НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____20__ р.

Дипломна робота

освітньо-кваліфікаційного рівня «_Бакалавр_»
(назва ОКР)

з напряму підготовки 6.050101 Комп'ютерні науки
(код та назва напряму підготовки)

на тему: «Конфігурація веб-серверів в умовах високого навантаження»

Виконав: студент _IV_ курсу, групи _ДА-31
(шифр групи)

Волошин Владислав Сергійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник _____доцент, к.т.н. Кисельов Г.Д._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний ___доц. Рощина Н.В. _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____доцент, к.т.н. Кучернюк П.В. ____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст.. викладач Британов О.А._____ ____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського

Інститут (факультет) ННК «Інститут прикладного системного аналізу» _
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 7.05010102, 8.05010102 Інформаційні технології проектування 7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту

Волошин Владислав Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи Конфігурація веб-серверів в умовах високого навантаження_____,

керівник роботи Кисельов Геннадій Дмитрович, к.т.н., доцент _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» травня 2017 р. № 1477

2. Термін подання студентом роботи 13.06.2017

3. Вихідні дані до роботи віртуальні машини з операційною системою Ubuntu 14.04 LTS, веб-сервер Nginx, Docker, форма реалізації – веб-система

4. Зміст роботи

1. Дослідження принципів, підходів до реалізації та організації систем з високим навантаженням.
2. Дослідження та вивчення особливостей роботи веб-серверів.
3. Аналіз підвищення продуктивності веб-серверів шляхом налаштування внутрішніх параметрів системи.
4. Аналіз підвищення продуктивності веб-серверів шляхом впровадження змін до архітектури системи.
5. Тестування навантаження на веб-сервер, розгляд прикладу високонавантаженої системи.
6. Провести функціонально-вартісний аналіз отриманої системи.
7. Проаналізувати результати роботи, зробити висновки.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Принцип роботи веб-сервера – плакат
2. Архітектурні способи підвищення продуктивності веб-серверів – плакат
3. Результати тестування навантаження – плакат

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., професор, д.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.02.2017	
2	Збір інформації	15.02.2017	
3	Вивчення варіантів реалізації та вибір варіанту для розробки	03.03.2017	
4	Розробка структури системи	10.03.2017	
5	Розробка плану тестування	17.03.2017	
6	Розробка програми	30.03.2017	

_____

7	Розробка опису системи	28.04.2017	
8	Тестування системи та аналіз результатів	10.05.2017	
9	Оформлення дипломної роботи	06.06.2017	
10	Отримання допуску до захисту та подача роботи в ДЕК	13.06.2017	

Студент _____ Волошин В.С.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Кисельов Г.Д.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

до бакалаврської дипломної роботи Волошина Владислава Сергійовича

на тему: «Конфігурація веб-серверів в умовах високого навантаження»

Дипломна робота присвячена дослідженню конфігурації веб-серверів в умовах високого навантаження. В роботі були розглянуті методи підвищення надійності та продуктивності веб-сервера, а саме: налаштування внутрішніх параметрів сервера та зміна архітектури системи.

Основними вимогами до системи були стабільна робота при високому навантажені на сервер та максимально зрозуміла та зручна конфігурація системи. Було проведено налаштування веб-серверу за допомогою розглянутих способів та визначено, які серед них забезпечують найбільшу доступність при роботі.

Загальний об’єм роботи – 64 сторінки, 15 рисунків, 6 таблиць, 17 посилань.

Ключові слова: веб-сервер, високонавантажена система, Nginx, конфігурація, кешування, балансування навантаження, черги повідомлень, продуктивність.

АНОТАЦИЯ

к бакалаврской дипломной работе Волошина Владислава Сергеевича

на тему: «Конфигурация веб-серверов в условиях высокой нагрузки»

Дипломная работа посвящена исследованию конфигурации веб-серверов в условиях высокой нагрузки. В работе были рассмотрены методы повышения надежности и производительности веб-сервера, а именно: настройка внутренних параметров сервера и изменения архитектуры системы.

Основными требованиями к системе были стабильная работа при высокой нагрузке на сервер и максимально понятная и удобная конфигурация системы. Была проведена настройка веб-сервера с помощью рассмотренных способов и определенно, какие среди них обеспечивают самую большую доступность при работе.

Общий объём работы – 64 страницы, 15 рисунков, 6 таблиц, 17 ссылок.

Ключевые слова: веб-сервер, высоконагруженная система, Nginx, конфигурация, кеширование, балансировка нагрузки, очередь сообщений, производительность.

ANNOTATION

On Vladyslav Voloshyn bachelor's degree

Thesis: «Web server's configuration in case of high load»

Thesis is devoted to research of web server's configuration in case of high load. There were investigated high availability and performance methods for web server, namely: web server's inner parameters configuration and architectural changes.

Main requirements to the system were stable work under high load and easy to understand configuration. A provisioning of web server was held and found out what methods guarantee the biggest availability alongside performance metrics.

Total volume of work – 64 pages, 15 figures, 6 tables, 17 references.

Keywords: web server, high load system, Nginx, configuration, caching, load balancing, message queue, performance.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP – HyperText Transfer Protocol

HTML – HyperText Markup Language

URL – Uniform Resource Locator

SPOF – Single Point Of Failure

DNS – Domain Name System

IP – Internet Protocol

XML – eXtensible Markup Language

CSS – Cascading Style Sheets

CDN – Content Delivery Network

ВСТУП

Сучасні інформаційні системи створюються з врахуванням навантаження на відповідну систему та визначенням допустимої межі операцій за одиницю часу. Оскільки в наш час більшість систем призначені для відкритого користування, проблема гарантування працездатності системи в умовах високого навантаження постає дуже гостро. Не є виключенням і Web-системи, їхнє розповсюдження через мережу Інтернет.

Задля забезпечення надійності системи необхідно враховувати її конфігурацію. Змінюючи доступні параметри, можливо налаштувати систему, що відповідатиме поставленим вимогам. Особливої уваги питання конфігурації системи заслуговує при її використанні в загальнодоступному середовищі. Високонавантажені системи потребують окремих підходів до архітектури ПЗ.

Актуальність теми дослідження полягає в тому, що сучасний стан відкритих веб-систем вимагає надійності та достатньої продуктивності для того, аби задовольнити великий попит на них. При зростанні популярності сервісу з’являються проблеми з навантаженням, що неодмінно призводить до погіршення роботи системи. Так як більшість сучасних веб-систем користуються в своїй роботі готовими рішеннями в обслуговуванні http-запитів, а саме веб-серверів, їхня конфігурація потребує ретельного аналізу.

Метою дипломної роботи є дослідження існуючих методів підвищення продуктивності веб-серверів та порівняння цих методів з налаштуваннями, що ставляться за замовченням у веб-серверів.

Налаштований веб-сервер має справлятися з навантаженням не менш як 50000 одночасних запитів, не втрачаючи при цьому швидкість обробки запитів. Крім того, конфігурація веб-сервера має бути максимально простою та легкою у підтримці і модернізації. В якості доповнення сервер має показувати високу доступність в разі відмови однієї з компонентів системи.

Потенційні застосування та практична цінність результатів дипломної роботи:

1) Методи та алгоритми запропоновані в даній роботі можуть бути використані як частина веб-системи, що має справу з великим навантаженням;
2) Сформульовано рекомендації з вибору правильного методу в залежності від очікуємого навантаження.

1 ПРИНЦИПИ ПОБУДОВИ СИСТЕМ З ВИСОКИМ НАВАНТАЖЕННЯМ

Високонавантажена система – це система, що опрацьовує велику кількість одночасних запитів. Система стає високонавантаженою в той самий момент, коли при налаштуванні зі стандартною конфігурацією перестає справлятись з навантаженням. Дане поняття є достатньо відносним, оскільки для різних систем існують різні показники цього визначення, такі як: кількість запитів, швидкість роботи системи та інші.

При побудові високонавантаженої системи відіграють важливу роль декілька основних принципів:

- *Відмовостійкість та доступність*

Система, що складається з багатьох життєво важливих елементів, створює певні складності у підтримці та забезпеченні стабільної роботи. Ризик відмови частини інфраструктури вирішується шляхом надмірності, тобто коли кожен вузол має резервну копію. І у випадку, коли робота основного елементу порушується, резервний бере на себе роботу основного.

Головне правило забезпечення відмовостійкості – це уникати числа 1, так звана ситуація SPOF (Single Point Of Failure), коли кожен компонент має бути зарезервований. Резервні компоненти мають повторювати конфігурацію основних. Таким чином вірогідність виходу з ладу двох серверів одночасно набагато менша, ніж одного.

Масштабування будь-якої програми, тобто розширення за рахунок створення додаткових вузлів, має попередньо бути підтверджене аналізом навантаження на систему. Завдяки цьому підходу визначаються найбільш навантажені ділянки, які потребують відокремлення та оптимізації.

- *Моніторинг*

Якісна робота будь-якої програми залежить від правильної та своєчасної діагностики. Задача моніторингу полягає максимально швидко дізнаватись про зміни в системі, особливо такій як web-програма.

- *Гнучкість*

Система, що розрахована на велике навантаження, повинна забезпечити необхідну гнучкість. Оскільки кількість запитів може збільшуватись доволі непередбачено з часом, неможливо побудувати детальне архітектурне рішення. Такі системи потребують постійного розвитку, а отже можливості внесення змін. Тобто саме забезпечення гнучкості є одним з найголовніших аспектів при побудові динамічних елементів.

- *Використання простих рішень*

Гнучка за своєю основою система апріорі не буває складною. Крім того, не треба намагатись одразу проектувати інфраструктуру, здатну витримати мільйони користувачів. Золоте правило – поступові рішення з можливістю подальшого розширення.

- *Акцентування лише на важливому*

Велика система вимагає рішення великої кількості задач. Проте, часу на вирішення всіх проблем буде недостатньо. Необхідно правильно розставляти пріоритети, займатися задачами, що стосуються абсолютної більшості користувачів.

1.1 Висновки до розділу 1

У цьому розділі було розглянуто поняття високонавантаженої системи та перераховані основні принципи, яких слід дотримуватися при побудові такої системи. Саме ці принципи дозволять побудувати надійну та стабільну систему та в подальшому з легкістю керувати нею.

2 ПОНЯТТЯ ВЕБ-СЕРВЕРА

Веб-сервер – це комп’ютерна система, що оперує даними за допомогою HTTP-протоколу. Веб-сервер відповідає за опрацювання вхідних запитів та відповідь на них.

При розгляданні веб-серверу потрібно виділити 2 сторони, що беруть участь у обміні даними: сам веб-сервер та клієнт. Під клієнтом розуміється будь-яка система, яка здатна генерувати HTTP-запити та опрацьовувати HTTP-віповіді.

HTTP (Hypertext transfer protocol) – це протокол прикладного рівня, що призначається для передачі даних між двома комп’ютерами у гіпертекстовому форматі, тобто зв’язані веб-документи. HTTP є текстовим протоколом без збереження стану. Ні клієнт, ні сервер не пам’ятають про попередні з’єднання. Наприклад, спираючись тільки на HTTP, сервер не зможе згадати введений вами пароль чи на якому кроці транзакції ви знаходитесь.

HTTP основується на певних строгих правилах:

- Тільки клієнти можуть відправляти HTTP запити, і тільки на сервера. Сервера в свою чергу відповідають лише на HTTP запити клієнта.
- Веб-сервер має відповісти на кожен HTTP запит, у крайному випадку з повідомленням про помилку з відповідним кодом.
- Коли запитується фізичний файл, клієнт повинен сформувати так званий file URL, який вказуватиме фізичне розташування файлу на сервері.
- При отриманні запиту, HTTP сервер спочатку перевіряє чи існує ресурс за даним URL. Якщо існує, веб-сервер відправляє зміст файлу назад у браузер. Якщо ж ні, веб-сервер відправляє

повідомлення про помилку до браузеру, попередньо сформувавши необхідно сторінку з кодом помилки (такі як 404, 502 та інші).

