата, розшифрування підпису)

Студент

_____ Зелінський В.І
(підпис, дата, розшифрування підпису)

Допускається до захисту
Завідувач кафедри

_____к.т.н. Пашкевич (
(підпис, дата, розшифрування підпису)

Керівник роботи :

_____ к.т.н. Мануляк
(підпис, дата, розшифрування підпису)

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ТА

ОПИС ОБ’ЄКТА ДОСЛІДЖЕННЯ 9
1.1 Аналіз компонентів формування, які використовуються у веб-сторінках. 9
1.2 Огляд компонентів контенту веб-сторінок та їх вплив на швидкість завантаження. 11
1.3 Огляд доступної літератури та проблема дослідження 14
1.4 Аналіз оптимізації контенту веб-сторінок 14
1.5 Аналогічні дослідження та їх проблеми 22
РОЗДІЛ 2. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ 24
2.1 Процес та інструмент дослідження. Проведення контрольного тестування 24
2.2 Принцип роботи та метод впровадження оптимізації паралельних завантажень 30
2.3 Принцип роботи та метод впровадження кешування часто використовуваних об’єктів 36
2.4 Принцип роботи та метод впровадження HTTP стиснення 42
РОЗДІЛ 3. ПІДВЕДЕННЯ ПІДСУМКІВ ДОСЛІДЖЕННЯ ТА ОГЛЯД ПОВНИХ РЕЗУЛЬТАТІВ 52
3.1 Висновки проведеного дослідження 52
3.1.1 Метод оптимізації паралельних завантажень 52
3.1.2 Метод кешування часто використовуваних об’єктів 57
3.1.3 Метод HTTP стиснення 61
3.2 Підведення підсумків тестування 67
ВИСНОВКИ 85
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 86

АНОТАЦІЯ

В даній кваліфікаційній роботі проведено дослідження доцільності використання певних методів оптимізації роботи веб-сайтів. Проведено аналіз існуючих досліджень, їх актуальність. Було протестовано метод оптимізації паралельних завантажень, метод кешування часто використовуваних об’єктів та метод HTTP стиснення.

Результати показують що вибрані методи оптимізації зберігають свою актуальність та спроможні впливати на продуктивність роботи веб-сайтів.

Методи оптимізації, аналіз, принцип роботи, тестування, дослідження, оптимізація паралельних завантажень, кешування часто використовуваних об’єктів, HTTP стиснення.

SUMMARY

In this qualification work, a study of the feasibility of using certain methods to optimize the work of websites. The analysis of existing researches, their urgency is carried out. The method of optimizing parallel downloads, the method of caching frequently used objects and the method of HTTP compression were tested.

The results show that the chosen optimization methods remain relevant and can affect the performance of websites.

Optimization methods, analysis, principle of operation, testing, research, optimization of parallel downloads, caching of frequently used objects, HTTP compression.

ВСТУП

Актуальність теми. Однією з найважливіших рис веб-сайту, для користувача, є його швидкість. Якщо сторінка не завантажується більше 3-х секунд - це зменшує шанси того що користувач дочекається її повного завантаження. Для уникнення таких випадків розробники використовують методи для оптимізації веб-сайтів. Тому потрібно розуміти який метод підходить краще для проекту, та який результат можна отримати при його впровадженні.

При отриманні даних про ефективність використання методів оптимізації можна краще розрахувати доцільність впровадження того чи іншого методу, що значно економить час та ресурси при розробці веб-продуктів.

Мета і завдання. Визначення актуальності застосування вибраних методів дослідження та оновлення існуючих даних.

Об’єкт дослідження. Процес опрацювання даних веб-контенту, які втрачають свою актуальність.

Предмет дослідження. Алгоритми кешування, стиснення та паралельних завантажень.

Методи дослідження базуються на методах стиснення інформації (оптимізація паралельних завантажень, кешування часто використовуваних об’єктів, HTTP стиснення), методах статистичного аналізу.

Наукова новизна одержаних результатів. Отримали подальший розвиток методи, алгоритми та їх програмні реалізації стиснення контенту інформаційних сайтів, який втрачає свою актуальність.

Практичне значення одержаних результатів. Розроблено алгоритм та програму, яка може бути використана для зниження часу завантаження сторінки інформаційного ресурсу.

Апробація результатів дослідження. Результати дослідження опубліковано на міжнародній науково-практичній конференції “Дослідження інновацій та перспективи розвитку науки і техніки у XXI столітті”.

Структура. Дослідження поділене на 3 частини. У першій частині проводиться аналітика попередніх опублікованих результатів та інформація про методи дослідження. У другій частині проводиться тестування впроваджених методів оптимізації. У третій частині проводиться узагальнення отриманих результатів та робиться висновок.

Загальний обсяг сторінок основної частини – 82, а список використаних джерел містить 40 позицій.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ОПИС ОБ'ЄКТА ДОСЛІДЖЕННЯ

1.1 Аналіз компонентів формування, які використовуються у веб-сторінках

Файл HTML відіграє кілька важливих ролей на веб-сторінці. По-перше, використовується структура, створена HTML-кодом, щоб посилатися, покращувати й маніпулювати елементами веб-сторінки за допомогою CSS та JavaScript. Наприклад, можна використовувати HTML, щоб позначити всі заголовки на сторінці веб-браузера, а потім вибрати розмір і колір, які хочеться застосувати до цих заголовків, щоб відображати бренд організації, або просто візуальний дизайн, розроблений для сайту.

По-друге, HTML-текст дозволяє вказувати ролі різних структурних елементів для пошукових систем та інших служб, які індексують вміст і підсумовують його для інших користувачів. Наприклад, позначення підпису зображення елементом "figcaption" і включення зображення та його підпису в мета-елемент "figure" допомагає пошуковій системі зрозуміти, що ці два фрагменти вмісту пов'язані і що підпис описує пов'язане зображення.

CSS є однією з основних мов відкритої мережі і стандартизований у всіх веб-браузерах відповідно до специфікацій W3C. Раніше розробка різних частин специфікації CSS відбувалася синхронно, що дозволяло версійність останніх рекомендацій.

Починаючи з CSS3, обсяг специфікації значно збільшився, і прогрес у різних модулях CSS почав настільки відрізнятися, що стало більш ефективним розробляти та випускати рекомендації окремо для кожного модуля. Замість версій специфікації CSS, W3C тепер періодично робить снапшот останнього стабільного стану специфікації CSS.

JavaScript.

Оскільки JavaScript стає все популярнішим, команди використовують його підтримку на багатьох рівнях у своєму стеку — інтерфейсний, внутрішній, гібридні додатки, вбудовані пристрої тощо.

Майже всі вже чули про V8 Engine як концепцію, і більшість людей знає, що JavaScript є однопотоковим або що він використовує чергу зворотного виклику.

На рисунку 1.1 зображено як це виглядає у спрощеному вигляді:

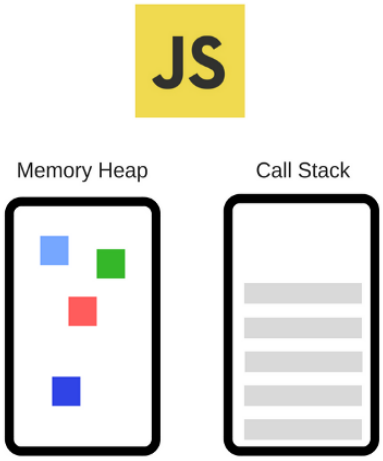

Рисунок 1.1 - Зображення двигуна на якому працює JavaScript

Популярним прикладом движка JavaScript є V8 від Google. Наприклад, V8 використовується в Chrome і Node.js.

Двигун складається з двох основних компонентів:

– memory heap — тут відбувається розподіл пам'яті;
– call stack — тут знаходяться кадри стека під час виконання коду;

На зображенні показано спрощений вигляд концепції двигуна на якому працює JavaScript та який використовується у розробці програмних продуктів.

Така концепція не дуже відома та отримує підтримку та подальший розвиток, що можна назвати прогресом у розробці ПЗ.

1.2 Огляд компонентів контенту веб-сторінок та їх вплив на швидкість завантаження

Зображення.

Найкращою практикою для SEO є збереження файлів зображень веб-сторінки якомога меншими, оскільки зображення, як правило, займають найбільшу частину загальної кількості переданих байтів. Якщо сайт не має належним чином оптимізованих зображень, він може працювати не так добре, як це можливо.

Згідно з дослідженням, проведеним Google, швидкість завантаження сторінки є одним з найбільших факторів, чому користувачі залишають сайти або “відмовляються” від них. Зараз відомо, що швидкість завантаження сторінки є важливим органічним фактором рейтингу в пошуковій видачі.

На рисунку 1.2 можна побачити як зросла кількість переданих даних, починаючи з 2012 року.

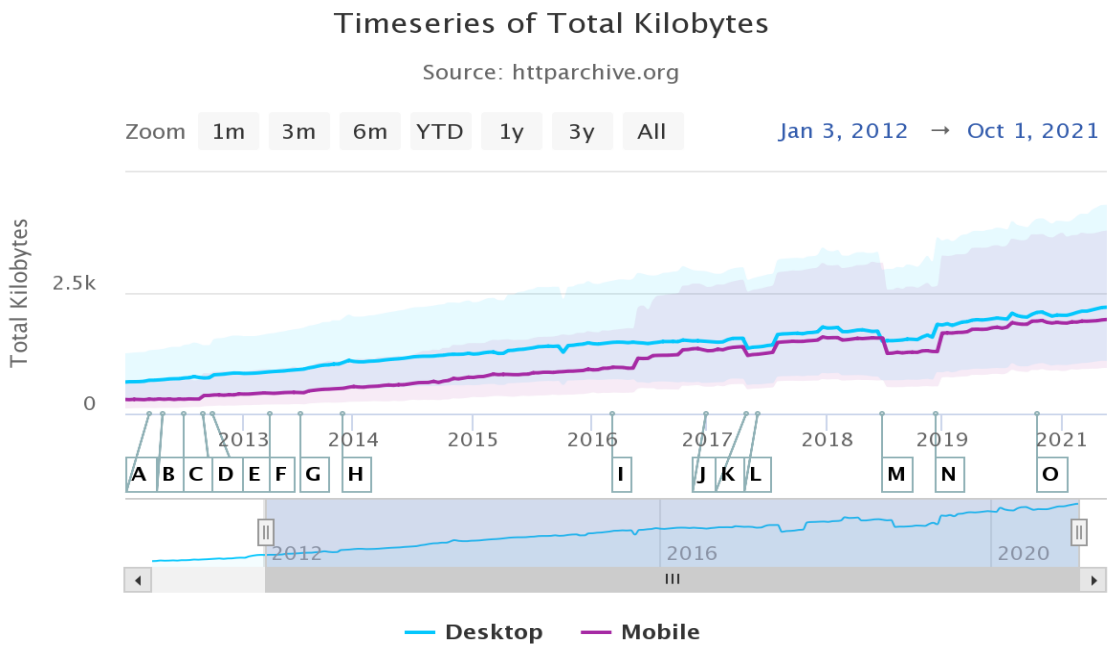

Рисунок 1.2 – Статистика кількості переданих даних

Які аспекти ваги сторінки найбільше впливають на продуктивність?

Якщо вивчити 1000 найкращих сайтів і витягнути деяку статистику ваги сторінки з HTTP-архіву, можна порівняти дані HTTP-архіву з CrUX.

На графіках нижче наведено відсоток сторінок із часом завантаження менше 2 секунд. Вісь Y – це середня кількість байтів, а вісь X – відсоток сайтів із швидким завантаженням сторінок.

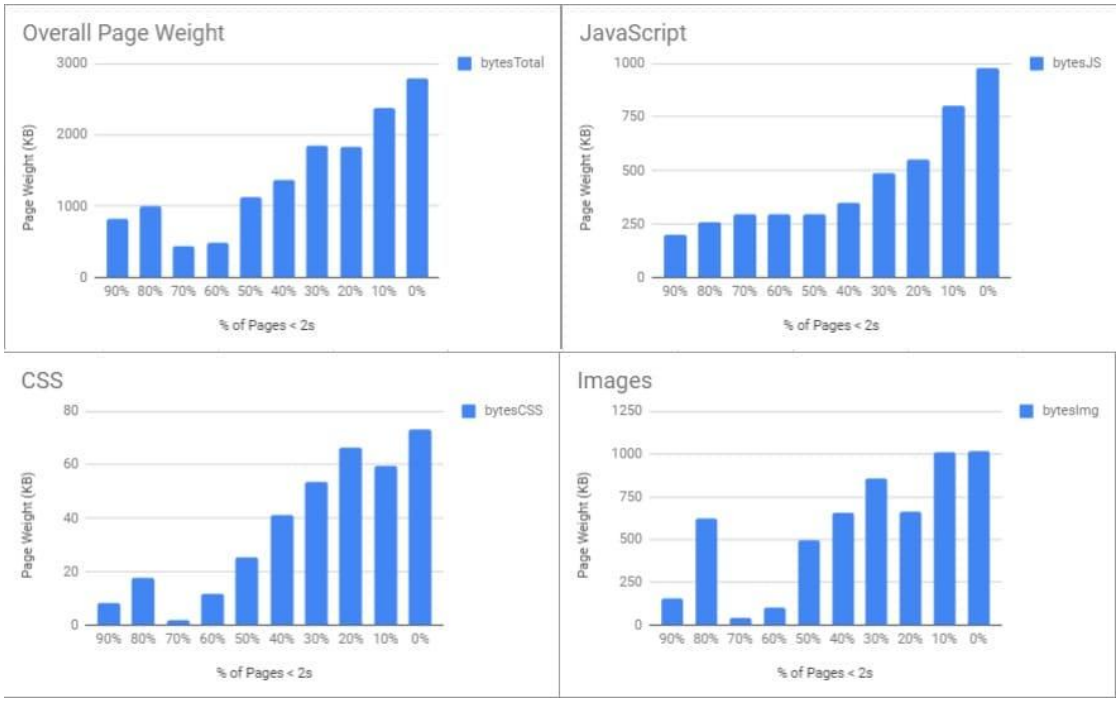

Рисунок 1.3 – Відсоток сторінок із часом завантаження менше 2 секунд

У верхньому лівому графіку вага сторінки показує сильну кореляцію, і сайти з менш швидкими сторінками, як правило, мають більшу вагу сторінки. Решта 3 графіки показують, як JavaScript, CSS та зображення сприяють збільшенню ваги та продуктивності сторінки. Виходячи з цих графіків, зображення та JavaScript вносять найбільший внесок у вагу сторінки, що впливає на час завантаження. А для деяких повільних сайтів кількість стисненого JavaScript насправді перевищує кількість байтів зображення.

Відео.

Згідно зі звітом Google за 2018 рік, трохи більше половини всіх користувачів мобільних пристроїв залишають веб-сайти, для завантаження яких потрібно більше трьох секунд.

У той же час Mach Metrics повідомляє, що останній індекс швидкості 2020 року становить 4,7 секунди для користувачів настільних комп'ютерів і 11,4 секунди для мобільних пристроїв.

Переважна більшість веб-сторінок в Інтернеті сьогодні втрачають більше половини потенційних клієнтів лише тому, що вони завантажуються недостатньо швидко. Експерти з SEO знають, що швидкість завантаження сторінки була важливим фактором рейтингу в алгоритмі пошуку Google. Так було багато років.

Чи вбудовані відео сповільнюють час завантаження сторінки?

Ось коротка відповідь: Залежно яке.

Справа в тому, що існує багато способів вбудовування відео на веб-сторінку.

Те, як можна вирішити це зробити, матиме значний вплив на час завантаження веб-сторінки.

Ось кілька прикладів різних способів, якими ви можете вставити відео на веб-сторінку сьогодні:

- самостійний хостинг: розміщення відео безпосередньо на сторінці;
- вбудовування YouTube: YouTube оптимізовано для потокового передавання й належить Google, тому це може бути непоганим варіантом;
- індивідуальний відеохостинг: найкращим варіантом є розміщення відео на спеціально створеній платформі з власним брендом;

Кожен з цих трьох варіантів має свій набір переваг і недоліків.

Відео, розміщене на власному хостингі, є найгіршими з точки зору часу завантаження. Вони, по суті, змушують кожного користувача завантажувати відео безпосередньо з сервера веб-сайту.

Вбудовані відео YouTube, безумовно, працюватимуть швидше, ніж відео, розміщені самостійно. Однак YouTube заради монетизації ставить під загрозу швидкість завантаження сторінки.

1.3 Огляд доступної літератури та проблема дослідження

Для проведення дослідження можна використати літературу, яка знаходиться у вільному доступі. Для базового розуміння методів оптимізації можна скористатися книгою “Website Optimization” автора Ендрю Б. Кінга, в ній описана базова інформація для ознайомлення з багатьма методами оптимізації, зокрема тими які використані у даному дослідженні. Для більш поглибленого вивчення можна скористатися книгою автора Патріка Кіллелеа під назвою “Web Performance Tuning”.

1.4 Аналіз оптимізації контенту веб-сторінок

Дослідження спрямоване на три методи оптимізації роботи веб-сервера, а саме:

- оптимізація паралельних завантажень;
- кешування часто використовуваних об’єктів;
- використання HTTP стиснення;

Оптимізація паралельних завантажень

Паралельні завантаження – це просто кілька завантажень, які відбуваються одночасно в кількох доменах, розміщених на одному хості. Основною перевагою паралельного завантаження є скорочення часу завантаження при завантаженні багатьох ресурсів.

Важливо зауважити, що веб-браузери не призначені для паралельного завантаження. Зазвичай веб-браузери обмежують кількість одночасних підключень до хоста. Зазвичай вони обмежуються 6 одночасними передачами.

Це призводить до відставання ресурсів, коли багато активів очікують на завантаження. Браузери встановлюють якомога більше одночасних з’єднань, але ставлять ресурси в чергу, щоб забезпечити їх завантаження по одному.

Це автоматично збільшує час, необхідний для завантаження ресурсів з того самого хоста. Паралельні завантаження, також відомі як розділення домену, скорочують час, необхідний для встановлення з'єднання, що призводить до більш швидкого завантаження сторінки та швидкості завантаження.

Переваги:

- більш висока швидкість завантаження: створюючи кілька субдоменів, кількість ресурсів, які одночасно завантажуються, збільшується. Оскільки браузери обмежують підключення на основі хостів, створення субдоменів додає нові пули підключень, які браузери можуть створити, дозволяючи більше одночасних завантажень;
- швидше завантаження сторінки: паралельне завантаження дозволяє розділити запити на ресурси між багатьма доменами. В результаті на блокування сторінок витрачається мало часу. Це в кінцевому підсумку збільшує час, необхідний для завантаження сторінок приблизно на 33%;

Хоча паралельне завантаження може допомогти скоротити час, необхідний для завантаження ресурсів/завантаження ресурсів на сторінках, перед використанням паралельного завантаження між доменами на вашому сайті необхідно враховувати ряд недоліків. Нижче наведено основні мінуси, які слід враховувати.

Недоліки:

- паралельні завантаження призводять до збільшення кількості пошуків DNS і збільшують час, необхідний для створення початкових з'єднань, коли відбувається занадто багато завантажень, що, у свою чергу, може негативно вплинути на продуктивність. Тому важливо переконатися, що час підключення і додаткові пошуки DNS того варті;
- активи втратять переваги кешування, якщо ви не переконаєтеся, що ресурси постійно обслуговуються з подібних субдоменів. Для досягнення позитивного ефекту важливо використовувати одну зону з псевдонімами зон;

- можна понести додаткові витрати, наприклад, витрати на покупку сертифіката SSL для субдоменів або купівлю всього сертифіката SSL із підстановкою, якщо вашому сайту потрібен HTTPS. Вам не потрібно купувати сертифікати SSL для субдоменів, якщо ваш сайт не HTTPS, оскільки ви можете передавати ресурси через основний домен;
- багато веб-переглядачів мають політику того самого походження, яка запобігає міждоменним порушенням. Паралельні завантаження можуть піддати ваш веб-сайт потенційним загрозам безпеки під час переміщення статичного вмісту в окремі домени або субдомени. Розгляньте можливість увімкнення CORS;
- SPDY, сприйнятливий HTTP/2 у 2016 році (оскільки він замінить SPDY), робить сегментування домену застарілим, але поки що не кожен браузер підтримує SPDY;

Специфікація HTTP 1.1 рекомендує браузерам обмежувати завантаження двома об'єктів на ім'я хоста. Ця рекомендація була створена в 1999 році, за часів комутованого зв’язку та менш надійних проксі-серверів. У більшості браузерів за замовчуванням це обмеження. Хоча користувачі можна змінити ці параметри за замовчуванням, більшість не турбується про це. Для сайтів, розміщених на одному домену, результатом є повільніше завантаження об’єктів, які завантажуються по два за раз. Тепер, коли пропускна здатність і проксі-сервери покращилися, можна покращити паралельність за допомогою кількох доменів (або субдоменів) для доставки об’єктів. Yahoo! вияснили що збільшення кількості імен хостів до двох було оптимальним.

Збільшення до чотирьох або більше імен хостів фактично погіршило продуктивність для більших зображень через зайві витрати на запити за межами сайту та “подрібнення ЦП”. Райан Брін з Гомеса повідомив про подібні покращення після збільшення кількості субдоменів для обслуговування об’єктів. Він використовував записи канонічних імен (CNAME) системи доменних імен (DNS) для створення таких субдоменів, як

images1.example.com, images2.example.com і images3.example.com, які вказували на головний сервер www.example.com. Потім він використав код, щоб призначити субдомени зображенням, хоча всі вони вказують на той самий сервер.

Кешування часто використовуваних об’єктів

Кешування — це тимчасове зберігання часто використовуваних даних у високошвидкісних носіях (як правило, SRAM або RAM) або на носіях, розташованих ближче до користувача, для більш ефективного пошуку. Веб-кешування зберігає часто використовувані об’єкти ближче до клієнта через кеш браузера, проксі-сервера або сервера. Зберігаючи “свіжі” об’єкти ближче до ваших користувачів, ви уникаєте непотрібних запитів HTTP і мінімізуєте “стрибки” DNS. Це зменшує споживання пропускної спроможності та навантаження на сервер, а також покращує час відповіді. Yahoo! За оцінками, від 62% до 95% часу, необхідного для отримання веб-сторінки, витрачається на виконання HTTP-запитів для об’єктів. Кешування допомагає зменшити дорогі HTTP-запити для підвищення продуктивності. На жаль, кешування використовується недостатньо і часто його неправильно розуміють в Інтернеті. Опитування, проведене в липні 2007 року на веб-сайтах компаній Fortune 1000, показало, що 37,9% використовували заголовки контролю кешу. Опитування не говорить вам про те, що більшість цих сайтів використовують заголовки “не керувати”. Розробники регулярно знищують кеш, боячись надати застарілий вміст.

