МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломної роботи

магістра
(освітній ступінь)

на тему: «Експериментальне дослідження методів балансування навантаження HighLoad-систем»

ХАІ.603.657П1.121.156313.20О

Виконав студент 6 курсу групи 657П1
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)

Левченко А.В.
(Прізвище та ініціали студента)
Керівник: Кузнецова Ю.А.
(Прізвище та ініціали)
Рецензент: Барковська О.Ю.
(Прізвище та ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)
Кафедра інженерії програмного забезпечення
(повне найменування)
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)
Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (ініціали та прізвище)
“_____” _____ 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Левченку Анатолію Васильовичу
(прізвище, ім’я, по батькові)

1. Тема дипломної роботи Експериментальне дослідження методів балансування навантаження HighLoad-систем
керівник дипломної роботи Кузнецова Юлія Анатоліївна, к.т.н, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Університету № _____ від “____” _____ 2020 року
2. Термін подання студентом роботи _____
3. Вихідні дані до роботи: результати огляду та аналізу проблем високонавантажених комп’ютерних систем
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити): огляд та аналіз проблем високонавантажених комп’ютерних систем, планування експерименту для досліджень розвантаження високонавантажених систем, аналіз результатів експерименту для досліджень розвантаження високонавантажених систем та формування практичних рекомендацій
5. Перелік графічного матеріалу: пояснювальна записка - 82 сторінки, 23 рисунка, 37 джерел, 5 таблиць, 1 додаток, 21 слайд презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Кузнецова Ю. А., доцент каф. 603		
2	Кузнецова Ю. А., доцент каф. 603		
3	Кузнецова Ю. А., доцент каф. 603		

Нормоконтроль _____ _____ _____ «___»_____2020_р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання «___9___»___вересня___2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і затвердження теми диплому	09.09.19	
2	Огляд та аналіз проблем високонавантажених комп'ютерних систем	10.09.19 - 20.12.19	
3	Планування експерименту для досліджень розвантаження високонавантажених систем	15.02.20 - 26.05.20	
4	Аналіз результатів експерименту для досліджень розвантаження високонавантажених систем та формування практичних рекомендацій	02.09.20 - 23.10.20	
5	Наповнення контентом розділів пояснювальної записки	24.10.20 - 10.11.20	
6	Форматування пояснювальної записки до дипломного проекту відповідно до правил та вимог оформлення письмової роботи Національного аерокосмічного університету ім. М.Є. Жуковського "ХАІ"	11.11.20 - 24.11.20	
7	Передзахист дипломного проекту	25.11.20	
8	Захист дипломного проекту	04.12.20	

Студент _____ Левченко А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Кузнецова Ю. А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота на тему «Експериментальне дослідження методів балансування навантаження HighLoad-систем»: 82 сторінки, 23 рисунка, 37 джерело, 5 таблиць.

Об'єкт дослідження – процеси, що протікають у високонавантажених системах типу Google, Facebook, YouTube.

Предмет дослідження – Використання методів статистичного аналізу, а саме методів планування експерименту для порівняльного аналізу алгоритмів балансування навантаження Round Robin та Least Connection.

Мета – аналіз ефективності високонавантажених систем на прикладі використання методів балансування навантаження.

Мета роботи полягає в використанні методів: статистичного аналізу, планування експерименту для аналізу ефективності високонавантажених систем.

На етапі аналізу проблеми проаналізовані високонавантажені системи: розглянуті відмінні риси «high load» систем, проаналізовано проблеми високонавантажених систем, розглянуті основні концепції «high load» систем, розглянуті основні технологічні рішення таких високонавантажених систем, як Google, Facebook і YouTube.

Актуальність даної роботи полягає в тому, що зростає значимість інформаційних технологій в сучасному світі, відповідно зростає чисельність відвідувачів веб-додатків і питання забезпечення відмово стійкості при високих навантаженнях встає на передній план. Таким чином, на сьогоднішній день стоїть завдання розробки оптимального програмного коду, організації баз даних, удосконалення апаратних ресурсів для підвищення швидкодії систем.

ВИСОКОНАВАНТАЖЕНІ СИСТЕМИ, КОНЦЕПЦІЇ HIGHLOAD, ПЛАТФОРМИ GOOGLE, FACEBOOK, YOUTUBE

ABSTRACT

Course thesis on the topic “Study of software features of high-load systems”: 82 pages, 23 figures, 37 sources.

The object of the research is high-load systems such as Google, Facebook and YouTube.

The study subject is technologies for developing high-load systems using examples of Google, Facebook and YouTube.

The goal is to analyze the efficiency of high-load systems using examples of Google, Facebook and YouTube.

The purpose of the work is based on the use of methods: statistical analysis, experimental planning methods for analyzing the effectiveness of high-load systems.

At the problem analysis stage, high-load systems were analyzed: the distinctive features of high-load systems were considered, the problems of high-load systems were analyzed, the basic concepts of high-load systems were considered, the main technological solutions of high-load systems such as Google, Facebook and YouTube were considered.

The relevance of this work is that the importance of information technologies in the modern world increases, the number of visitors of web applications is increasing accordingly, and the issue of ensuring resiliency under high loads comes to the fore. So, today there is still and issue with of developing an optimal software, organizing databases, improving hardware resources to improve the speed of systems.

HIGH-LOAD SYSTEMS, HIGH-LOAD CONCEPTS, GOOGLE PLATFORM, FACEBOOK, YOUTUBE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API – інтерфейс створення додатку [1].

IMDG – InMemory Data Grid – це розподілене сховище об'єктів, схоже по інтерфейсу зі звичайною багато потокової хеш-таблицею [2].

GFS – Google File System – розподілена файлова система [3].

HighLoad – високонавантажених система.

HTML – Hyper Text Markup Language – мова гіпертекстової розмітки.

Nginx – веб-сервер і поштовий проксі-сервер, що працює на Unix-подібних операційних системах [4].

NoSQL – not only SQL – ряд підходів, спрямованих на реалізацію сховищ баз даних [5].

БД – база даних.

СУБД – система управління базами даних.

ІС – інформаційна система

ВСТУП

Щороку кількість інтернет-користувачів неухильно зростає. Високонавантажені ресурси в першу чергу розраховані на багато користувачів додатків, більшість з яких розподілені системи, які працюють на більш ніж одному сервері. Ця конфігурація необхідна для ввімкнення обробки великих обсягів даних, створених під час пікових навантажень, а також його реплікації. Друге найважливіше завдання, описане в конфігурації застосунку ресурсів, полягає в забезпеченні стійкості системи .

В даний час немає добре налагодженого визначення дуже завантаженої системи, але деякі функції, які присутні в таких системах, можуть бути ідентифіковані з існуючої літератури. Як правило, високонавантажені системи реактивні це означає, що вони не просто повинні робити розрахунки, але і , якщо в такій системі щось виходить зі строю, користувачі не повинні цього відчувати. Навантаження в таких системах зазвичай починаються з 1000 RPS / QPS вони повинні швидко реагувати на прохання [7].

Високонавантажені ресурси в першу чергу розраховані на багато користувачів додатків, більшість з яких розподілені системи, які працюють на більш ніж одному сервері. Таким чином, дуже завантажені системи є безупинними системами доступу, тобто тими структурами, чиї дані запити дозволяють отримувати інформацію без тривалого переривання роботи при постійній роботі.

Актуальність цієї роботи полягає в тому, що важливість інформаційної технології в сучасному світі зростає, кількість відвідувачів веб-додатків зростає, і питання забезпечення стійкості при високих навантажень виходить на перший план. Таким чином, сьогодні завдання полягає у розробці оптимального коду, організації баз даних, поліпшенню апаратних ресурсів для підвищення швидкості роботи систем.

Мета дипломної роботи – аналіз ефективності високонавантажених систем на прикладі використання методів балансування навантаження.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1 Проаналізувати особливості та проблеми при розробці високонавантажених систем на прикладах Google, Facebook и YouTube.

2 Спланувати експериментальне дослідження для оцінки ефективності роботи високонавантажених систем.

3 Провести експериментальне дослідження і проаналізувати результати експерименту.

4 Розробити практичні рекомендації для вирішення задачі вибору методу балансувальника навантаження.

Об'єкт дослідження – процеси, що протікають у високонавантажених системах типу Google, Facebook, YouTube.

Предмет дослідження – технології для оптимізації високонавантажених систем.

Методи дослідження – використання методів статистичного аналізу, а саме методів планування експерименту для порівняльного аналізу алгоритмів балансування навантаження Round Robin та Least Connection.

Наукова новизна дослідження. Дістали подальший розвиток методи оптимізації навантаження на сервер за рахунок використання балансувальника навантаження.

Практичне значення отриманих результатів полягає у тому, що рекомендації, отримані в результаті проведеного експериментального дослідження, можуть бути використані розробниками високонавантажених систем для зменшення навантаження на веб-сервер, що дозволить підвищити доступність серверу.

1 ОГЛЯД ТА АНАЛІЗ ПРОБЛЕМ ВИСОКОНАВАНТАЖЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ

1.1 Відмінні риси високонавантажених систем «High load»

Так склалося, що в нашій країні термін «high load» використовують в основному щодо веб-сервісів, тобто розрахованих на багато користувачів інтернет-сайтів. Але це тільки частина високонавантажених систем, до них ще належать і різні керуючі, і бізнес-додатки.

«High load» – це додаток з високим навантаженням, яка виникає з наступних причин:

– великої кількості одночасних користувачів;

– великий обсяг оброблюваних даних;

– наявність численних складних розрахунків і обчислень в бізнес-додатках.

Всі перераховані вище фактори як разом, так і окремо характеризують високонавантажених додаток. Для роботи такої системи потрібно багато ресурсів.

Щоб зрозуміти, чим «high load» додатки відрізняються від звичайних, зупинимося докладніше на їх особливості:

– жорсткість. Високонавантажених додаток це жорстка система, в якій передбачені варіанти зміни лише деяких частин. Таку систему неможливо зробити гнучкою. Складно зробити універсальним все, що пов'язано з доступом до даних. Щоб система функціонувала стабільно, потрібно чітко розуміти, з якою базою даних вона буде працювати та використовувати при реалізації проекту переваги цієї БД, виходячи з обсягів даних і частоти звернень. Якщо в високонавантаженої системі спробувати щось зробити гнучким потрібно невиправдано велика кількість ресурсів.

– Швидкий час відгуку**.** Це важлива якість «high load»-додатків. Спілкування користувача з системою відбувається через запит, і відповідь на нього повинен приходити якщо не миттєво, то через прийнятний час.

– Масштабування**.** Як багато людей можуть одночасно користуватися ресурсом, даними, що здатна вмістити база до того, як перестане справлятися з навантаженням? Високонавантажений проект обов'язково повинен бути масштабованим. Існують два шляхи масштабування:

– вертикальне – збільшення продуктивності окремих компонентів системи для покращення загальної продуктивності. Досягається шляхом заміни елементів, які не справляються з роботою, на більш потужні і швидкодіючі. Це найпростіший тип масштабування, який не потребує внесення змін до програми.

– горизонтальне – передбачає поділ системи на структурні компоненти, рознесені по різних машинах, і збільшення числа серверів, паралельно використовуючи окремі функції. Тягне за собою додавання в систему додаткових вузлів, що працюють як єдине ціле, може зажадати внесення змін в програму для ефективного використання збільшеного кількості ресурсів.

Вертикальне масштабування не безкінечна, тому, коли мова йде про «high load»-систему, багато хто вважає другий підхід більш кращим. Але повністю відмовлятися від вертикального масштабування не варто. Воно швидше і дешевше – перестав сервер справлятися з навантаженням, простіше купити новий, більш потужний, і запустити його в роботу, ніж переписувати програму [9].

– Модульність. Застосовуючи горизонтальний підхід до масштабування, все намагаються розділити додаток на окремі модулі, які можна рознести по різних серверах. І зробити так, щоб самий високонавантажених ділянка була мультиплікована. Оптимальне рішення – оптимально поєднувати вертикальний і горизонтальний підходи в залежності від унікальних особливостей системи і змін вимог.