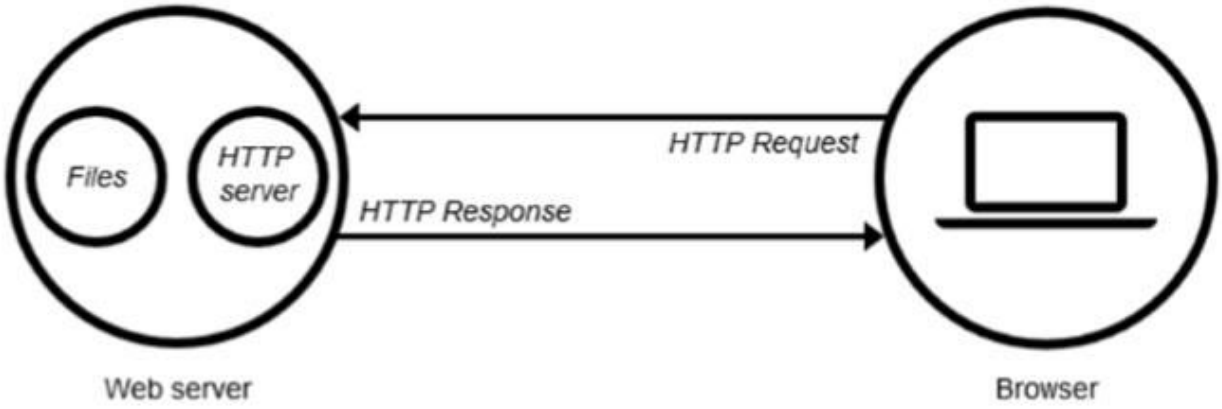

Рис 2.1 – Загальний вигляд роботи веб-сервера в стандартній конфігурації [1]

Веб-сервер може віддавати статичний або динамічний контент сторінки. Статичний означає, що вміст сайту віддається як є. Динамічний – сервер опрацьовує дані або навіть генерує їх на льоту. Це забезпечує більше гнучкості, проте технічно складніше в обслуговуванні, що робить динамічний контент більш складним для створення веб-сайту.

2.1 Архітектура популярних веб-серверів

На даний момент найбільш популярними веб-серверами у світі є Apache та Nginx. Разом вони обслуговують більш ніж 50% трафіку у всьому Інтернеті.

Різниця між цими серверами полягає в тому як вони опрацьовують підключення та відповідний їм трафік. В Apache за паралельну обробку запитів клієнтів відповідають модулі мультипроцесингу (multi-processing modules або MPM). Найпростіший з цих модулів – mpm_prefork. Він працює за схемою «один запит – один системний процес з одним потоком». Очевидний недолік такого методу – як тільки запитів стає більше ніж

процесів з mpm_prefork, продуктивність веб-серверу різко падає. З цієї причини mpm_prefork погано підходить для хостингу з слабким апаратним забезпеченням.

Модуль mpm_worker працює за приблизно такою ж схемою, з тією різницею, що кожне підключення опрацьовує один потік, а не процес – а це означає, що продуктивність такого підходу значно вища.

В модулі mpm_event додатково добавлена робота з keep-alive з'єднаннями, які оптимізують завантаження потоків.

Nginx початково замислювався як система з високим степенем паралелізму, її асинхронний код не створює блокувань, а підключення опрацьовуються за схемою подій. Процеси Nginx здатні опрацьовувати до 1 тисячі підключень одночасно, при появі нової події. Він не створює додаткових процесів для підключення та використовується з врахуванням одного процесу на одне ядро процесора, що підвищує ефективність роботи серверу.

Обробка динамічного контенту в Apache виконується через інкапсуляцію інтерпретатора (наприклад, php) в окремі процеси веб-сервера. Nginx не має вбудованих засобів для динаміки і тому використовує зовнішні інтерпретатори і бібліотеки, або передає запити динамічному веб-серверу.

Налаштування Apache для різних сайтів виконується через головний конфігураційний файл, а також через файли «.htaccess» для кожного окремо. При цьому налаштування перечитуються при кожному запиті до відповідного каталогу, без необхідності перезавантажувати сам веб-сервер.

Nginx у більшості випадків не дозволяє користувачам змінювати свої налаштування, що в певній мірі обмежує гнучкість конфігурації, проте дає перевагу в безпеці та швидкості роботи.

Обидва веб-сервери можуть розширювати свій функціонал через додаткові модулі, але схеми їх підключення суттєво різняться. Модулі

Apache завантажуються і відключаються динамічно, тобто доданням «include-директиви» в конфігураційний файл.

Модулі Nginx компілюються в монолітний код для кожної модифікації веб-сервера. Більшість функціональних пакетів включається в стандартну специфікацію, однак для додаткових завдань може знадобитися перекомпіляція пакетів з вихідного коду.

Завдяки особливостям своєї роботи Nginx показує кращі результати під навантаженням, що робить його одним з найпопулярніших рішень. Крім того, Nginx має вбудований функціонал балансування та HTTP кешування. За даними компанії Netcraft Nginx обслуговував 19,6 % найбільш навантажених сайтів станом на лютий 2017 року [5].

2.2 Висновки до розділу 2

У цьому розділі було розглянуто поняття веб-сервера та найпопулярніші на даний момент його реалізації, такі як: Apache та Nginx. Було детально проаналізовано принцип роботи веб-серверу, протоколу HTTP та особливості архітектури популярних веб-серверів. Крім того, була проведена порівняльна характеристика між Apache та Nginx, яка показала, що веб-сервер Nginx здатний краще справлятися з навантаженням.

3 ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ-СЕРВЕРІВ ШЛЯХОМ КОНФІГУРАЦІЇ ВНУТРІШНІХ ПАРАМЕТРІВ

Всі можливі способи підвищення швидкості сайтів поділяються на 5 категорій:

- максимальне використання кешу;
- мінімізація кількості запитів до сервера;
- мінімізація об’єму службових даних в запиті;
- мінімізація розміру даних передачі;
- оптимізація відображення змісту сайту веб-браузером;

3.1 Gzip

Використовуючи веб-сервер, перш за все слід звернути увагу на налаштування самого серверу, покращення його конфігурацій.

Веб-сервер без внутрішнього налаштування скоріш за все не використовує всі доступні ресурси. Існує декілька простих методів підвищення ефективності роботи веб-серверів.

Перш за все, слід звернути увагу на стискання запитів. Всі сучасні браузери підтримують роботу зі стисканням Gzip. Веб-сервер стискає зміст відповіді перед відправленням його клієнту, а браузер розпаковує його в момент отримання. Таким чином можливо зекономити до 70% розміру файлу. Для клієнтів це буде означати більш високу швидкість роботи сайту.

Стискання працює тільки для файлів текстового формату (HTML/XML, CSS, Javascript). Але треба пам’ятати, що стискання приводить до додаткового навантаження на сам фізичний сервер. Це навантаження є незначним доки об’єм файлів не зросте до певної позначки.

Gzip підтримує декілька рівней стискання – від швидкого і найгіршого до повільного і найкомпактнішого результату. Проте, коли потрібно стискати великі об’єми тексту, gzip може виявитись не настільки ефективним. Gzip працює в один потік і відповідно ефективно буде використовувати тільки одне ядро. Існує декілька альтернативних утиліт, які стискають дані, використовуючи всі ядра, серед них: pigz та pgzip.

Для того, щоб перевірити чи використовує сайт компресію можна скористатися Gzip Checker [6].

3.2 HTTP/2

Всі сучасні веб-сервери як і веб-браузери підтримують протокол HTTP/2. Він прийшов на заміну своєму попереднику – HTTP 1.1. Нова специфікація набагато швидша та продуктивніша за HTTP 1.1. За даними W3Techs більш як 7,1% сайтів вже підтримують HTTP/2 [7]. Наприклад, популярний ресурс для програмістів «Stack Overflow» перевів нещодавно повністю свій сайт на HTTP/2 саме через його переваги у швидкості обробки даних [8].

На відміну від тестового HTTP 1.1, HTTP/2 – бінарний. Тому протокол більш ефективний при парсингу, більш компактний при передачі та схильний до меншої кількості помилок.

В HTTP 1.1 браузери використовують декілька підключень до серверу для завантаження веб-сторінки, при цьому, кількість таких підключень обмежена. Але це не вирішує проблему с блокуванням каналу повільними пакетами. Тоді як в HTTP/2 використовується мультиплексування, яке дозволяє використовувати браузеру одне TCP з’єднання для всіх запитів (рис.2).

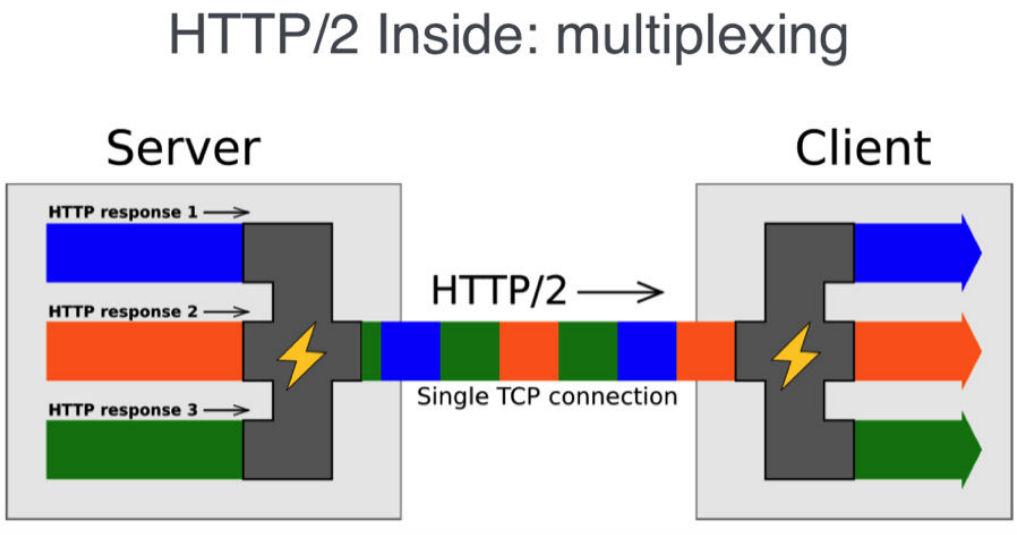

Рис.3.1 – Мультиплексування в HTTP/2 [2]

Всі файли підвантажуються паралельно. Запити і відповіді розділяються за фреймами з метаданими, які асоціюють запити і відповіді. Так що вони не перекривають один одного. При цьому відповіді отримуються по мірі їх готовності, а це означає, що важкі запити не будуть блокувати опрацювання і видачу більш простих даних.

Разом із мультиплексуванням з’явилася і пріоритезація трафіку. Запитам можна назначити пріоритет на основі важливості та залежності. При завантаженні веб-сторінки браузер буде в першу чергу отримувати важливі дані, наприклад, CSS-код, а все другорядне буде опрацьовуватись в останню чергу.

Протокол HTTP побудований таким чином, що при передачі запитів разом з ними передаються і заголовки, які містять додаткову інформацію. Сервер, в свою чергу, також прикріплює заголовки до відповідей. Враховуючи, що веб-сторінки складаються з багатьох файлів, всі заголовки можуть займати вагомий об’єм. Тому в HTTP/2 присутнє стискання заголовків, яке дозволяє істотно зменшити обсяг додаткової інформації, так що браузер зможе відправити всі запити одразу.