Браузери також не допомогли ситуації. Щоб уникнути запитів “304”, які надходять від повторної перевірки раніше завантажених об’єктів, розробники прийняли схему перевірки один раз за сеанс. Однак, якщо користувач не вимкне свій веб-переглядач, він зможе побачити застарілий вміст. Одним із рішень є кешування веб-об’єктів на тривалі періоди (деякі розробники встановлюють термін їх дії на 20 років у майбутнє), зміна назв файлів об’єктів для оновлень і використання коротших термінів дії для документів HTML, які, як правило, змінюються частіше. Кешування

призначене не тільки для статичних сайтів; навіть динамічні сайти можуть скористатися цим. Кешування динамічно створеного вмісту є менш корисним, ніж кешування всіх залежних об’єктів, таких як сценарії, стилі, зображення та Flash, які часто повторно запитуються або принаймні перевіряються браузерами чи посередниками. Залежні об’єкти, такі як мультимедійні об’єкти, зазвичай не змінюються так часто, як файли HTML. Графіці, яка рідко змінюється, наприклад логотипам, заголовкам і панелям навігації, можна надати довший термін дії, тоді як ресурсам, які змінюються частіше, таким як файли HTML і XML, можна надати менший термін дії. Розробляючи свій сайт з урахуванням кешування, ви можете націлюватися на різні класи ресурсів, щоб надати їм різний термін дії лише за допомогою кількох рядків коду.

Останнім часом схема паралельного завантаження (PD) була прийнята рядом програм для завантаження файлів в Інтернеті. Згідно зі схемою PD, клієнт, який запитує файл, відкриє одночасне з’єднання з кількома відправниками, які можуть бути серверами або одноранговими, і різні частини файлу будуть передані від відправників до клієнта. Були експерименти, які показують, що PD призводить до більш високого агрегованого пропускна спроможність завантаження і, отже, менший час завантаження для клієнтів за рахунок більшої кількості сигналів через необхідність координації серверів.

Попередні роботи над паралельним завантаженням були зосереджені на часі завантаження окремого клієнта. Дослідження продуктивності паралельного завантаження, коли воно використовується великою кількістю клієнтів, не проводилося. В результаті не можна зробити висновок, чи PD є хорошою схемою на основі попередньої роботи. Насправді існують дві загальні, але суперечливі точки зору: перша точка зору полягає в тому, що паралельне завантаження, як правило, ефективне, оскільки воно прискорює завантаження, використовуючи потужність серверів більш збалансовано. Однак інша думка полягає в тому, що паралельне завантаження нічим не

краще, якщо не гірше, ніж схема завантаження з одним сервером, оскільки скорочення часу завантаження стане менш значним, якщо кожен клієнт вирішить виконувати PD. Також не зрозуміло, який компроміс між середнім часом завантаження кожного клієнта та загальною швидкістю блокування запитів. На жаль, попередні експериментальні дослідження не дали відповіді на ці фундаментальні питання.

Ідея паралельного доступу до кількох серверів CDN була запропонована в останній літературі як емпірична методика. Родрігес і Бірсак вивчають схему динамічного паралельного доступу для доступу до кількох дзеркальних серверів. У їхньому дослідженні клієнт завантажує файли з дзеркальних серверів, що знаходяться в глобальній мережі. Вони показують, що їхня динамічна схема паралельного завантаження досягає значного прискорення завантаження відносно однієї схеми сервера. Однак вони вивчають лише сценарій, коли один клієнт використовує паралельне завантаження. Автори не враховують ефект і наслідки, коли всі клієнти вирішують прийняти ту саму схему. Інші роботи щодо паралельного завантаження надають експериментальні результати щодо підвищення продуктивності з точки зору окремого клієнта, не розглядаючи вплив, якщо всі клієнти використовують паралельне завантаження. Крім того, їхні результати отримані з експериментів під певними налаштуваннями та параметрами системи. Щоб здійснити паралельний доступ до кількох серверів, клієнт повинен або визначити частину документа яка йому потрібна з кожного із серверів, або закодувати документ таким чином, щоб він потребував меншої синхронізації між клієнтом і серверами. Небат і Сіді розглядають сценарій, коли "пакети" запитуються з кількох серверів, і аналізують кількість буфера на стороні клієнта, необхідного для обробки паралелізму. Byers et al. запропонувати нову схему кодування Tornado Code, яка працює як FEC, але з меншою складністю обчислень, щоб мінімізувати потребу в координації між серверами. Перевага використання Tornado Code полягає в тому, що клієнти, які отримують дані з кількох серверів, не

потребують складної сигналізації, щоб визначити, скільки і яких даних очікувати від кожного сервера, всупереч. Клієнт може просто наказати серверам зупинитися, як тільки він отримає “достатньо” закодованих сегментів даних для відновлення файлу. Недоліком Tornado Code є те, що клієнт повинен буферизувати всі дані, перш ніж мати змогу відновити файл.

HTTP стиснення

Стиснення HTTP – це загальнодоступний спосіб стиснення текстового вмісту, що передається з веб-серверів у браузери. Для стиснення HTML, JavaScript, CSS, XML та інших текстових файлів на сервері використовуються загальнодоступні алгоритми стиснення, такі як gzip і compress. Цей заснований на стандартах метод доставки стисненого вмісту вбудований у HTTP 1.1. Усі сучасні браузери, які підтримують файли HTTP 1.1 і PNG, підтримують ZLIB інфляцію роздутих документів (див. майбутню бічну панель “Браузери, які підтримують стиснення HTTP”). Іншими словами, вони можуть автоматично розпаковувати стислі файли, що заощаджує час і пропускну здатність.

У форматі Portable Network Graphics (PNG) використовується алгоритм стиснення ZLIB. ZLIB також може розпаковувати дані, запаковані gzip. Отже, браузери, які можуть обробляти PNG-файли, вже мають необхідне програмне забезпечення для розпакування даних gzip. Internet Explorer 4 і новіших версій (окрім IE Mac версій 4.5 і 5), Firefox і Opera 5.12+ підтримують стиснення HTTP.

Хоча сьогодні існує багато різних алгоритмів стиснення без втрат, більшість із них є варіаціями двох популярних схем: кодування Хаффмана та алгоритму Лемпеля-Зіва. Кодування Хаффмана працює шляхом присвоєння двійкового коду кожному із символів (символів) у вхідному потоці (файлі). Це досягається шляхом спочатку побудови двійкового дерева символів на основі частоти їх появи у файлі. Призначення двійкових кодів символам здійснюється таким чином, що найбільш часто зустрічаються символи

отримують найкоротші двійкові коди, а найменш часто зустрічаються символи — найдовший. Це, у свою чергу, створює менший стиснутий файл.

Алгоритм Лемпеля–Зіва, також відомий як LZ-77, використовує надлишкову природу даних для забезпечення стиснення. Алгоритм використовує те, що називають ковзаючим вікном, щоб відстежувати останні n байтів даних. Кожного разу, коли зустрічається фраза, яка існує в буфері ковзного вікна, вона замінюється вказівником на початкову позицію попередньої фрази у розсувному вікні разом із довжиною фрази.

Основною метрикою для алгоритмів стиснення даних є коефіцієнт стиснення, який відноситься до відношення розміру вихідних даних до розміру стиснутих даних. Наприклад, якби у нас був файл розміром 100 кілобайт і ми могли стиснути його до 20 кілобайт, ми б сказали, що коефіцієнт стиснення становить 5 до 1, або 80%. Вміст файлу, зокрема надмірність і впорядкованість даних, може сильно вплинути на коефіцієнт стиснення.

Багато веб-ресурсів, такі як HTML, JavaScript, CSS і XML, є просто текстовими файлами ASCII. Враховуючи той факт, що такі файли часто містять багато повторюваних послідовностей ідентичної інформації, вони є ідеальними кандидатами для стиснення. Інші ресурси, такі як зображення JPEG і GIF, а також потокові аудіо- та відеофайли, попередньо стискаються і, отже, не отримають вигоди від подальшого стиснення. Таким чином, під час стиснення HTTP фокус зазвичай обмежується текстовими ресурсами, які отримують найбільшу економію байтів від стиснення.

Схеми кодування для таких текстових ресурсів повинні забезпечувати стиснення даних без втрат. Як випливає з назви, алгоритм стиснення даних без втрат — це той, який може відтворити вихідні дані, біт за бітом, зі стиснутого файлу. Можна легко уявити, як втрата або зміна одного біта у файлі HTML може вплинути на його значення.

Метою стиснення HTTP є зменшення розміру певних ресурсів, які передаються між сервером і клієнтом. Зменшуючи розмір веб-ресурсів,

стиснення може ефективніше використовувати пропускну здатність мережі. Стиснутий вміст може також заощадити гроші тим особам, які сплачують комісію залежно від обсягу споживаної пропускної здатності. Що ще важливіше, оскільки передається менше байтів, клієнти зазвичай отримають ресурс за менший час, ніж якби він був надісланий у нестисненому вигляді. Особливо це стосується вузькосмугових клієнтів. Модеми зазвичай представляють те, що називають найслабшою ланкою або найдовшою милей у передачі даних; тому методи скорочення часу завантаження особливо актуальні для цих користувачів.

Крім того, стиснення потенційно може полегшити деяку частину навантаження, пов’язаного з фазою повільного запуску TCP. Фаза повільного запуску TCP є засобом контролю кількості перевантажень у мережі. Вона працює, створюючи невелике початкове вікно перевантаження для кожного нового TCP-з’єднання, обмежуючи тим самим кількість пакетів максимального розміру, які спочатку можуть бути передані відправником.

У випадку, коли HTML-документ надсилається у стисненому форматі, імовірно, що перші кілька пакетів даних міститимуть більше HTML-коду, а отже, і більшу кількість вбудованих посилань на зображення, ніж якби той самий документ був надісланий без стиснення. Як наслідок, клієнт може згодом надсилати запити на ці вбудовані ресурси швидше, що полегшує деяке навантаження на повільний запуск. Крім того, вбудовані об’єкти, ймовірно, будуть на тому ж сервері, що й цей HTML-документ.

1.5 Аналогічні дослідження та їх проблеми

Раніше проводилися тести даних методів оптимізації, проте проблема полягає в тому що ці дані досить складно найти, вони не повні, а головна проблема - застаріла інформація.

В наш час прогрес розвивається дуже стрімко. Звідси можна зробити висновок що тести які проводилися біля 10 років назад уже втрачають свою актуальність. Кількість людей які одночасно підключені до мережі набагато більша, порівняно з 2006 роком, отже і кількість цих користувачів, яка одночасно заходить на одну сторінку більша в декілька раз. Навантаження на веб-сервери набагато більше ніж тоді, при цьому ми досі керуємося даними за 2006-2010 роки, які не передбачають такого стрімкого збільшення користувачів до 2021 року.

При отриманні нових результатів можна краще розуміти ситуацію в наш час, чи справляються методи оптимізації зі сторінками сьогоднішніх стандартів, адже вони явно вищі ніж стандарти багаторічної давності.

РОЗДІЛ 2. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

2.1 Процес та інструмент дослідження. Проведення контрольного тестування

Дослідження проводиться у декілька етапів:

- навантаження сайту без використання методу оптимізації та запис даних;
- навантаження сайту з використанням методу оптимізації та запис даних;
- порівняння результатів;
- підведення підсумку проведеного тесту;

Після завершення дослідження проводиться підсумок і узагальнення отриманих результатів.

WAPT

WAPT – Тестування навантаження, стресу та продуктивності веб-додатків – економічно ефективний і простий у освоєнні інструмент тестування веб-завантаження.

WAPT дозволяє виконувати завантаження веб-сайту та тестування продуктивності, створюючи велике навантаження з однієї або кількох робочих станцій. Ви можете налаштувати та запустити свої тести за допомогою цього інструмента за лічені хвилини, а також швидко отримати звіти про ефективність свого веб-сайту чи веб-програми.

Вигляд початкового вікна програми з боковим меню зображено на рисунку 2.1.

WAPT використовує потужних віртуальних користувачів (які є такими ж, як і реальних користувачів) з повним контролем над тим, як налаштувати цих віртуальних користувачів.

WAPT – інструмент продуктивності веб-сайту виконує тест, емулюючи активність багатьох віртуальних користувачів. Кожен віртуальний користувач може мати власні налаштування профілю. Можна мати тисячі віртуальних користувачів, які одночасно діють на веб-сайті, виконуючи будь-яку діяльність, як-от читання чи запис за допомогою веб-сервера.

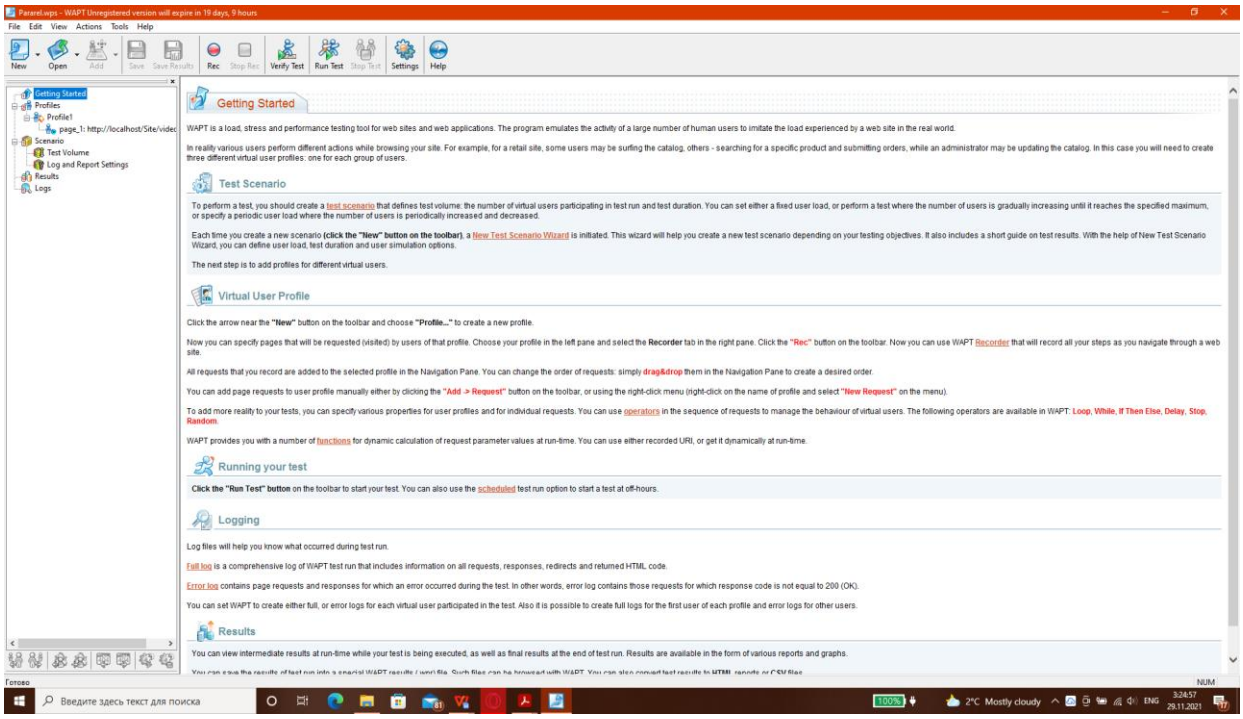

Рисунок 2.1 – Основний інтерфейс програми WAPT

WAPT використовує вбудований Microsoft Internet Explorer, який використовується для запису вашої взаємодії з веб-сайтом. Під час запису тесту всі динамічні параметри записуються як статичні значення, які можна налаштувати пізніше під час виконання тесту.

Потім потрібно налаштувати кожного користувача різними параметрами, такими як унікальні сеанси, кількість віртуальних користувачів, значення динамічних параметрів тощо.

Після того, як завершується запис і налаштування, достатньо просто перевірити, чи готовий тест до запуску, а потім потрібно виконати тести продуктивності, щоб перевірити, чи все виглядає нормально.

Вигляд вікна налаштування сценарію зображено на рисунку 2.2.

Нарешті, потрібно проаналізувати звіти, щоб визначити, чи були тести ефективності веб-сайту прийнятними чи провальними за набором визначених стандартів.

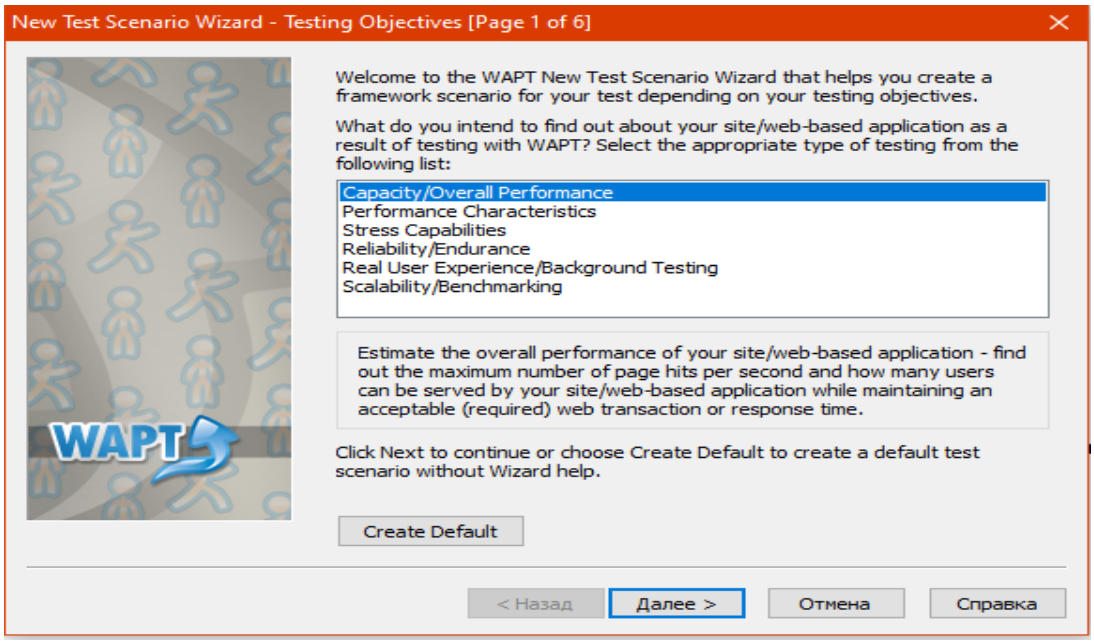

Рисунок 2.2 – Вікно налаштування нового сценарію

Для порівняння результатів дослідження необхідно провести контрольне тестування без впровадження методів оптимізації.

Для цього в програмі потрібно створити новий сценарій та вибрати профілі, яким буде задано певний сценарій, тобто перелік дій які вони повинні виконати на сайті.

В розділі налаштування сценарію можна вибрати за яких умов він буде завершуватися, кількість користувачів і час виконання тестування.

Після виконаних дій потрібно очікувати завершення тестування та виведення результатів на екран.

У боковому меню з’являться опції вибору загального графіку та графіку із даними швидкості завантаження стоорінок.

На рисунку 2.3 зображено налаштування створеного сценарію, де показано які профілі будуть виконувати сценарій, скілький часу це займе і візуально зображено приріст користувачів на певних відрізках часу.

Меню дає користувачеві можливість налаштувати під свої потреби.

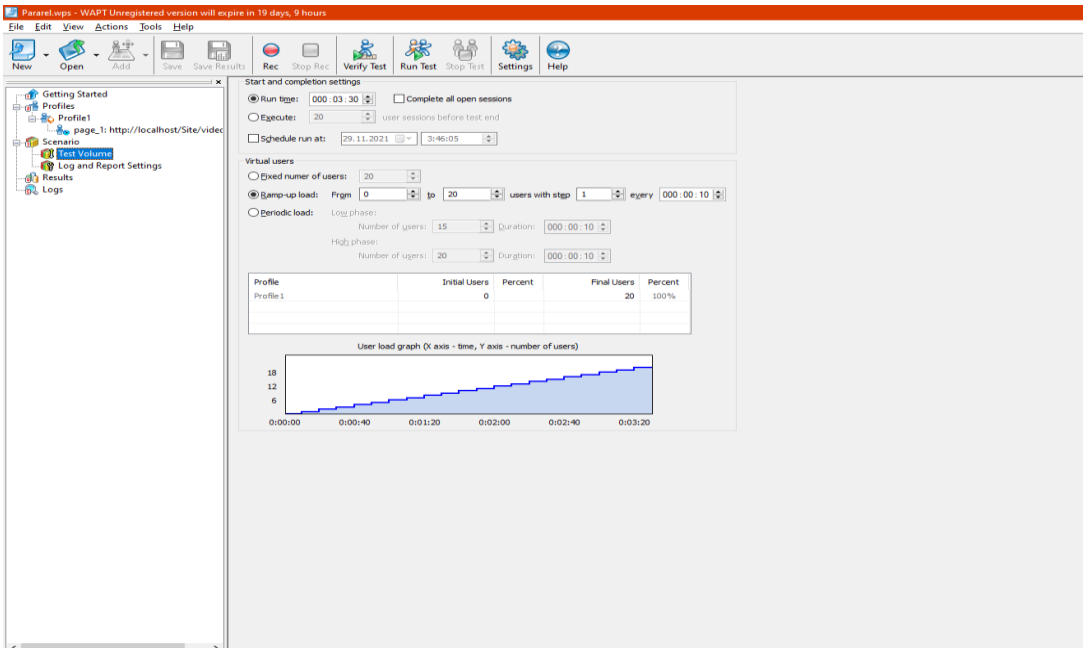

Рисунок 2.3 – Меню налаштування створеного сценарію

На рисунку 2.4 можна побачити як виглядає вікно з загальним графіком результатів.