– Високе навантаження на інтеграційний рівень. Це ще одна особливість високонавантажених додатків. Чим більше ми його розбиваємо на модулі і чим

більше зростає навантаження на них, тим сильніше стає навантаження і на інтеграційний шар.

– Ексклюзивність. З вищесказаного випливає ще одна особливість високонавантаженого навантаження – воно нетривіально і обмежене навантаженням на інтеграційний шар. Проектуючи такі додатки, потрібно розуміти, що немає стандартних рішень, які б підійшли для будь-якої «high load» -системи, в кожному випадку рішення ексклюзивно і орієнтоване на бізнес-вимоги [10].

– Дублювання критичних вузлів. Всі особливо важливі частини «high load» -системи, від яких залежить її життєдіяльність, потрібно дублювати і в програмному плані, і в плані «заліза». І при цьому вони не обов'язково повинні працювати паралельно. Завжди потрібно мати failover-сервер або кластер серверів, якими можна скористатися, якщо робочі машини перестануть справлятися з навантаженням [11].

1.2 Проблеми високонавантажених інформаційних систем

При розробці будь-якої інформаційної системи (ІС) передбачається складна робота з великими масивами даних. Аналіз і обробка значної кількості одночасних запитів і звернень вимагають від ІС високої продуктивності, надійності та відмово стійкості. Створення подібних рішень передбачає вибір оптимальної архітектури, а також взаємозв'язку безлічі складних компонентів [12].

Розглядаючи архітектуру високих навантажень при роботі в ІС основне і пікове навантаження на систему буде формуватися з обсягів даних, якими оперує сам додаток, кількості користувачів, які цим додатком користуються, і швидкістю обробки, аналізу і видачі отриманих підсумкових даних користувачеві за запитом . Для високонавантажених ІС завжди буде характерна позитивна динаміка збільшення зростання кількості користувачів і це збільшення кількості користувачів в свою чергу породжує збільшення кількості

даних, якими ці користувачі оперують, тим самим структура ІС набуває масштабів високонавантаженої ІС [13].

Архітектурні рішення – фундамент для будь-яких додатків. У тому числі і для додатків з високими навантаженнями. Важливо розуміти, що архітектура додатка визначає успішність його роботи в цілому. У тому числі і його змоги справлятися з навантаженнями.

Архітектура системи, що розробляється куди важливіше використовуваних в ній алгоритмів і мов програмування, але ще важливіше вміти вибудовувати найбільш підходящі структури даних і вибирати найактуальніші протоколи обміну даних. Від структур даних залежать алгоритми, якими вони будуть оброблятися, а з поганими структурами можна написати хороший алгоритм. Від протоколів залежить робота сервера і швидкість обміну даними між компонентами системи (наприклад, між СУБД та сервером, сервером і клієнтом, двома різними серверами або сервісами). Для розробки ІС має сенс вибирати таку мову програмування і таку платформу, яка має хороший менеджер пам'яті і здатну тримати в пам'яті великий об’єм даних, ефективно обробляти і оперувати великими структурами даних. Рекомендується використовувати мови побудови запитів до InMemory даними. Основне завдання InMemory Data Grid (IMDG) – забезпечити надвисоку доступність даних шляхом зберігання їх в оперативній пам'яті в розподіленому стані. Сучасні IMDG здатні задовольнити більшість вимог до обробки великих масивів даних. Сенс полягає в тому, щоб мінімізувати роботу з СУБД, знизити кількість звернень до дисків на серверах і як найменше робити або взагалі не робити зайвих операцій введення / виводу, окрім безпосередньо роботи клієнтом або оператором по мережі [14].

Піковий момент високого навантаження на систему настає тоді, коли система перестає якісно вирішувати поставлені перед нею завдання. Симптомами такої проблеми на прикладі веб-сервера можуть бути: випадкові помилки сервера, повільне завантаження сторінок, часткове або неповне відображення контенту і обірвані з'єднання при запиті або зверненні до системи

користувачем [15].

Присутність високих навантажень в ІС передбачає використання великих по масштабу додатків (в рамках використовуваних апаратних можливостей). У разі якщо спочатку при розробці проекту ІС передбачається і планується, що система може перейти в режим високонавантаженої, то при її створенні, розгортання і підтримки необхідно буде врахувати принципи побудови високо навантажених ІС:

– неможливо передбачити всі нюанси динаміки системи. Набагато легше забезпечити її гнучкість, і поступове зростання згідно з потребами і запитами. Успішність роботи над великим додатком має на увазі зовсім не детальне планування всіх аспектів. Основне зусилля повинне бути націлене на забезпечення гнучкості самої системи. Гнучкість дозволяє швидко вносити зміни, а це найбільш важлива властивість будь-якої швидкозростаючою і високо навантаженої системи;

– необхідно розробляти найбільш прості рішення і вирішувати в першу чергу необхідно ті проблеми, які виникають у абсолютної більшості користувачів [16].

Ці принципи будуть застосовні як до програмної, так і до апаратної частини ІС.

Для того щоб отримати максимальну швидкість при істотному навантаженню на сервери, можна буде скористатися більш простим і тонким стеком технологій, всі інструменти якого повинні бути знайомі ІС і передбачувані у використанні. Складові конструкції утворюють ІС повинні бути одночасно простими, достатніми для вирішення поточних проблем, але і до того ж повинні мати досить значний запас міцності і гнучкість. Найкращим рішенням тут буде застосування горизонтального масштабування і використовування кеш-контролю продуктивності системи. Обов'язковою параметром для життєзабезпечення будь-якого високонавантаженого інформаційного проекту є так само логування і моніторинг стану системи. А система розгортання дозволить в значній мірі спростити життя і зекономити нерви, час і гроші як з

боку розробника, так і з боку замовника даної інформаційної системи [17].

1.3 Основні концепції «high load» систем

Відкладені обчислення. Відкладені обчислення або ліниві обчислення – концепція, згідно з якою обчислення слід відкладати до тих пір, поки не знадобиться їх результат.

Основна ідея ледачих обчислень полягає в тому, що економиться час на проведення обчислень, результати яких свідомо не будуть використані в подальшому програмою. Відповідно, за рахунок зниження об'ємів обчислень підвищується і продуктивність програми, а за рахунок відсутності необхідності зберігати в пам'яті результати обчислень знижуються і вимоги програми до пам'яті. Крім цього, ледачі обчислення позбавляють програміста від необхідності стежити за тим, які саме обчислення будуть надалі затребувані програмою, а які, навпаки, виявляться повністю марними [18].

Попередні обчислення**.** Якщо обчислення трудомісткі, такі як розрахунок статистики або підрахунок топів / голосів / рейтингів, але немає необхідності чекати першого запиту, щоб здійснювати обчислення, якщо є можливість їх підготувати заздалегідь. Нема чого витрачати настільки дороге процесорний час навантаженого фронт-сервера. Такі, розрахункові завдання виконуються по розкладую або через "сервера завдань", на віддалених комп'ютерах.

Вся суть «high load» проектів зводиться до принципу "Швидко взяв – швидко віддав". Як мінімізувати швидкість віддачі сторінки. Логіка віддачі така:

– частина даних винести в окрему логіку, готувати їх окремих скриптами або «демонами». Віддавати їх окремими AJAX запитами, а HTML формувати на стороні клієнта (браузера). Тим самим ми зменшуємо час на формування і передачу вихідних даних. Такими даними можуть бути лічильники повідомлень, переглядів, "Що нового у друзів" і так далі;

– мінімізувати звернення до БД [18].

Однією з різновидів попередніх обчислень є кешування.

Кешування – організація обчислень з використанням проміжного буфера з швидким доступом, що містить інформацію, яка може бути запрошена з найбільшою ймовірністю.

Кешування сьогодні є невід'ємною частиною будь-якого Webпроекту, не обов'язково високонавантаженого. Для кожного ресурсу критичною для користувача є така характеристика як час відгуку сервера. Збільшення часу відгуку сервера призводить до відтоку відвідувачів. Отже, необхідно мінімізувати час відгуку: для цього необхідно зменшувати час, необхідний на формування відповіді користувачу, а відповідь користувачеві вимагає отримати дані з якихось зовнішніх ресурсів (backend). Цими ресурсами можуть бути як бази даних, так і будь-які інші щодо повільні джерела даних (наприклад, віддалений файловий сервер, на якому ми уточнюємо кількість вільного місця). Для генерації однієї сторінки досить складного ресурсу нам може знадобитися вдосконалення десятків подібних звернень. Багато з них будуть швидкими: 20 мілі-секунд і менше, проте завжди існує деякий невелику кількість запитів, час обчислення яких може обчислюватися секундами або хвилинами (навіть в самій оптимізованої системі вони можуть бути, хоча їх кількість повинна бути мінімально) . Якщо скласти весь час, яке ми витратимо на очікування результатів запитів (якщо ж ми будемо виконувати запити паралельно, то візьмемо час обчислення самого довгого запиту), ми отримаємо незадовільний час відгуку [19].

Рішенням цього завдання є кешування: ми поміщаємо результат обчислень в деяке сховище (наприклад, memcached), яке володіє відмінними характеристиками по часу доступу до інформації. Тепер замість звернень до повільних, складних і важких backend'ам нам достатньо виконати запит до швидкого кешу.

Memcached є величезною хеш-таблицю в оперативній пам’яті, доступну з мережевого протоколу. Він забезпечує сервіс зі зберігання значень, асоційованих з ключами. Доступ до хешу ми отримуємо через простий мережевий протокол, клієнтом може виступати програма, написана на довільній мові програмування

(існують клієнти для C / C ++, PHP, Perl, Java) [20]

Найпростіші операції – отримати значення зазначеного ключа (get), встановити значення ключа (set) та код (del). Для реалізації ланцюжка атомарних операцій (за умови конкурентного доступу до memcached зі стоку паралельних процесів) використовуються додаткові операції: інкремент / декремент значення ключа (incr / decr), дописати дані до значення ключа в верхню чи нижню частину (append / prepend), атомарна зв'язка отримання / установки значення (gets / cas) та інші були реалізований Бредом Фітцпатріком (Brad Fitzpatrick) в рамках роботи над проектом ЖЖ (LiveJournal). Він користувався ними для розвантаження бази даних від запитів при віддачі контенту сторінок. Сьогодні memcached знайшов своє застосування в ядрі багатьох великих проектів, наприклад, Wikipedia, YouTube, Facebook і інші [20].

Розпаралелювання завдань. Віддача контенту в «high load» проектах зводиться до задачі "якомога швидше віддати". Як цього досягти:

– швидко отримати або розрахувати необхідні дані;

– швидко сформувати HTML;

– якомога швидше віддати, отриманий HTML.

Як можна швидко зібрати дані – це або вилучити вже готові дані (кешування). Або розкидати пошук даних за кількома машинам, якщо численні дані у нас розподілені по декількох серверах.

Фонові обчислення. Віддача і формування динамічного контенту – це завдання, що вимагає відносно великої кількості процесорного часу. Якщо в проекті є завдання, наприклад розрахункові або по обробці графіки, то їх бажано віддати на інші сервера. Зазвичай це робиться через спеціальні сервера завдань або черги. Наприклад, завдання завантаження медіа контенту можна доручити окремому серверу і не задіяти frontend, використовуючи спеціальні модулі nginx для завантаження файлів: ngx_http_upload_module, ngx_http_upload_progress_module [21].

Балансування навантаження. Балансування (вирівнювання) навантаження – розподіл процесу виконання завдань між декількома серверами мережі з метою

оптимізації використання ресурсів і скорочення часу обчислення.

Продуктивність Web вузла в цілому збільшиться, якщо збільшити кількість фронт серверів, з розміщенням на вузлі "дзеркальних" копій WEB серверів (горизонтальне масштабування) (рисунок 1.1). Розподіляючи загальне навантаження за всіма компонентами системи, в результаті скорочується загальний час оброблення інформації, обробляючи її на одному з серверів. Крім збільшення потужності, горизонтальне масштабування додає надійності системі - при виході з ладу одного з серверів, навантаження буде збалансована між працюючими і додаток буде жити [22].