При використанні протоколу HTTP 1.1 браузер запитує сторінку, сервер відправляє у відповідь HTML і чекає, поки браузер його опрацює і отримає всі необхідні файли: Javascript, CSS та фото. Для того, аби зменшити час очікування відповіді, в новому протоколі застосували функцію Server Push. Вона дозволяє серверу одразу, не очікуючи відповіді веб-браузера, додати необхідні на його розсуд файли в кеш для швидкої віддачі.

Протокол HTTP/2 не потребує шифрування каналу. Тим не менш, всі сучасні браузери працюють з HTTP/2 тільки у парі з TLS. Таким чином, масове застосування протоколу має посприяти розповсюдженню шифрування в мережі. До того ж HTTP/2 спрощує процес шифрування та зменшує час підключення за рахунок того, що при створенні зашифрованого з'єднання відбувається лише один TLS Handshake.

3.3 Google Pagespeed

Google Pagespeed – це набір інструментів, розроблений компанією Google, який використовується для аналізу швидкості сайту та її оптимізації. Модуль pagespeed для веб-серверів, таких як Apache та Nginx, дозволяє автоматизувати багато задач пов'язаних з оптимізацією продуктивності. Таким чином він дозволяє покращити характеристики веб-сервера без складної конфігурації. Модуль може стискати статичний контент сайту та навіть оптимізувати картинки.

Існує також окрема утиліта з аналогічною назвою, що дозволяє проаналізувати продуктивність клієнтської частини веб-сайту. Доволі детальний аналіз зможе вказати на те, що необхідно оптимізувати на сайті, підвищивши швидкість та зручність для користувача.

3.4 Мініфікація файлів

З іншої сторони існують спеціальні програмні продукти, “project builders”, що опрацьовують текстові файли, прибираючи пусті символи. Вони створюють мініфіковані копії файлів. При створенні великого проекту, дані копії здатні оптимізувати роботу сайту. Прикладами таких програм є Gulp, Grunt, написані на Javascript.

Мініфікація (minify) – це простий підхід для зменшення розмірів файлів css, js та html. В процесі стискання всі коментарі до коду, переноси рядків, зайві символи табуляції та порожні символи видаляються, тобто ті символи, які не впливають на працездатність сайту. Такий підхід дозволяє зекономити 10-20% від оригінального розміру файлу.

Ще одним способом збільшення швидкості завантаження сайту може слугувати склеювання файлів. Даний метод підходить для CSS та Javascript файлів. Таким чином кількість файлів зменшиться, а відповідно зменшиться і кількість додаткових запитів. До того ж, не слід забувати і про оптимізацію зображень на сайті за рахунок стискання без втрати якості.

Щодо медіа контенту слід зауважити, що правильна оптимізація цього типу ресурсів може значно вплинути на швидкість роботи сайту. За статистикою HTTP Archive зображення займають в середньому 64% розміру веб-сторінки [10]. Частіше за все файли зображень зберігають купу додаткової інформації, наприклад деталі про фото, коментарі, текстовий опис, географічні дані і т.д. При використанні в Web ці дані не потрібні. Існують інструменти, які дозволяють вирізати всі непотрібні метадані, що значно зменшує розмір зображення. Найбільш оптимізованим для зображень ж формат WebP, який здатний замінити звичні JPEG та PNG.

3.5 Висновки до розділу 3

У даному розділі було розглянуто способи збільшення можливої кількості одночасних запитів, що здатний опрацювати веб-сервер, через налаштування доступних внутрішніх параметрів серверу. Значного приросту в продуктивності можливо досягти, застосувавши всі перераховані в розділі методи, а саме: стискання Gzip, використання протоколу HTTP/2, мініфікація файлів та використання додаткового модулю Google PageSpeed. Найкращі результати здатний дати модуль PageSpeed. Найменший вплив здійснює принцип мініфікації файлів, який дає помітний виграш лише при використанні дуже великих об’ємів веб-сторінок.

4 АРХІТЕКТУРНІ СПОСОБИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ-СЕРВЕРІВ

4.1 Фронтенд та бекенд сервери

Коли навантаження росте, web-система стає працювати повільніше. В певний момент причина вже буде стояти в самій реалізації системи, а не конфігурації окремих її частин.

Важливо пам’ятати, що постійно збільшувати ресурси серверу не вийде, в якийсь момент вертикальне масштабування не даватиме очікуємих результатів. Тоді доречним є розділення web-серверів на дві функціонально різні за своїм призначенням частини: frontend та backend веб-сервери.

Frontend сервер – це сервер, який отримує запити від клієнтів та відправляє їх відповідь. Вихід із ладу такого сервера приводить до недоступності всієї web-системи. Він буде передавати важкі запити (запити з динамічним контентом) на backend сервери.

Резервування frontend серверів зручно реалізовувати за допомогою віртуальних IP-адрес UCARP. Тоді один з двох ідентичних серверів отримує віртуальний IP, до якого прив’язане доменне ім’я. При виході з ладу робочого серверу, ця IP-адреса назначається резервному серверу.

Водночас з цим frontend сервери часто використовуються для балансування навантаження. Запити будуть автоматично розподілятися між вказаними backend серверами. Крім цього, деякі frontend-сервери можуть в реальному часі аналізувати помилки у відповідях від backend. У випадку виявлення помилки, він перестає відправляти запити на поломаний сервер. Час і кількість спроб можливо в такому випадку налаштувати. Найчастіше в якості frontend сервера використовується Nginx.

Backend сервер – це сервер, на якому працює основна веб-програма (наприклад, PHP). Зазвичай для них створюють резервні копії, які є

абсолютно ідентичними. Це дозволяє використовувати активне резервування. Тобто всі backend-частини починають опрацьовувати запити. Якщо один з серверів виходить із ладу, він просто виключається з загального списку доступних серверів і на нього перестають приходити запити від frontend-сервера.

Однак, слід враховувати, що при відключенні одного з серверів, навантаження розподілиться між залишеними серверами. Це призведе до збільшення кількості запитів на них. Тому кількість backend серверів потрібно розраховувати виходячи з того, що в будь-який момент може вийти з ладу 25% їх загальної кількості.

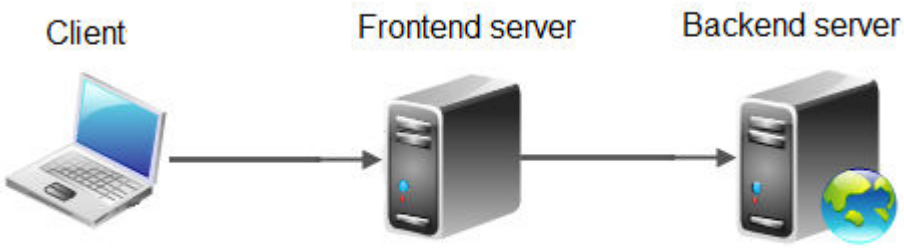

Рис.4.1 – Модель роботи фронтенд та бекенд серверів у зв’язці

4.2 Кешування

Кешування – це один із способів оптимізації web-систем. Дані, що лежать далеко від користувача потребують значного часу на їх передачу. В будь-якій програмі зустрічаються повільні операції, результати яких можна зберегти на деякий час. Це дозволить виконувати менше таких операцій, а більшості користувачів видавати заздалегідь збережені дані. Тобто основна мета кешування для веб-серверів – зменшення часу на отримання контенту, аби користувач якомога швидше отримав свої дані.

За даними компанії Yahoo, до 80% трафіку сайту можна «зрізати» за рахунок грамотного кешування. Тому кешування, як відносно просту технологію, впроваджують усюди: в браузерах, в проксі-серверах та ін.

Проте, допоки не буде прямої необхідності у використанні кешування, його не слід одразу використовувати. Це доволі проста техніка, але вона знижує гнучкість системи.

Розмір кешу постійно обмежений. Частіше за все він менше об’єму даних, який можна в цей кеш вложити. З цієї причини компоненти, які розміщені у кеші, рано чи пізно будуть витіснені. Сучасні фреймворки, що орієнтовані на кешування, дають можливість досить еластично розпоряджатися застаріванням, приймаючи до уваги цінність, період застарівання, розміри даних та інше.

Якщо одні і ті ж самі дані попадають у різні кеші, то з’являється питання про когерентність кешу. Наприклад, однакові дані можуть застосовуватись для формування різних веб-сторінок. Сторінки, що сформовані пізніше будуть включати обновлені дані, в той час як ті, що були закешовані раніше, будуть включати застарілі дані. Таким чином буде порушена цілісність системи. Аби запобігти таким випадкам кеш спорожнюється при зміні даних.

Визначення ефективності кешування здійснюється за рахунок кількості попадань запросів у кеш. Постійні очищення кешу, кешування рідко запитуємих даних, невідповідний розмір кешу – все це приводить до розтрати пам’яті, ніяк при цьому не збільшуючи результативність роботи. Інколи дані змінюються так швидко і несподівано, що процент попадань буде близький до нуля. Проте, як правило, дані зчитуються частіше, аніж записуються, в такій ситуації кеші будуть результативними.

Існують три основних типи кешування:

- Lazy cache (лінивий кеш) – найпростіший в реалізації тип кешування. Кеш просто зберігає дані і віддає їх допоки вони не застаріють.
- Synchronized cache (синхронізований кеш) – кеш, при якому клієнт окрім даних отримує відмітку останньої зміни інформації. Він відповідно запитує у постачальника чи не змінилися дані. Якщо дані не змінилися, запит на оновлення кешу повторно не буде відісланий. Такий тип кешування дозволяє завжди мати актуальні дані, проте досить складний у реалізації.
- Write-through cache (кеш наскрізного запису) – будь-яка зміна даних відразу відображається в кеші. Цей тип кешу може ніколи не застарівати, але виникають проблеми с «когерентністю».

4.3 Черги задач

Будь-яка програма – це набір послідовних інструкцій. Для того, щоб виконати наступну інструкцію, треба дочекатися завершення виконання попередньої. Час виконання усієї програми – це сумарний час виконання всіх інструкцій. А це означає, що повільна частина програми буде робити повільною всю програму.

В такому випадку слід використовувати принцип асинхронності, переносячи певні операції в асинхронне виконання, тобто таке, яке можна «відкласти», продовживши роботи основної програми. Має сенс відкладати важкі операції, які можуть зайняти велику кількість часу:

- Відправлення пошти
- Обробка зображень і медіа даних
- Відправлення зовнішніх запитів до третіх серверів та опрацювання відповідей (наприклад, використання різноманітних API)

Всі ці операції можуть бути виконані пізніше, а користувач отримає швидку відповідь від системи і продовжить її виконання («Дякуємо, ваше

відео оброблюється», «Ви отримаєте повідомлення протягом хвилини» і т.д.). Використання цього принципу передбачає прибирання з асинхронних задач всієї логіки взаємодії з користувачем з тієї причини, що очікування відповіді від користувача порушить сам підхід.

Черги задач дозволяють виконувати важкі операції асинхронно, не сповільнюючи систему. Цей принцип дозволить збільшити швидкість роботи певної ділянки системи, обслуговувати більшу кількість користувачів, використовувати різні мови розробки в одній програмі. Черги повідомлень є сполучною ланкою між різними процесами в системі і забезпечують надійний і здатний до масштабування інтерфейс взаємодії з іншими підключеними системами.