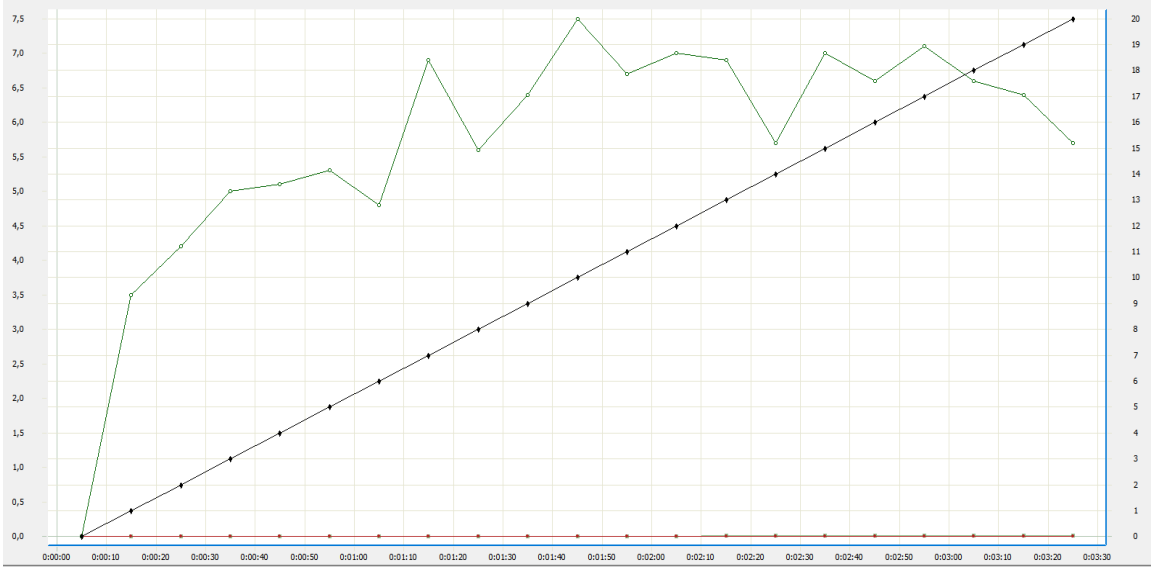

Рисунок 2.4 - Вікно з графіком завантаження кількості сторінок в секунду

В даному випадку зображено вікно з графіком кількості завантажених сторінок в секунду.

Такий графік корисний при необхідності візуального представлення процесу завантаження сторінки.

Плюсом є те що він кастомізований, тобто на ньому можна відключити непотрібні для користувача дані і залишити тільки ті які його цікавлять

Для порівняння створено дві сторінки з графічними елементами та анімаційними елементами. Графічні статичні елементи навантажують сторінку більше ніж анімаційні, адже роздільна здатність статичних елементів більша, в даному випадку 3840x2160 пікселів.

Анімовані графічні елементи у форматі GIF мають набагато меншу роздільну здатність - 800x600 пікселів.

На рисунку 2.5 зображено приклад статичного графічного елементу з роздільною здатністю 3840x2160.

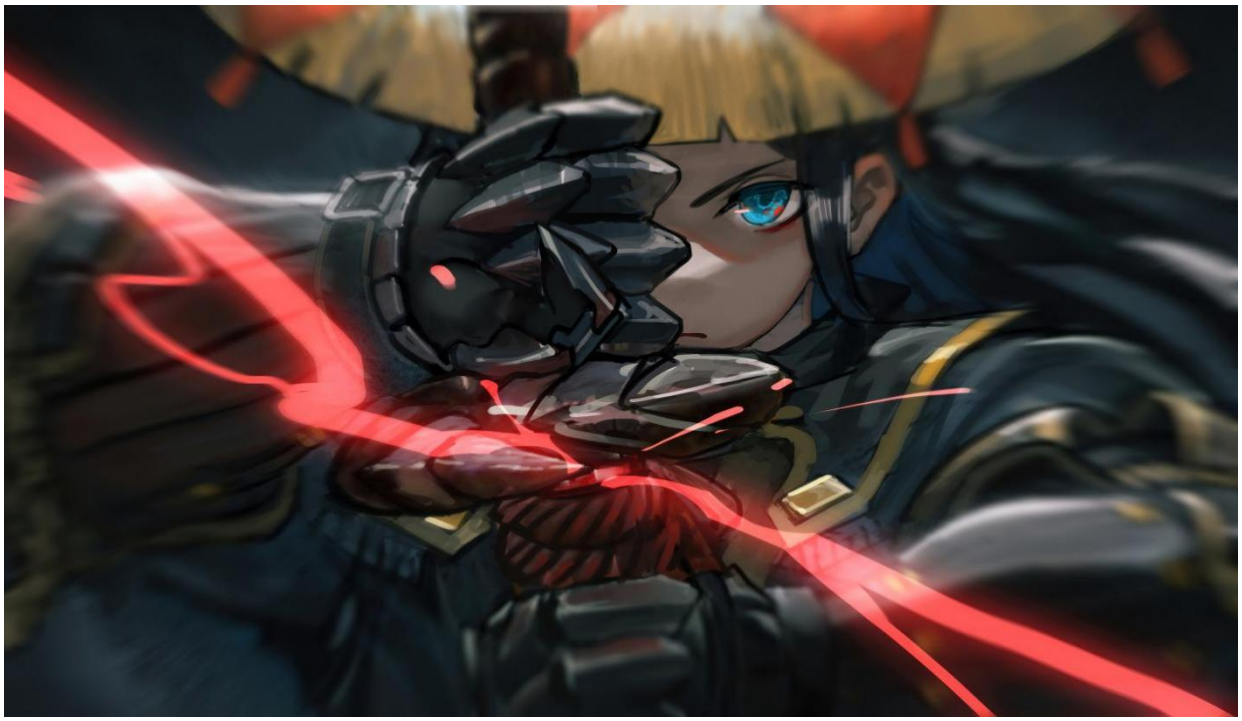

Рисунок 2.5 – Один із графічних елементів, використаних при тестуванні

Звичайні графічні елементи не навантажують сторінку так як анімаційні, але проблема полягає в тому що вони можуть довго завантажуватися якщо їх досить велика кількість на одній сторінці, тим більше якщо вони завантажуться з одного сервера.

На рисунку 2.6 можна побачити як виглядає вікно з графіком конкретного направлення, в даному випадку зображено графік швидкості завантаження сторінок.

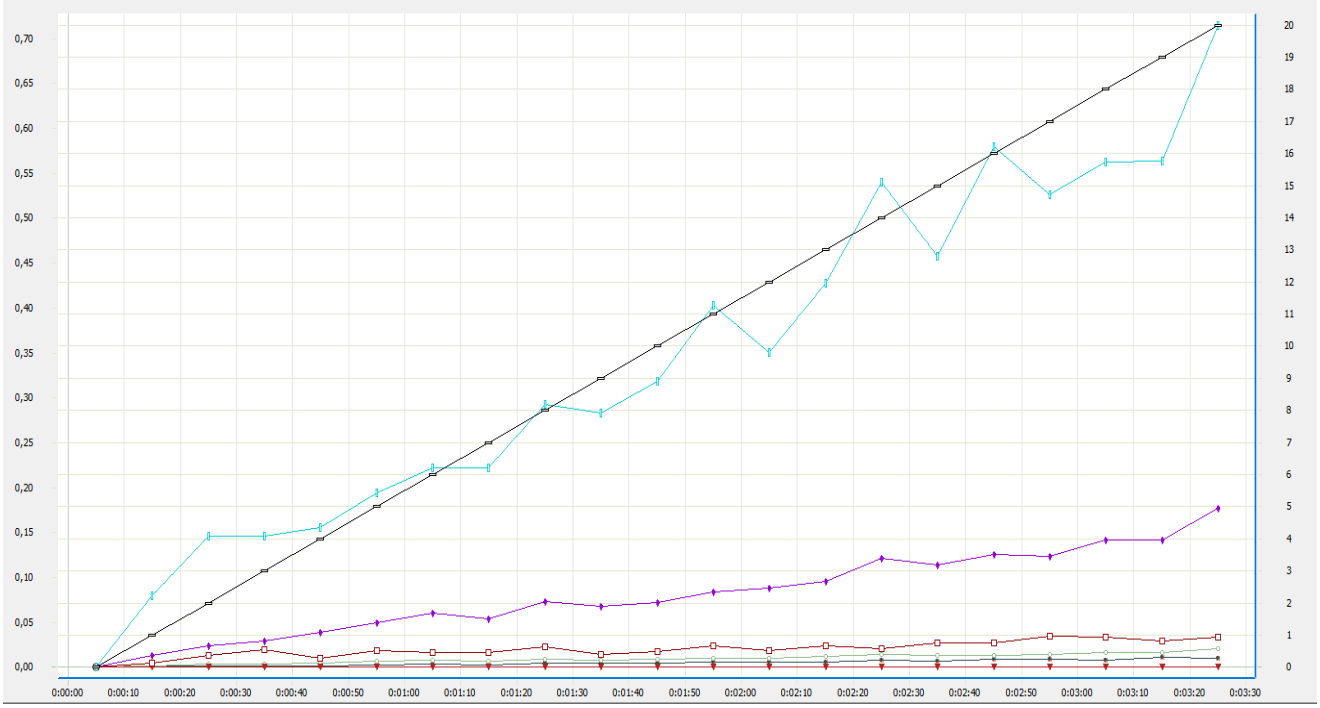

Рисунок 2.6 - Вікно з графіком швидкості завантаження сторінки

Маючи дані про навантаження на сторінку без оптимізаційних методів можна провести дослідження і вияснити як впливають ці методи на завантаження сторінки.

До кожного методу описується спосіб його впровадження. Після цього проводиться тестування і узагальнення підсумків тестування, також в кінці роботи можна ознайомитися з графіками та результатами наведеними у таблицях.

2.2 Принцип роботи та метод впровадження оптимізації паралельних завантажень

Трафік і популярність Інтернету значно зросли. В результаті популярні веб- та ftp-сайти змушені мати справу з більшим навантаженням.

З іншого боку, очікування користувачів зросли; вони хочуть завантажити потрібну інформацію в найкоротші терміни.

Було запропоновано два основних підходи для боротьби зі збільшенням навантаження та досягнення короткого часу завантаження. Перший — використовувати проксі для кешування інформації ближче до клієнтів. Інший — реплікувати інформацію на дзеркальних серверах і спрямовувати кожного клієнта на той, який може забезпечити найкращу продуктивність. Клієнт направляється автоматично або повинен вибрати сервер вручну зі списку всіх дзеркальних серверів. На рисунку на 2.7 зображено схему принципу роботи паралельних завантажень.

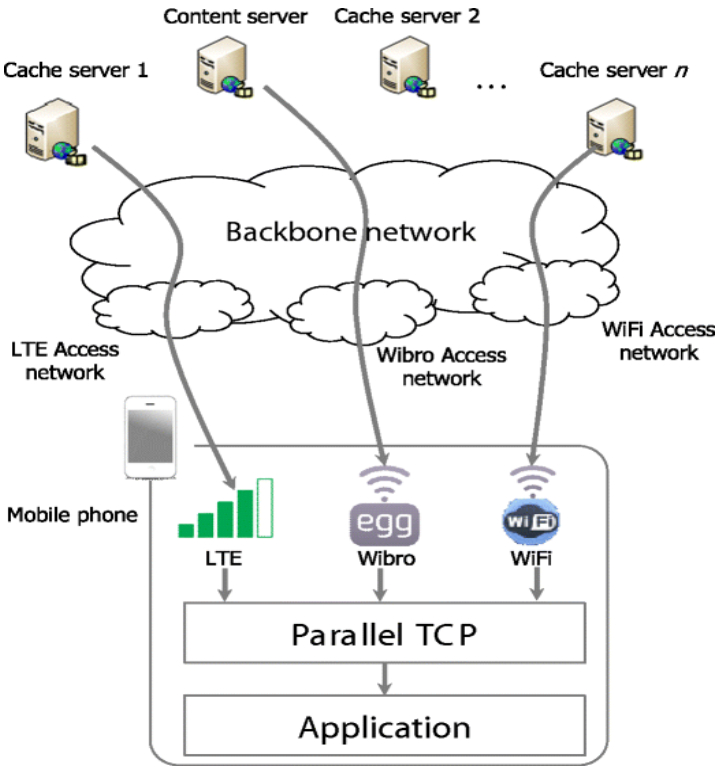

Рисунок 2.7 - Схема принципу роботи паралельного завантаження

За наявності кількох дзеркальних серверів, які можуть передавати ту саму інформацію, новий підхід до підвищення продуктивності полягає в паралельному підключенні до багатьох серверів. “Паралельний” у цьому контексті означає, що різні частини інформації запитуються з різних серверів

(використовуючи, наприклад, запит діапазону HTTP). Такий підхід може значно скоротити час очікування клієнта.

Суть паралельних завантажень полягає у тому щоб зменшити час на завантаження об’єктів шляхом завантаження їх з різних доменів, що економить час, на від міну від того якби елементи завантажувалися по черзі з одного домену.

Проблема в тому що браузери обмежують такі завантаження, на це вказує специфікація HTTP 1999 року. Для деяких браузерів цей параметри можна налаштувати. В інших випадках можна скористатися методом для обходу цього обмеження.

Алгоритми паралельного завантаження

Можна виділити три паралельні схеми завантаження:

1) Static-Equal: у цьому підході клієнт завантажує рівну частину потрібного файлу з кожного з доступних серверів. Цей підхід є “статичним”, оскільки рішення про те, яку частину файлу отримати з якого сервера, приймається до початку завантаження.

2) Static-Unequal Ідея Static-Unequal полягає в тому, щоб зробити кількість даних, запитуваних від кожного сервера, пропорційною пропускній здатності сервера (на відміну від схеми Static-Equal, де всі сервери обслуговують частини файлу однакового розміру). Перед початком завантаження клієнт оцінює пропускну здатність кожного сервера. Спосіб отримання цієї оцінки детально описано в Розділі IIID. Потім клієнт запитує від кожного сервера частину файлу, яка пропорційна оцінці пропускної здатності сервера. Назва схеми походить від того, що файл може бути поділений на нерівні частини і рішення приймається до початку завантаження. Подібна ідея, хоча і з критичною різницею в методі, що використовується для оцінки повсюдно, описується як TCP, заснований на історії. Цей же принцип також використовується комерційним програмним забезпеченням, таким як Speedbit’s Download Accelarator.

3) Динамічний: Інший підхід полягає в тому, щоб динамічно регулювати частку файлу, отриманого з кожного сервера під час процесу завантаження, щоб запитувати більше інформації від серверів, які зараз працюють краще. Основна схема працює наступним чином. Файл розділений на багато рівних частин, і клієнт запитує шматок у кожного з серверів. Коли сервер завершує роботу, клієнт просить ще один фрагмент і так далі.

Найбільш практичним рішенням буде налаштування декількох піддоменів, які будуть вказувати на один і той же сервер. Завдяки ньому браузери думають, що об’єкти відвантажуються з різних хостів і таким чином дозволяють завантажувати більше двох об’єктів з сервера.

Для застосування такого методу для початку потрібно налаштувати піддомени, з яких будуть завантажуватися потрібні елементи сторінки.

Першим кроком потрібно внести зміни у файл конфігурації віртуальних хостів “httpd-vhosts.conf” і додати наступний код:

```
<VirtualHost *:80>
DocumentRoot E:/Server/data/htdocs/Site2/
ServerName [Site2.example.com](<http://Site2.example.com/>)
<Directory E:/Server/data/htdocs/Site2/>
AllowOverride All
Order Allow,Deny
Allow from All
</Directory>
ErrorLog ${APACHE_LOG_DIR}/Site2-error.log
CustomLog ${APACHE_LOG_DIR}/Site2-access.log combined
RewriteEngine on
RewriteCond%{SERVER_NAME}=
[Site2.example.com](<http://Site2.example.com/>)
RewriteRule^https://%{SERVER_NAME}%{REQUEST_URI}
[END,NE,R=permanent]
</VirtualHost>
```

Після збереження змін потрібно перезавантажити сервер. Змінений файл конфігурації віртуальних хостів зображено на рисунку 2.7.

Після зміни файлу конфігурації потрібно налаштувати HTTPS. Цей крок потрібний для захисту субдомену з допомогою HTTPS.

Для цього можна використати Certbot.

Certbot - це повнофункціональний клієнт, що розширюється, для CA Let's Encrypt (або будь-якого іншого CA, який використовує протокол ACME), який може автоматизувати завдання отримання сертифікатів та налаштування веб-серверів для їх використання.

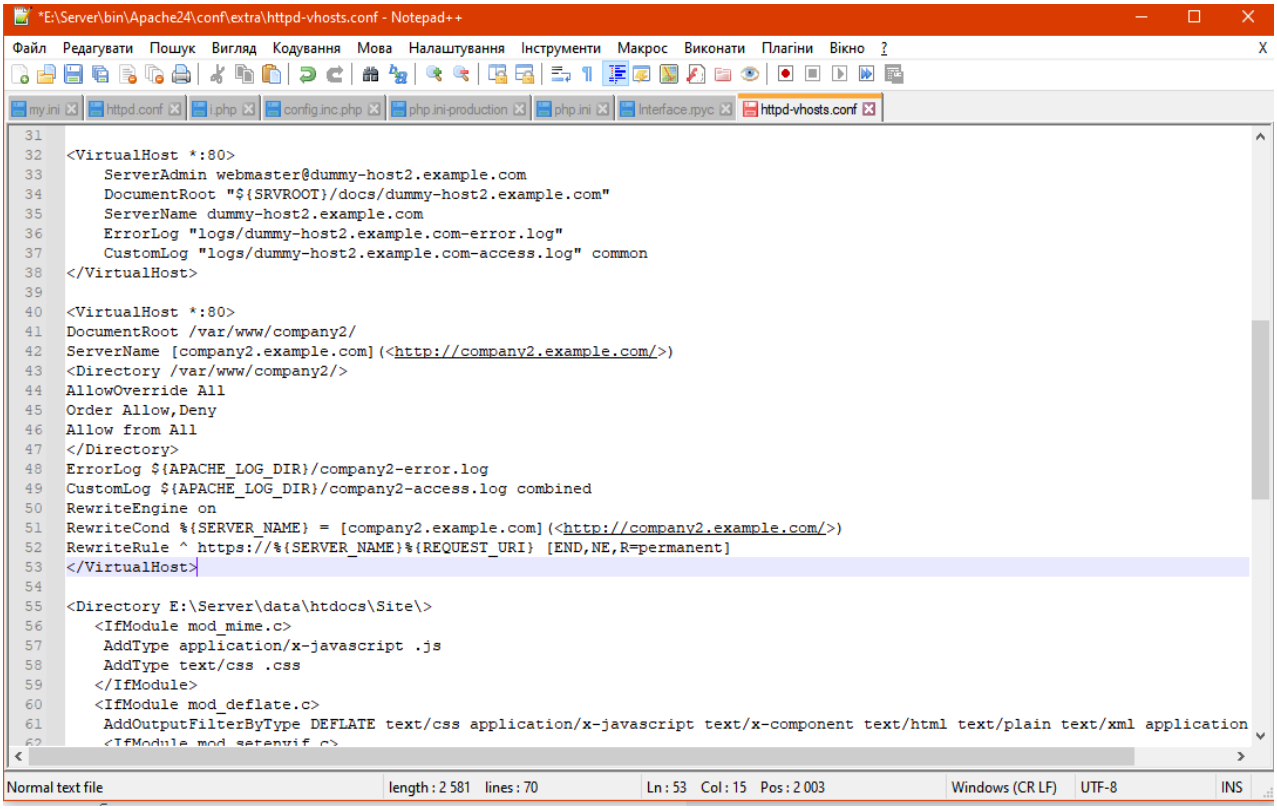

Рисунок 2.7 - Файл конфігурації віртуального хоста

Після завантаження та установки програми, потрібно її запустити та вказати потрібний субдомен і як підтвердження повинно появитися наступне повідомлення:

Congratulations! You have successfully enabled https://Site2.example.com

Оптимальна кількість хостів для мінімізації витрат на доставку об'єктів.

Пара досліджень, проведених незалежно Yahoo! і Gomez продемонстрували, як збільшення числа хостів здатне зменшити затримки при паралельних завантаженнях. А Steve Souders та Tenni Theurer з Yahoo! виклали звіт про результати тестів у своєму блозі Performance UI (Theurer та Souders 2007). Змінюючи кількість доступних хостів поряд з розміром об'єктів, інженери з Yahoo! встановили, що підключення двох хостів дає найшвидший відгук великих файлів. Theurer виклав графік, який містить більш детальну статистику за часом відгуку, числом хостів та розміром файлів.

Зараз середня веб-сторінка складається з більш ніж 50 об'єктів (для Рунета, за статистичними даними webo.in, до речі, ситуація дуже схожа, але кількість об'єктів коливається в межах 20–30), тому мінімізація витрат на доставку об'єктів є дуже критичною. клієнтської продуктивності. Також можна зменшити кількість об'єктів на сторінці, якщо використовувати техніку CSS sprites, і об'єднання CSS або JavaScript файлів (а може, навіть усі в одному) на сервері. Так як в даний момент у користувачів досить швидкий канал, то можна досягти зменшення часу завантаження до 40% (автори трохи занижують це число, тільки використання кількох хостів здатне збільшити швидкість завантаження до 4 разів, а скорочення на 75%). Можна використовувати 2 або 3 хости для обслуговування об'єктів з одного сервера, щоб обдурити браузери в їх обмеженнях на завантаження декількох об'єктів паралельно.

Автори не торкаються ще одного, дуже цікавого моменту в оптимізації часу завантаження шляхом збільшення числа паралельних потоків. Полягає він у вирівнюванні і збільшенні розміру об'єктів, що одночасно завантажуються, щоб максимально використовувати наявні з'єднання. Наприклад, якщо у вас є 40 картинок по 5Кб, то набагато вигідніше буде віддавати 10 картинок по 20Кб з двох хостів, ніж 20 (по 10Кб) з 4 хостів або 40 з 8. Загальні затримки в першому випадку будуть мінімальними через максимізацію ефективної швидкості завантаження даних клієнта.

Можна піти і далі і завантажувати, наприклад, 4 картинки по 50Кб у 4 потоки, досягаючи просто феноменального прискорення. Однак, тут бере участь психологічний фактор: користувачеві буде не комфортно, якщо він бачитиме сторінку взагалі без картинок весь той час, поки вантажиться 50Кб, і він може просто піти з сайту.