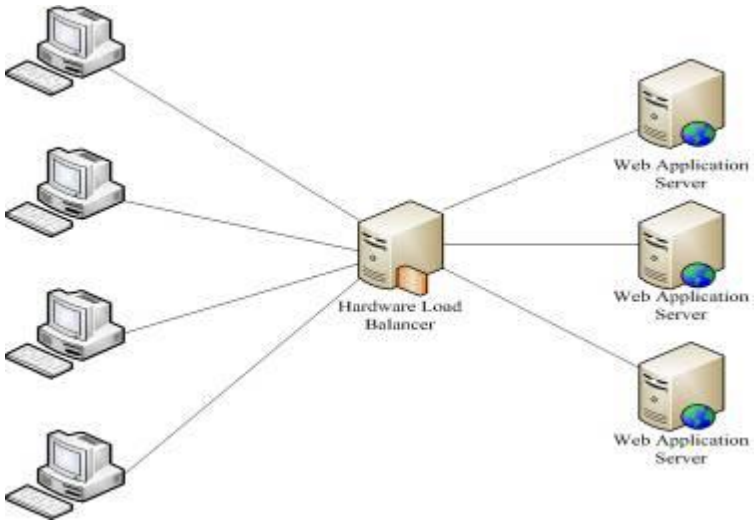

Рисунок 1.1 – Балансування WEB-сервера

Типи балансування:

– WEB-сервера;

– кластер серверів БД;

– кластер кешей.

Всі запити проходять через балансувальник, який визначає кому з серверів віддати на обробку. Про його налаштування і піде мова.

При отриманні запиту від клієнта, балансувальник потрібно визначити, якому з WEB-серверів переслати запит. Алгоритм прийняття рішення називаються методом або стратегією балансування, найбільш поширені стратегії:

– Round robin. З доступних серверів будується черга і балансувальник вибирає перший в черзі. Після виконання запиту сервер переміщається в кінець черги;

– Least Connection. Балансувальник веде облік кількості незакритих з'єднань і вибирає той сервер, у якого таких з'єднань менше;

– використання "ваги" серверів. Кожному сервера в залежності від потужності присвоюється вага, який використовується для ранжирування [21, 22].

Очевидно, що стратегія, що не включає перевірку стану серверів або хоча б працездатності, не придатна для використання, тому що не гарантує обробку запиту. Тому алгоритм повинен вміти перевіряти дієздатність сервера, його завантаженість і вибирати найбільш здатний.

При балансуванні часто виникає проблема зберігання сесій, адже сесія доступна тільки на тому, який створив її, що потрібно враховувати в алгоритмі перенаправлення запиту. Або ж зберігати сесії на окремому сервері або в БД (рисунок 1.2).

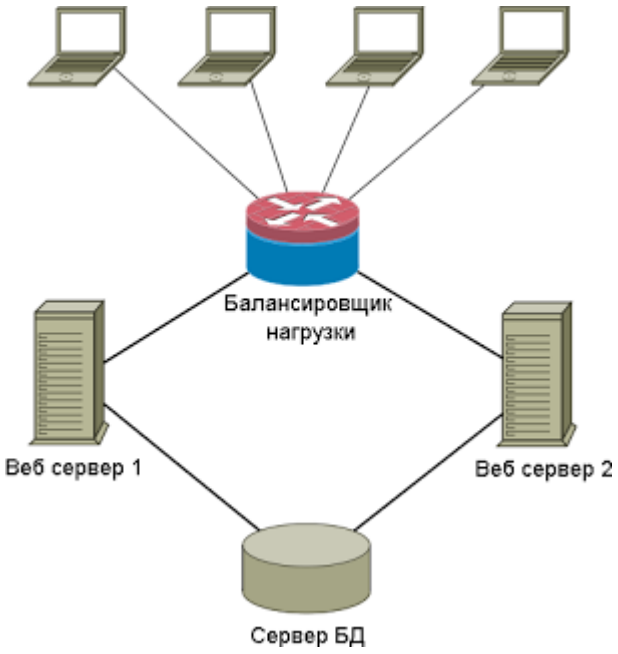

Рисунок 1.2 – Проста схема балансувальника навантаження двох WEB-серверів

Для програмної реалізації балансувальника існує багато рішень, наприклад:

– Nginx;

– Apache + mod_proxy_balancer;

– проксі сервер.

Балансування серверів БД, здійснюється спеціальними проксі серверами. Для MySQL – це MySQLProxy. Для PostgreSQL це поєднання PgProxy і PgBouncer. Один є пулом з'єднань, а другий балансувальник навантаження.

Такий продукт, як memcached і redis мають вбудовані засоби балансування.

Якщо для балансування WEB запитів і запитів до БД використовується алгоритм round robin, то для балансування кешів використовується алгоритм ketama, суть якого в тому, щоб закріпити ключі за серверами і при додаванні нового сервера в кластер кешів не порушити розподіл ключів по серверів кластера. Наприклад, якщо номер сервера буде вибиратися хаотично (хоч і постійно при однаковій кількості серверів memcached), то при додаванні нового сервера ключі змінять своє положення на серверах і закешовані дані будуть втрачені.

Використовуючи ketama можна безболісно додавати нові сервери memcached, старі ключі будуть лежати на старих серверах. Доступ до даних буде збережений [23].

1.4 Огляд використовуваних технологій в високо навантажених системах

1.4.1 Технології Google

Google – величезна інтернет-компанія, є беззаперечним лідером на ринку пошуку в Інтернет і власник великої кількості продуктів, багато з яких також домоглися певного успіху в своїй ніші.

На відміну від більшості інтернет-компаній, які займаються лише одним продуктом (проектом), архітектура Google не може бути представлена як єдине

конкретне технічне рішення.

Всі продукти Google ґрунтуються на програмній платформі, яка спроектована з урахуванням роботи на мільйонах серверів, що знаходяться в різних дата центрах по всьому світу.

У Google дуже рано зіткнулися з проблемами ненадійності обладнання і роботи з величезними масивами даних. Програмна платформа, спроектована для роботи на багатьох недорогих серверах, дозволила їм абстрагуватись від збоїв і обмежень одного сервера [24].

Основними завданнями в ранні роки була мінімізація точок відмови і обробка великих обсягів слабо структурованих даних. Рішенням цих задач стали три основних шару платформи Google, що працюють один поверх іншого:

Google File System: розподілена файлова система, що складається з серверу з метаданими і теоретично необмеженої кількості серверів, зберігаючих довільні дані в блоках фіксованого розміру.

BigTable: розподілена база даних, яка використовує для доступу до даних два довільних байтових рядки-ключа (що позначають рядок і стовпчик) і дату / час (що забезпечують версіювання).

MapReduce: механізм розподіленої обробки великих обсягів даних, який оперує парами ключ-значення для отримання необхідної інформації [25].

Google Colossus. Нова архітектура GFS була спроектована для мінімізації затримок при доступі до даних (що критично для додатків на зразок Gmail і YouTube), не на шкоду основним властивостям старої версії: відмово стійкості і прозорою масштабованості.

В оригінальній же реалізації упор був зроблений на підвищення загальної пропускної здатності: операції об'єднувалися в черзі і виконувалися разом, при такому підході можна було прочекати пару секунд ще до того, як перша операція в черзі почне виконуватися. Крім цього в старій версії було велике слабке місце у вигляді єдино майстер-сервера з метаданими, збій в якому погрожував недоступністю всієї файлової системи в протягом невеликого проміжку часу (поки інший сервер не підхопить його функції, спочатку це займало близько 5

хвилин, в останніх версіях близько 10 секунд) – це також було цілком допустимо при відсутності вимоги роботи в реальному часі, але для додатків, безпосередньо взаємодіють з користувачем, це було неприйнятно з точки зору можливих затримок.

Основним нововведенням в Colossus стали розподілені майстер-сервера, що дозволило позбутися не тільки від єдиної точки відмови, а й істотно зменшити розмір одного блоку з даними (з 64 до 1 мегабайта), що в цілому дуже позитивно позначилося на роботі з невеликими обсягами даних. Як бонус зникла теоретична межа кількості файлів в одній системі.

Деталі розподілу відповідальності між майстер-серверами, сценаріїв реакції на збої, а також порівняння з затримок і пропускної здатності обох версій, на жаль, як і раніше конфіденційні [26].

Google Percolator. MapReduce відмінно справлявся із завданням повної перебудови пошукового індексу, але не передбачав невеликі зміни, що зачіпають лише частину сторінок. Через потокової, послідовної природи MapReduce для внесення змін в невелику частину документів все одно довелося б оновлювати весь індекс, так як нові сторінки неодмінно будуть якимось чином зв’язані зі старими. Таким чином, затримка між появою сторінки в Інтернеті і в пошуковому індексі при використанні MapReduce була пропорційна загального обсягу індексу (а значить і Інтернету, який постійно зростає), а не розміром набору змінених документів.

Ключові архітектурні рішення, що лежать в основі MapReduce, що не дозволяють вплинути на цю особливість і в підсумку система індексації була побудована заново з нуля, а MapReduce продовжує використовуватися в інших проектах Google для аналітики і інших задач, як і раніше не пов'язаних з реальним часом.

Нова система отримала назву Percolator. Являє собою надбудову над BigTable, що дозволяє виконувати комплексні обчислення на основі наявних даних, що зачіпають багато рядків і навіть таблиць одночасно (в стандартному API BigTable це не передбачено).

Веб-документи або будь-які інші дані змінюються / додаються в систему за допомогою модифікованого API BigTable, а подальші зміни в решті базі здійснюються за допомогою механізму "оглядачів". Якщо говорити в термінах реляційних СУБД, то оглядачі – щось середнє між тригерами і збереженими процедурами. Оглядачі представляють собою підключаємий до бази даних код (на C ++), який виконується в разі виникнення змін в певних колонках. Всі використовувані системою метадані також зберігаються в спеціальних колонках BigTable. При використанні Percolator всі зміни відбуваються в транзакціях, які відповідають принципу ACID, кожна з яких зачіпає саме ті сервера в кластері, на яких необхідно внести зміни. Механізм транзакцій на основі BigTable розроблявся в рамках окремого проекту під назвою Google Megastore.

Таким чином, при додаванні нового документа (або його версії) в шуканий індекс, викликається ланцюгова реакція змін до старих документах, обмежена за своєю рекурсивності. Ця система при здійсненні випадкового доступу підтримує індекс в актуальному стані.

Як бонус в цій схемі вдалося уникнути ще двох недоліків MapReduce:

– проблеми "відстаючих": коли один з серверів (або одна з конкретних підзадач) опинявся істотно повільніше інших, що також значно затримувало загальний час завершення роботи кластера;

– пікове навантаження: MapReduce не є безперервним процесом, а розділяється на роботи з обмеженою метою і часом виконання. Таким чином крім необхідності ручного налаштування робіт і їх типів, кластер має очевидні періоди простою і пікового навантаження, що веде до неефективного використання обчислювальних ресурсів.

Але все це виявилося не безкоштовно: при переході на нову систему вдалось досягти тієї ж швидкості індексації, але при цьому використовувалося вдвічі більше обчислювальних ресурсів. Продуктивність Percolator знаходиться десь між продуктивністю MapReduce і продуктивністю традиційних СУБД. Так як Percolator є розподіленою системою, для оброблення фіксованої невеликої кількості даних їй доводиться використовувати ти істотно більше ресурсів, ніж

традиційна СУБД; така ціна масштабування. У порівнянні з MapReduce також довелося платити додатковими споживаними обчислювальними ресурсами за можливість випадкового доступу з низькою затримкою. Проте, при обраній архітектурі Google вдалося досягти практично лінійного масштабування при збільшенні обчислювальних потужностей на багато порядків. Додаткові накладні витрати, пов'язані з розподіленою природою рішення, в деяких випадкових до 30 разів перевершують аналогічний показник традиційних СУБД, але у даної системи є солідний простір для оптимізації в цьому напрямку, ніж Google активно і займається [26].