Важливо розуміти, що система черг – це принцип, а не конкретна технологія. Система черг складається з 2 основних компонент:

- Сервер черги
- Обробник

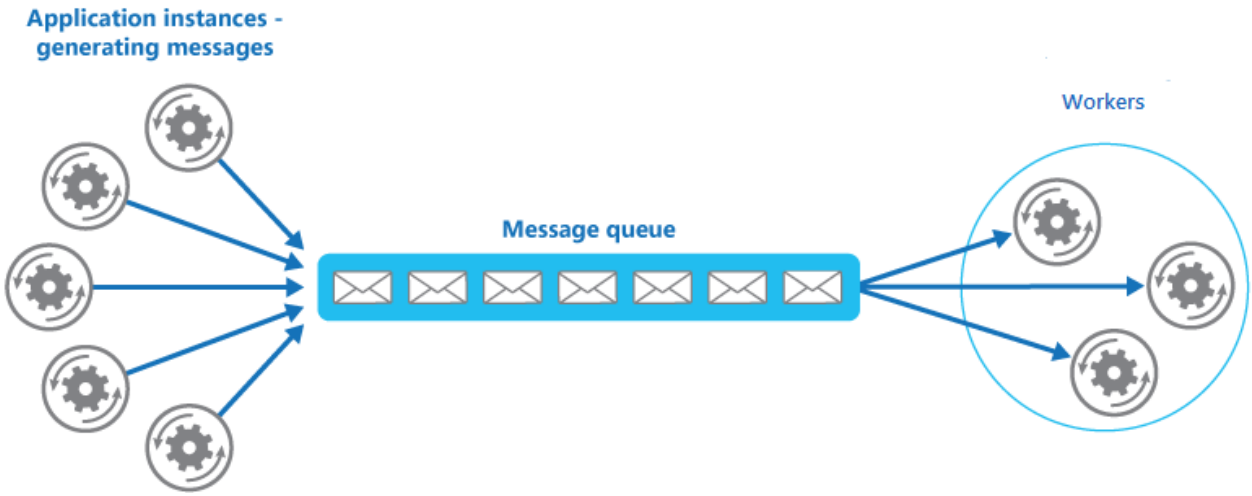

Рис.4.2 – Принцип роботи черг задач

Сервер черги зберігає список повідомлень чи задач, так звана «job queue», які відправляє йому основна програма. Задача серверу – зберігання інформації про те, що і як треба виконати. Сам сервер нічого не виконує.

Обробник (чи воркер) – це частина основної програми, яка працює з чергою у зворотному порядку. Він отримує нові повідомлення з черги та виконує відповідні дії. Таким чином все опрацювання програми переноситься в обробник. Відповідно обробник можливо запустити в декілька потоків, що є однією з основних його переваг (рис.4).

Асинхронні операції в програмах дозволяють зробити її більш ефективною, а головне швидкою для користувача. З точки зору масштабування система черг надає широкі можливості:

- Можливість реалізовувати обробники на різних технологіях дозволить приймати найбільш ефективні рішення;
- Використання декількох серверів повідомлень дозволить зробити програму надійною та здатною до масштабування; черга гарантує, що повідомлення буде доставлено та опрацьовано у випадку, коли є хоча б один обробник;
- Можливість запуску декількох обробників дозволить пришвидшити повільні операції;
- Більшість систем повідомлень підтримує пріоритезацію, що дозволить швидше виконувати важливі задачі;
- Черги дозволяють уникнути випадків неекономного використання ресурсів процесу в результаті зберігання неопрацьованої інформації.

Системи черг можна і треба використовувати не тільки на великих веб-сайтах. Повільні операції значно погіршують досвід використання програми, а черги дозволять забезпечити високу швидкість роботи для користувачів.

4.4 Балансування навантаження

Балансування навантаження є ключовою складовою високо доступних систем, що часто використовується для покращення продуктивності та надійності веб-сайтів, програм та інших сервісів, розподіляючи навантаження

серед багатьох серверів. Коли питання стоїть не тільки в тому, аби забезпечити продуктивність системи, але й гарантувати її стабільність та безвідмовність – балансування навантаження стає необхідним інструментом.

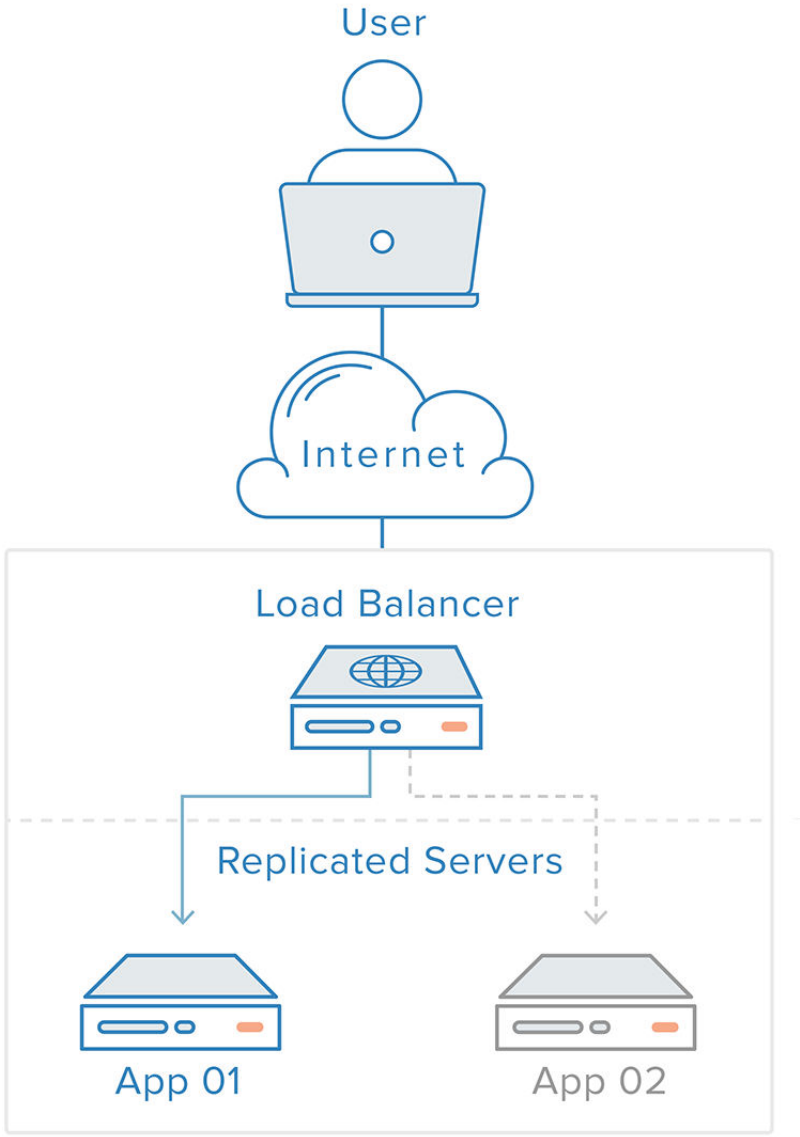

Рис.4.3 – Схема роботи балансувальника навантаження [11]

Користувач відсилає запит на сервер, на якому встановлене спеціальне програмне забезпечення, що відповідає за балансування навантаження. Далі сервер пересилає запит користувача на backend сервер, який вже віддає контент користувачу. При такому архітектурному рішенні, слабким місцем стає сам сервер, що відповідає за балансування. Він стає так званим SPOF,

якого треба всіляко уникати при побудові високонавантажених систем. Дану проблему можливо вирішити шляхом додавання другого балансувальника, який буде працювати синхронно з першим, образуючи тим самим кластер.

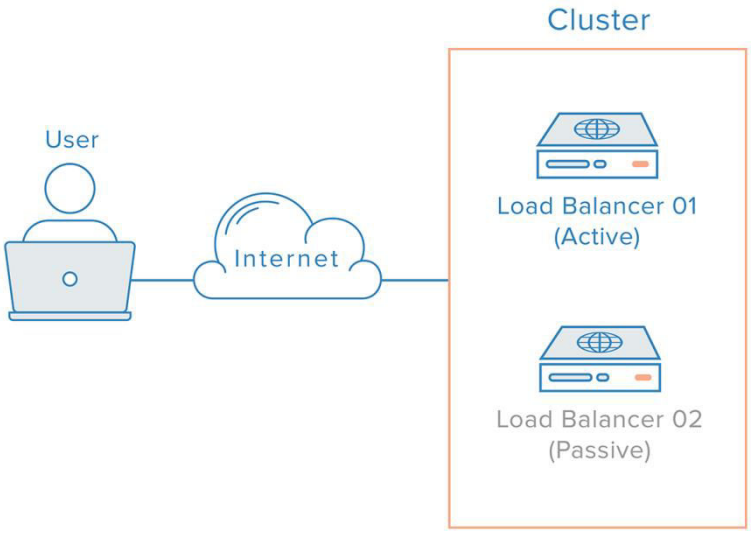

Рис.4.4 – Відмовостійка архітектура балансувальника навантаження[12]

Кожен з балансувальників відстежує стан іншого і в разі поломки одного здатен прийняти на себе частину навантаження. Відповідно можливо утворювати кластери більш ніж з двох серверів, збільшуючи надійність всієї системи.

Як зрозуміло з призначення балансування, сервери, що слугують балансувальниками, повинні передавати запити лише на «здорові» backend сервери, тобто ті, які здатні опрацьовувати трафік користувача. Для відстежування стану backend серверів регулярно з певним інтервалом проводяться перевірки. Використовуючи певний протокол та порт доступу, програмне забезпечення відправляє запити на сервери, очікуючи підтвердження. У разі, якщо відповідь невірна або взагалі не прийшла, балансувальник автоматично виключає цей сервер зі списку доступних. Трафік не буде відсилатися до серверу, допоки він не відповість на перевірку.

Найвідоміший відкритий програмний продукт, що дозволяє встановлювати та налаштовувати сервери для балансування навантаження – HAProxy.

HAProxy – це безкоштовне, дуже швидке і надійне рішення, що пропонує високу доступність і балансування навантаження для TCP і HTTP-додатків, за допомогою розподілу вхідних запитів на кілька обслуговуючих серверів. Програма написана на C і має репутацію швидкого, ефективного (в плані використання процесора і оперативної пам'яті) і стабільного рішення.

HAProxy використовується в ряді високо-навантажених веб-сайтів, включаючи Твіттер, Інстаграм Github, Stack Overflow, Reddit, Tumblr і OpsWorks product з Amazon Web Services, W3C (W3C Validator), а також є складовою частиною хмарної платформи Red Hat OpenShift і балансувальник по -умолчанію в хмарної платформі OpenStack. HAProxy є програмою з відкритим вихідним кодом і поширюється відповідно до GNU General Public License (GNU GPL v2) [9].

4.4.1 Алгоритми балансування навантаження

Алгоритми балансування навантаження використовуються для визначення одного зі здорових “backend” серверів, до якого буде перенаправлений запит.

Існують наступні алгоритми:

- “round robin”
- Алгоритм пошуку найменшої кількості підключень
- hash та ip-hash

Для демонстрування прикладів використаємо веб-сервер Nginx.

4.4.1.1 Round robin

Кожному бекенду виділяється окремий IP адрес. Веб-сервер за замовченням розподіляє запити рівномірно між бекендами. В цьому випадку різні клієнти будуть отримувати різні IP адреси при звертанні до сайту. Це дозволить розподілити навантаження між декількома серверами, які обслуговують веб-ресурс. Round robin визначає умову, при якій IP адреса будуть видаватися в певному порядку, не обов'язково послідовно. Такий метод є стандартним у більшості веб-браузерів, а тому використовується за замовченням. За даним принципом працює служба DNS (Domain Name System).

Використання даного методу також передбачає можливість призначення ваги серверам. Це визначатиме пріоритет опрацювання запитів. Застосування вагів може пригодитись у випадку, якщо, наприклад, певні сервери мають більш високі характеристики, порівняно з іншими у стеку. За замовченням всі ваги дорівнюють 1.

```
upstream backend {
        server backend1.somesite.com weight=10;
        server backend2.somesite.com weight=5;
        server backend3.somesite.com;
        server 192.0.0.1 backup;
}
```

В даному прикладі з 16 запитів перший бекенд буде опрацьовувати 10, другий 5, а третій – 1. При цьому backup сервер буде отримувати запити тільки в тому випадку, якщо три інших недоступні.