Цей підхід застосовується і до інших ресурсних (у тому числі, і HTML-) файлів, однак, варто пам'ятати про дуже жорсткі обмеження браузерів на завантаження CSS-і JavaScript-файлів (наприклад, при відображенні сторінки IE вантажить останні строго в один потік).

2.3 Принцип роботи та метод впровадження кешування часто використовуваних об’єктів

Кешування (або кеш) – це проміжний буфер, в якому зберігаються дані. Завдяки кешуванню сторінка сайту не відтворюється наново для кожного користувача. Кешування дозволяє здійснювати роботу з великою кількістю даних у максимально стислий термін і при обмежених ресурсах (серверних та користувальницьких).

Кешування економить час при завантаженні елементів сторінки. Проте, коли на сторінці оновлюється вміст, то в кеші він не з’являєтся доки користувач на неї не повернеться.

Найвідоміший кеш – це кеш веб-браузера, що зберігається на машині користувача. Коли веб-браузер, такий як IE, Firefox або Chrome, отримує ресурс, він не відображає його один раз і не викидає: він зберігає вміст на локальному жорсткому диску користувача.

Веб-сервер може використовувати ряд методів (розглянутих нижче), щоб інструктувати веб-браузер, коли дозволено використовувати цю локальну копію вмісту замість повторного завантаження вмісту з веб-сервера.

Веб-сервер надає різні типи даних, включаючи (якщо назвати лише деякі) HTML-код, зображення, таблиці стилів та бібліотеки Javascript. Частина цієї інформації, наприклад, результат PHP або іншого сценарію CGI, є дуже динамічною та непостійною, і може повертати різні результати кожного разу, коли до неї звертаються. Інші файли, такі як зображення та клієнтські сценарії, можуть змінюватися набагато рідше. Ці файли є головними кандидатами для кешування.

Контроль кешу веб-сервера використовує HTTP-заголовки, щоб регулювати, які ресурси витягуються з кешу та як часто. Налаштування ефективних заголовків контролю кешу для вмісту має вирішальне значення для продуктивності веб-програми, оскільки воно безпосередньо впливає на те, як браузери запитують файли з вашого веб-сервера.

Заголовок Expires повідомляє веб-браузеру, що термін дії заголовка певного типу закінчується в певний день і час. Заголовок Cache-control використовує комбінацію директив кешування та модифікаторів віку, щоб інструктувати веб-клієнта про те, чи можна кешувати чи ні певний вміст, і як довго. Заголовок керування кешом описано в розділі 14.9 RFC2616.

Веб-сервери також можуть використовувати заголовок ETag, який призначає унікальний ідентифікатор кожній версії файлу, зображення чи іншого компонента. Коли веб-сервер доставляє вміст, він ставить на нього значення ETag. Пізніше, якщо клієнт вважає, що термін дії вмісту закінчився, він робить HTTP-запит, включаючи заголовок If-None-Match, зі значенням цього заголовка, встановленим на останнє значення ETag для цього компонента. Якщо вміст змінився після того, як було видано значення ETag, сервер відповість новим вмістом. В іншому випадку, якщо вміст не змінився, сервер відповість зі статусом 304 HTTP і порожнім тілом відповіді HTTP. Теги ET перешкоджають веб-серверу повторно доставляти вміст, який ще свіжий, тим самим зберігаючи ресурси сервера, пропускну здатність і скорочуючи час завантаження сторінки.

Неймовірне поширення мультимедіа стало можливим завдяки збігу двох фактів: існування успішних стандартів, таких як MPEG, та розробки нових елементів архітектури комп'ютера, спеціально орієнтованих на мультимедійні додатки. Насправді, на відміну від загальноприйнятої тенденції до проектування високопродуктивних комп'ютерів загального призначення, ринок мультимедіа нав'язав виробникам обладнання також інтеграцію деяких новинок у стандартні ПК, щоб задовольнити попит на мультимедіа.

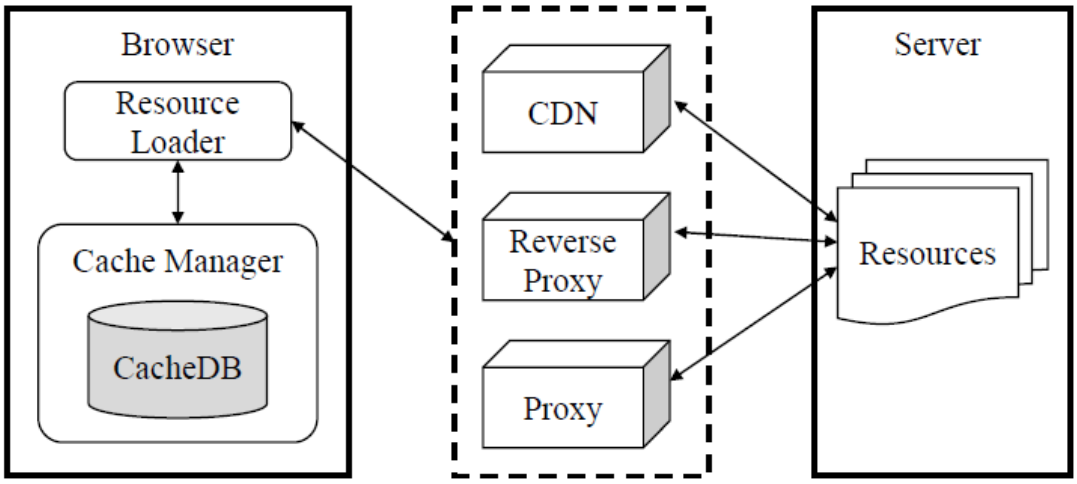

Рисунок 2.8 - Архітектура веб-кешування

Відповідь була у двох напрямках: по-перше, додавання мультимедійних процесорів або високопродуктивних DSP в обчислювальних системах, щоб обробляти зображення та звуки одночасно зі звичайною роботою ЦП; по-друге, інтеграція в обчислювальні блоки деяких покращень SIMD (таких як Pentium MMX, HP MAX2 або Ultrasparc VIS) для покращення паралельних обчислень на пікселях зображення. Далі запропоновано системи обробки середовищ на основі паралельності грубих зерен.

Загалом, продуктивність нових комп'ютерних архітектур для мультимедіа оцінюється з посиланням на найпоширеніші етапи обробки зображень, такі як стиснення та декодування MPEG. У недавньому опитуванні Курода та Нішітані порівняли стандартні високопродуктивні ПК

та робочі станції, DSP та мультимедійні процесори, використовуючи програми MPEG як еталон. Як підкреслювалося в, архітектура кешу, яка зіграла головну роль у вибуху потужності ПК для додатків загального призначення, частково недостатня, якщо не марна для багатьох мультимедійних завдань.

Найважливішою проблемою кешу при обробці зображень є те, що обробка зображень демонструє внутрішню 2D просторову локальність замість класичної одновимірної просторової локальності, типової для звичайних обчислень і векторної обробки.

Найбільш підходящим та ефективним видом кешування для сторінок з великою кількістю вмісту та зображень являється браузерне кешування. Він полягає в тому що браузеру віддається заголовок “expired” та заголовка “304 Not Modified”, що в свою чергу означає, що даний вміст втратив актуальність і на сервері є оновлена версія, або навпаки що файл не модифікований.

Також слід виділити кешування мобільних браузерів, адже вони бувають досить ресурсоємкими.

Як працює веб-кешування?

Кешування веб-сервера реалізується одним із двох способів. Веб-сервер може кешувати відносно статичний вміст у пам'яті, що значно скорочує час, який витрачається на отримання вмісту зі сховища. Деякі веб-сервери, такі як G-WAN, оптимізовані для статичного вмісту та автоматично кешують дані в пам'яті. Інші веб-сервери дозволяють кешувати пам'ять за допомогою файлів конфігурації. Apache, наприклад, підтримує модуль mod_cache, який забезпечує як кеш пам’яті для статичного вмісту (mod_mem_cache), так і кеш диска для кешування вмісту, зібраного з динамічних джерел або отриманого від зовнішнього постачальника (mod_disk_cache).

Кеш веб-сервера також може мати форму спеціального сервера, який називається проксі-сервером веб-кешу, який перехоплює клієнтські запити та обслуговує вміст, попередньо кешований із фактичного хост-сервера, на

якому знаходиться програма. Ці проксі-кеші зазвичай використовуються в різних місцях по всьому світу, і запити динамічно направляються на відповідний проксі-кеш-сервер залежно від місцезнаходження користувача. Така близькість користувача прискорює доставку вмісту користувачеві, зменшуючи затримку мережі. Це також зменшує кількість запитів, зроблених до цільового сервера.

Проксі-кеш працює так само, як і кеш веб-браузера: він використовує інформацію, включену в заголовки HTTP, наприклад Expires і Cache-control, щоб визначити, чи є даний компонент свіжим чи застарілим. Налаштування точних заголовків контролю кешу на вміст має вирішальне значення для успіху проксі-сервера веб-кешу.

На рисунку 2.9 показана спрощена процедура, щоб проілюструвати, як працює веб-кешування. Веб-сторінка складається з набору ресурсів, таких як HTML, CSS, JavaScript та зображення. Коли користувачі введуть URL-адресу та натиснуть кнопку “Перейти” або натиснуть гіперпосилання, браузер завантажить цільову веб-сторінку, отримавши відповідні ресурси з модуля Resource Loader.

Завантажувач ресурсів спочатку спробує завантажити ресурси з локального кешу через модуль Cache Manager. Відповідно до RFC 2616, HTTP/1.1 визначає модель закінчення терміну дії кешу та модель перевірки кешу. Модель закінчення терміну дії кешу дозволяє веб-серверу встановлювати час закінчення терміну дії для кожного ресурсу, вказуючи, як довго ресурс може кешуватися клієнтським пристроєм. Встановлення терміну дії не є обов’язковим. Якщо ресурс не має терміну дії, явно встановленого сервером, браузери можуть застосувати власні евристичні алгоритми, щоб визначити час закінчення терміну дії кешу ресурсу. Такий механізм називається евристичною експірацією.

Менеджер кешу перевіряє, чи можна знайти запитаний ресурс (ідентифікований за протоколом його URL-адреси, хостом, портом, шляхом і запитом) у базі даних кешу. Якщо ні, браузер налаштує з’єднання HTTP для

отримання ресурсу з віддаленого веб-сервера. Якщо ресурс знайдено і термін його дії не закінчився відповідно до моделі закінчення терміну дії кешу, він буде повернений безпосередньо з кешу.

Якщо термін дії ресурсу закінчився, модель перевірки кешу HTTP/1.1 вимагає, щоб браузер перевіряв із сервером, чи змінено ресурс чи ні (через час останньої зміни або ідентифікатор Etag). Якщо сервер вирішує, що відповідний ресурс не змінився, він відповідає повідомленням “304 Not Modified”, і ресурс буде витягнуто з кешу клієнта. В іншому випадку сервер надсилає оновлений ресурс назад у браузер.

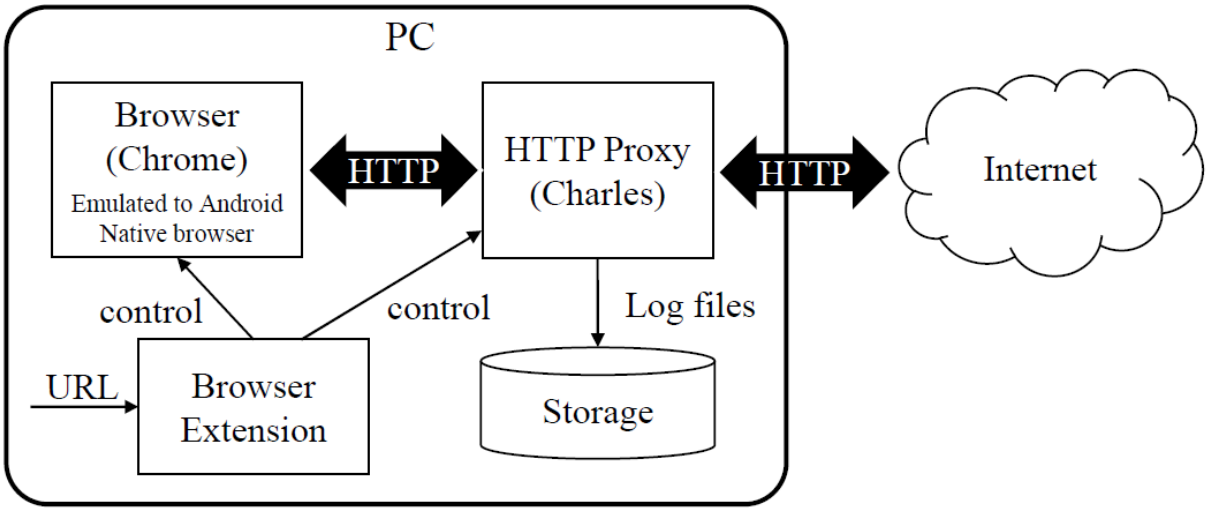

Рисунок 2.9 - Інструмент збору даних

Варто зазначити, що веб-сайт може бути набагато складнішим, ніж один веб-сервер. Великий веб-сайт (наприклад, Google) може мати багато внутрішніх веб-серверів. Також можуть існувати сервери мережі доставки вмісту (CDN), щоб забезпечити масштабованість і географічне поширення. У такому випадку частина запиту URL-адреси часто містить хеш-код, щоб вказати, який сервер CDN повинен обробити запит, і, таким чином, один і той самий ресурс може мати різні URL-адреси, які обслуговуються різними серверами CDN. Крім того, проксі-сервери та зворотні проксі-сервери часто діють посередині, щоб направляти запити на різні серверні сервери для

балансу робочого навантаження. Різні серверні сервери також можуть приєднувати різні URL-адреси до одного ресурсу. Як обговорювалося в Розділі 5.3, ця проблема “кілька URL-адрес одного ресурсу” спричиняє небажану продуктивність веб-кешу, і її слід дослідити.

В кінці дослідження, результати кеш тестування порівняються з рештою результатів, отриманих під час дослідження. Повні результати дослідження даного методу можна переглянути в розділі , де будуть підбиватися підсумки проведеної роботи.

2.4 Принцип роботи та метод впровадження HTTP стиснення

Передача пакетів даних через мережу ефективніша, коли перед передачею застосовується стиснення даних без втрат. Зберігання тих самих даних вимагає менше місця, якщо стиснення застосовується до запису даних на носій. Вищі швидкості передачі даних і швидко зростаючі вимоги до пам’яті викликають потребу в кращих рішеннях для стиснення. Сьогодні стратегії енергозбереження також відіграють важливу роль у розробці систем керування даними. Технологічні досягнення дозволяють продукції задовольняти ці потреби, що розвиваються та зростають.

Хоча сьогодні існує багато різних алгоритмів стиснення без втрат, більшість із них є варіаціями двох популярних схем: кодування Хаффмана та алгоритму Лемпеля-Зіва. Кодування Хаффмана працює шляхом присвоєння двійкового коду кожному із символів (символів) у вхідному потоці (файлі). Це досягається шляхом спочатку побудови двійкового дерева символів на основі частоти їх появи у файлі. Призначення двійкових кодів символам здійснюється таким чином, що найбільш часто зустрічаються символи отримують найкоротші двійкові коди, а символам, що зустрічаються рідше, — найдовші коди. Це, у свою чергу, створює менший стиснутий файл.

Алгоритм Лемпеля–Зіва, також відомий як LZ-77, використовує надлишкову природу даних для забезпечення стиснення. Алгоритм використовує те, що називають ковзаючим вікном, щоб відстежувати останні n байтів даних. Кожного разу, коли зустрічається фраза, яка існує в буфері ковзного вікна, вона замінюється вказівником на початкову позицію попередньої фрази у розсувному вікні разом із довжиною фрази.

Основною метрикою для алгоритмів стиснення даних є коефіцієнт стиснення, який відноситься до відношення розміру вихідних даних до розміру стиснутих даних. Наприклад, якби у нас був файл розміром 100 кілобайт і ми могли стиснути його до 20 кілобайт, ми б сказали, що коефіцієнт стиснення становить 5 до 1, або 80%. Вміст файлу, зокрема надмірність і впорядкованість даних, може сильно вплинути на ступінь стиснення.

Користувачі з низькою пропускною здатністю та/або з високою затримкою підключення до Інтернету, наприклад користувачі модемів комутованого зв’язку, найімовірніше відчують переваги стиснення даних. Наразі модеми реалізують стиснення самостійно, однак воно не є оптимальним і не настільки ефективно, як стиснення HTTP. Більшість модемів реалізують стиснення на апаратному рівні, хоча його можна вбудувати в програмне забезпечення, яке взаємодіє з модемом. Протокол V.44, який є поточним стандартом стиснення модему, використовує алгоритм Lempel-Ziv-Jeff-Heath (LZJH), варіант LZ-77, для виконання стиснення.

Алгоритм LZJH працює шляхом побудови словника, представленого деревом, що містить часто повторювані групи символів і рядків. Для кожного входження рядка, що з’являється в словнику, потрібно вивести лише індекс словника, який називається кодовим словом, а не кожен окремий символ у рядку. Коли починається передача даних, кодер (відправник) і декодер (одержувач) починають будувати ідентичні дерева словника. Таким чином, щоразу, коли декодер отримує кодове слово, надіслане кодером, він може

використовувати його як індекс у дереві словника, щоб перебудувати вихідний рядок.

Модемне стиснення працює лише над невеликими блоками даних фіксованої довжини, які називаються фреймами, а не над великим фрагментом файлу, як у випадку з високорівневими алгоритмами, такими як gzip. Алгоритм постійно контролює стисливість цих кадрів, щоб визначити, чи слід надсилати стиснену чи нестисну версію кадру. Модеми можуть швидко перемикатися між стиснутим і прозорим режимом за допомогою спеціального escape-коду.

Крім того, модеми пропонують лише стиснення «точка-точка». Лише коли дані передаються по аналоговій лінії, зазвичай між кінцевим користувачем та постачальником послуг Інтернету, дані стискаються. Таким чином, стиснення HTTP зазвичай вважається кращим, оскільки воно забезпечує наскрізне рішення між вихідним сервером і кінцевим користувачем.

Apache не надає вбудованого механізму для стиснення HTTP. Однак існує досить популярний і широко протестований модуль з відкритим кодом, який називається mod_gzip, який забезпечує стиснення HTTP для Apache. Увімкнути стиснення досить просто, оскільки mod_gzip можна або завантажити як зовнішній модуль Apache, або зібрати безпосередньо на веб-сервер Apache. Оскільки mod_gzip є стандартним модулем Apache, він працює на будь-якій платформі, яка підтримується сервером Apache. Подібно до IIS, mod_gzip використовує існуючі стандарти кодування вмісту, як описано в HTTP/1.1. На відміну від IIS, mod_gzip дозволяє активувати стиснення для кожного каталогу, таким чином надаючи адміністратору веб-сайту більший контроль над функціональністю стиснення для його/її сайтів. Mod_gzip, як і IIS, може стискати як статичні, так і динамічні документи. У випадку статичних документів mod_gzip може спочатку перевірити, чи існує попередньо стиснута версія файлу, і, якщо так, надіслати цю версію. Інакше mod_gzip стискає документ на льоту. Таким чином, mod_gzip відрізняється

від IIS, оскільки mod_gzip може стискати статичний документ під час першого доступу. Крім того, mod_gzip можна налаштувати для збереження стиснутих файлів у тимчасовий каталог. Однак, якщо такий каталог не вказано, статичний документ буде просто стискатися при кожному доступі. Передбачається, що Mod_gzip підтримує майже всі типи CGI, включаючи: Perl, PHP, ColdFusion, скомпільований код C тощо. І IIS, і mod_gzip дозволяють адміністратору вказати, який статичний і динамічний вміст слід, а який не повинен стискатися на основі mime тип або розширення імені файлу ресурсу. Крім того, на відміну від IIS, mod_gzip за замовчуванням надсилає вміст клієнтам незалежно від версії HTTP, яку вони підтримують. Це можна легко змінити, щоб лише запити від HTTP/1.1-сумісних клієнтів могли отримувати стиснений вміст.

Для даного методу можна використати GZIP Compression. Він забезпечує стиснення без втрат, іншими словами, вихідні дані можна повністю відновити під час розпакування. GZIP заснований на алгоритмі DEFLATE, який використовує комбінацію алгоритму LZ77 та алгоритму Хаффмана.

Апаратне стиснення GZIP розвантажує стиснення даних без втрат і звільняє центральний процесор (ЦП) системи. GZIP — це стандарт формату файлів, де основний алгоритм стиснення називається Deflate. Стиснуті блоки, що стискаються, обгортаються верхнім і нижнім колонтитулами, щоб стати файлами GZIP. Deflate — це алгоритм стиснення з відкритим кодом і широко доступний як програмний інструмент. Ліцензія на використання програмного забезпечення не потрібна. Коли процесор загального призначення виконує стиснення GZIP, зазвичай продуктивність стиснення зменшується, щоб максимізувати швидкість передачі даних, або вона працює повільно. Ефективним рішенням цього є перевантаження завдання стиснення на апаратний співпроцесор GZIP. Співпроцесор приймає нестиснені вхідні дані, стискає їх і виводить дані у стиснутому вигляді. Апаратний співпроцесор виконує багато завдань паралельно, усуваючи багатопрохідну та ітераційну

природу, яка є типовою для ЦП загального призначення, що виконує алгоритм Deflate. Це дозволяє співпроцесору працювати на набагато вищій швидкості передачі даних.

Стиснення в пристроях зберігання даних або в мережевих каналах має бути без втрат. Дані, як правило, захищаються за допомогою генерації та перевірки CRC, щоб гарантувати, що після декомпресії вихідні дані на сто відсотків відновлені та точні, а також без пошкоджень або додаткових артефактів. Після стиснення пакета даних або файлу, створення та приєднання CRC його можна зашифрувати з метою безпеки. Якщо до даних потрібно застосувати шифрування, важливо, щоб воно було застосовано після стиснення, оскільки зашифровані дані дуже випадкові і не стискаються.

Ефективність стиснення вимірюється двома показниками: зменшенням розміру та пропускною здатністю даних. Зменшення розміру зазвичай відображається як відношення нестиснутого вихідного розміру, поділеного на стиснений розмір. Пропускна спроможність даних вимірюється в байтах на секунду, як вимірюється на нестисненій стороні пристрою співпроцесора.