Google Spanner. Spanner являє собою єдину систему автоматичного управління ресурсами всього парку серверів Google.

Основні особливості:

– єдиний простір імен:

а) ієрархія каталогів;

б) незалежність від фізичного розташування даних;

– підтримка слабкою і сильною цілісності даних між дата центрами;

– автоматизація:

а) заміна або додавання реплік даних;

б) виконання обчислень з урахуванням обмежень і способів використання;

в) виділення ресурсів на всіх доступних серверах;

– зони напівавтономного управління;

– відновлення цілісності після втрат з'єднання між дата центрам;

– можливість вказівки користувачами високорівневих вимог, наприклад:

а) 99% затримок при доступі до цих даних повинні бути до 50 мс;

б) розташувати ці дані на як мінімум 2 жорстких дисках в Європі, 2 в США і 1 в Азії;

– інтеграція не тільки з серверами, а й з мережевим обладнанням, а також системами охолодження в дата центрах;

– проектувалася з розрахунку на:

а) 1-10 мільйонів серверів;

б) 10 трильйонів директорій;

в) 1 000 петабайт даних;

г) 100-1000 дата центрів по всьому світу;1 мільярд клієнтських машин.

Про цей проект Google відомо дуже мало, офіційно він був представлений публіці лише одного разу в 2009 році, з тих пір лише місцями згадувався співробітниками без особливої конкретики. Точно не відомо розгорнута ця система на сьогоднішній день і якщо так, то в якій частині дата центрів, а також який статус реалізації заявленого функціоналу [26].

Інші компоненти платформи. Платформа Google в кінцевому підсумку зводиться до набору мережевих сервісів і бібліотек для доступу до них з різних мов програмування (в основному використовуються C / C ++, Java, Python і Perl). Кожен продукт, що розробляється Google, в більшості випадків використовує ці бібліотеки для здійснення доступу до даних, виконання комплексних обчислень і інших задач, замість стандартних механізмів, що надаються операційною системою, мовою програмування або open-source бібліотеками.

Вищевикладені проекти складають лише основу платформи Google, хоча вона включає в себе куди більше готових рішень і бібліотек, кілька прикладів з публічно доступних проектів:

– GWT (Google Web Toolkit – вільний Java-фреймворк для AJAX-додатків) для реалізації користувальницьких інтерфейсів на Java;

– Closure – набір інструментів для роботи з JavaScript;

– Protocol Buffers – не залежить від мови програмування і платформ-ми формат бінарної серіалізації структурованих даних, використовується при взаємодії більшості компонентів системи всередині Google;

– LevelDB – високопродуктивна вбудована СУБД;

– Snappy – швидка компресія даних, використовується при зберіганні даних в GFS.

1.4.2 Технології Facebook

Можна по-різному ставитися до соціальних мереж взагалі і до Facebook зокрема, але з точки зору технологічності це один з найцікавіших проектів. Особливо приємно, що розробники ніколи не відмовлялися ділитись досвідом створення ресурсу, що витримує подібні навантаження. В цьому є велика практична користь. Адже в основі системи лежать загальнодоступні компоненти, які використовувати кожен. Більш того, багато хто з тих технологій, які розробляється валися всередині Facebook, зараз опубліковані з відкритим кодом. І використовувати їх, знову ж таки, може будь-хто. Розробники соціальної мережі по можливості використовували лише відкриті технології та філософію Unix: кожен компонент системи повинен бути максимально простим і продуктивним, при цьому рішення задач досягається шляхом їх комбінування. Всі зусилля інженерів спрямовані на масштабованість, мінімізацію кількості точок відмови і, що найважливіше, простоту [27].

Основні технології, які зараз використовуються всередині Facebook:

Memcached — один з найбільш широко відомих проектів в Інтернеті. Його розподілена система кешування інформації використовується як кешується між веб-серверами і MySQL (оскільки доступ до баз даних відносно повільний). Пройшли роки, і Facebook справив величезне кількість модифікацій коду Memcached і супутнього ПО.

У Facebook працюють тисячі Memcached серверів з десятками терабайт кешованих даних в будь-який момент часу [20].

HipHop для PHP. PHP, оскільки він є скриптовою мовою, досить повільною, якщо порівнювати його з рідним кодом процесора, виконуваним на сервері. HipHop конвертує скрипти на PHP в вихідні коди на C ++, які потім компілюються для забезпечення гарної продуктивності. Це дозволяє Facebook отримувати більшу віддачу від меншої кількості серверів, оскільки PHP в Facebook використовується практично повсюдно [28].

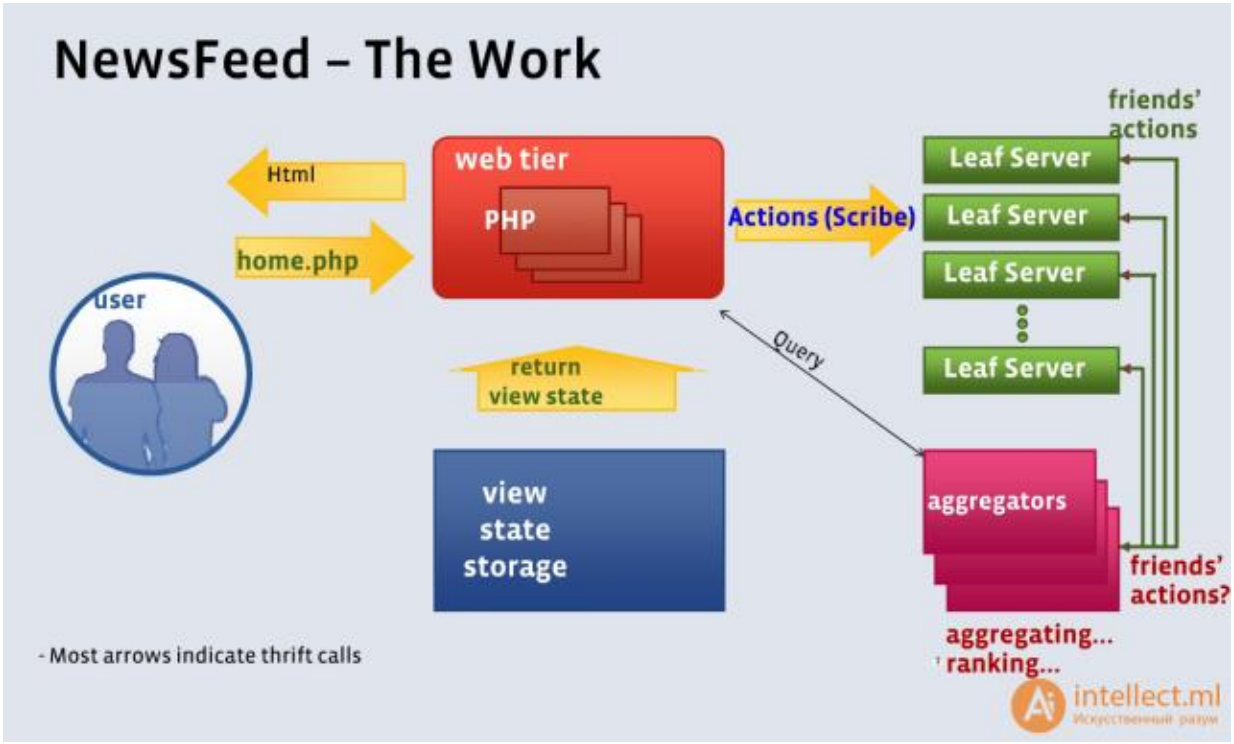

Рисунок 1.3 – Схема формування стрічки новин

Haystack — це високопродуктивна система зберігання / отримання фотографій (Haystack – сховище об'єктів, тому він може зберігати будь-які дані, а не тільки фотографії). На частку цієї системи випадає величезна кількість роботи. На Facebook завантажено понад 20 мільярдів фотографій, і кожна зберігається в чотирьох різних дозволах, що в підсумку дає більше 80 мільярдів одиниць фотографій.

Haystack повинен не просто вміти зберігати фото, але і віддавати їх дуже швидко. Facebook віддає більше 1,2 мільйона фотографій в секунду. Ця кількість не включає в себе фотографії, які віддаються системою доставки контенту Facebook і постійно зростає.

BigPipe – система динамічної доставки web-сторінок, розроблена в Facebook. Вона використовується для доставки кожної веб-сторінки секціями (які називаються pagelets) для оптимізації продуктивності.

Наприклад, вікно чату, стрічка новин та інші частини сторінки запитуються окремо. Їх можна отримувати паралельно, що збільшує продуктивність і дозволяє користувачам використовувати веб-сайт навіть в тому

випадку, якщо якась його частина відключена або несправна [29].

Cassandra – розподілене відмово стійке сховище даних. Це одна з систем, яку завжди згадують, говорячи про NoSQL. Cassandra стала проектом з відкритим вихідним кодом і навіть стала дочірнім проектом Apache Foundation. На Facebook вона використовується для пошуку по папці «Вхідні». В принципі, її використовують багато проектів. Наприклад, Digg. Планується її використання в проекті Pingdom.

Scribe –зручна система протоколювання, яка використовується відразу для кількох речей одночасно. Вона була розроблена для забезпечення ведення протоколів в масштабі всього Facebook і підтримує додавання нових категорій подій, як тільки вони з'являються (на Facebook їх сотні).

Hadoop і Hive – реалізація алгоритму map-reduce з відкритим результатним кодом, що дозволяє робити обчислення на величезних обсягах даних. У Facebook його використовують для аналізу даних. Hive був розроблений в Facebook і дозволяє використовувати SQL запити для отримання інформації від Hadoop, що полегшує роботу непрограмістів.

І Hadoop, і Hive мають відкриті вихідні коди і розвиваються під егідою Apache Foundation. Їх використовує велику кількість інших проектів. Наприклад, Yahoo і Twitter.

Thrift. Facebook використовує різні мови програмування в розособистих компонентах системи. PHP використовується як фронтенд, Erlang для чату, Ява і C ++ теж не залишилися без діла. Thrift – кросмовний фреймворк, який пов'язує всі частини системи в єдине ціле, дозволяючи їм спілкуватися один з одним. Thrift розробляється як проект з відкритим вихідним кодом і в нього вже додана підтримка деяких інших мов програмування.

Varnish – HTTP акселератор, який може служити в якості балансувальника навантаження і кеша вмісту, яке потім може доставлятися з великою швидкістю. Facebook використовує Varnish для доставки фотографій і картинок профілів, витримуючи навантаження в мільярди запитів в день. Як і всі, що використовується Facebook, Varnish – програмне забезпечення з відкритим

вихідним кодом [29].

Східчасті релізи та неявна активація нових функцій. У Facebook використовується система, звана GateKeeper, яка дозволяє обслуговувати різних користувачів за допомогою різних версій вихідного коду системи. Це дозволяє проводити релізи покроково, активувати деякі функції тільки для працівників Facebook і так далі.

GateKeeper також дозволяє Facebook виконувати неявну активацію нових функцій для виконання, наприклад, тестування навантаження і виявлення повільно працюють компонентів системи. Неявна активація включає деяку функцію або можливість, але не показує її в інтерфейсі користувача. Зазвичай неявна активація виконується за два тижні до того, як нова можливість надається користувачам.

Тест продуктивності системи. Facebook проводить ретельний моніторинг продуктивності системи і, що цікаво, проводиться моніторинг продуктивності виконання кожної PHP функції системи. Це забезпечується використанням XHProf [30].

Часткове відключення функцій. Якщо проект починає працювати повільно, відключаються деякі можливості (з досить великої кількості) для того, щоб підняти продуктивність ключових компонентів системи.

Балансувальник навантаження вибирає php-сервер для обробки кожного запиту, де HTML генерується на основі різних джерел (таких як MySQL, memcached) і спеціалізованих сервісів. Таким чином, архітектура Facebook має традиційний трирівневий вигляд:

– веб-додаток;

– розподілений індекс;

– постійне сховище.