4.4.1.2 Алгоритм пошуку найменшої кількості підключень

При даному алгоритмові запити спочатку відправляються до серверу з найменшою кількістю активних підключень. Проте, якщо використовуються ваги, то пошук активних підключень відбувається з урахуванням пріоритету.

Приклад використання алгоритму пошуку найменшої кількості підключень:

```
upstream backend {
    least_conn;
    server backend1.somesite.com;
    server backend2.somesite.com;
}
```

У випадку якщо кількість активних підключень є однаковою, то додатково використовується алгоритм round robin.

4.4.1.3 Hash та IP hash

За допомогою цього методу можна створити свого роду постійні з’єднання між клієнтами та серверами.

Для кожного запиту сервер вираховує хеш, який складається з URL, змінних веб-серверу або їх комбінації, а потім зіставляє його з серврами.

Приклад використання алгоритму «hash»:

```
upstream backend {
    hash $scheme$request_uri;
    server backend1.somesite.com;
    server backend2.somesite.com;
    server backend3.somesite.com;
}
```

IP hash це різновид hash методу, в якому хеш вираховується за IP адресою користувача. Якщо адреса клієнта IPv4, то для хеша використовуються перші 3 октети, якщо IPv6, то вся адреса.

```
upstream backend {
        ip_hash;
        server backend1.somesite.com;
        server backend2.somesite.com;
        server backend3.somesite.com;
}
```

4.5 CDN

CDN (Content Delivery Network) – це спеціальна технологія, яка дозволяє відвідувачу отримувати контент сайту з різних географічних місць.

При великих відстанях сервера від користувача має місце затримка при передачі даних. Це призводить до того, що швидкість сайту буде відрізнятися для різних місцезнаходжень. Для рішення цієї проблеми і існує CDN. Принцип його роботи досить тривіальний. Для того, аби користувач отримав дані швидше, вміст сайту переноситься територіально ближче до нього. Тобто сервера добавляються в потрібних місцях і туди копіюється сайт.

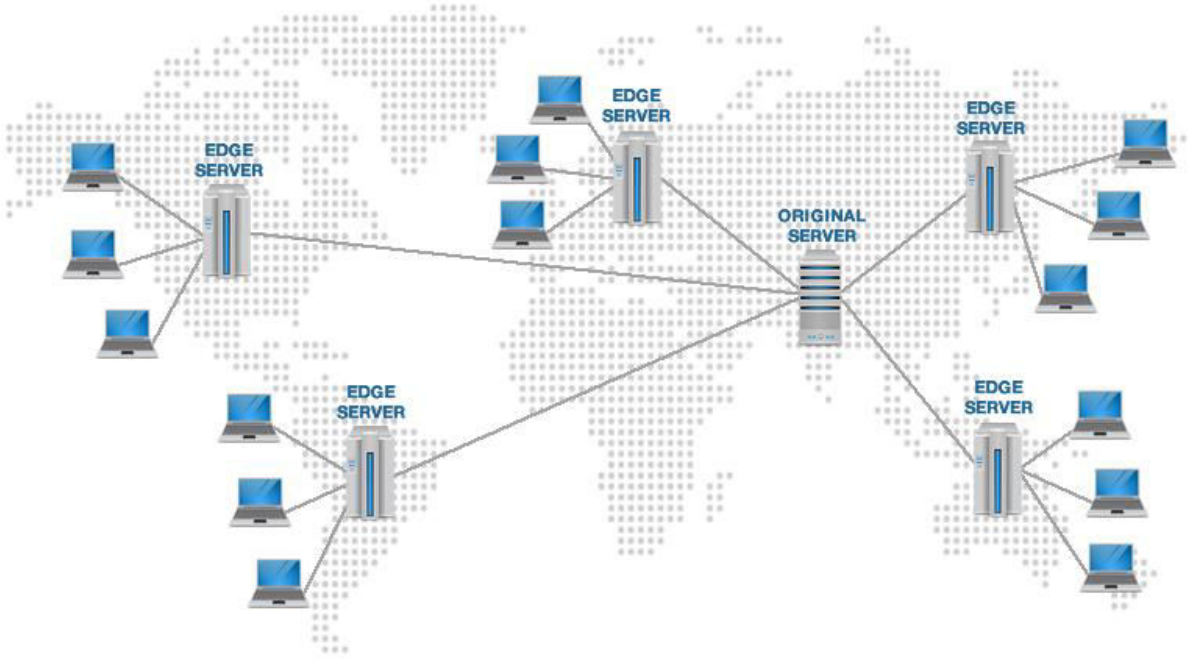

Рис.4.5 – Приклад роботи CDN [13]

CDN дозволяє зменшити навантаження на сайт, посилити захист від DDOS-у та пришвидшує завантаження сайту у віддалених регіонах. Працює дана технологія за принципом Anycast, – система, яка використовує мережеві адреса і методи маршрутизації для того, аби відправити дані на найближчий доступний вузол в межах групи, які використовують один і той же IP-адрес.

Проте, слід зауважити, що сенс використовувати даний метод є тільки у випадку, коли сайт розрахований на аудиторію, яка знаходиться на значній відстані від сервера (тисячі кілометрів).

Існує велика кількість сервісів, які надають послуги CDN. Необхідно тільки визначити перелік файлів, які будуть доступні з різних місць і передати цей набір файлів в систему. Найбільш популярні провайдери мереж доставки контенту: CloudFlare, MaxCDN.

4.6 Висновки до розділу 4

У даному розділі були розглянуті архітектурні підходи до покращення роботи веб-серверу, його модернізації, а саме: кешування, черги задач, балансування навантаження та використання CDN. Згідно розглянутих методів можна зробити висновок, що для веб-сервера обов’язково необхідно використовувати кешування, особливо кешування важких та найчастіше запитуємих запитів. Проте, у випадку, якщо дані на сервері часто змінюються даний метод не буде найкращим рішенням.

Крім того, у випадку, коли веб-сервер не може впоратися з вхідним навантаженням, навіть з налаштованими параметрами, – необхідно звернути увагу на асинхронні операції та використання черг повідомлень, які дозволять розподілити опрацювання запитів на декілька стадій: записування даних з багатьох потоків в буфер та зчитування і обробка цих даних у кілька потоків. Таким чином досягається найбільш раціональне використання ресурсів системи та впроваджується здатність до масштабування.

Для забезпечення також і надійності роботи веб-системи необхідно використовувати балансування, яке дозволить, створюючи резервні копії веб-серверів, розподіляти навантаження між ними за одним із алгоритмів: round robin, алгоритм пошуку найменшої кількості підключень, ip hash та різні модифікації перерахованих алгоритмів.

5 ТЕСТУВАННЯ НАВАНТАЖЕННЯ НА ВЕБ-СЕРВЕР

Для того, аби перевірити допустиме навантаження на веб-сервер застосовуються спеціальні програмні продукти, що симулюють потік користувачів (тобто відповідну кількість запитів) за допомогою синтетичних тестів. Ці тести дозволять зрозуміти орієнтовно-можливу кількість запитів, що здатна витримати веб-система. Прикладами таких тестів є Apache Benchmark, Httperf, Tsung та інші.

Для того, аби визначити слабкі місця вашої архітектури, проблеми з продуктивністю системи, необхідно застосувати інструменти для моніторингу і збору статистики.

Розглянемо конфігурацію популярних веб-серверів зі стандартними налаштуваннями. Для встановлення образів використаємо docker-контейнери. Запустити ці контейнери можна наступним чином:

docker run –d –p 80 –name name_of_container –v /path/to/content:/usr/share/nginx/html:ro –d nginx

Для розгортання контейнерів використовується робоча станція Ubuntu 14.04 LTS з 4 Гб оперативної пам’яті та 4 ядрами ЦПУ. Контейнери запускатимуться окремо один від одного задля максимально можливого використання ресурсів. Крім того, саме тестування відбуватиметься з іншого сервера з аналогічними характеристиками. В даному випадку можна знехтувати мережевими затримками, адже вони не впливатимуть на кінцевий результат порівняння отриманих систем.

В якості веб-сервера було обрано Nginx. Для чистоти експерименту в якості веб-ресурсу було використано просту html-сторінку, що містить в собі тільки статичний контент, тобто без обробки даних та пошуку в базі даних.

Запустимо контейнер зі стандартними налаштуваннями веб-серверу. Тестування проводиться з наступними характеристиками: 100000 запитів, які запускаються у 100 потоків, Apache Benchmark в якості тестової платформи. Моделювання графіків зроблено за допомогою gnuplot. Отримуємо наступні результати:

ab –n 100000 –c 100 http://10.12.42.225/

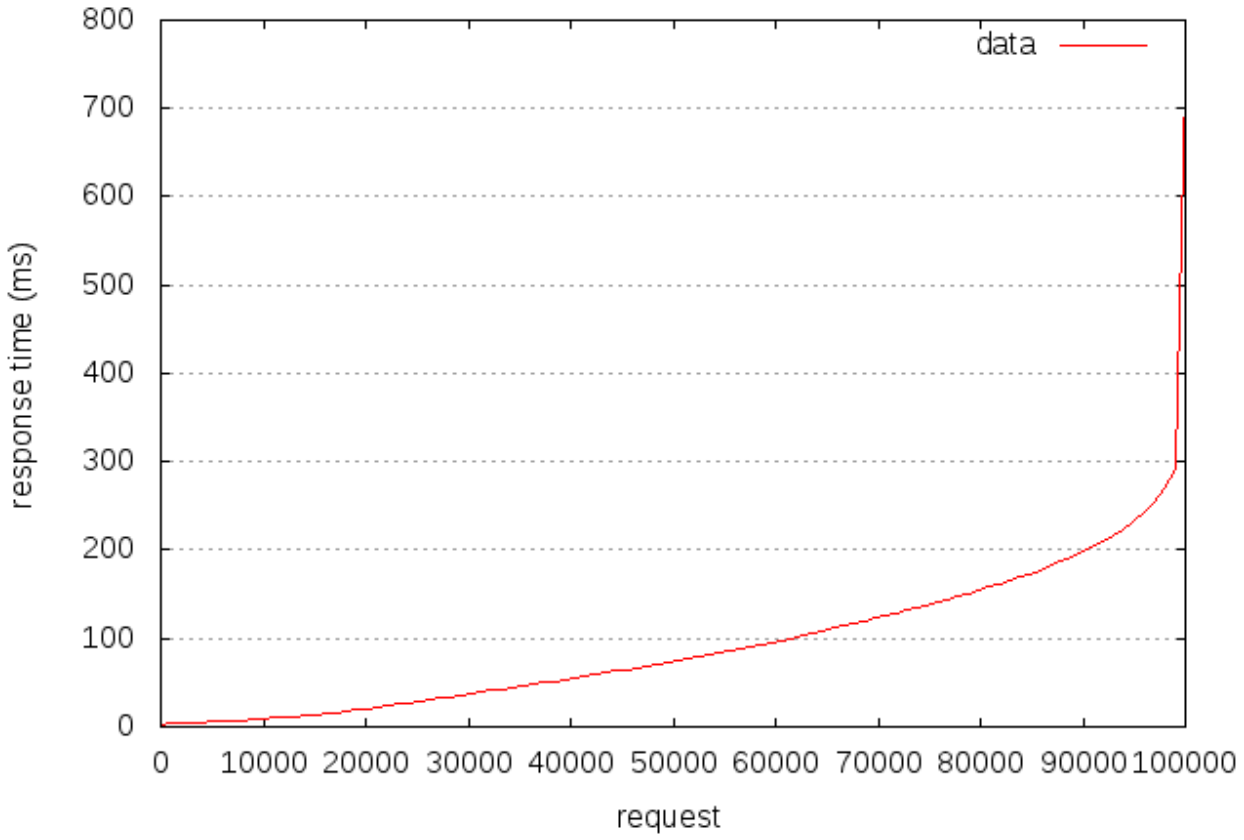

Рис.5.1 – Графік навантаження веб-сервера зі стандартними налаштуваннями

Дані результати були перевірені шляхом повторних тестів. Отримані дані збігаються з наведеними вище.