Складність даних не впливає на пропускну здатність даних. Легкі для стиснення файли даних, які стискаються з високим коефіцієнтом, проходять через співпроцесор з такою ж високою швидкістю передачі даних, як і дуже складні дані, що забезпечує нижчий ступінь стиснення. У таблиці 1 наведено порівняння продуктивності ко-процесора AHA GZIP у порівнянні з іншими стандартними алгоритмами і стандартними наборами файлів, визначеними двома відомими університетами. Це Корпус Калгарі та Кентерберійський корпус. Набір файлів HTML із колекції динамічного вмісту Інтернету.

Таблиця 2.1 – Коефіцієнт стиснення

Стандартна колекція файлів	Ко-процесор GZIP	LZS	ALDC
Корпус калгари	2.73	2.24	2.10

Продовження таблиці 2.1

Кентерберійський корпус	3.6	2.75	2.68
Набір файлів HTML	4.4	Н/Д	2.65

Під час пошуку рішення для стиснення GZIP слід враховувати енергоспоживання. Порівняння мікросхеми стиснення AHA GZIP з чотирьохядерним процесором, призначеним для роботи програмного забезпечення для стиснення GZIP, показує, що пристрій AHA потребує приблизно в 8 разів менше енергії. У центрі обробки даних потенціал для економії енергії великий. Перевантаження стиснення на співпроцесор замість використання ЦП економить значну кількість енергії. Передача менших стиснутих пакетів призводить до додаткової економії енергії. Зберігання стиснених даних на носіях також економить електроенергію.

Алгоритм LZ77 замінює повторні входження даних на "посилання". Тобто, якщо в наявних даних якийсь ланцюжок елементів зустрічається більше одного разу, то наступні її входження замінюються “посиланнями” на її перший екземпляр.

Кодування Хаффмана є методом кодування зі змінною довжиною, яка призначає більш короткі коди до частіших “символів”. Проблема зі змінною довжиною коду, як правило, в тому, що нам потрібен спосіб дізнатися, коли код закінчився і почався новий, щоб розшифрувати його.

Кодування Хаффмана вирішує цю проблему, створивши код префікса, де жодне кодове слово не є префіксом іншого.

Застосування та рішення

Багато компаній із мережевих пристроїв і пристроїв зберігання даних віддали б перевагу впроваджувати плату стиснення GZIP, що підключається, замість того, щоб розвивати свою систему на основі пристрою ASIC зі стисненням GZIP. AHA пропонує плату PCI Express з драйверами та

підтримкою розробки драйверів, що спрощує процес проектування, дозволяючи компаніям використовувати переваги цієї нової технології з меншими зусиллями щодо дизайну. Ця плата GZIP (AHA363-PCIe) забезпечує пропускну здатність даних 5 гігабіт в секунду за допомогою двох співпроцесорних пристроїв AHA. Плата здатна працювати в дуплексному режимі, стискаючи та розпаковуючи декілька потоків або файлів.

На рисунку 2.10 показані плати прискорювача стиснення, встановлені в різних мережевих пристроях.

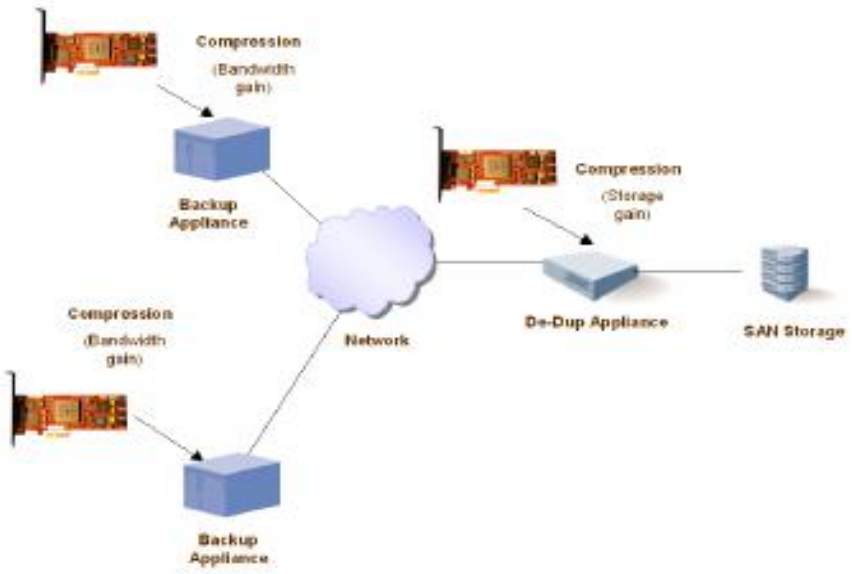

Рисунок 2.10 – Плати прискорювача стиснення в мережевих приладах

Зменшення часу отримання динамічного вмісту під час доступу до веб-сайту є однією з проблем, яку мережеві пристрої можуть покращити, реалізувавши прискорювач стиснення. Це також називають прискоренням програми і важливо під час отримання динамічного вмісту, наприклад фінансових даних.

Реалізація стиснення GZIP на основі апаратного забезпечення є ідеальним рішенням, оскільки воно перевантажує функцію стиснення на співпроцесор і залишає ЦП вільним для виконання інших завдань. Інша причина вибору GZIP (Deflate) як алгоритм полягає в тому, що більшість веб-

браузерів мають вбудований декомпресор і можуть автоматично розпаковувати ці пакети під час їх отримання.

Це добре працює, оскільки процес декомпресії для GZIP не настільки інтенсивний, як процес стиснення.

Деякі інсталяції використовують апаратне стиснення для прискорення мережевого трафіку. Інші використовують апаратне стиснення, щоб зменшити необхідний простір на носіях даних.

Пристрої зберігання даних, такі як віртуальні стрічкові бібліотеки та пристрої для прискорення глобальної мережі (WAN), можуть скористатися платою стиснення GZIP на основі PCI Express, стискаючи вміст даних перед записом на носій.

Для цієї програми найважливішою вимогою є високий коефіцієнт стиснення, за яким слід висока швидкість передачі даних.

У минулому LZS і ALDC були популярними рішеннями для стиснення, але обидва алгоритми недостатньо ефективні в порівнянні з останніми пропозиціями продуктів AHA, які реалізують GZIP (або Deflate).

Високе зростання корпоративних баз даних, відновлення резервних копій та електронної пошти, а також вимоги до довгострокового збереження даних викликають потребу в постійно зростаючій ємності зберігання даних. Дата-центри будуються поблизу гідроелектростанцій через їх величезний попит на електроенергію.

Стиснення даних без втрат зараз є необхідністю при розробці мереж або систем зберігання даних. GZIP (Deflate) є найпопулярнішим і найефективнішим стандартом стиснення даних без втрат.

Передача стиснення GZIP на співпроцесор значно покращує ефективність передачі даних, вищі коефіцієнти стиснення та економію енергії.

Для тестування стиснення, GZIP буде застосовано на сервері Apache.

Для цього потрібно включити модуль GZIP в файлі конфігурації Apache (рисунок 2.11).

А саме для початку потрібно додати рядок команди: “LoadModule deflate_module modules/mod_deflate.so”. Цією командою роблоковується модуль GZIP.

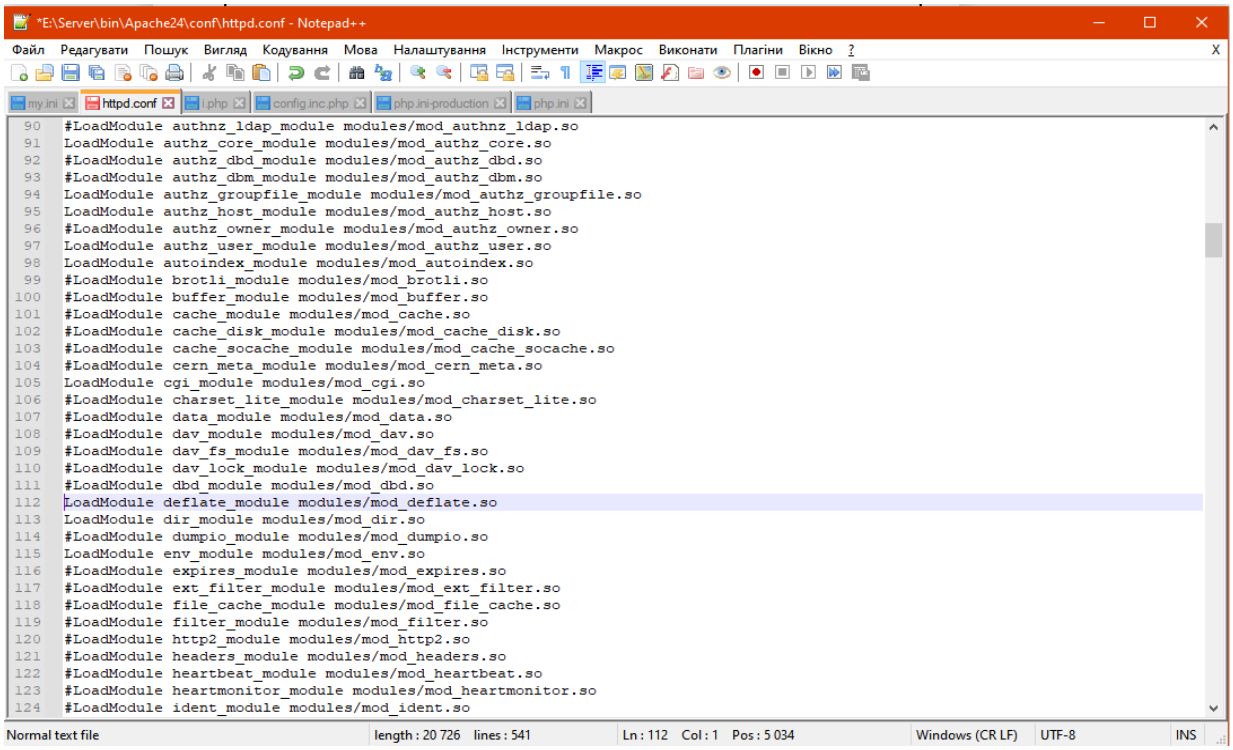

Рисунок 2.11 - Файл конфігурації Apache

Наступним кроком потрібно відкрити файл конфігурації з допомогою команди “nano /E:/Server/bin/Apache24/conf” і добавити наступні команди:

AddOutputFilterByType DEFLATE text/plain

AddOutputFilterByType DEFLATE text/html

AddOutputFilterByType DEFLATE text/xml

AddOutputFilterByType DEFLATE text/css

AddOutputFilterByType DEFLATE application/xml

AddOutputFilterByType DEFLATE application/xhtml+xml

AddOutputFilterByType DEFLATE application/rss+xml

AddOutputFilterByType DEFLATE application/javascript

AddOutputFilterByType DEFLATE application/x-javascript.

Після збереження змін потрібно перезавантажити сервер в Windows PowerShell командою “e:\Server\bin\Apache24\bin\httpd.exe -k restart” для того щоб зміни вступили в силу.

Потрібно ввести наступне налаштування в віртуальний хост Apache для увімкнення стиснення GZIP для веб-сайту. Також можна додати цей код у файл .htaccess веб-сайту в корені сайту:

```
<Directory /var/www/html/>
  <IfModule mod_mime.c>
AddType application/x-javascript .js
AddType text/css .css
  </IfModule>
  <IfModule mod_deflate.c>
AddOutputFilterByType DEFLATE text/css application/x-javascript text/x-component text/html text/plain text/xml application/javascript
<IfModule mod_setenvif.c>
        BrowserMatch ^Mozilla/4 gzip-only-text/html
        BrowserMatch ^Mozilla/4.0[678] no-gzip
        BrowserMatch bMSIE !no-gzip !gzip-only-text/html
</IfModule>
   </IfModule>
   Header append Vary User-Agent env=!dont-vary
</Directory>
```

Після збереження змін, включення модуля стиснення можна вважати завершеним.

Наступним кроком є тестування сторінки з анімованими графічними елементами.

Зі збільшенням потоку користувачів на сервер, такі результати мають все більше і більше значення.

Отже, можна підвести підсумок тестування. Стиснення є важливим способом підвищення продуктивності веб-сайту. Для деяких документів

зменшення розміру до 70% знижує потребу в пропускній здатності. З роками алгоритми також стали більш ефективними, а нові підтримуються клієнтами та серверами.

На практиці веб-розробникам не потрібно впроваджувати механізми стиснення, вони вже реалізовані як у браузерах, так і на серверах, але вони повинні бути впевнені, що сервер налаштований належним чином.

Такі налаштування було проведено перед тестом і описано. Якщо провести налаштування правильно, то сервер буде надсилати запит до браузера з методом перевірки можливості підтримки браузером методу стиснення, а після підтвердження запиту, сервер може надсилати браузеру стиснені документи.

РОЗДІЛ 3. ПІДВЕДЕННЯ ПІДСУМКІВ ДОСЛІДЖЕННЯ ТА ОГЛЯД ПОВНИХ РЕЗУЛЬТАТІВ

Після проведеного дослідження та ознайомлення з отриманими результатами можна стверджувати що зазначені методи оптимізації мають значний вплив на продуктивність роботи сервера та час його відклику при впровадженні цих методів у проекти.

Для повного ознайомлення нижче приведено повні результати дослідження кожного методу у вигляді таблиць.

Метод стиснення досить важливий при великих обсягах програмного коду, завдяки чому при отриманні стиснених фалів браузер виконує швидше програмний код і швидше завантажує потрібні елементи.

3.1 Висновки проведеного дослідження

3.1.1 Метод оптимізації паралельних завантажень

Даний метод найкраще підходить для веб-серверів, які повинні підвантажувати велику кількість елементів, зокрема зображення.

Розподілення елементів на піддомени повинно дати відчутний приріст продуктивності веб-сервера та зменшити час завантаження окремих елементів.

На наведених нижче графіках можна побачити приріст кількості сторінок які може завантажити веб-сервер в секунду та приріст швидкості завантаження сторінок.

На рисунку 3.1 можна побачити що кількість сторінок які завантажуються в секунду тримається на приблизно на одному рівні.

Кількість сторінок які завантажуються в секунду ваірюється від 20 до 19, що є значно більшим за результат контрольного тесту, який показав що сервер завантажує 7 сторінок в секунду.

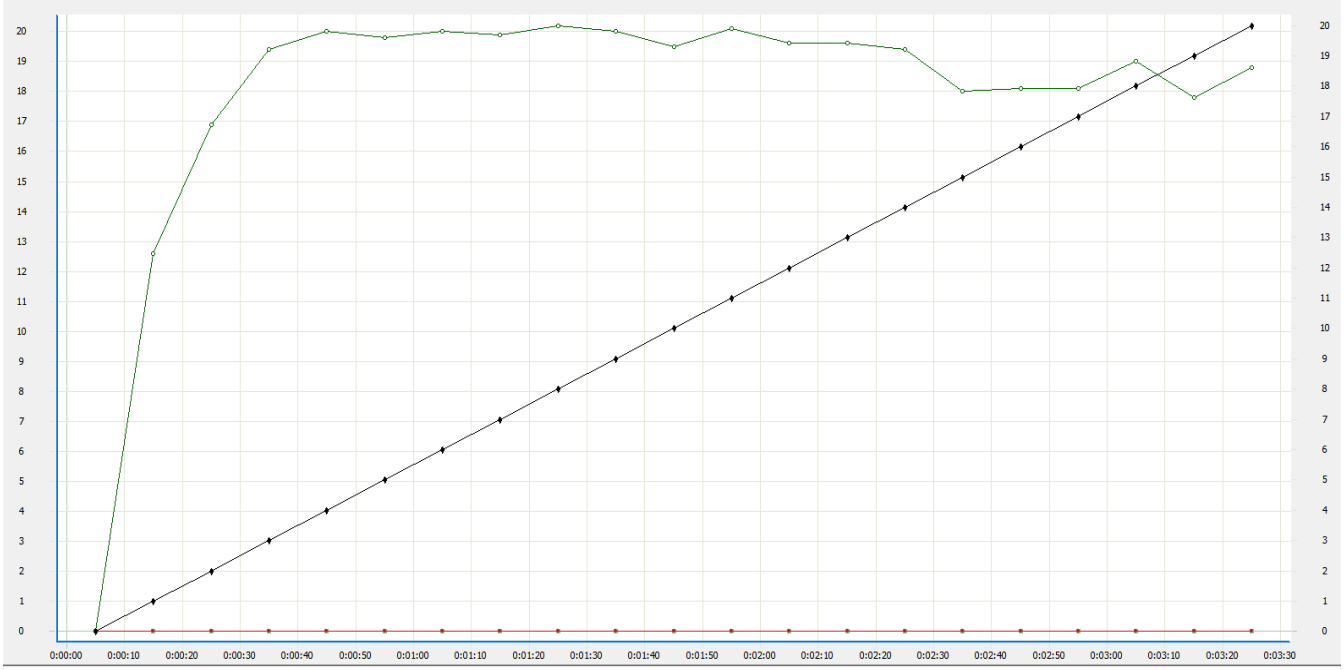

Рисунок 3.1 – Графік завантаження сторінок в секунду методом паралельного завантаження

Такий приріст зумовлений ефективністю застосування методу оптимізації паралельних заавнтажень, який зменшив навантаження на сторінку.

Швидкість завантаження сторінки з статичними графічними елементами (Рисунок 3.2) варіюється від 0.12 до 0.20 секунд, що також є швидшим ніж у контрольного тесту.

Тестування сторінки з анімованими графічними елементами також показало приріст продуктивності сервера, порівняно з контрольним тестом.

Тест показує, що кількість завантажених сторінок в секунду сягає 121, а швидкість завантаження сторінки сягає від 0.19 до 0.2 секунд.

В таблиці 3.1 представлено дані по кількості завантажених сторінок в секунду.

Таблиця 3.1 – Кількість завантажених сторінок за секунду.

Pages per second							
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	Total
Profile1	6,3	18,1	19,9	20,1	19,7	19,5	17,9
Total	6,3	18,1	19,9	20,1	19,7	19,5	17,9

Порівняно з контрольним тестом швидкість відклику отримала певний приріст, що грає велику роль при великій кількості запитів на сервер.

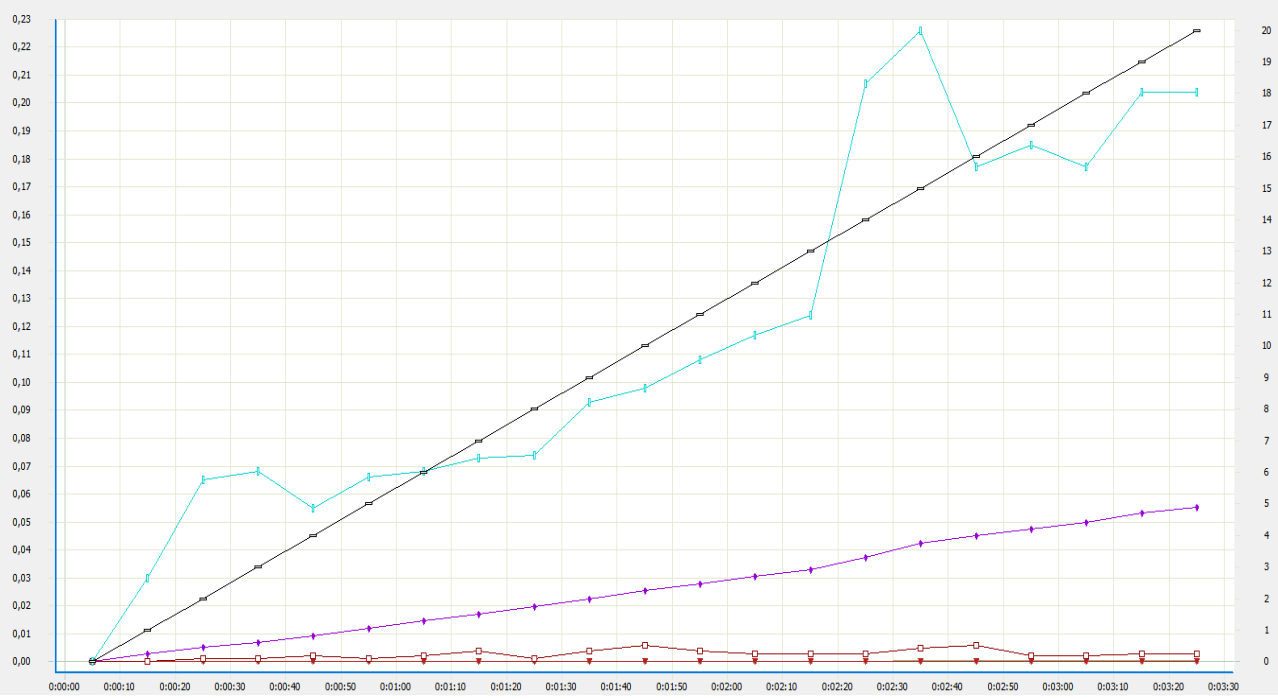

Рисунок 3.2 – Графік часу завантаження сторінки з графічними елементами методом паралельного завантаження

На рисунку 3.3 зображено графік з часом відклику сторінки з анімованими графічними елементами на запити клієнта.

Згідно графіка, швидкість завантаження сторінки варіюється від 0.19 до 0.2 секунд, що значно швидше від контрольного тесту, який показав результат в 0.35 секунд. Можна ввпевнено стверджувати, що паралельне

завантаження позитивно впливає на час завнтаження не тільки статичних, а й анімованих графічних елементів сторінки.