1.4.3 Технології YouTube

– Linux – загальна назва UNIX-подібних операційних систем на основі

однойменного ядра. Це один із найвидатніших прикладів розробки вільного (free) та відкритого (з відкритим кодом, open source) програмного забезпечення (software). На відміну від власницьких операційних систем (на кшталт Microsoft Windows та MacOS X), їхні вихідні коди доступні всім для використання, зміни та поширення абсолютно вільно (в тому числі безкоштовно).;

– Apache – основний HTTP-сервер – це безкоштовне програмне забезпечення для міжплатформенних веб-серверів із відкритим кодом, випущене на умовах ліцензії Apache 2.0. Apache розробляється та підтримується відкритою спільнотою розробників під егідою Фонду програмного забезпечення Apache. Переважна більшість екземплярів сервера Apache HTTP працює на дистрибутиві Linux, але нинішні версії також працюють на Microsoft Windows, OpenVMS та на різноманітних Unix-подібних системах. Минулі версії також працювали на NetWare, OS / 2 та інших операційних системах, включаючи порти до мейнфреймів;

– lighttpd – віддача відео з YouTube CDN;

– Zookeeper – розподілені блокування, зберігання конфігурацій;

– Python:

а) wiseguy – FastCGI-прошарок між Apache і Python;

б) pycurl – найкраща доступна реалізація HTTP-клієнта, але в підсумку її все одно замінили на власне низькорівневе рішення, вигравши 8% в споживанні обчислювальних ресурсів;

в) spitfire – високопродуктивний шаблонізатор на основі абстрактного синтаксичного дерева з регульованим рівнем оптимізації (як в gcc);

г) bson як формат серіалізації.

– BigTable – Кожна таблиця має багато розмірностей (одна з яких є полем для відміток часу, що дозволяє використовувати системи керування версіями та збирання сміття). Таблиці оптимізовані для файлової системи Google (GFS) шляхом розбиття на кілька таблиць або таблетів (англ. tablet) — сегменти таблиці діляться по рядкам, таким чином, щоб їх розмір складав ~200 мегабайт. Коли розміри загрожують вийти за встановлену межу, таблиці стискуються за

допомогою алгоритмів BMDiff та Zippy — останній алгоритм широко відомий за відкритою бібліотекою Snappy[en], яка є варіацією LZ77 не дуже ефективною за об'ємом стиснення, але більш ефективною за часом обчислень. Розташування складових таблиці (таблети) в GFS вносяться як записи бази даних у кількох спеціальних таблетах, які називаються «META1»-таблетами. META1 таблети знаходяться за запитом єдиного таблета «META0», який звичайно розміщується на власному сервері, оскільки до нього часто йдуть запити від клієнтів щодо розташування таблета «META1», який сам містить відповідь на питання про те, де знаходяться фактичні дані. Як і головний сервер GFS, сервер META0 взагалі не є вузьким місцем[en], оскільки час і пропускна спроможність процесора, необхідні для пошуку та передачі адреси META1, є мінімальними, а клієнти активно кешують координати для мінімізації запитів.;

– MySQL – використовується просто як сховище даних, версія 5.1.52 з InnoDB. База даних представляє собою структуровану сукупність даних. Ці дані можуть бути будь-якими – від простого списку попередніх покупок до перегляду експонатів картинної галереї або величезної кількості інформації в корпоративній мережі. Для записів, вибору та обробки даних, зберігання в комп'ютерній базі даних, необхідна система управління базовими даними, яка є і є РО MySQL. Після того, як комп'ютери замінюються, пропонуються з обробкою більших об'єктів даних, управління базами даних грає центральну роль у виплатах. Реалізовано таке управління може бути по-різному – як у вигляді окремих утиліт, так і у вигляді коду, що знаходиться в складі інших додатків. MySQL є дуже швидким, надійним і легким у використанні. Якщо вам потрібні саме ці якості, спробуйте попрацювати з даними сервера. MySQL також володіє зручними можливостями, розробленими в тісному контакті з користувачами. Порівняльні характеристики MySQL та інших засобів управління базами даних, представлені на нашій сторінці тестів продуктивності;

– Vitess – це рішення для баз даних для розгортання, масштабування і управління великими кластерами примірників MySQL. Він спроектований так, щоб працювати в архітектурі публічного або приватного хмари так само

ефективно, як і на виділеному обладнанні. Він об'єднує і розширює багато важливих функцій MySQL з масштабністю бази даних NoSQL [31].

1.5 Висновки з розділу 1

Перш за все, «high load» – вкрай відносне поняття. Воно ніколи не вимірюється кількістю запитів або швидкістю роботи сайту, тому що попросту немає такого поняття, як "середній сайт". Всі сайти специфічні і однакову кількість запитів може призводити до абсолютно різних навантажень на різні ресурси.

Кількість користувачів Інтернету з року в рік швидко збільшується, що, в свою чергу, призводить до перевантаження серверів баз даних. Системи просто не в змозі обробляти таку велику кількість запитів користувачів одночасно, що спричиняє несправність Інтернет-ресурсів. Проекти, що враховують баланс швидкості повернення динамічного ("важкого") та статичного ("легкого") вмісту, називаються проектами з високим навантаженням. Як правило, вони унікальні і реалізуються для конкретних потреб.

Таким чином, було вирішено перше завдання дипломної роботи – проаналізовано особливості та проблеми при розробці високонавантажених систем на прикладах Google, Facebook и YouTube.

2 ПЛАНУВАННЯ ЕКСПЕРЕМЕНТУ ДЛЯ ДОСЛІДЖЕНЬ РОЗВАНТАЖЕННЯ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ

В даному розділі необхідно вирішити такі завдання:

– експериментально порівняти ефективність методів та засобів для розвантаження серверів, що використовуються в високонавантажених системах YouTube, Google, Facebook;

– вибрати методи для порівняльного оцінювання ефективності використання існуючих підходів в розвантаженні високонавантажених систем.

2.1 Постановка цілей і завдань експериментального дослідження

Мета експерименту – отримання інформації про ефективність використання існуючих засобів в розвантаженні високонавантажених систем. Провести аналіз і порівняти отримані результати.

Для досягнення мети необхідно виконати наступні завдання:

– проаналізувати відгуки та фактори експерименту;

– описати завдання експерименту;

– обрати метод обробки результатів експерименту;

– визначити технічні засоби реалізації;

– визначити програмні засоби реалізації.

2.2 Аналіз факторів та відгуків експерименту

2.2.1 Основні положення теорії планування експерименту

Для всіх видів фізичних експериментів, послідовність їх організації стандартизована і складається з представлених нижче етапів.

Аналітичний огляд раніше проведених досліджень. Проводиться з метою уникнення дублювання вже відомих результатів, а також з метою отримання інформації для відпрацювання методики даного експерименту.

Вибір залежних змінних (відгуків). Кількість відгуків повинна бути достатнім для повного опису поведінки об'єкта в аспекті даного дослідження. Відгуки повинні бути однозначними, кількісними і мати ясний фізичний зміст. Однозначність необхідна для запобігання невизначеності при постановці експерименту. Кількісність означає можливість вимірювання даного параметра за допомогою зіставлення його величини з еталоном. Ясний фізичний зміст параметра необхідний при інтерпретації результатів експерименту. Фізично безглуздими часто виявляються безрозмірні критерії подібності в модельних експериментах, що не дозволяє зрозуміти механізм досліджуваного явища.

Вибір незалежних змінних (факторів). Як і відгуки, фактори повинні бути однозначними, кількісними і мати ясний фізичний зміст. Однак при їх виборі немає таких жорстких обмежень по кількості, як при виборі відгуків. В ідеалі експеримент повинен враховувати всі фактори, що впливають на даний об'єкт дослідження [32].

Рішення про те, які параметри і структурні допущення вважати фіксованими показниками моделі, а які експериментальними чинниками, залежить від цілей дослідження. На рисунку 2.1 представлена схема експериментальних досліджень.

2.2.2 Визначення факторів експерименту

Після вибору параметра оптимізації потрібно розглянути всі фактори, які можуть впливати на процес. Якщо який-небудь істотний фактор виявиться неврахованих або прийме довільні значення, контрольовані експериментатором, то це значно збільшить похибку досліду. При підтримці цього фактору на певному рівні може бути отримано неправильне уявлення про оптимум, так як немає гарантії, що отриманий рівень є оптимальним [33].

Факторами даного дослідження будуть:

– модель процесора сервера, на якому розвернута система;

– модель процесора балансувальника навантаження;

– тип материнської плати;

– обсяг оперативної пам’яті;

– обсяг пам’яті жорсткого диску;

– тип материнської плати;

– тип бази даних;

– тип операційної системи;

– кількість одночасних користувачів.

Рисунок 2.1 – Схема експериментальних досліджень

2.2.3 Визначення відгуків експерименту

Параметром оптимізації є характеристика мети, задана кількісно. Параметр оптимізації це реакція (відгук) на вплив чинників, які визначають поведінку обраної системи. Він повинен бути кількісним, задаватися числом.

Реальні об'єкти або процеси, як правило, дуже складні. Вони часто вимагають одночасного обліку декількох, іноді дуже багатьох, параметрів. Кожен об'єкт може характеризуватися всією сукупністю параметрів, або одним – єдиним параметром оптимізації. В останньому випадку інші характеристики процесу вже не виступають в якості параметра оптимізації, а служать обмеженнями [33].

Параметрами оптимізації для даного дослідження є:

– час з моменту посилання запиту до його отримання користувачем;

– продуктивність системи при одночасному зверненні сотні користувачів.

2.3 Методика обробки результатів експерименту

2.3.1 Статистична обробка

В результаті виконання експерименту будуть отримані чисельні значення набору метрик для кожного протестованого набору конфігурацій серверу. При обробці результатів експериментів в вимірювальній техніці виникає необхідність оцінки характеристик випадкової величини. Необхідність попередньої статистичної обробки та оцінки експериментальних даних викликана тим, що в фізичному експерименті поряд з керованими вхідними факторами практично завжди присутні некеровані випадкові впливи, тому значення вихідних показників є величинами випадковими. Крім того, завдання параметрів режиму завжди виконується з якоюсь помилкою. Тобто, навіть у випадку постійних значень вхідних факторів, значення вихідних показників будуть різні. Для побудови математичних моделей може бути використано тільки ті результати експерименту, де вплив керованих вхідних факторів буде істотно вище впливу

випадкових [34].

Сутність статистичної обробки експериментальних даних полягає в перевірці достовірності результатів експерименту, розрахунку статистичних характеристик та оцінки приналежності результатів до однієї генеральної сукупності. Тільки після статистичної обробки можливо зробити висновок про доцільність подальшого використання результатів експерименту. При цьому також одержується додаткова інформація про процес, який досліджується.

2.3.2 Попередня обробка даних

Для кожного набору значень по одній метриці отримаємо наступні статистичні характеристики:

– мінімальне значення метрики;

– максимальне значення метрики;

– математичну очікування – середнє значення випадкової величини:

– середньоквадратичне відхилення – показник розсіювання значень випадкової величини відносно її математичного очікування.

2.3.3 Гістограми

Гістограма – це спосіб графічного представлення табличних даних. Являє собою діаграму, що складається з прямокутників без розривів між ними. Кількісні співвідношення деякого показника представлені у вигляді прямокутників, площі яких пропорційні. Найчастіше для зручності сприйняття ширину прямокутників беруть однакову, при цьому їх висота визначає співвідношення відображуваного параметра.

На основі даних, упорядкованих у рядки або стовпці на аркуші, можна побудувати гістограму. Гістограми ілюструють порівняння окремих елементів. Зазвичай у гістограмі категорії відкладаються по осі абсцис, а значення – по осі ординат.

Так як значення метрик є випадковими величини, слід оцінити характер їх розподілу. Це необхідно для вибору коректних методів їх подальшої обробки, тому що випадкові величини з різною функцією розподілу вимагають різних підходів до їх аналізу.

Для кожного набору метрик якості вихідного коду досліджуваної системи побудуємо гістограму.