Тепер змінимо конфігурацію Nginx шляхом додавання pagespeed модуля. Він налічує в собі одразу декілька модифікацій, а саме: використання gzip для стискання запитів, мініфікація файлів та кешування. Отримуємо такі результати:

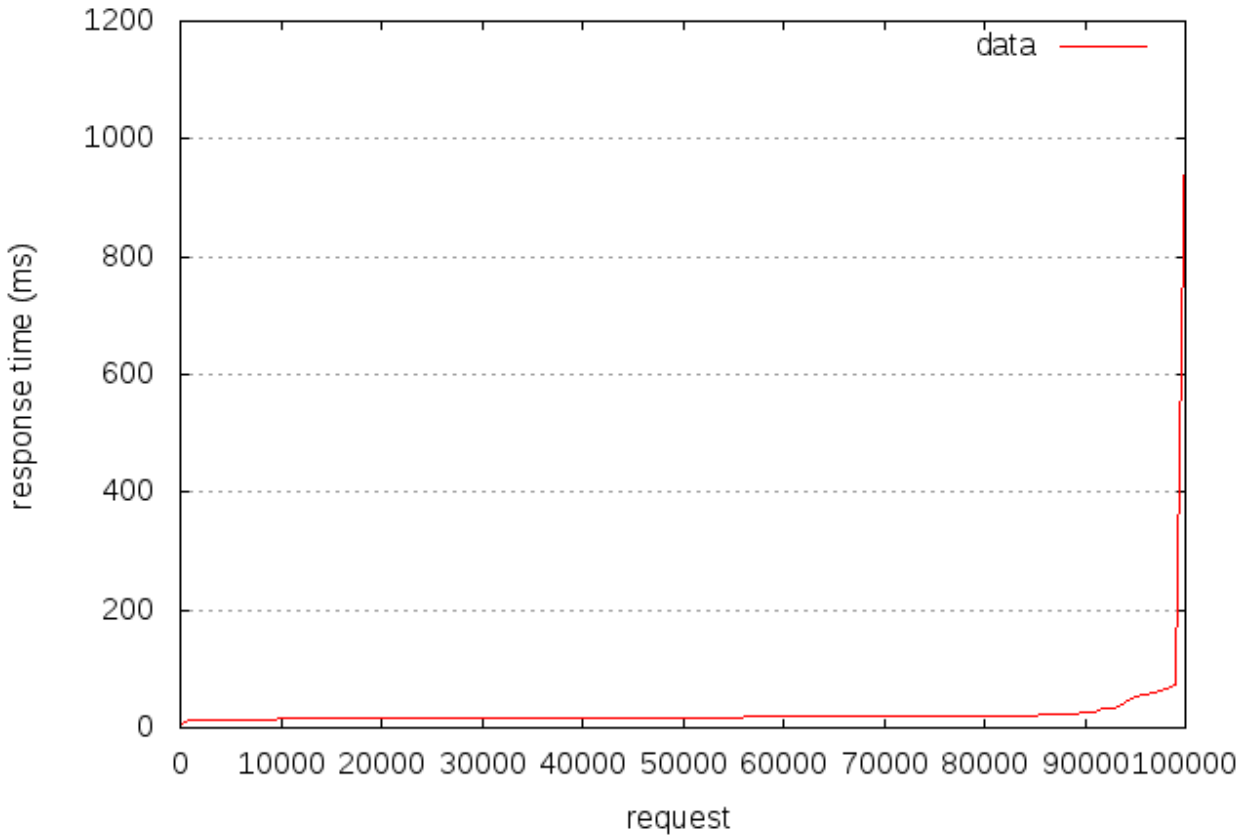

Рис.5.2 – Графік навантаження веб-сервера з використанням модуля pagespeed

Порівнюючи вихідні дані, бачимо, що результати виглядають доволі переконливо. При стандартному налаштуванні веб-сервер починає відчувати навантаження вже при 30000 одночасних запитів. Проте, при використанні кешування та стискання запитів ефективність веб-сервера збільшується більш як в 3 рази.

Розглянемо досвід Твіттера – популярної соціальної мережі.

Проект стартував з простого прототипу і досить швидко зіткнувся з великими труднощами при масштабуванні. Наразі сервіс обслуговує 150 мільйонів активних користувачів по всьому світові та успішно справляється з 300000 запитів в секунду.

Статистика на початку:

- проста веб-програма, побудована на фреймворці Ruby on Rails
- орієнтовно 350 000 користувачів
- 600 запитів в секунду
- запит оброблюється в середньому 50-100 мілісекунд
- серверний парк налічував всього 8 серверів

Але невдовзі сервіс почав швидко набирати популярність (рис.9). Їх тодішня архітектура не була розрахована на таке навантаження. Довелось абсолютно змінювати підхід до обробки даних.

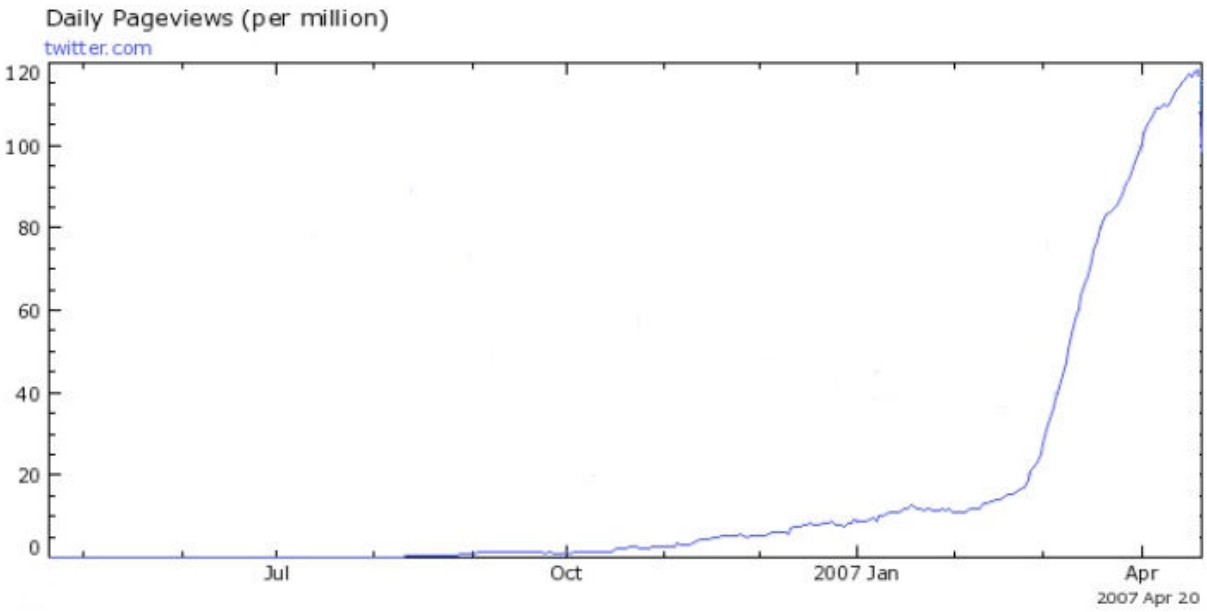

Рис.5.3 – Крива росту популярності Твіттера

Що було зроблено?

- кешування даних
- впровадження черг як основної системи Твіттера
- використання CDN для зображень (планується і відео з впровадженням його у сервіс)
- моніторинг всього, що тільки можливо

Наразі соціальна мережа Твіттер справляється з навантаженням в 150 мільйонів постів щодня [11]. Як наслідок команда Твіттеру дійшла висновку, що більшість проблем з продуктивністю пов’язано не з мовою програмування, а з архітектурою.

5.1 Висновки до розділу 5

В розділі було проведено тестування навантаження на веб-сервер. Проведено порівняльний аналіз допустимого навантаження на сервер зі стандартною конфігурацією та сервер, що налаштований за допомогою додаткового модуля pagespeed. В результаті отримуємо, що впровадження додаткового модулю здатне підвищити продуктивність веб-сервера більш як втричі, що є чудовим результатом та задовольняє поставлені вимоги.

При зростанні навантаження необхідно змінювати архітектуру системи, принцип її роботи. Дане твердження продемонстровано на прикладі досвіду всесвітньої компанії Твіттер, яка, змінивши спосіб обробки даних, досягла приросту допустимого опрацювання запитів в 500 разів.

6 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик архітектурних підходів веб-системи, призначених для оптимізації її роботи.

Нижче наведено аналіз різних варіантів реалізації з метою вибору оптимальної, з огляду як на економічні фактори, так і на характеристики системи, що впливають на продуктивність роботи і на її сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій. Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

6.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи, призначеної для оптимізації роботи веб-серверів в умовах високого навантаження. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу гетероскедастичних процесів в економіці та фінансах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- продукт має функціонувати на серверах під управлінням операційних систем Linux/Unix;
- забезпечувати високу швидкість обробки великих об’ємів даних в реальному часі;
- передбачати мінімальні витрати на впровадження програмного продукту.

6.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який налаштовує веб-сервери для середовищ з високим навантаженням. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір операційної системи, на яку буде встановлюватись програмне забезпечення;

F_2 – вибір веб-серверу, який буде обслуговувати систему;

F_3 – вибір архітектурного підходу для реалізації веб-системи.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) ОС Debian GNU/Linux;

б) ОС FreeBSD;

Функція F_2:

а) Apache веб-сервер;

б) Nginx веб-сервер;

Функція F_3:

а) конфігурація внутрішніх параметрів системи;

б) зміна архітектури системи шляхом впровадження нових елементів;

6.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 6.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 6.1).

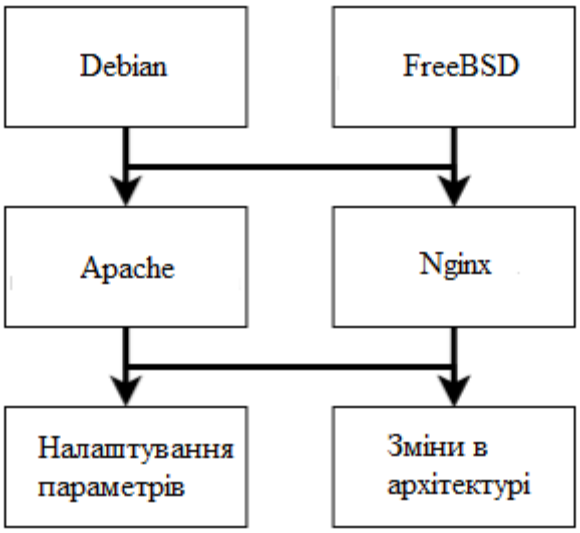

Рис.6.1 – Морфологічна карта

Морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП. На основі аналізу позитивно-негативної матриці робимо висновок, що при обрані програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Таблиця 6.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F_1	А	Простота в керуванні системою	Може де в чому програвати досвідченішим системам
	Б	система має більш універсальне застосування	Складна в керуванні
F_2	А	Використовує динамічні модулі, що полегшують роботу з продуктом	За своєю архітектурою не розрахований на велику кількість одночасних підключень
	Б	Здатний витримувати більшу кількість одночасних підключень	Не має підтримки динамічних модулів
F_3	А	Легкість у налаштуванні	Незначний виграш в продуктивності
	Б	Помітне підвищення продуктивності, забезпечення безвідмовності	Складність в керуванні

Функція F_1:

Оскільки для конфігурації системи при виборі операційної системи слід орієнтуватись на зручність у керуванні, варіант б) має бути відкинутий.