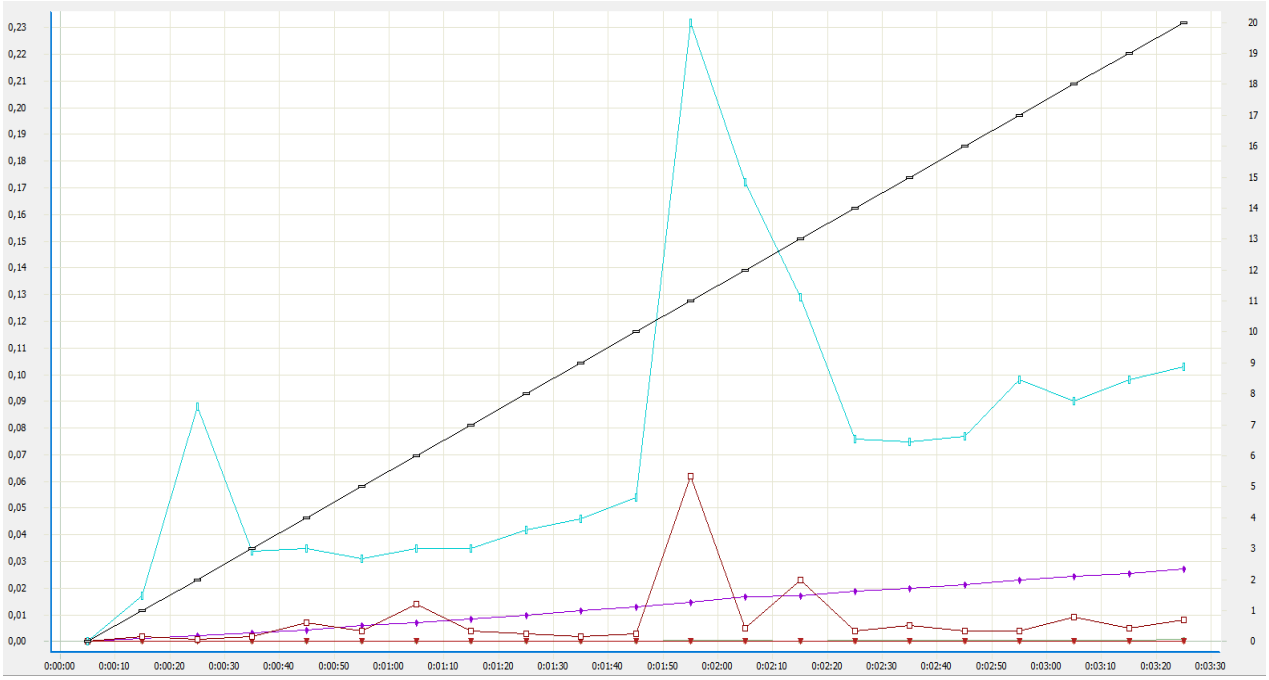

Рисунок 3.3 – Графік часу завантаження сторінки з анімованими графічними елементами методом паралельного завантаження

Для більш наглядного порівняння можна переглянути діаграму-порівняння результатів контрольного тесту та методу паралельного завантаження, зображеного нижче.

На ньому можна побачити, що в порівнянні з контрольним тестом, тест паралельного завантаження показав значний приріст відклику сервера.

За допомогою діаграми можна візуально побачити різницю роботи веб-сервера з використанням методу оптимізації паралельних завантажень та без нього.

Паралельні завантаження зменшують час відклику сервера та збільшують його продуктивність більше як на половину, що являється дуже хорошим показником, враховуючи те що цей метод використовується досить давно.

Також можна порівняти збільшення продуктивності роботи веб-сервера згідно таблиці 3.2.

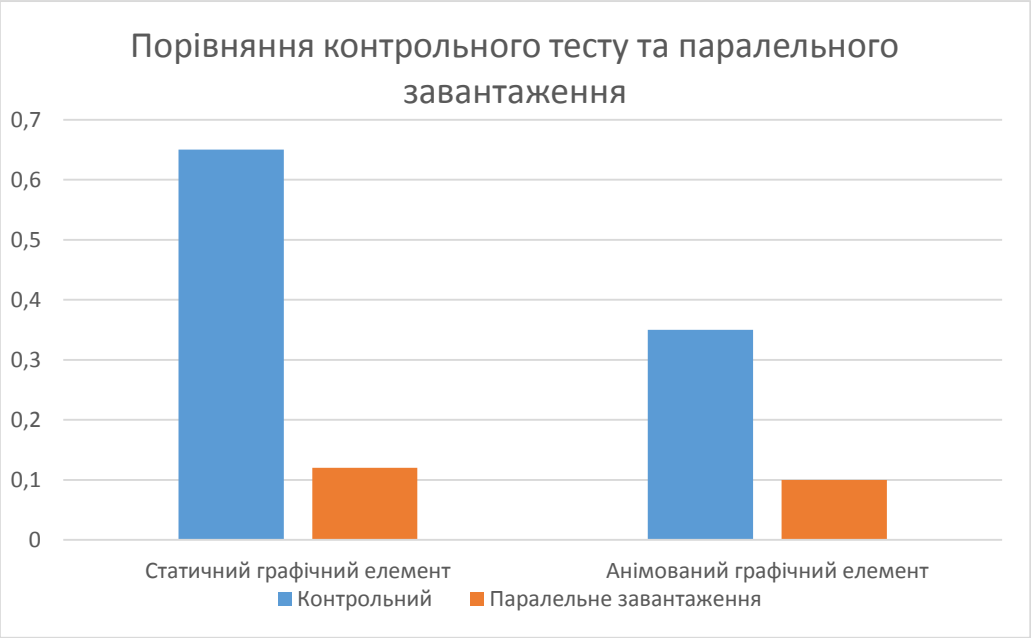

Рисунок 3.4 – Даграма Порівняння контрольного тесту та паралельного завантаження

В ній показано час завантаження сторінки з анімованими та статичними графічними елементами, а також кількість завантажених сторінок в секунду.

Таблиця 3.2 – Порівняння результатів методу оптимізації паралельних завантажень та контрольного тестування

Тип тестування	Сторінки в сек. Анімовані елементи	Сторінки в сек. Статичні елементи	Швидкість завантаження. Анімовані елементи	Швидкість завантаження. Статичні елементи
Контрольне тестування	40	7	0,35	0,7
Тестування методу оптимізації паралельних завантажень	121	19-20	0,19 - 0,2	0,12 - 0,2

Отже можна стверджувати що даний метод оптимізації не втрачає своєї актуальності та здатен оптимізувати роботу веб-сервера, навіть якщо може знадобитися завнтаженні досить важких об’єктів. Приріст продуктивності роботи сервера більше половини, а саме кількість сторінок з анімованими графічними елементами які завантажуються в секунду збільшилась на 67% в порівнянні з контрольним тестуванням. Кількість сторінок з статичними графічними елементами, які завантажуються в секунду збільшилась на 185%, в порівнянні з контрольним тестуванням. Також збільшилась швидкість завантаження сторінки з анімованими та статичними графічними елементами на 43% (анімовані графічні елементи) та 71% (статичні графічні елементи).

3.1.2 Метод кешування часто використовуваних об’єктів

Даний метод зберігає елементи сторінки веб-сайту в кеш браузера, тим самим зменшуючи час завантаження при повторному відвідуванні сайту.

Недоліком є те що дані не оновляються самі і для того щоб воно оновилися в кеші потрібно знову відвідати сторінку сайту.

Найкращим та найефективнішим варіантом кешування для сайтів з великою кількістю графічних елементів, особливо зображень, виявилось браузерне кешування.

Після проведення тестів можна побачити наступні графіки, де можна помітити приріст швидкості відклику сервера.

На зображених нижче рисунках можна побачити що графіки показують значний приріст швидкості завантаження сторінок.

Проте, у такого методу є мінус. При накопиченні великої кількості збережених даних у кеші браузера цей метод не тільки зменшує свою ефективність, а й спроможний погіршувати продуктивність роботи веб-сторінки.

Дані з швидкості завантаження сторінки з анімованими графічними елементами представлено в рисунку 3.5.

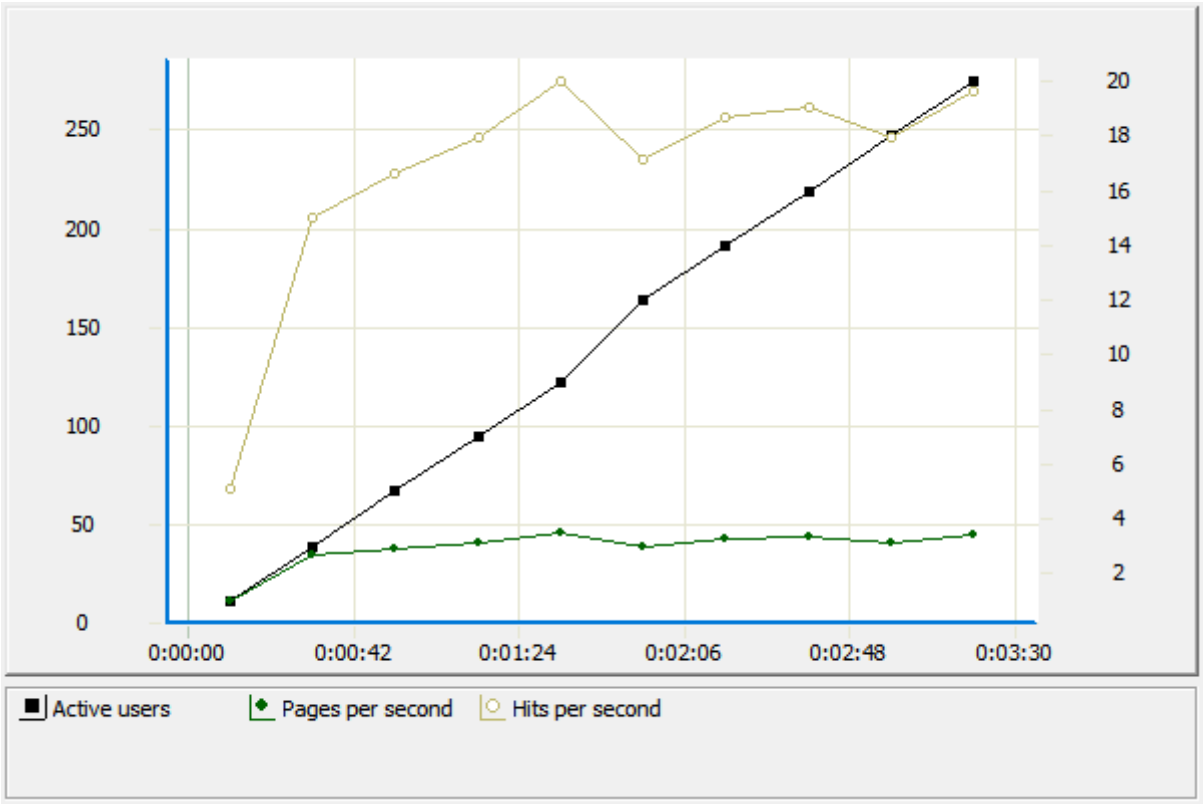

Рисунок 3.5 – Графік завантаження сторінок в секунду методом кешування

На графіку можна побачити, що сервер завантажує білизько 50 сторінок в секунду. Такий приріст зумовлений тим що браузер зберіг елементи сторінки, які мають високу роздільну здатність, через що довго підвантажуються, і при повторних запусках не отримував заголовків від веб-сервера про зміну елементу або його оновлення і завантажував його швидше.

В результаті застосування методу кешування швидкість завантаження сторінки з анімованими елементами виросла і варіюється від 0,017 до 0,02 секунд, в порівнянні з контрольним тестуванням без впроваджених методів з результатом 0,35 секунд.

Кешування стосується не тільки до зображень формату PNG, JPEG та інш., але й до анімованих графічних зображень формату GIF.

Тому цей метод є досить ефективним при застосуванні.

Графік швидкості завантаження сторінки з анімованими графічними елементами можна переглянути на рисунку 3.6.

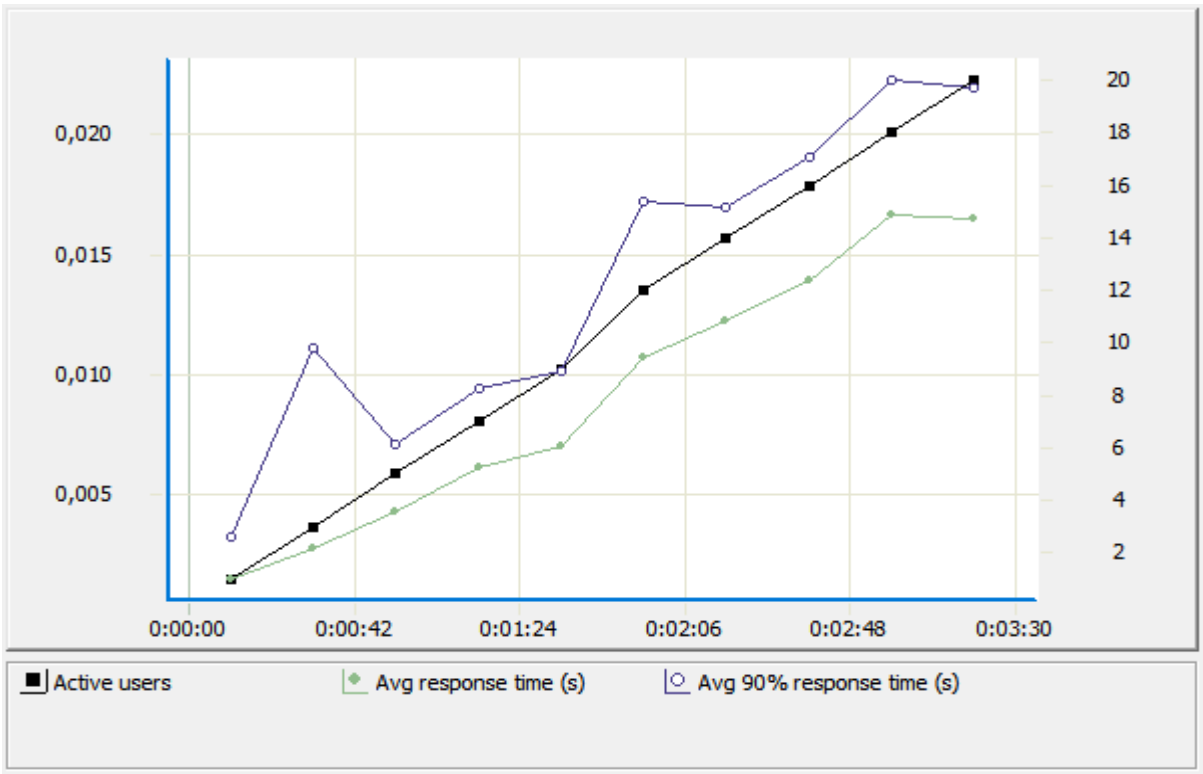

Рисунок 3.6 – Графік швидкості завантаження сторінки з анімованими графічними елементами

Для більш точного ознайомлення з результатами тестування можна ознайомитися з таблицею 3.3, в якій представлені результати контрольного тестування та методу кешування для наглядного порівняння.

Таблиця 3.3 – Порівняння результатів методу кешування та контрольного тестування

Тип тестування	Сторінки в сек. Анімовані елементи	Сторінки в сек. Статичні елементи	Швидкість завантаження. Анімовані елементи	Швидкість завантаження. Статичні елементи
Контрольне тестування	40	7	0,35	0,7
Метод кешування	50	40	0,02	0,3

Для візуального порівняння можна зобразити на діаграмі різницю між швидкістю завантаження контрольного тестування та методу кешування.

Можна помітити, що найбільший ефект кешування дало на швидкість завантаження сторінки з анімованими графічними елементами.

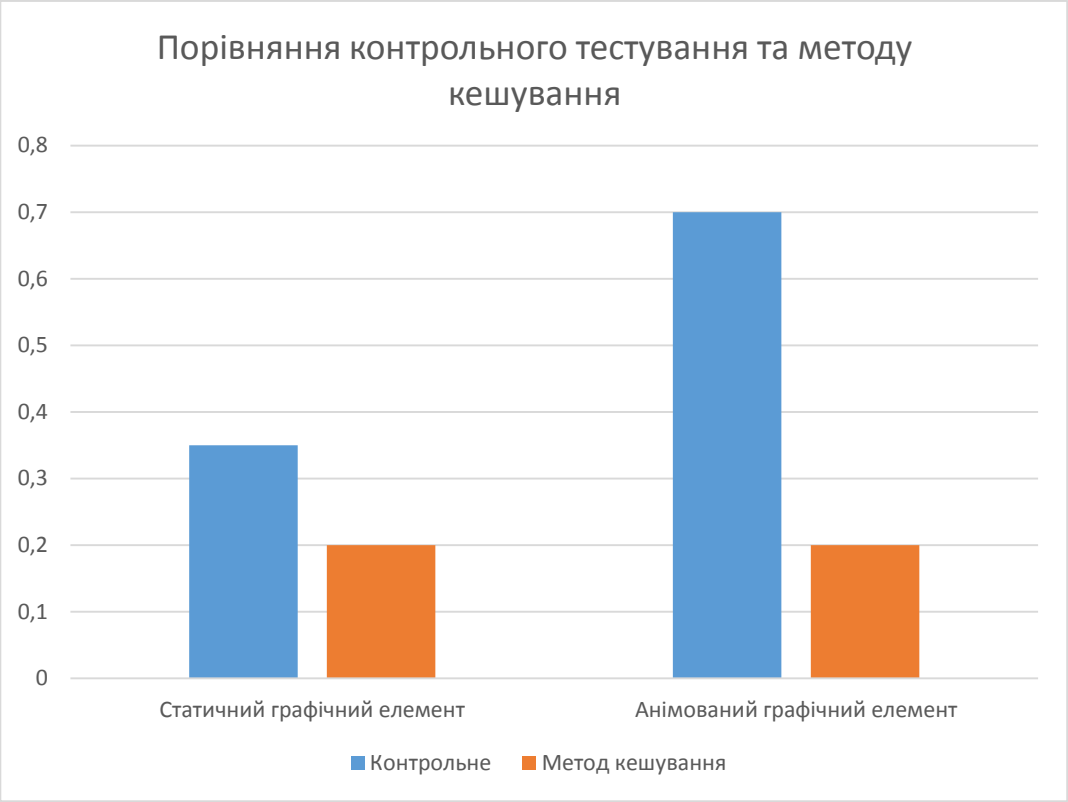

Рисунок 3.6 – Графік порівняння контрольного тестування та методу кешування

Такі результати являються гарним показником ефективності даного методу оптимізації.

Виходячи з вищеперерахованих результатів можна стверджувати що метод кешування досить добре підходить для веб-сайтів з великою кількістю графічних елементів, зокрема зображень, адже після його застосування швидкість завантаження та кількість сторінок, які сервер може відкрити за секунду зростає.

Метод кешування збільшив кількість сторінок з анімованими графічними елементами, які завантажуються в секунду на 25%, а сторінок з

статичними графічними елементами на - 471%. Швидкість завантаження сторінок з анімованими графічними елементами зросла на 94%, а сторінки з статичними графічними елементами - на 57%.

Також варто відмітити, що у такого методу є мінус, який здатен уповільнити роботу браузера.

Незважаючи на певний недолік, цей метод є одним із основних і найрозповсюдженіших, тому застосовується у кожному браузері для забезпечення швидкого завантаження сторінок веб-сайтів і зберігає свою актуальністьі на даний час.

3.1.3 Метод HTTP стиснення

Метод стиснення дуже корисний для проектів в яких є багато програмного коду. Велика кількість такого коду може сповільнити роботу веб-сервера, який буде приймати файли цього коду, тому для зменшення навантаження на сервер можна ці файли стиснути і в такому стані відправляти серверу, який уже буде їх обробляти.

Зокрема варто виділити GZIP стиснення, яке і використовувалося для тестування у даній роботі. Такий тип дає змогу стиснення без втрат.

Для застосування даного методу потрібно було внести зміни в файл конфігурації Apache, після чого перезапустити сервер з внесеними налаштуваннями.

Якщо браузер підтримує стиснення, тоді сервер надсилає йому стиснуті дані, а браузер їх розпаковує при отримані, що значно зменшує час очікування при великих об’ємах програмного коду.

Згідно проведених тестів можна побачити, що кількість сторінок з статичними графічними елементами, які завантажуються в секунду збільшилась до 20, що переважує результати при контрольному тестуванні. Результати для порівняння розміщені у кінці підрозділу.

На рисунку 3.7 зображено графік кількості сторінок з статичними графічними елементами, які завантажуються в секунду.

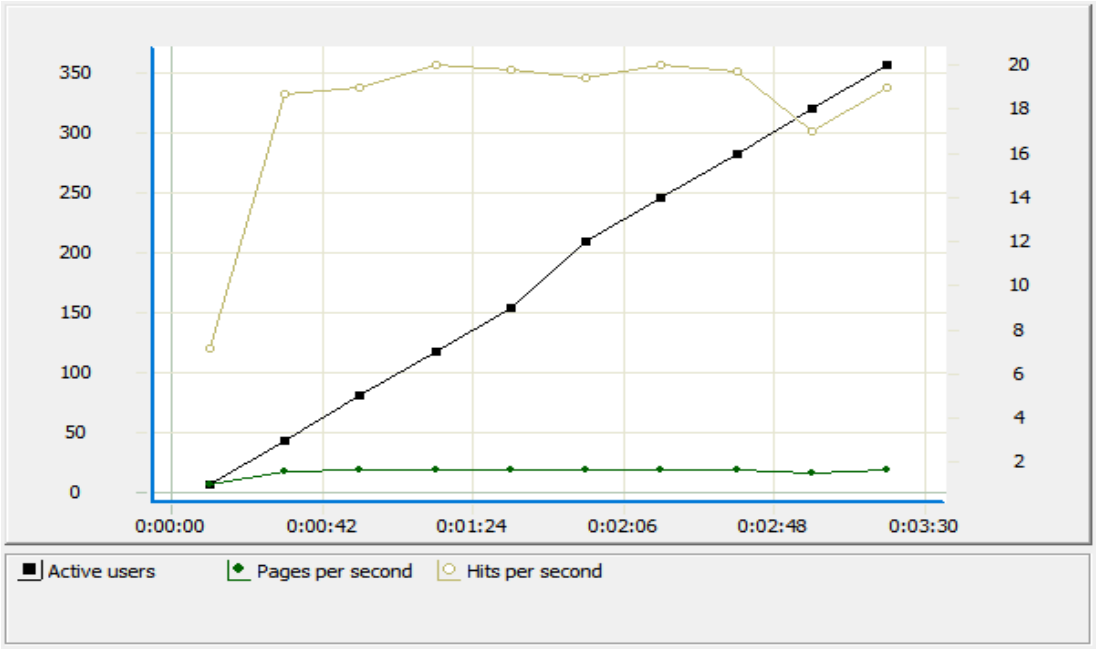

Рисунок 3.7 – Кількість сторінок з статичними графічними елементами, які завантажуються в секунду

А кількість сторінок з анімованими графічними елементами – сягає 128. Такі дані значно впливають при великих кількостях користувачів.