2.3.4 Кореляція

Кореляція – статистичний взаємозв'язок двох або кількох випадкових величин (або величин, які можна з деякою допустимої ступенем точності вважати такими). При цьому зміни значень однієї або декількох з цих величин супроводжують систематичного зміни значень іншої або інших величин. Математичної мірою кореляції двох випадкових величин служить коефіцієнт кореляції.

Деякі види коефіцієнтів кореляції можуть бути позитивними або негативними. У разі позитивного коефіцієнта кореляції передбачається, що можна оцінити лише сам факт наявності або відсутності взаємозв'язку випадкових величин, в разі негативного можна також визначити її напрямок. Збільшення значення першої випадково величини в такому випадку веде до зменшення значення другої. Відповідно, позитивне значення коефіцієнта кореляції в таких умовах вказує на такий зв'язок, при якій збільшення першої змінної пов'язано зі збільшенням і другий. Крім того, можливо ситуація, коли коефіцієнт кореляції близький до нуля, – це говорить про відсутність статистичного взаємозв'язку між аналізованими випадковими величинами.

Метод обчислення коефіцієнта кореляції залежить від виду шкали, до якої відносяться змінні. Так, для вимірювання змінних з інтервального і кількісної шкалами необхідно використовувати коефіцієнт кореляції Пірсона. Якщо щонайменше одна з двох змінних має порядкову шкалу, або не є нормально розподіленою, необхідно використовувати рангову кореляцію Спірмена.

Коефіцієнт лінійної кореляції Пірсона [35] – найбільш часто використовуваний коефіцієнт кореляції. Призначений для розрахунку сили і напряму лінійної залежності між змінними дослідження. Передбачається, що змінні виміряні в інтервального шкалою або в шкалі відносин.

Коефіцієнт кореляції набуває значень від −1 до 1. Значення +1 означає, що залежність між X та Y є лінійною, і всі точки функції лежать на прямій, яка відображає зростання Y при зростанні X. Значення −1 означає, що всі точки лежать на прямій, яка відображає зменшення Y при зростанні X. Якщо коефіцієнт кореляції Пірсона = 0, то саме лінійної кореляції між змінними немає.

Якщо досліджувані пари змінних розмістити на координатному полі, і значення змінних розглядати як координати точки, то коефіцієнт кореляції буде відображати середня відстань від точок до усередненої прямий: чим ближче точки з усередненою прямий, тим вище кореляція (див. рисунок 2.2).

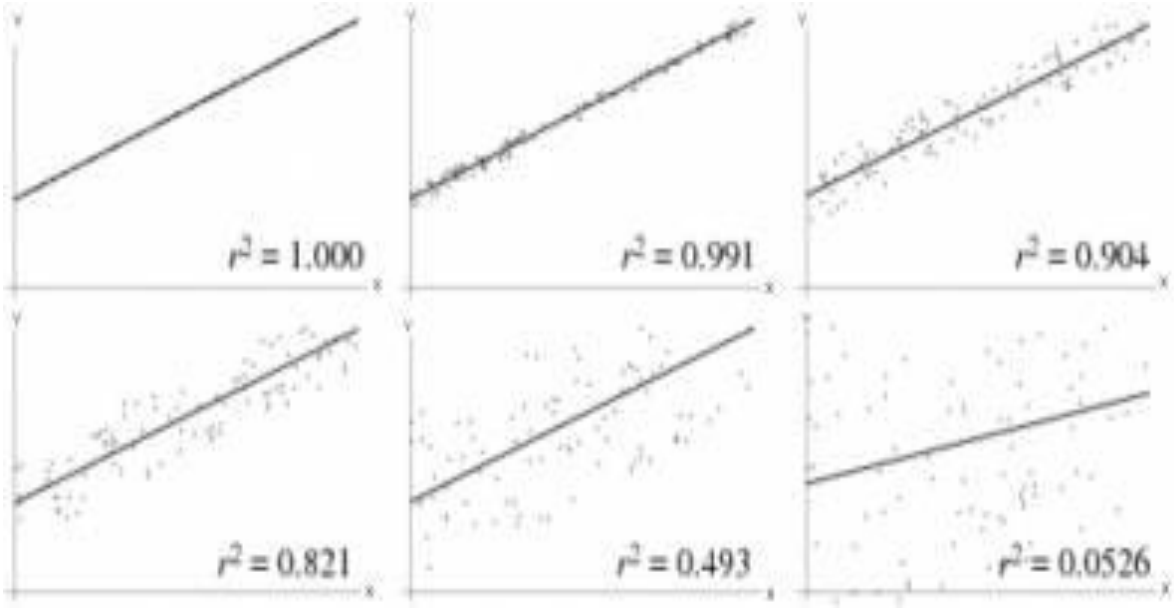

Рисунок 2.2 – Коефіцієнт кореляції Пірсона

Розрахунок коефіцієнта кореляції Пірсона припускає, що змінні X і Y розподілені нормально.

Рангова кореляція Спірмена – найпростіший спосіб визначення міри зв'язку між факторами. Назва методу свідчить про те, що зв'язок визначають між

рангами, тобто рядами одержаних кількісних значень, ранжованих у порядку зниження або зростання. Треба мати на увазі, що, по-перше, рангову кореляцію не рекомендовано проводити, якщо зв'язок пар менший чотирьох і більший двадцяти; по-друге, рангова кореляція дає змогу визначати зв'язок і в іншому випадку, якщо значення мають напівкількісний характер, тобто не мають числового виразу, відображають чіткий порядок прямування цих величин; по-третє, рангову кореляцію доцільно застосовувати в тих випадках, коли достатньо одержати приблизні дані.

Коефіцієнт кореляції рангів Спірмена відноситься до непараметричних показників зв'язку між змінними, вимірюваними в рангової шкалою. При розрахунку цього коефіцієнта не потрібно ніяких припущень про характер розподілів ознак у генеральній сукупності. Цей коефіцієнт визначає ступінь тісноти зв'язку порядкових ознак, які в цьому випадку є ранги порівнюваних величин.

Величина коефіцієнта кореляції Спірмена також лежить в інтервалі [-1; 1]. Він, як і коефіцієнт Пірсона, може бути позитивним і негативним, характеризуючи спрямованість зв'язку між двома ознаками, виміряними в рангової шкалою.

Як видно з визначень, при різному характері вхідних даних, використовуються різні методи підрахунку коефіцієнта кореляції. При роботі з метриками програмного забезпечення досліджувані значення лежать в просторі інтервального шкали вимірювань і шкали відносин [36].

Конкретний метод підрахунку коефіцієнта кореляції для поточного дослідження буде обраний після вивчення характеру розподілу значень.

2.3.5 Аналіз використання методу Weighted Round Robin для розвантаження трафіку на сервери

Weighted Round Robin (WRR) – метод планування, який використовується в мережах для планування потоків даних, але також використовується для

планування процесів.

Weighted Round Robin – це узагальнення планування круглолистого обертання. Він обслуговує набір черг або завдань. Тоді як кругообіг переходить на черги / завдання та надає одну можливість обслуговування за цикл, зважена кругла робота пропонує кожному фіксовану кількість можливостей, робочу вагу, встановлену при конфігурації. Потім це дозволяє впливати на частку потужності, отриману кожною чергою / завданням.

У комп'ютерних мережах можливістю обслуговування є видача одного пакету, якщо вибрана черга не порожня. Якщо всі пакети мають однаковий розмір, WRR є найпростішим наближенням узагальненого обміну процесорами (GPS).

WRR – це модифікація Round Robin, яка враховує вимоги якості обслуговування до кожного класу. Статичні ваги використовуються для розрізнення вимог якості обслуговування кожного класу, а також для визначення необхідної пропускної здатності. Кожен клас має список черг, і кожен з них має відповідний лічильник ваги. Якщо для обслуговування обрана черга, її (статична) вага присвоюється її лічильнику, який визначає кількість пакетів, які потрібно передавати в раунді. Вага кожної черги обчислюється з урахуванням вимог якість обслуговування кожного класу. Використовуються різні параметри якість обслуговування: пріоритет руху, MSTR (максимальна стабільна швидкість руху) та MRTR (мінімальна зарезервована швидкість руху). Алгоритм обслуговує кожну чергу в режимі RR від найвищого до нижчого пріоритетного класу. Пакети з кожної черги подаються в кожному раунді від тих, чия вага не має негативного значення. Якщо пакет надісланий, лічильник ваги черги зменшується. Пакети продовжують надсилати, поки лічильник не досягне нуля або черги в усіх класах не будуть порожніми. Нарешті, всі лічильники черг скидаються до своїх значень ваги. Тому WRR є справедливим, коли всі черги кожного класу використовують рівні ваги та рівні пакети. Однак використання статичного зважування для обслуговування класів зі змінними рівнями пріоритетності під час вхідного трафіку може призвести до швидкого зростання

їх черг і, отже, до подальшої затримки пакетів, а також до втрати пакетів. Також несправедливим є черги зі змінною довжиною пакетів. Крім того, на відміну від черг у режимі реального часу, яким надається більша вага, чергам у режимі реального часу надаються менші ваги, тим самим накладаючи затримку на нижчі пріоритетні класи (див. рисунок 2.3).

Модифікований зважений rounded robin (MWRR) – це варіант, запропонований для подолання цього штрафу для нижчих пріоритетних класів. Він починається з обчислення ваги WRR для кожної черги, виходячи з пріоритету та кількості всіх непорожніх черг. Це зважування дозволяє кожній черзі передавати певну кількість пакетів за один сервісний раунд. Таким чином, загальна кількість пакетів, які планувальник WRR може доставити за цикл обслуговування, фіксується. Планувальник MWRR множить кожен лічильник ваги WRR на постійне ціле значення, щоб збільшити кругообіг обслуговування. Цей алгоритм зменшує середню затримку і збільшує середню пропускну здатність, особливо для нижчих пріоритетних класів, подовжуючи розмір сервісного раунду над WRR. Однак використовуваний множник є статичним, що може призвести або до збільшення затримки, або до зниження пропускної здатності, якщо він неправильно обраний.

Рисунок 2.3 – Візуалізація роботи алгоритму WRR

Адаптивний зважений round robin (AWRR) – це ще один варіант, запропонований, щоб уникнути проблеми голодування нижчих пріоритетних класів. Він використовує два планувальники: планувальник вводу має на меті визначити пріоритетність відеопотоків з вищою якістю (HD та SD) і складається з буфера високого пріоритету (HP) для, наприклад, UGS, ertPS і rtPS, і низького пріоритету (LB) буфер, який включає rtPS-web-телебачення, rtPS-mobile-TV, nrtPS і BE. З іншого боку, мета планувальника вихідних даних - регулювати потоки даних та керувати всіма класами обслуговування. Обидва планувальники використовують AWRR для регулювання зважування класів обслуговування. Для кожного класу встановлюється порогове значення, яке запускає динамічне регулювання ваги при кожному перевищенні порогу. Черги HP мають нижчі пороги, ніж черги LP. Планувальник введення контролює ваги черг HP та LP залежно від завантаження трафіку і розміру буфера. Контроль ваг з черги дозволяє алгоритму домогтися мінімальної пропускної здатності трафіку BE в умовах напруги в мережі, а також надавати переважне трафік трафіку HP. Динамічна вага розраховується за допомогою алгоритму, в якому ваги повинні бути позитивними, щоб забезпечити мінімальну пропускну здатність для трафіку LP, а довільна константа використовується для переваги класів трафіку HP, таких як UGS та rtPS. Однак він використовує складний розрахунок для обчислення ваг і застосовується в мережах WiMAX multihop.

На сьогоднішній день жоден із цих алгоритмів не виявив здатності до подальшого зменшення затримок та втрат пакетів, одночасно покращуючи пропускну здатність в мережах IEEE 802.16, враховуючи навантаження трафіку в кожній черзі. Пропонований нами алгоритм планування пакетів відрізняється від цих зусиль тим, що він динамічно коригує статичні ваги відповідно до завантаження трафіку кожної черги у кожному сервісному класі.