Функція F_2:

Оскільки в даному дослідження пріоритетом являється саме продуктивність системи в умовах високого навантаження, варіант а) має бути відкинутий.

Функція F_3:

Так як обидва варіанти мають як сильні так і слабкі сторону, тому обидва варіанти а) і б) гідні розгляду.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2б – F3а
2. F1а – F2б – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

6.2 Обґрунтування системи параметрів ПП

6.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня. Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – час відповіді від сервера на запит клієнта;

– *X2* – кількість одночасних запитів до системи;

– *X3* – об’єм пам’яті, що використовується;

– *X4* – навантаження на процесор.

*X1:*Відображає час, що необхідний серверу для обробки запиту клієнту та відправлення відповіді.

*X2:*Відображає кількість одночасних запитів, що здатний опрацьовувати сервер.

*X3:*Показує кількість оперативної пам’яті, що необхідні для стабільної роботи серверу.

X4: Показує максимально допустиме навантаження процесора для стабільної роботи серверу.

6.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 6.2.

Таблиця 6.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Час відповіді від сервера	X1	с	20	15	10
Кількість одночасних запитів	X2	шт.	1000	5000	10000
Об’єм пам’яті	X3	Мб	32	16	8
Навантаження на процесор	X4	%	90	70	50

За даними таблиці 6.2 будуються графічні характеристики параметрів – рис. 6.2 – рис. 6.5.

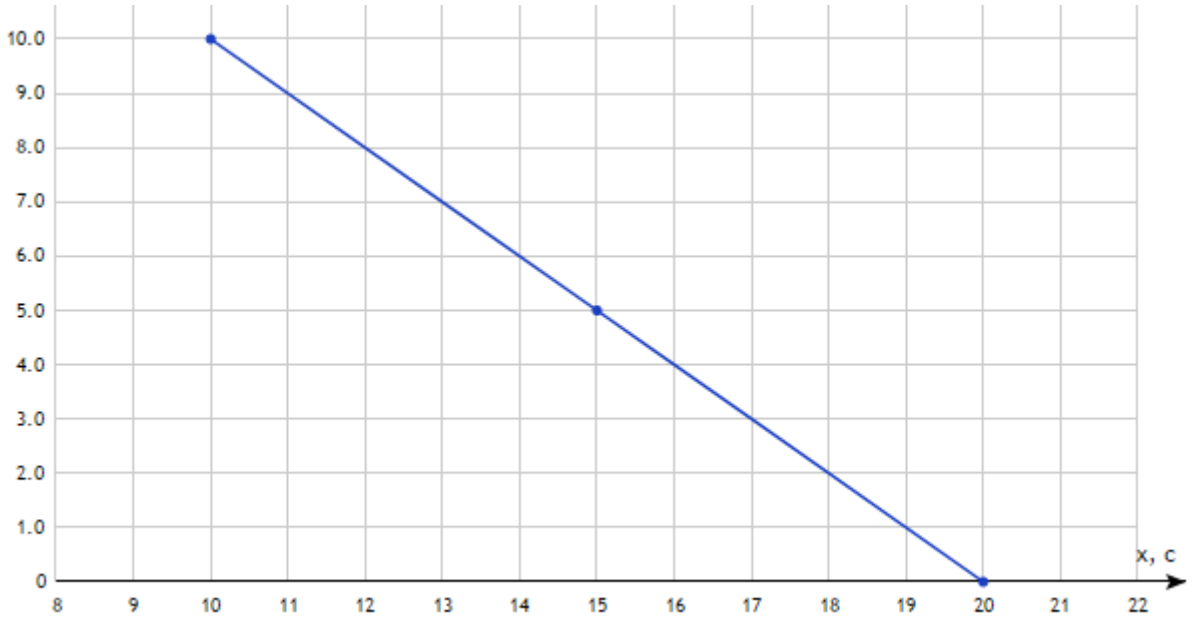

Рисунок 6.2 – X1, Час відповіді від сервера на запит клієнта

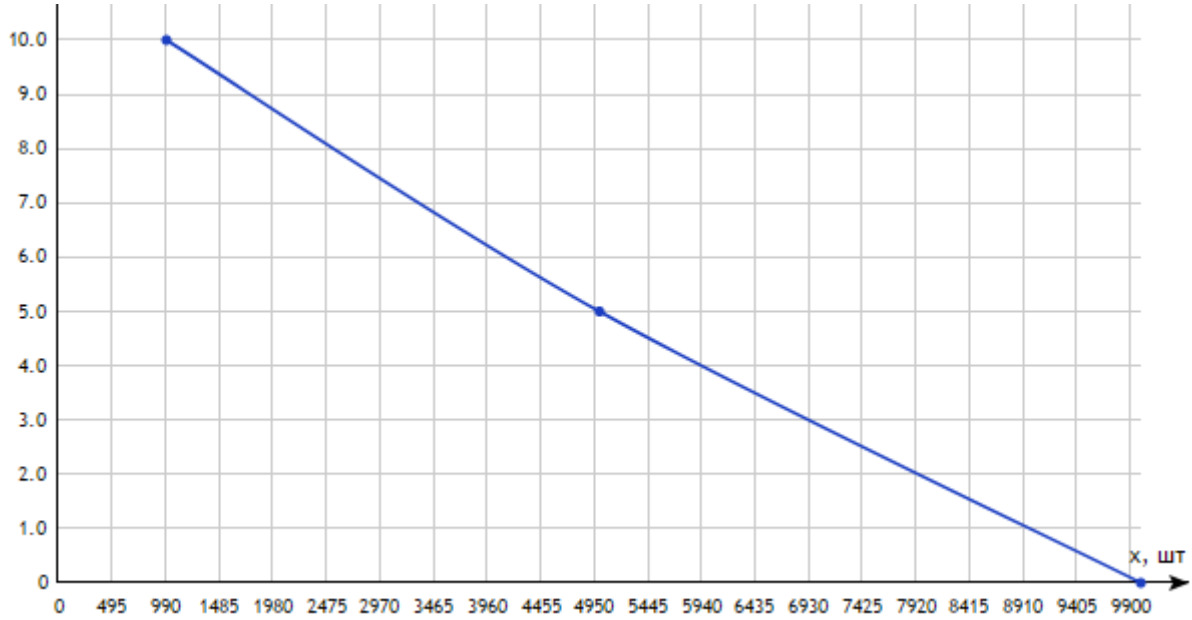

Рисунок 6.3 – X2, Кількість одночасних запитів до системи

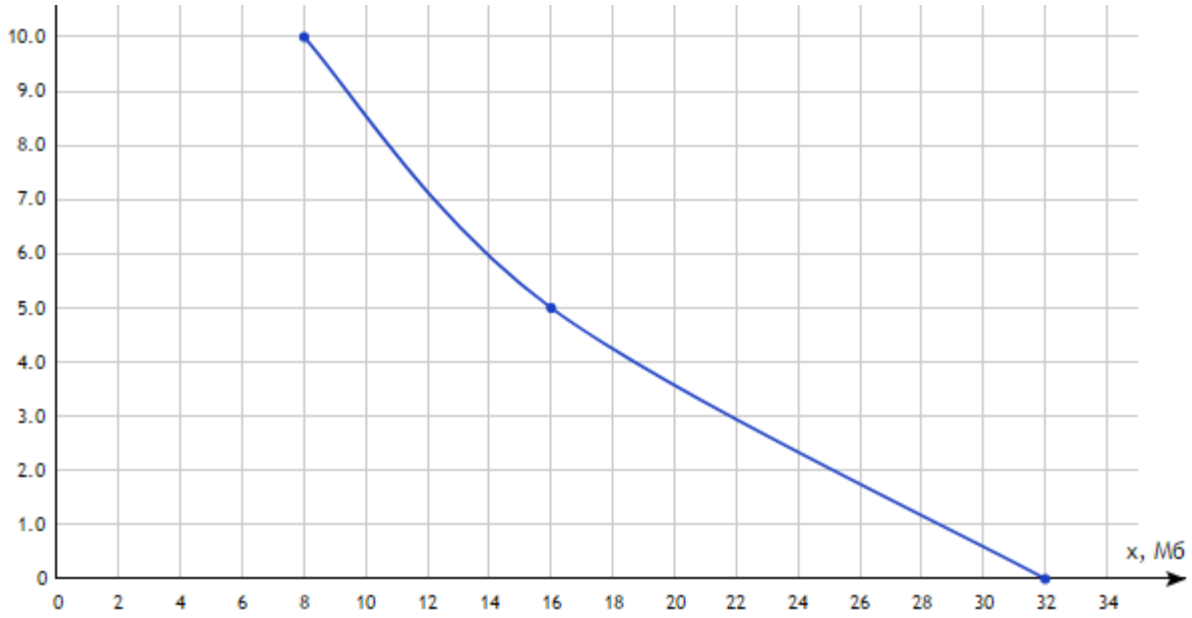

Рисунок 6.4 – X3, Об’єм пам’яті, що використовується

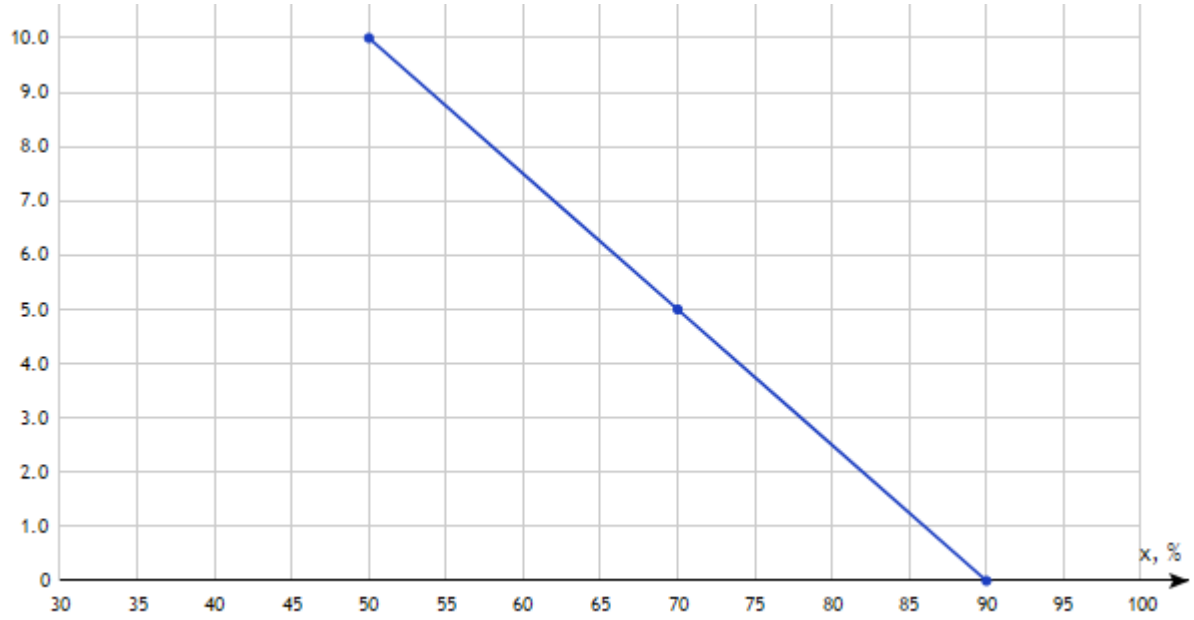

Рисунок 6.5 – X4, Навантаження на процесор

6.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при

знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 6.3.