В таблиці 3.4 можна ознайомитися з точнішими даними, отриманими при тестуванні кількості сторінок з статичними графічними елементами, які завантажуються в секунду.

Таблиця 3.4 – Кількість сторінок з статичними графічними елементами, які завантажуються в секунду

Pages						
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:02:48-0:03:09
Profile1	126	350	356	376	372	318
Total	126	350	356	376	372	318

Можна побачити що кількість завантажених сторінок зростає.

Завдяки стисненню файлів, які надсилаються на сервер, час відклику стає меншим, що позитивно впливає на загальну продуктивність веб-сервера та зменшує час очікування користувача на запит.

На рисунку 3.8 зображено графік завантаження кількості сторінок з анімованими графічними елементами, які завантажуються в секунду.

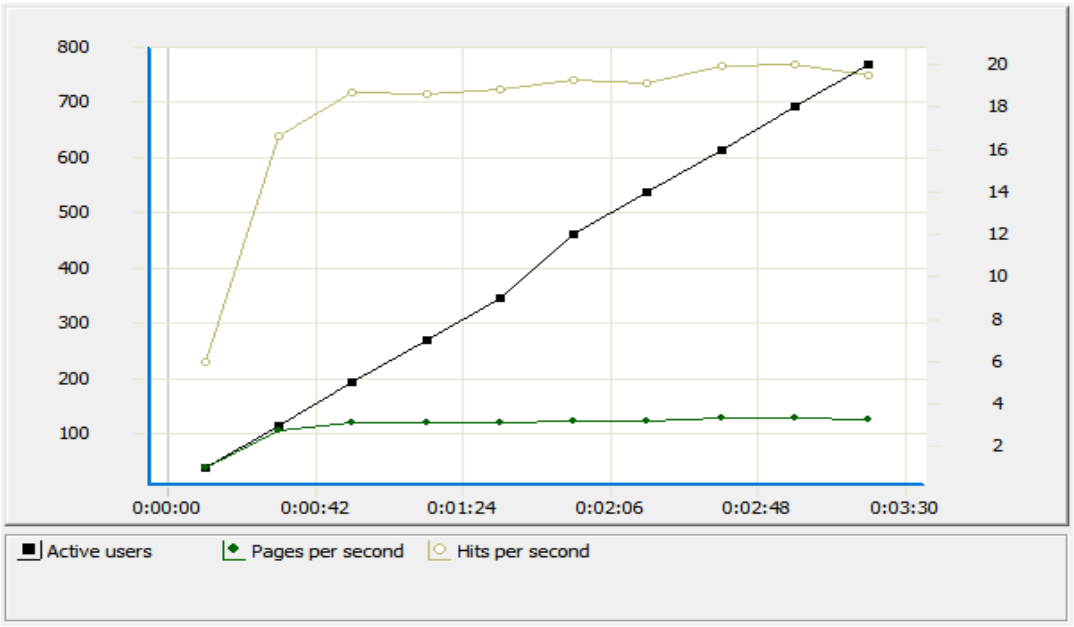

Рисунок 3.8 – Кількість сторінок з анімованими графічними елементами, які завантажуються в секунду

Подальші результати відображають вплив стиснення на швидкість завантаження сторінок веб-сторінки.

Виходячи з результатів, можна стверджувати що даний метод позитивно впливає на час завантаження.

При введені методу стиснення сторінка з статичними графічними елементами зменшила час завантаження до 0,02 – 0,04 секунд, що в порівнянні з контрольним тестуванням швидше на 71%.

На рисунку 3.9 зображено графік часу завантаження сторінок з статичними графічними елементами методом стиснення.

Можна побачити що час відгуку “не скаче” на приведенному графіку і залишається приблизно на одному місці.

Приблизно такі ж результати відображені у наступному тестуванні.

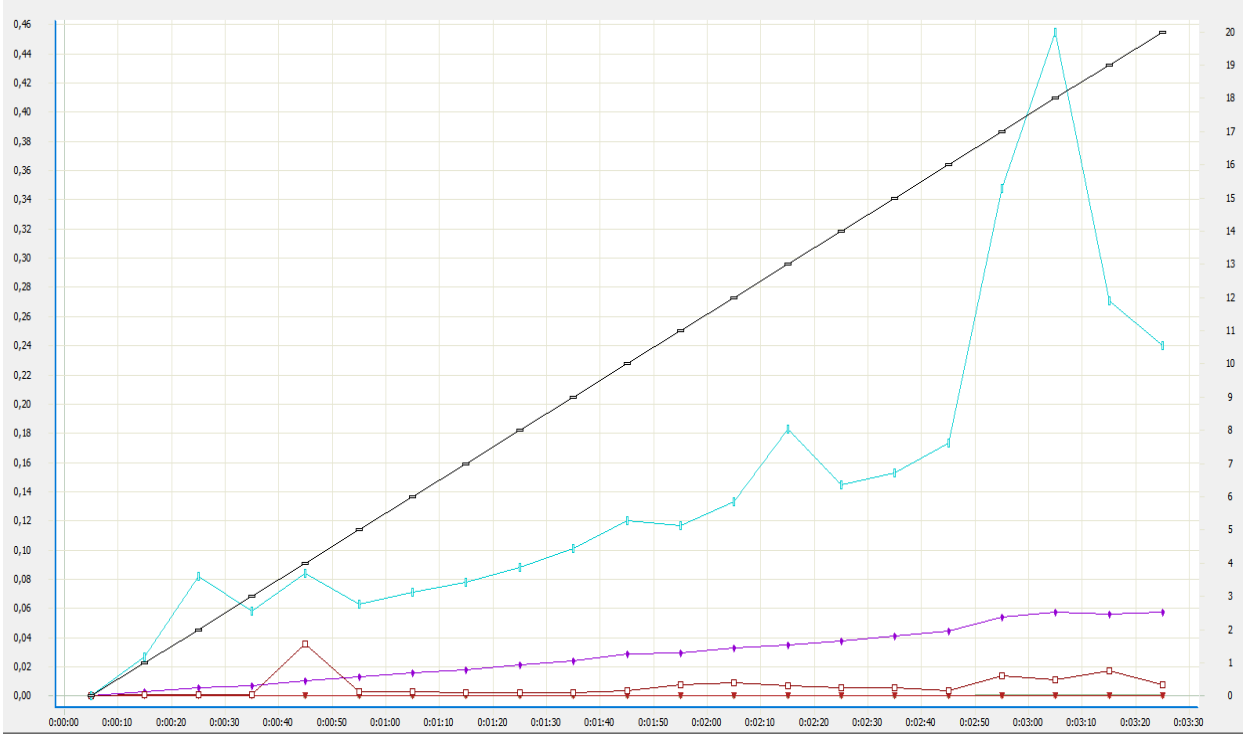

Рисунок 3.9 – Графік часу завантаження сторінки з статичними графічними елементами методом стиснення

Для детальної інформації можна переглянути таблицю 3.4.

Таблиця 3.4 – Час відгуку сторінки з статичними графічними елементами методом стиснення

Response time, sec								
Name	Time	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27
Profile 1	Min	0	0	0	0	0	0	0
	Avg	0,000008	0,00004	0,0003	0,0002	0,0002	0,0003	0,0003
	Avg90	0,0001	0,0003	0,002	0,0008	0,0008	0,001	0,001
	Max	0,001	0,001	0,04	0,003	0,002	0,009	0,007

Час завантаження сторінки з анімованими графічними елементами також зменшився до 0,009 - 0,01 секунди.

На рисунку 3.10. можна краще побачити як змінився час завантаження сторінки з використанням даного методу при зростанні кількості користувачів, які одночасно знаходяться на сторінці.

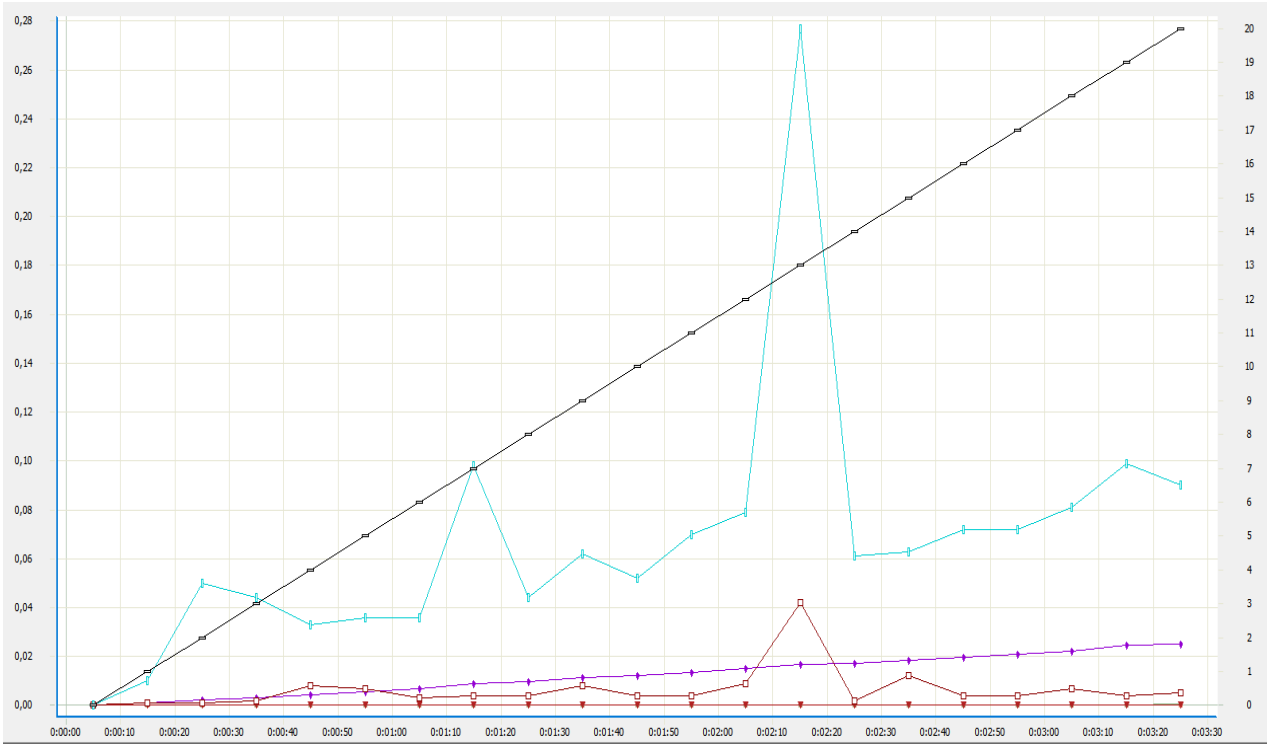

Рисунок 3.10 – Графік часу завантаження сторінки з анімованими графічними елементами методом стиснення

Цей метод можна успішно поєднувати з методом паралельних завантажень, що дасть високий приріст продуктивності веб-серверу.

Адже тоді сервер буде надсилати браузеру стиснені файли що пришвидчить час його роботи і при цьому завдяки паралельному завантаженню навантаження на сервер стане меншим, що збільшить його продуктивність.

Багато розробників та компаній користуються цим методом у зв’язці з іншими, що дає ще більший приріст продуктивності сервера і зменшує час очікування користувачів на обробку їхнього запиту.

В таблиці 3.5 можна порівняти результати проведеного тестування і порівняти їх з результатами контрольного тестування

Таблиця 3.5 – Порівняння результатів методу HTTP стиснення та контрольного тестування

Тип тестування	Сторінки в сек. Анімовані елементи	Сторінки в сек. Статичні елементи	Швидкість завантаження. Анімовані елементи	Швидкість завантаження. Статичні елементи
Контрольне тестування	40	7	0,35	0,7
Метод HTTP стиснення	128	20	0,01	0,04

Згідно таблиці можна побачити що ефект від застосування методу стиснення досить помітний.

Для порівняння можна переглянути діаграму на рисунку 3.11.

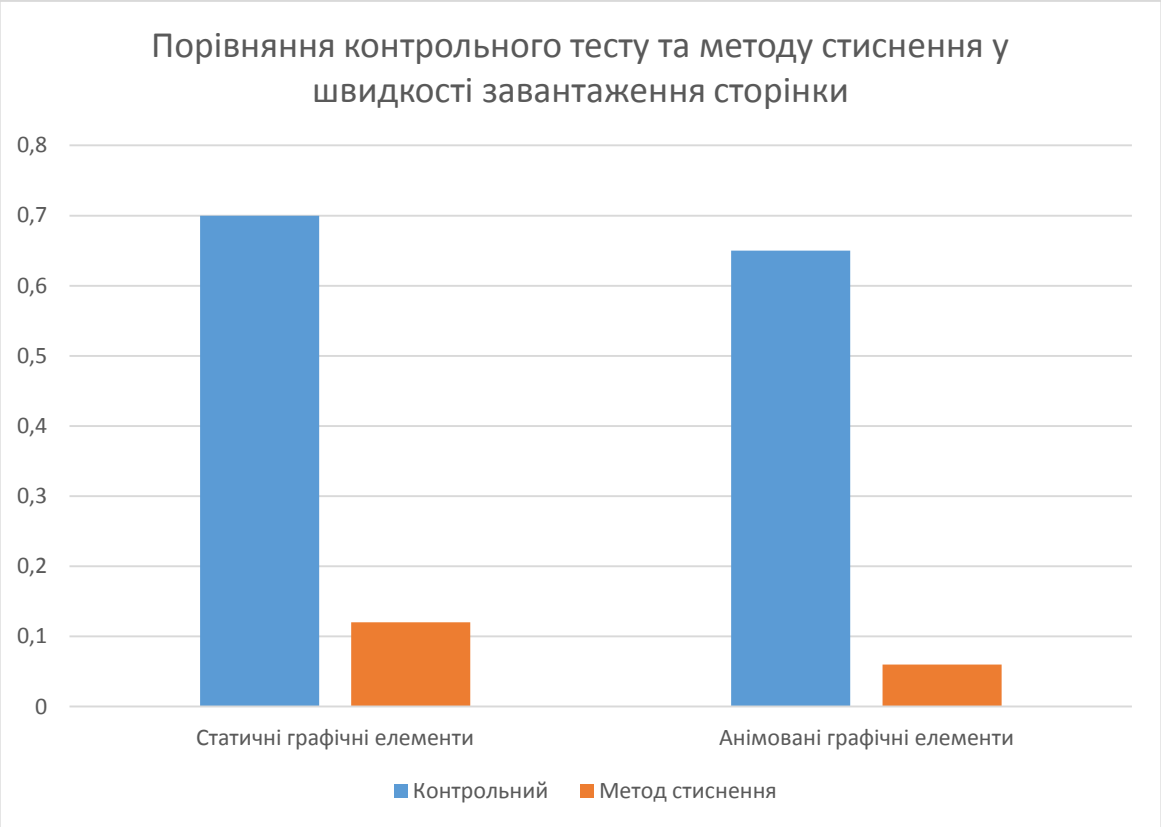

Рисунок 3.11 – Діаграма порівняння контрольного тесту та методу стиснення у швидкості завантаження сторінки

Комбінування такого методу з іншими методами дасть ще більш відчутнійший результат.

На діаграмі можна побачити, що метод стиснення є досить дієвим і його використання значно зменшить навантаження на сервер.

Завдяки методу стиснення можна побачити що сервер почав працювати краще та швидше відповідати на запити користувачів.

Отже після завершення тестування методу HTTP стиснення можна впевнено стверджувати, що такий метод має місце у використанні для оптимізації веб-серверів з метою зменшення нагрузки на них, використовуючи стиснення файлів, які сервер передає браузеру.

Результати показують, що даний метод оптимізації не втрачає своєї актуальності і може використовуватися для оптимізації веб-серверів і на далі.

3.2 Підведення підсумків тестування

Після проведення тестування було отримано результати кожного методу оптимізації та зроблено висновки.

Після цього можна сказати, що дані методи не втрачають своєї актуальності і можуть активно використовуватися в проектах і на далі задля забезпечення оптимізації сайтів..

Вони можуть активно використовуватися розробниками та допоможуть оптимізувати роботу веб-сайтів

Завдяки методу оптимізації паралельного завантаження графічні елементи почали завантажуватися з піддоменів, що зменшило навантаження на веб-сервер і дало приріст продуктивності сервера, що зменшило час на завантаження сторінок веб-сайту.

Для тестування даного методу, його було підключено до Apache черз файл конфігурації віртуального сервера для здобуття бажаного результату у дослідженні.

Основні результати даного методу оптимізації зображено в таблицях 3.6 – 3.15 з описом та поясненнями.

Таблиця 3.6 – Загальні результати проведеного дослідження

Summary								
Profile	Sessions performed	Sessions with errors	Pages performed	Pages with errors	Hits performed	Hits with errors	Total KBytes sent	Total KBytes received
Profile 1	3 549	0	3 549	0	67 431	0	20 000	143 879 645
Total	3 549	0	3 549	0	67 431	0	20 000	143 879 645

В таблиці відображено загальні результати проведеного дослідження. З більш точними результатами можна ознайомитися в подальших таблицях. До кожної таблиці передбачене загальне пояснення.

Таблиця 3.7 – Кількість активних користувачів

Number of active users										
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30
Profile1	1	3	5	7	9	12	14	16	18	20
Total	1	3	5	7	9	12	14	16	18	20

В даній таблиці зображено активних користувачів у певні відрізки часу, в які проходило тестування, по мірі їх збільшення до максимуму. В кінці тестування число користувачів, які одночасно знаходилися на сервері дорівнювало – 20.

Таблиця 3.8 – Кількість відкритих сесій

Sessions							
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	Total

Продовження таблиці 3.8

Profile1	126	350	356	376	372	548	3 549
Total	126	350	356	376	372	548	3 549

В даній таблиці наведено кількість відкритих, віртуальними користувачами, сесій у певні відрізки часу, в які проходило тестування. В загальному кількість сесій досягла – 3594.

Таблиця 3.9 – Кількість запущених сторінок

Pages											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	126	350	356	376	372	548	376	371	318	356	3549
Total	126	350	356	376	372	548	376	371	318	356	3549

В даній таблиці зображено кількість сторінок, які були запущені на певних відрізках часу, під час тестування. Можна побачити що загальна кількість запущених сторінок дорівнює – 3594 сторінки.

Таблиця 3.10 показує дані по кількості відкритих сесій за секунду та середню кількість цих сесій.

Таблиця 3.10 – Кількість відкритих сесій за секунду

Sessions per second											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9

Продовження таблиці 3.10

Total	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9

Кількість відкритих сесій на протязі усього тесту тримається приблизно на одному рівні, а середній результат сягає – 16,9 сесій в секунду.

Таблиця 3.11 – Кількість сторінок в секунду

Pages per second											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9
Total	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9

На даній таблиці можна побачити кількість сторінок, які завантажуються за секунду, у певні відрізки часу. Кількість залишається приблизно на одному рівні, досить стабільна. Середня кількість сягає – 16,9 сторінок в секунду.

Таблиця 3.12 – Кількість сторінок з помилками

Pages with errors											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	0	0	0	0	0	0	0	0	0	0	0
Total	0	0	0	0	0	0	0	0	0	0	0

В даній таблиці зображена кількість сторінок, які не змогля завантажитися черех певні причини та конфлікти. Як можна побачити –

таких сторінок немає, що говорить про успішне застосування та функціонування данного методу оптимізації.

Таблиця 3.13 – Кількість надісланих даних, КБайт

KBytes sent											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	710	1 972	2 006	2 119	2 096	3 088	2 119	2 091	1 792	2 006	20 000
Total	710	1 972	2 006	2 119	2 096	3 088	2 119	2 091	1 792	2 006	20 000

Згідно наведеної таблиці, загальна кількість надісланих даних дорівнює – 20000 Кбайт. В середньому це 2000 Кбайт в певні проміжки часу, в який виконувалося тестування.

У наступній таблиці можна переглянути скільки даних було отримано натомість.

Таблиця 3.14 – Кількість отриманих даних, КБайт

KBytes received										
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	Total
Profile1	5 108 153	14 189 314	14 432 559	15 243 377	15 081 214	22 216 412	15 243 377	15 040 673	12 892 005	143 879 645
Total	5 108 153	14 189 314	14 432 559	15 243 377	15 081 214	22 216 412	15 243 377	15 040 673	12 892 005	143 879 645

В наведеній таблиці можна побачити, що загальна кількість отриманих даних дорівнює – 143879645 Кбайт.

В кожен проміжок часу тестування було отримано від 14000 до 15000 Кбайт даних.

Ця статистика важлива для контролю даних.

Таблиця 3.14 – Час відгуку сторінки

Response time, sec											
Name	Time	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30
Profile1	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0,000008	0,00004	0,0003	0,0002	0,0002	0,0003	0,0003	0,0003	0,0008	0,0007
	Avg90	0,0001	0,0003	0,002	0,0008	0,0008	0,001	0,001	0,001	0,003	0,003
	Max	0,001	0,001	0,04	0,003	0,002	0,009	0,007	0,006	0,01	0,02

В таблиці 3.14 наведено дані часу відгуку сторінки на запити віртуальних користувачів, а саме мінімальний, середній та максимальний час відклику.

Таблиця 3.15 – Загальний відсоток помилок

Total errors %											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	0	0	0	0	0	0	0	0	0	0	0
Total	0	0	0	0	0	0	0	0	0	0	0

Дана таблиця показує кількість помилок, які з’являлися під час процесу тестування.

Як можна побачити, помилок не виникало, отже метод працює справно і імплементувався без проблем і конфліктів.

Отже метод паралельних завантажень має вплив не тільки на кількість завантажених сторінок, але й на час їх завантажень. Це говорить про те що він і в теперішній час досить актуальний для використання.

Метод кешування зберігав елементи сторінки в кеші браузера, що допомогло швидше завантажувати сторінки веб-сайту. У такого метода є мінус, при якому він починає уповільнювати роботу браузера, але якщо час від часу очищувати вміст кешу то проблем не виникає. Також незручність полягає в тому що при оновлені даних на сторінці вони не заносяться в кеш і щоб вони оновилися потрібно знову зайти на сторінку де оновилися дані, тоді браузер отримає заголовок від сервера про те, що дані оновилися чи видалилися і відбувається заміна старих даних в кеші.