Наприклад, якщо $q_{i,1}$, 1 являє собою чергу i (i = 1, 2,…, n) у 1 раунді, тоді планувальник на рисунку 2.4 планує пакети як:

$$\rightarrow q_{2,1} \rightarrow q_{3,1} \rightarrow q_{1,1} \rightarrow q_{2,1} \rightarrow q_{3,1} \rightarrow q_{4,1}.$$

Завершення вищезазначеної послідовності становить раунд. Таким чином,

планувальник передає два пакети з кожної черги за один раунд.

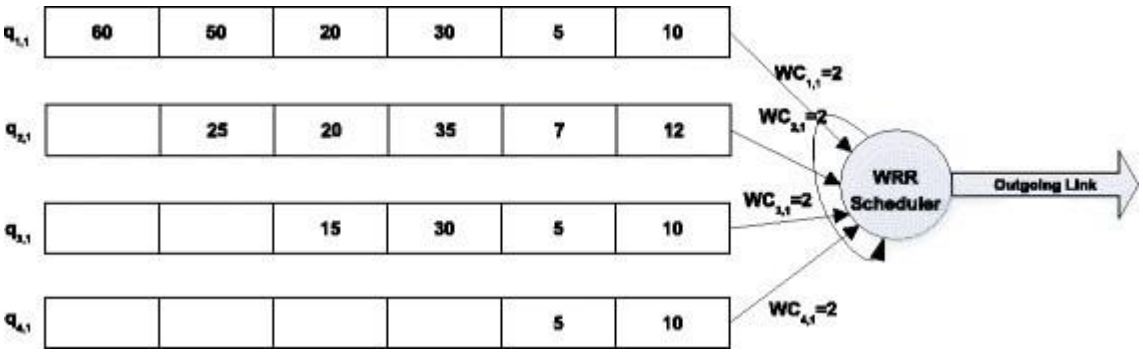

Рисунок 2.4 – Ілюстрація алгоритму WRR: показано стани черги та планувальника.

Щоб вирішити описані вище недоліки WRR, ми пропонуємо алгоритм, який змінює статичну вагу кожного класу на динамічну. Вони обчислюються динамічно відповідно до характеристик трафіку черги та статичної ваги кожного класу на початку кожного скидання лічильника. Вони відслідковують зміни трафіку кожної черги для зменшення затримки та втрати пакетів, а також для покращення пропускної здатності в мережі IEEE 802.16 під вхідним пакетним трафіком.

Розрахунок динамічної ваги – це чотири стадійний процес. По-перше, ми обчислюємо дисперсію навантаження: припустимо, що в турі k навантаження черги i в мережі IEEE 802.16 задається $\mathrm{load_i}$, k, де 1≤i≤n для класу з n чергами. Варіантність черг у круглі k (α_k) обчислюється як:

$$\alpha_k = \frac{1}{n}\sum_{i=1}^{n}(load_{i,k} - \langle load_k \rangle)^2$$

де $\langle load_k \rangle$ – середнє значення від круглого k,

$$\langle load_k \rangle = \frac{1}{n}\sum_{i=1}^{n} load_{i,k}$$

По-друге, середньоквадратичні похибки квадратного округленого k (β_k) визначаються як:

$$\beta_k = \sqrt{\alpha_k}$$

По-третє, динамічний коефіцієнт черги і в круглому k ($\omega_{i,k}$) обчислюється як:

$$\omega_{i,k} = \left\lceil \frac{load_{i,k}}{\beta_k + 1} \right\rceil$$

Ми беремо максимальне (⌈y⌉ мінімальне ціле число ≤ y), оскільки ($\omega_{i,k}$) – динамічний коефіцієнт динамічної ваги і повинен бути цілим числом, оскільки динамічна вага визначає кількість пакетів, які планувальник може передавати з черги за один лічильник скидання (див. рисунок 2.5).

Нарешті, динамічна вага черги і в круглому k ($\omega_{i,k}$) задається:

$\omega_{i,k} = \omega_{i,k} W_i$.

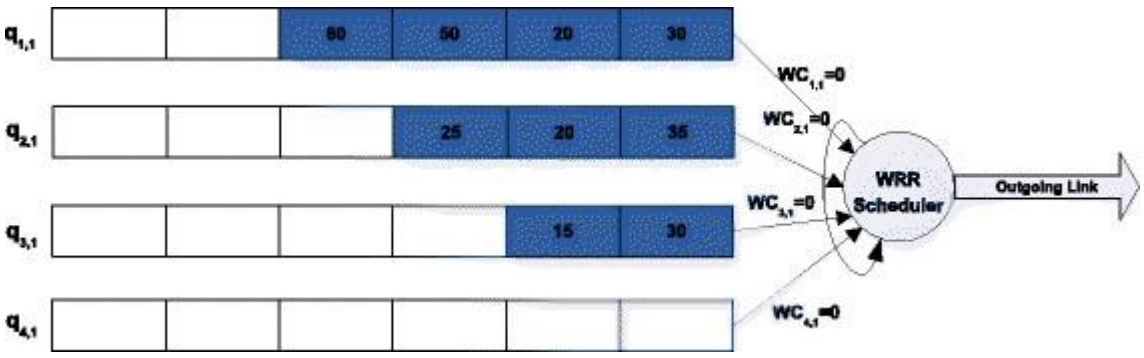

Рисунок 2.5 – Стан алгоритму WRR перед скиданням лічильника: Черги відображаються після вичерпання ваг.

2.3.6 Аналіз використання методу LeastConn для розвантаження трафіку на серверу

У галузевому стандартному алгоритмі балансування навантаження "Найменші з'єднання" використовується кількість поточних з'єднань до кожного екземпляра програми (члена) для прийняття рішення про балансування навантаження. Вибирається член з найменшою кількістю активних з'єднань.

Передумовою цього алгоритму є загальне припущення, що менше з'єднань (а отже, і меншої кількості користувачів) означає менше навантаження та, отже, більш високу продуктивність. Це операційна аксіома №2 на роботі – якщо продуктивність зменшується зі збільшенням навантаження, це означає, що продуктивність збільшується зі зменшенням навантаження.

Це було б правдою (і в перші дні балансування навантаження це було правдою), якби всі робочі навантаження в додатку вимагали однакових ресурсів. На жаль, це вже не відповідає дійсності, а результат – нерівномірний розподіл навантаження, що призводить до непередбачуваних коливань продуктивності в міру збільшення попиту (рисунок 2.6)

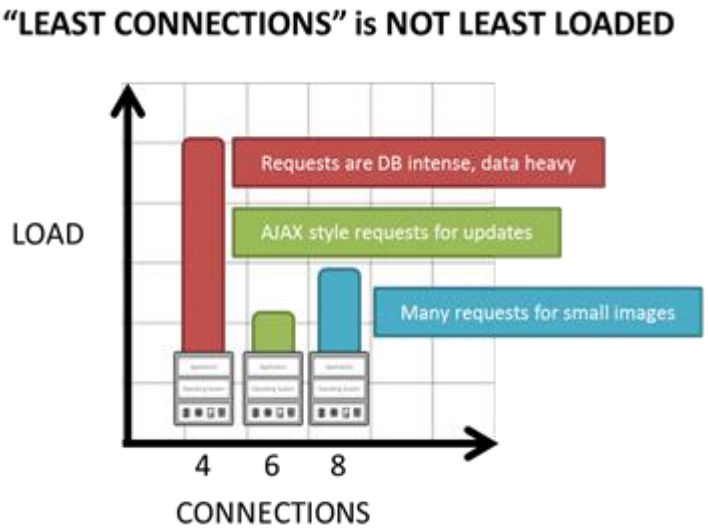

Рисунок 2.6 – Зображення пропорції кількості з’єднань до завантаження серверу

Розглянемо простий приклад: користувач, що входить у систему, займає хоча б один, якщо не більше запитів бази даних, щоб перевірити облікові дані, а потім оновити систему, щоб вказати на активність. Залежно від характеру програми, інші види внутрішньоприкладних заходів потребуватимуть різної кількості ресурсів. Деякі є важкою оперативною пам’яттю, інші – важким процесором, третім файлом або базою даних. Крім того, залежно від конкретного користувача, схема використання сильно відрізнятиметься. Сто користувачів можуть увійти в одну і ту ж систему (вимагає як мінімум десять підключень), але якщо всі вони відносно непрацюючі, система буде легко завантажена і працюватиме добре.

І навпаки, інший екземпляр програми може похвалитися лише 50 підключеннями, але всі п’ятдесят користувачів активно активні із запитами бази даних, що повертають великі обсяги даних. Система набагато більш завантажена, і продуктивність може вже починати страждати.

Коли наступний запит надходить, але балансир навантаження, використовуючи алгоритм "найменших з'єднань", вибере останнього члена, збільшуючи навантаження на цей член і, ймовірно, ще більше погіршуючи продуктивність.

Передумова алгоритму найменшого з'єднання полягає в тому, що екземпляр програми з найменшою кількістю з'єднань є найменш завантаженим. За винятком, це не так.

Єдиний спосіб дізнатися, який екземпляр програми є найменш завантаженим – це безпосередньо контролювати його системні змінні, збираючи використання процесора та пам'яті та порівнюючи їх з відомими максимумами. Для цього, як правило, потрібні або SNMP, агенти, або інші активні механізми моніторингу, які можуть надмірно оподатковувати систему самі по собі за рахунок споживання ресурсів.

Це важко для операцій, тому що "завантаженість додатків" просто занадто широке узагальнення. Безумовно, деякі програми більш важкі для вводу / виводу, ніж інші, а інші більш важкі для процесора або з'єднання. Але всі програми

мають як загальний профіль навантаження, так і профіль завантаження всередині додатків. Розуміння шаблонів використання – - профілю завантаження внутрішньої програми – важливо для того, щоб можна було визначити, як найкраще не тільки вибрати алгоритм балансування навантаження, але вказати будь-які обмеження, які можуть забезпечити кращу загальну продуктивність та використання потужностей під час виконання.

На рисунку 2.7 віртуальний сервер вибирає послугу для кожного вхідного з'єднання, вибираючи сервер з найменшою кількістю активних транзакцій.

З'єднання передаються наступним чином:

– Сервіс-HTTP-3 отримує перший запит, оскільки він не обробляє жодних активних транзакцій. Спочатку вибирається послуга без активної транзакції.

– Сервіс-HTTP-3 отримує другий і третій запити, оскільки служба має найменшу кількість активних транзакцій.

– Сервіс-HTTP-1 отримує четвертий запит Оскільки Service-HTTP-1 і Service-HTTP-3 мають однакову кількість активних транзакцій, віртуальний сервер використовує метод кругового обміну для вибору між ними.

– Сервіс-HTTP-3 отримує п'ятий запит.

– Service-HTTP-1 отримує шостий запит і так далі, поки і Service-HTTP-1, і Service-HTTP-3 не обробляють таку ж кількість запитів, що і Service-HTTP-2. У цей час прилад NetScaler починає переадресовувати запити на Service-HTTP-2, коли це найменш завантажений сервіс, або його черга з’являється в черзі круглих роботів. Якщо підключення до сервісу-HTTP-2 закриваються, воно може отримати нові з'єднання, перш ніж кожен з двох інших сервісів має 15 активних транзакцій.