Таблиця 6.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів Ri	Відхилення Δi	Δi2
			1	2	3	4	5	6	7			
X1	Час відповіді від сервера	с	2	1	1	2	2	1	2	11	-6,5	42,25
X2	Кількість одночасних запитів	шт.	3	4	2	4	3	4	4	24	6,5	42,25
X3	Об’єм пам’яті	Мб	4	3	4	3	4	3	3	24	6,5	42,25
X4	Навантаження на процесор	%	1	2	3	1	1	2	1	11	-6,5	42,25
	Разом		10	10	10	10	10	10	10	70	0	169

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$\boldsymbol{S = \sum_{i=1}^{N} \Delta i^2 = 169}$$

Порахуємо коефіцієнт узгодженості:

$$\boldsymbol{W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 169}{7^2(4^3 - 4)} = 0,69 > W_k = 0,67}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6.4.

Таблиця 6.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	>	<	<	>	>	<	>	>	1.5
X2 і X3	<	>	<	>	<	>	>	>	1.5
X2 і X4	>	>	<	>	>	>	>	>	1.5
X3 і X4	>	>	>	>	>	>	>	>	1.5

Числове значення, що визначає ступінь переваги і–го параметра над j–тим, аijвизначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю А=‖ aij ‖.

Для кожного параметра зробимо розрахунок вагомості Кві за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 6.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 6.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X1	1,0	0,5	0,5	1,5	3,5	0,219	12,25	0,208	44,875	0,207
X2	1,5	1,0	1,5	1,5	5,5	0,344	21,25	0,360	77,875	0,361
X3	1,5	0,5	1,0	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X4	0,5	0,5	0,5	1,0	2,5	0,156	9,25	0,157	34,125	0,158
Всього:					16	1	59	1	216	1

6.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2(Кількість одночасних запитів до системи) та X3 (Об‘єм пам‘яті, що використовується) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1(Час відповіді від сервера на запит клієнта) буде найкращим у випадку обрання у F3 варіанта Б і становитиме 12, для варіанту А це значення буде 18.

Абсолютне значення параметра X4 (навантаження на процесор) буде найкраще при обрані варіанту А 55, а при обрані варіанту Б воно буде 70.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 6.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j},$$

де n – кількість параметрів; в – коефіцієнт вагомості і–го параметра;Bi – оцінка і–го параметра в балах.

Таблиця 6.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне забезпечення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X3	16	5	0,274	1,37
F2	Б	X2	10000	5	0,361	1,805
F3	А	X1	18	2	0,207	0,414
		X4	55	9	0,158	1,422
	Б	X1	12	9	0,207	1,863
		X4	70	5	0,158	0,79

За даними з таблиці 6.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 1,37 + 1,805 + 0,414 + 1,422 = 5,011$

$K_{K2} = 1,37 + 1,805 + 1,863 + 0,79 = 5,828$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

6.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості. Програмний продукт, що буде встановлюватись на сервер кафедри вже розроблено, проте необхідно увесь рік тримати під контролем сервер, на якому буде встановлено програмне забезпечення. Отже

Т1 = 365 людино-днів на рік.

ТІ = 365 * 8 = 2920 людино-годин;

В адмініструванні беруть участь два адміністратори з окладом 8000 грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{M}{T_m * t} \text{ грн.},$$

де М – місячний оклад працівників; – кількість робочих днів тиждень; – кількість робочих годин в день.

$$С_Ч = \frac{2 * 8000}{2 * 21 * 8} = 47{,}62 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$С_{ЗП} = С_Ч * Т_і * К_Д$,

де $С_Ч$– величина погодинної оплати праці програміста; – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$С_{ЗП}$ = 47,62 * 2920 * 1,2 = 166880,48 грн. на рік

Відрахування на єдиний соціальний внесок незалежно від групи професійного ризику становить 22%:

$С_{ВІД} = С_{ЗП}$ * 0,22 = 166880,48 * 0,22 = 36709,31 грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного адміністратора з окладом 8000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г$ = 12 * М * $К_З$ = 12 * 8000 * 0,2 = 19200 грн

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г * (1+ К_З) = 19200 * 1,2 = 23040$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} * 0,22 = 23040 * 0,22 = 5068,80$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} * К_А * Ц_{ПР} = 1,15 * 0,25 * 10000 = 2875$ грн.

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} * К_{ТМ} * К_Р = 1,15 * 10000 * 0,05 = 575$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин

де $Д_К$ – календарна кількість днів у році; ДВ,

$Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot 1.93819 =1706,4 \cdot 0,156 \cdot 1.93819= 515,94$ грн.,

де NC – середньо-споживча потужність приладу;

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 =6700$ грн

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} =С_{ЗП}+ С_{ВІД}+ С_А + С_Р+ С_{ЕЛ} + С_Н$

$С_{ЕКС} = 23040 + 5068,80 + 2875 + 575 + 515,94 + 6700 = 38774,74$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 38774,74 / 1706,4 = 22,72$ грн/год

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т$$

$С_М = 22,72 * 1706.4 = 38769,41$грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

$С_Н = 166880,48 * 0,67 = 111809,92$ грн.

Отже, вартість розробки ПП становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

$С_{ПП} = 166880,48 + 36709,31 + 38769,41 + 111809,92 = 354169,12$ грн.

6.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = \frac{К_{Кj}}{С_{Фj}},$$

$К_{ТЕР1} = 5,011 / 354169,12 = 1,41 * 10^{-5}$

$К_{ТЕР2} = 5,828 / 354169,12 = 1,65 * 10^{-5}$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 1,65 * 10^{-5}$.

6.6 Висновки до розділу 6

В даному розділі проведено повний функціонально-вартісний аналіз програмного забезпечення, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

У першій проведено дослідження програмний продукт з технічної точки зору: були поставлені основні параметри, що повинні бути головними при обранні кращої реалізації. На основі отриманих значень параметрів, оцінок експертів було обчислено коефіцієнт технічного рівня, який надалі у другій частині допоміг обрати найкращий варіант з техніко-економічної точки зору.

У другій частині виконувалося економічне обґрунтування альтернативних варіантів реалізації. Порівняння робились з урахуванням витрат на заробітні плати, електроенергії, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється. Після проведення першої частини аналізу було виявлено, що перший варіант є найбільш оптимальним для реалізації. Його показник техніко-економічного рівня якості $К_{ТЕР2} = 1,65 * 10^{-5}$;

Даний варіант виконання програмного комплексу є простим у написанні, достатньо швидким для користувача та гнучким у налаштованні та масштабуванні.

ВИСНОВКИ

В результаті виконання даної роботи було здійснено аналіз методів налаштувань веб-серверів у високонавантажених системах. Тема конфігурації веб-серверів є досить цікавою і представляє собою широке поле для подальших досліджень.

У роботі висвітлені основні вимоги до високонавнтаженої системи та теоретична база кожного з методів підвищення продуктивності веб-серверів, а саме: використання gzip та http/2, мініфікація файлів, використання модулю Google PageSpeed, кешування, черги задач, балансування навантаження, використання CDN. Також показані можливі значення приросту продуктивності при відтворенні даних процедур на реальних об’єктах.

Результати показали, що використання простого pagespeed модулю здатне підвищити продуктивність веб-сервера більш як в 3 рази. Саме використання pagespeed виявилось найбільш вдалим способом конфігурації внутрішніх параметрів веб-сервера, оскільки він об’єднує в собі декілька налаштувань, включаючи gzip та мініфікацію файлів. У разі збільшення кількості одночасних запитів до 90000 варто змінювати архітектуру веб-системи, переходячи на впровадження черг, балансування та обов’язково кешування. Якщо ж система використовується користувачами з різних куточків планети, ще варто замислитись і про застосування CDN.

Крім того, було продемонстровано на прикладі досвіду компанії Twitter, що за допомогою кешування, використання черг задач та CDN можливо підвищити початкову допустиму здатність веб-сервера обробляти одночасні запити в 500 разів. Тобто можна зробити висновок, що для збільшення показників продуктивності веб-серверів в реальних навантажених проектах слід переходити від простого налаштування параметрів до модернізації архітектурної складової системи.

Всі програмні продукти, використані в роботі, являються загальнодоступними та відкритими для користування під ліцензіями BSD та Apache License 2.0, що передбачає широке використання наведених методів.

Отже, внаслідок проведеного дослідження стало зрозуміло, що потрібно зосереджувати увагу на розширенні веб-інфраструктури за рахунок розподілу на взаємопов’язані частини, що складатимуть єдину систему, тим самим звільняючись від проблеми масштабування та нестачі ресурсів. Таким чином досягаються кращі результати у надійності системи та її працеспроможності. Сучасні веб-системи орієнтовані на широку аудиторію користувачів, тому використання даного підходу матиме великий попит та можливості для розвитку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Приклад роботи веб-сервера. – Режим доступу: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRwSUz7rgCu0YsVJlaAP5eUHoCTbP2t8s9trDtxa-86R52Wr_w7. Дата доступа: 03.05.2017
2. Мультиплексування в HTTP/2. – Режим доступу: https://blog.planethoster.net/wp-content/uploads/2015/11/HTTP2-explication.jpg. Дата доступу: 14.05.2017
3. DigitalOcean: What is Load Balancing? – Режим доступу: https://www.digitalocean.com/community/tutorials/what-is-load-balancing. Дата доступу: 14.05.2017
4. DigitalOcean: Introduction to DigitalOcean Load Balancers. – Режим доступу: https://www.digitalocean.com/community/tutorials/an-introduction-to-digitalocean-load-balancers. Дата доступу: 14.05.2017
5. Netcraft: February 2017 Web Server Survey. – Режим доступу: https://news.netcraft.com/archives/2017/02/27/february-2017-web-server-survey.html . Дата доступу: 10.05.2017
6. Website Gzip Checker. – Режим доступу: http://highloadtools.com/gzip . Дата доступу: 27.05.2017
7. Офіційний сайт компанії W3Techs. – Режим доступу: https://w3techs.com/. Дата доступу: 27.05.2017
8. HTTPS on Stack Overflow: The End of a Long Road. – Режим доступу: https://nickcraver.com/blog/2017/05/22/https-on-stack-overflow/?utm_source=telegram.me&utm_medium=social&utm_campaign=vchera-stack-overflow-vklyuchili-https-po-d#performance-http2. Дата доступу: 18.04.2017
9. Офіційний сайт програмного продукту HAProxy. – Режим доступу: http://www.haproxy.org/. Дата доступу: 28.05.2017

10. Http Archive: Interesting Stats – Режим доступу: http://httparchive.org/interesting.php. Дата доступу: 28.05.2017
11. Схема роботи балансувальника навантаження. – Режим доступу: https://assets.digitalocean.com/articles/high-availability/Diagram_2.png. Дата доступу: 28.05.2017
12. Відмовостійка архітектура балансувальника навантаження. – Режим доступу:https://assets.digitalocean.com/articles/HAProxy/layer_4_load_balancing.png. Дата доступу: 28.05.2017
13. Приклад роботи CDN. – Режим доступу: https://creativeboro.com/wp-content/uploads/2017/02/how-cdn-works-300x188.png. Дата доступу: 01.06.2017
14. High Scalabilty: Twitter by the numbers. – Режим доступу: http://highscalability.com/blog/2011/3/14/twitter-by-the-numbers-460000-new-accounts-and-140-million-t.html. Дата доступу: 14.05.2017
15. ITEduCenter: Apache или Nginx? – Режим доступу: https://blog.iteducenter.ua/sysadmin-basics/apache-ili-nginx-ili-apache-i-nginx/. Дата доступу: 18.05.2017
16. Оптимізація і масштабування Web програм. – Режим доступу: https://ruhighload.com/. Дата доступу: 03.05.2017
17. Habrahabr: «Стратегия кеширования в приложении»– Режим доступу: https://habrahabr.ru/post/168725/. Дата доступу: 10.05.2017