Такий метод є найпоширенішим і застосовується у всіх браузерах, доступних на даний момент.

Основні результати даного методу оптимізації зображено в таблицях 3.15 – 3.25.

Таблиця 3.15 – Загальні результати проведеного дослідження

Summary								
Profile	Sessions performed	Sessions with errors	Pages performed	Pages with errors	Hits performed	Hits with errors	Total KBytes sent	Total KBytes received
Profile 1	8 036	0	8 036	0	48 216	0	13 489	45 464 196
Total	8 036	0	8 036	0	48 216	0	13 489	45 464 196

В таблиці відображено загальні результати проведеного дослідження. З більш точними результатами можна ознайомитися в подальших таблицях. До

кожної таблиці передбачене загальне пояснення.

Таблиця 3.16 – Кількість активних користувачів

Number of active users										
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30
Profile1	1	3	5	7	9	12	14	16	18	20
Total	1	3	5	7	9	12	14	16	18	20

В даній таблиці зображено активних користувачів у певні відрізки часу, в які проходило тестування, по мірі їх збільшення до максимуму. В кінці тестування число користувачів, які одночасно знаходилися на сервері дорівнювало – 20.

Таблиця 3.17 – Кількість відкритих сесій

Sessions											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	228	687	761	821	916	1 175	856	872	822	898	8 036
Total	228	687	761	821	916	1 175	856	872	822	898	8 036

В даній таблиці наведено кількість відкритих, віртуальними користувачами, сесій у певні відрізки часу, в які проходило тестування.

В загальному кількість сесій досягла – 8036.

Таблиця 3.18 – Кількість запущених сторінок

Pages							
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	Total

Продовження таблиці 3.18

Profile1	228	687	761	821	916	1 175	8 036
Total	228	687	761	821	916	1 175	8 036

В даній таблиці зображено кількість сторінок, які були запущені на певних відрізках часу, під час тестування. Можна побачити що загальна кількість запущених сторінок дорівнює – 8036 сторінок.

Таблиця 3.19 показує дані по кількості відкритих сесій за секунду та середню кількість цих сесій.

Таблиця 3.19 – Кількість відкритих сесій за секунду

Sessions per second											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	11,4	34,4	38,0	41,0	45,8	39,2	42,8	43,6	41,1	44,9	38,3
Total	11,4	34,4	38,0	41,0	45,8	39,2	42,8	43,6	41,1	44,9	38,3

Кількість відкритих сесій на протязі усього тесту тримається приблизно на одному рівні, а середній результат сягає – 38,3 сесій в секунду.

Таблиця 3.20 – Кількість сторінок в секунду

Pages per second								
Profile	0:00:00 - 0:00:21	0:00:21 - 0:00:42	0:00:42 - 0:01:03	0:01:03 - 0:01:24	0:01:24 - 0:01:45	0:01:45 - 0:02:06	0:02:06 - 0:02:27	Total
Profile 1	11,4	34,4	38,0	41,0	45,8	39,2	42,8	38,3
Total	11,4	34,4	38,0	41,0	45,8	39,2	42,8	38,3

На даній таблиці можна побачити кількість сторінок, які завантажуються за секунду, у певні відрізки часу. Кількість залишається приблизно на одному рівні і показує досить стабільний результат, завдяки чому користувачі можуть користуватися ресурсомбез скачків трафіку.

Середня кількість сягає – 38,3 сторінок в секунду.

Таблиця 3.21 – Кількість сторінок з помилками

Pages with errors											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	0	0	0	0	0	0	0	0	0	0	0
Total	0	0	0	0	0	0	0	0	0	0	0

В даній таблиці зображена кількість сторінок, які не змогля завантажитися черех певні причини та конфлікти. Як можна побачити – таких сторінок немає, що говорить про успішне застосування та функціонування данного методу оптимізації.

Таблиця 3.22 – Кількість надісланих даних, КБайт

KBytes sent											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	383	1 153	1 277	1 378	1 538	1 972	1 437	1 464	1 380	1 507	13 489
Total	383	1 153	1 277	1 378	1 538	1 972	1 437	1 464	1 380	1 507	13 489

Згідно наведеної таблиці, загальна кількість надісланих даних дорівнює – 13489 Кбайт.

В середньому це 1300 Кбайт в певні проміжки часу, в який виконувалося тестування.

Такі дані важливі при контролі потоку інформації для розробників та фірм які випускють даний продукт.

Таблиця 3.23 – Кількість отриманих даних, КБайт

KBytes received								
Profile	0:00:00 - 0:00:21	0:00:21 - 0:00:42	0:00:42 - 0:01:03	0:01:03 - 0:01:24	0:01:24 - 0:01:45	0:01:45 - 0:02:06	0:02:06 - 0:02:27	Total
Profile1	1 289 925	3 886 747	4 305 407	4 644 861	5 182 330	6 647 639	4 842 876	45 464 196
Total	1 289 925	3 886 747	4 305 407	4 644 861	5 182 330	6 647 639	4 842 876	45 464 196

В наведеній таблиці можна побачити, що загальна кількість отриманих даних дорівнює – 45464196 Кбайт. В кожен проміжок часу тестування було отримано від 3000 до 5000 Кбайт даних.

Таблиця 3.24 – Час відгуку сторінки

Response time, sec											
Name	Time	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30
Profile1	Min	0	0	0,001	0,003	0,003	0,003	0,002	0,007	0,009	0,009
	Avg	0,001	0,003	0,004	0,006	0,007	0,01	0,01	0,01	0,02	0,02
	Avg90	0,003	0,01	0,007	0,009	0,01	0,02	0,02	0,02	0,02	0,02
	Max	0,01	0,17	0,02	0,02	0,02	0,05	0,03	0,04	0,04	0,04

В таблиці 3.24 наведено дані часу відгуку сторінки на запити віртуальних користувачів, а саме мінімальний, середній та максимальний час відклику.

Такі дані важливі при розумінні зручності для користувача який очікує на завнтаження викликаної ним сторінки.

Таблиця 3.25 – Загальний відсоток помилок

Total errors %										
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	Total
Profile1	0	0	0	0	0	0	0	0	0	0
Total	0	0	0	0	0	0	0	0	0	0

Дана таблиця показує кількість помилок які з’являлися під час процесу тестування. Як можна побачити, помилок не виникало, отже метод працює справно і імплементувався без проблем і конфліктів.

З отриманими даними можна сказати що метод кешування досить замітно знизив час очікування завантаження сторінок.

Останнім методом, який підлягав тестуванню був метод HTTP стиснення. Даний метод полягає у стисненні файлів, які сервер відправляє браузеру для оптимізації роботи сервера та зменшенню часу передачі файлів.

Існує декілька варіантів такого стиснення, але для тестування було вибрано GZIP стиснення, яке розуміє під собою передачу без втрат.

Даний метод найкраще підходить для застосування у проектах з великими об’ємами коду, які передаються сервером браузеру.

Основні результати даного методу оптимізації зображено в таблиці 3.10.

Таблиця 3.26 – Загальні результати проведеного дослідження

Summary								
Profile	Sessions performed	Sessions with errors	Pages performed	Pages with errors	Hits performed	Hits with errors	Total KBytes sent	Total KBytes received
Profile 1	3 549	0	3 549	0	67 431	0	20 000	143 879 645

Продовження таблиці 3.26

Total	3 549	0	3 549	0	67 431	0	20 000	143 879 645

В таблиці відображено загальні результати проведеного дослідження. З більш точними результатами можна ознайомитися в подальших таблицях, представлених в даному розділі.

До кожної таблиці передбачене загальне пояснення.

Таблиця 3.27 – Кількість активних користувачів

Number of active users										
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30
Profile1	1	3	5	7	9	12	14	16	18	20
Total	1	3	5	7	9	12	14	16	18	20

В даній таблиці зображено активних користувачів у певні відрізки часу, в які проходило тестування, по мірі їх збільшення до максимуму. В кінці тестування число користувачів, які одночасно знаходилися на сервері дорівнювало – 20.

Таблиця 3.28 – Кількість відкритих сесій

Sessions											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	126	350	356	376	372	548	376	371	318	356	3549
Total	126	350	356	376	372	548	376	371	318	356	3549

В даній таблиці наведено кількість відкритих, віртуальними користувачами, сесій у певні відрізки часу, в які проходило тестування. В загальному, кількість відкритих, віртуальними користувачами, сесій досягла – 3549.

Таблиця 3.29 – Кількість запущених сторінок

Pages							
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	Total
Profile 1	126	350	356	376	372	548	3 549
Total	126	350	356	376	372	548	3 549

В даній таблиці зображено кількість сторінок, які були запущені на певних відрізках часу, під час тестування. Можна побачити що загальна кількість запущених сторінок дорівнює – 8036 сторінок.

Таблиця 3.30 – Кількість відкритих сесій за секунду

Sessions per second											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9
Total	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9

Таблиця 3.30 показує дані по кількості відкритих сесій за секунду та середню кількість цих сесій.

Кількість відкритих сесій на протязі усього тесту тримається приблизно на одному рівні, а середній результат сягає – 16,9 сесій в секунду.

На таблиці 3.31 можна побачити статистику відкритих сторінок за секунду, що показує результат оптимізації данного методу.

Таблиця 3.31 – Кількість сторінок в секунду

Pages per second											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9
Total	6,3	17,5	17,8	18,8	18,6	18,3	18,8	18,6	15,9	17,8	16,9

На даній таблиці можна побачити кількість сторінок, які завантажуються за секунду, у певні відрізки часу. Кількість залишається приблизно на одному рівні . Середня кількість сягає – 16,9 сторінок в секунду.

Таблиця 3.32 – Кількість сторінок з помилками

Pages with errors											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	0	0	0	0	0	0	0	0	0	0	0
Total	0	0	0	0	0	0	0	0	0	0	0

В даній таблиці зображена кількість сторінок, які не змогля завантажитися черех певні причини та конфлікти. Як можна побачити – таких сторінок немає, що говорить про успішне застосування та функціонування данного методу оптимізації.

Таблиця 3.33 – Кількість надісланих даних, КБайт

KBytes sent											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total

Продовження таблиці 3.30

Profile1	710	1 972	2 006	2 119	2 096	3 088	2 119	2 091	1 792	2 006	20 000
Total	710	1 972	2 006	2 119	2 096	3 088	2 119	2 091	1 792	2 006	20 000

Згідно наведеної таблиці, загальна кількість надісланих даних дорівнює – 20000 Кбайт.

В середньому це 2000 Кбайт в певні проміжки часу, в який виконувалося тестування.

Таблиця 3.34 – Кількість отриманих даних, КБайт

KBytes received											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	5 108 153	14 189 314	14 432 559	15 243 377	15 081 214	22 216 412	15 243 377	15 040 673	12 892 005	14 432 559	143 879 645
Total	5 108 153	14 189 314	14 432 559	15 243 377	15 081 214	22 216 412	15 243 377	15 040 673	12 892 005	14 432 559	143 879 645

В наведеній таблиці можна побачити, що загальна кількість отриманих даних дорівнює – 143879645 Кбайт. В кожен проміжок часу тестування було отримано від 14000 до 15000 Кбайт даних.

Таблиця 3.35 – Час відгуку сторінки

Response time, sec											
Name	Time	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30
Profile1	Min	0	0	0	0	0	0	0	0	0	0
	Avg	0,000 008	0,00 004	0,00 03	0,00 02	0,00 02	0,00 03	0,00 03	0,00 03	0,00 08	0,00 07

Продовження таблиці 3.35

	Avg90	0,0001	0,0003	0,002	0,0008	0,0008	0,001	0,001	0,001	0,003	0,003
	Max	0,001	0,001	0,04	0,003	0,002	0,009	0,007	0,006	0,01	0,02

В таблиці 3.35 наведено дані часу відгуку сторінки на запити віртуальних користувачів, а саме мінімальний, середній та максимальний час відклику.

Таблиця 3.36 – Загальний відсоток помилок

Total errors %											
Profile	0:00:00-0:00:21	0:00:21-0:00:42	0:00:42-0:01:03	0:01:03-0:01:24	0:01:24-0:01:45	0:01:45-0:02:06	0:02:06-0:02:27	0:02:27-0:02:48	0:02:48-0:03:09	0:03:09-0:03:30	Total
Profile1	0	0	0	0	0	0	0	0	0	0	0
Total	0	0	0	0	0	0	0	0	0	0	0

Дана таблиця показує кількість помилок які з’являлися під час процесу тестування. Як можна побачити, помилок не виникало.

Отже метод працює справно і імплементувався без проблем і конфліктів.

Результати проведеного дослідження показали, що кожен метод сприяє покращенню продуктивності роботи веб-сервера та зменшує його час відклику.

Результати, представлені в дослідженні мають найбільшу практичну користь для розуміння як конкретно впливає кожен метод оптимізації на продуктивність сервера.

Також для порівняння можна ознайомитися з таблицею 3.36, в якій представлено результати усіх проведених тестувань для більшого розуміння отриманих результатів та підведення підсумків проведеного дослідженння.

Таблиця 3.36 – Результати тестування усіх методів оптимізації та контрольного тестування

Тип тестування	Сторінки в сек. Анімовані елементи	Сторінки в сек. Статичні елементи	Швидкість завантаження. Анімовані елементи	Швидкість завантаження. Статичні елементи
Контрольне тестування	40	7	0,35	0,7
Метод HTTP стиснення	128	20	0,01	0,04
Метод кешування	50	40	0,02	0,3
Метод оптимізації паралельних завантажень	121	19-20	0,19 - 0,2	0,12 - 0,2

Користуючись цими даними можна більш грамотніше впроваджувати потрібні елементи оптимізації в свої продукти.

Ще однією ціллю проведеного дослідження було оновлення застарілих результатів попередніх досліджень і виявлення актуальності методів оптимізації, які використовують і по сьогодні. Результати показали, що впроваджені методи і до теперішнього часу зберігають свою доцільність у веб-розробці, та рекомендуються до використаннях у проектах.

ВИСНОВКИ

В ході проведення дослідження було проаналізовано можливі методи для оптимізації веб-сайтів, зокрема їх веб-серверу, відібрано 3 методи оптимізації для проведення тестів та порівняння результатів.

Згідно з результатами, усі три методи дають вплив на збільшення продуктивності роботи веб-сайту та відклику сервера на запити користувача.

Цінність проведення такого дослідження полягає у отриманні більш точних даних про вплив конкретних методів оптимізації на роботу веб-ресурсів та оновлення існуючих даних, які протягом довгого часу застаріли.

Можна стверджувати що наявні методи оптимізації досі мають значний вплив на оптимізацію роботи веб-сайтів та можуть активно використовуватися у проектах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дж. К. Могул, Ф. Дугліс, А. Фельдманн і Б. Крішнамурті. Потенційні переваги дельта-кодування та стиснення даних для HTTP. В матеріалах симпозіуму ACM SIGCOMM '97, Канни, Франція, с. 344, вересень 1997 р.
2. Б. Крішнамурті, Дж. Рексфорд. Веб-протоколи та практика. Аддісон-Веслі, с.230, травень 2001 р.
3. Дж. Байерс, Дж. Консідайн, М. Мітценмахер і С. Рост. Доставка інформованого вмісту через адаптивні накладні мережі. У Proc. ACM SIGCOMM, с. 128, 2002.
4. Дж. В. Байерс, М. Любі та М. Мітценмахер. Паралельний доступ до кількох дзеркальних сайтів: використання кодів Tornado для прискорення завантаження. У Proc. INFOCOM, Нью-Йорк, Нью-Йорк, с. 340, березень 1999 р.
5. Дж. В. Байерс, М. Любі, М. Мітценмахер та А. Реге. Підхід цифрового фонтану до надійного розподілу масових даних. У Proc. ACM SIGCOMM, Ванкувер, Канада, с. 211, 1998.
6. Р. Л. Картер і М. Кровелла. Вибір сервера за допомогою динамічної характеристики шляху в глобальних мережах. У Proc. INFOCOM, том 3, Кобе, Японія, с. 225, квітень 1997 р.
7. З. Фей, С. Бхаттачарджі, Е. В. Зегура та М. Х. Аммар. Нова техніка вибору сервера для покращення часу відповіді реплікованого сервісу. У Proc. ІНФОКОМ, том 2, с.233, 1998.
8. Л. Чіалінгеон, “Вплив мультимедійних стандартів на мультимедійну індустрію”, Proceedings of IEEE, v. 86, n. 16, с.210, 1998.
9. К. Діфендофф, П.К. Дюбі, “Як мультимедійні навантаження змінять дизайн процесора”, IEEE Computer, с. 357, вересень 1997 р.
10. Дж.А. Уотлінгтон, В.М. Боув-молодший, “Система для паралельної обробки медіа”, Parallel Computing 23, с. 123, 1997.

11. А. Гонсалес, К. Аліагас, М. Валеро, “Кеш даних з кешуванням кількох стратегій, налаштованих на різні типи місцевості”, Proc. з ACM Int. конф. про суперкомп'ютери, Барселона, Іспанія, с. 242, 1995.
12. Р. Філдінг., "Протокол передачі гіпертексту - HTTP/1.1", консорціум World Wide Web, с. 221, Червень 1999.
13. Б. Крішнамуртий. і К. Вільямс, "Кішки-мишки: витрати на доставку змісту при доступі до веб-сторінок", WWW 2006, 23-26 травня, 2006, Едінбург, Шотландія.
14. Н. Ф. Максемчук, «Дисперсний маршрут». У Proc. Міжнародної конференції зі зв'язку, с. 340, 1975.
15. С. МакКанне, В. Джейкобсон та М. Веттерлі, «Рівневе багатоадресне пересилання на основі приймача». У Proc. ACM SIGCOMM ’96, с. 130, 1996.
16. К. К. Міллер, «Надійні протоколи багатоадресної передачі: практичний погляд». У Proc. 22-ї щорічної конференції з локальних комп’ютерних мереж, с. 331, 1997 рік.
17. М. Мітценмахер, Сила двох варіантів у випадковому балансуванні навантаження, докторська дисертація, Каліфорнійський університет в Берклі, с. 226, 1996.
18. Дж. Нонненмахер та Е. В. Бірсак, «Асинхронна багатоадресна передача: AMP». У Proc. Міжнародної конференції з комп’ютерних комунікацій, Канни, Франція, с. 321, листопад 1997 р.
19. Дж. Нонненмахер, Е. В. Бірсак та Д. Тоуслі, «Відновлення втрат на основі паритету для надійної багатоадресної передачі». У Proc. ACM SIGCOMM ’97, с. 322, 1997.
20. М. О. Рабін, «Ефективне розповсюдження інформації для безпеки, балансування навантаження та відмовостійкості». У журналі ACM, том 38, с.48, 1989.

21. Л. Ріццо, «Ефективні коди стирання для надійних протоколів комп’ютерного зв’язку». In Computer Communication Review, с.322, квітень 1997 року.
22. Л. Ріццо та Л. Вічізано, «Надійний протокол розповсюдження даних багатоадресної передачі на основі програмних методів FEC». У Proc. HPCS ’97, Греція, с. 221, червень 1997 року.
23. E. Schooler and J. Gemmell, «Використання FEC для багатоадресної передачі для вирішення проблеми опівнічного божевілля», Microsoft Research Technical Report MS-TR-97-25, с. 337, вересень 1997 р.
24. С. Сешан, М. Стемм і Р. Кац, «SPAND: Виявлення продуктивності спільної пасивної мережі», у Proc. симпозіуму Usenix з Інтернет-технологій та систем ’97, Монтерей, Каліфорнія, с. 113, грудень 1997 року.
25. Л. Вічізано, Л. Ріццо та Дж. Кроукрофт. «Контроль перевантаження, подібний до TCP, для багатошарової передачі даних». У Proc. ІНФОКОМ ’98, с.233.
26. М. Яжнік, Дж. Курос та Д. Тоуслі, «Кореляція втрат пакетів у мережі багатоадресної передачі MBone». In Proceedings of IEEE Global Internet ’96, Лондон, с. 334, листопад 1996 року.
27. Р. Яваткар, Дж. Гріффоен та М. Судан, «Надійний протокол розповсюдження для інтерактивних спільних додатків». In Proceedings of ACM Multimedia ’95, Сан-Франциско, 1995, с. 344.
28. С. Флойд, В. Джейкобсон, Ч. Г. Лю, С. МакКенн та Л. Чжан, «Надійний
29. Multicast Framework для легких сеансів і фреймування на рівні додатків». В ACM SIGCOMM ’95, ср. 342-356, серпень 1995 р.
30. Дж. Геммелл, «ECSRM – масштабована надійна багатоадресна передача, яка виправляє стирання», технічний звіт Microsoft Research MS-TR-97-20, с. 332, червень 1997 року.
31. С. Н¨anle. «Порівняння архітектури та продуктивності між надійними протоколами багатоадресної передачі через MBONE». Дипломна робота,

Інститут телематики Університету Карлсруе. URL: www.starburstcom.com.w papers.htm, дата звернення: липень 2021.

32. C. Huitema, «Приклад для пакетного рівня FEC». У Proc. 5-го міжнародного семінару IFIP з протоколів для високошвидкісних мереж, Софія-Антиполіс, Франція, с. 339, жовтень 1996 р.
33. М. Лубі, М. Мітценмахер, А. Шокроллахі, Д. Спілман та В. Стеманн, «Практичні коди, стійкі до втрат». У матеріалах 29-го симпозіуму ACM з теорії обчислень, с. 128, 1997 рік.
34. М. Лубі, М. Мітценмахер та А. Шокроллахі, «Аналіз випадкових процесів за допомогою оцінки дерева І-або». У матеріалах 9-го щорічного симпозіуму ACM-SIAM з дискретних алгоритмів, с. 240, січень 1998 року.
35. Максемчук Н.Ф., Дисперсність маршрутизації в мережах зберігання та пересилання,
36. к.т.н. дисертація, Університет Пенсільванії, с. 59, 1975.
37. Веб-сайт Tucows. URL: http://www.tucows.com, дата звернення: червень 2021 р.
38. Всесвітня мережа консорціум. URL: http://www.w3.org, дата звернення: вересень 2021 р.
39. Моше Сіді Йоав Небат. Паралельно повторюємо міркування завантаження. В ІНФОКОМ, с. 223, 2002.
40. В. Стівенс М. Оллман, В. Пакссон. Контроль перевантаження Tcp. IETF RFC 2581, с. 332, квітень 1999 р.