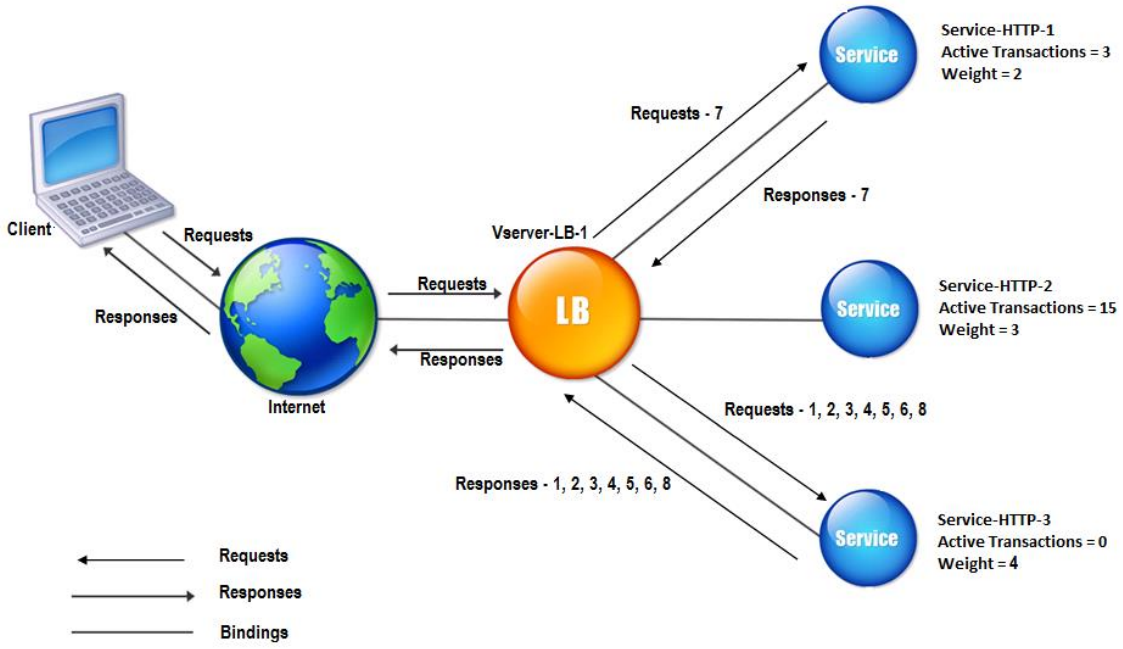

Рисунок 2.7 – Механізм балансувальника навантаження з використанням методу LeastConn (Least Connections)

Наступна таблиця пояснює, як з'єднання розподіляються в налаштуваннях балансування навантаження на три служби, описаному вище. Як для алгоритму Round Robin існує версія з вагами, що називається Weighted Round Robin, алгоритму так і для алгоритму LeastConn існує версія Weighted LeastConn**.** Як випливає з назви, внесок цього алгоритми полягають у тому, що він враховує змінну вагу.

Таблиця 2.1 – Розприділення з’єднань балансування навантаження

Вхідне з'єднання	Служба обрана	Поточна кількість активних з'єднань	Примітки
Запит-1	Service-HTTP-3; (N = 0)	1	Сервіс-HTTP-3 має найменше активне з'єднання.
Запит-2	Service-HTTP-3; (N = 1)	2	Сервіс-HTTP-3 має найменше активне з'єднання.
Запит-3	Service-HTTP-3; (N = 2)	3	

Продовження таблиці 2.1

Вхідне з'єднання	Служба обрана	Поточна кількість активних з'єднань	Примітки
Запит-4	Service-HTTP-1; (N = 3)	4	Сервіс-HTTP-1 і Service-HTTP-3 мають однакову кількість активних з'єднань.
Запит-5	Service-HTTP-3; (N = 3)	4	Сервіс-HTTP-1 і Service-HTTP-3 мають однакову кількість активних з'єднань.
Запит-6	Service-HTTP-1; (N = 4)	5	
Запит-7	Service-HTTP-3; (N = 4)	5	
Запит-8	Service-HTTP-1; (N = 5)	6	

Weighted LeastConn – спосіб знаходження вузла j з найменшим навантаженням в а кластер – це обчислення відношення числа з'єднання (C_j) і фіксовану вагу (W_j) по черзі, і вибирає вузол з мінімальним значенням. Однак одного не взяли до уваги, що серверів з різним вмістом TCP з'єднання мають різні накладні витрати, звідси і цей алгоритм втратить ефективність, коли накладні різниці між серверами не можна ігнорувати. У існуючому мережі з мультимедійними серверами, накладні витрати TCP з'єднання можуть бути досить вагомими, що впливають на нього робить для алгоритму зваженого найменшого зв'язку слід враховувати. І це є мотивація врахування змінної ваги в алгоритмі. Компоненти ваги: загалом кажучи, то параметри, що відображають здатність обробки вузла сервер – швидкість обробки процесора, швидкість холостого ходу, розмір пам'яті та ємності вводу / виводу. Визначаємо вагу вузла наступним чином:

$$W_j = \sqrt{(p_{mem} \times Q_{mem} \times R_{mem})^2 + (p_{cpu} \times V_{cpu} \times R_{cpu})^2}$$

де R_{mem} позначає швидкість холостого ходу пам’яті вузла

R_{cpu} позначає швидкість холостого ходу вузла CPU

Q_{mem} позначає розмір пам'яті одиницею є K

V_{cpu} позначає швидкість ЦП (одиниця - МГц),

p_{mem} і p_{cpu} позначають коефіцієнт пропорції

Після отримання рівняння ваги, ми можемо отримати алгоритми Least-Connection на основі змінної ваги. Припустимо, що існує n вузлів, кожен вузол i має вагу $W_i(i = 1, \dots, n)$ та живі з'єднання $C_i(i = 1, \dots, n)$, наступне мережеве з'єднання буде спрямоване на вузол j, і вузол j має використання нижче:

$$\frac{\mathrm{C}_j}{W_j} = \min\left(\frac{\mathrm{C}_j}{W_j}\right), \text{де } i = 1, \dots, n$$

Слід зазначити, що) процесорна швидкість і розмір пам'яті, в основному, постійні Необхідно врахувати пам'ять та швидкість холостого ходу процесора яка постійно змінюється.

2.4 JMeter

Apache JMeter – це проект Apache, який може бути використаний як засіб тестування навантаження для аналізу та вимірювання продуктивності різноманітних служб з акцентом на веб-додатках (рисунок 2.8).

JMeter можна використовувати як інструмент модульного тестування для з’єднань з базою даних JDBC, FTP, LDAP, веб-служб, JMS, HTTP, загальних TCP-з’єднань та ОС – власні процеси. Можна також налаштувати JMeter як монітор, хоча це, як правило, використовується як основне рішення моніторингу, а не як розширений моніторинг. Він також може бути використаний для деяких функціональних випробувань. Крім того, JMeter підтримує інтеграцію з Selenium, що дозволяє йому запускати сценарії автоматизації поряд з тестами продуктивності чи навантаження.

JMeter підтримує параметризацію змінних, твердження (перевірка

відповіді), файли cookie по потоку, змінні конфігурації та різноманітні звіти. Архітектура JMeter базується на плагінах. Більшість функцій "нестандартно" реалізовані за допомогою плагінів.

Рисунок 2.8 – Приклад виконання навантаження програмою JMeter

JMeter не є браузером, він працює на рівні протоколу. Що стосується веб-послуг та віддалених служб, JMeter виглядає як браузер (вірніше, кілька браузерів); однак JMeter виконує не всі дії, що підтримуються браузерами. Зокрема, JMeter не виконує JavaScript, знайдений на HTML-сторінках. Він також не відображає HTML-сторінки так, як це робить браузер (можна переглядати відповідь як HTML тощо, але таймінги не входять до жодних зразків, і одночасно відображається лише один зразок в одному потоці).

2.5 Azure services

Azure – це платформа хмарних обчислень Майкрософт. Azure - це постійно розширюваний набір хмарних служб, які допомагають вашій організації вирішувати поточні і майбутні бізнес-завдання. Azure надає можливість створення, розгортання додатків і управління ними в великих глобальних мережах за допомогою ваших улюблених засобів і платформ.

Хмарні обчислення – це надання обчислювальних служб через Інтернет за

допомогою цінової моделі з оплатою в міру використання. Іншими словами, це спосіб орендувати обчислювальні ресурси і сховище в сторонньому центрі обробки даних.

Замість того щоб використовувати фізичні ЦП і сховище в своєму центрі обробки даних, ви орендуєте їх на потрібний вам час. Всі турботи, пов'язані з підтриманням вашої інфраструктури, бере на себе постачальник хмари.

Azure надає більше 100 служб, що дозволяють вирішувати будь-які завдання: від запуску існуючих додатків в віртуальних машинах до освоєння нових парадигм програмного забезпечення, таких як інтелектуальні боти і змішана реальність.

Служби хмарних служб Azure є прикладом платформи як послуги (PaaS). Як і служба служб додатків Azure, ця технологія розроблена для підтримки програм, які масштабовані, надійні та недорогі в експлуатації. Так само, як Служба програм розміщується на віртуальних машинах (ВМ), так само як і хмарні служби Azure. Однак ви більше контролюєте віртуальні машини. Ви можете встановити власне програмне забезпечення на віртуальних машинах, які використовують хмарні служби Azure, і отримати до них доступ віддалено.

За допомогою хмарних служб Azure ви не створюєте віртуальні машини. Натомість ви надаєте файл конфігурації, який повідомляє Azure, скільки з них вам потрібно, наприклад, "три екземпляри веб-ролі" та "два екземпляри робочої ролі". Потім платформа створює їх для вас. Ви все ще вибираєте, якого розміру мають бути ті резервні віртуальні машини, але ви не створюєте їх явно самостійно. Якщо ваша програма повинна обробляти більше навантаження, ви можете попросити більше віртуальних машин, і Azure створює ці екземпляри. Якщо навантаження зменшується, ви можете закрити ці екземпляри та припинити платити за них.

Програма хмарних служб Azure, як правило, стає доступною для користувачів за допомогою двоетапного процесу. Розробник спочатку завантажує програму в інтерактивну зону платформи. Коли розробник готовий запустити програму в реальному часі, вони використовують портал Azure для

обміну постановкою на виробництво. Цей перехід між постановкою та виробництвом можна здійснити без простоїв, що дозволяє оновити запущену програму до нової версії, не заважаючи користувачам.

2.6 Алгоритм проведення експерименту

Для проведення порівняльного експерименту для дослідження розвантаження високонавантажених систем з використанням 2 методів балансування навантаження потрібно виконати наступні кроки:

– використовуючи Azure services створити різні конфігурації серверів (різні типи процесорів, оперативної пам’яті, материнської плати).

– також за допомогою Azure services створити 2 типи балансування навантаження. Один для тестування методу Weighed Round Robin, інший – для тестування методу Least Connections).

– використовуючи безкоштовну програму для вимірювання ефективності обробки запитів JMeter, провести тестування навантаженням, моделюючи одночасну кількість користувачів та та отримати результати.

– отримані результати обробити та створити гістограми для візуалізації даних.

– сформулювати висновки.

2.7 Висновки з розділу 2

В даному розділі були визначені параметри експерименту. Факторами експерименту являється конфігурація серверів, за допомогою яких буде проводитись експеримент та кількість одночасних користувачів, що будуть моделювати одночасний доступ до серверів. Відгуками експерименту являється результати продуктивності системи при одночасному зверненні сотні користувачі: час відгуку запиту від його посилання до отримання відповіді з серверу, кількість провалених запитів, що сервер не зміг обробити та пропускна

здатність серверу.

Були описані програмні забезпечення для моделювання, проведення експерименту та аналіз отриманих результатів. Для конфігурування тестового серверу використано сервіси Azure. Для проведення вимірювання навантаження використано безкоштовну програму JMeter. Для візуалізації даних, побудова гістограм в програмі Microsoft Excel.

Для моделювання експерименту були вирішені такі питання:

- вибір апаратних засобів;
- вибір програмного середовища для обробки даних;
- проектування процесу балансування навантаження.

Подальше дослідження передбачає проведення експерименту за двома зазначеними методами з метою визначення ефективності та доцільності їх практичного використання при оптимізації високонавантажених систем та надання рекомендацій щодо ефективного зменшення навантаження на сервери, на яких розгорнута високо-навантажена система.

Таким чином, було вирішено друге завдання дипломної роботи – планування експериментального дослідження для оцінки ефективності роботи високонавантажених систем.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТУ ДЛЯ ДОСЛІДЖЕНЬ РОЗВАНТАЖЕННЯ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ ТА ФОРМУВАННЯ ПРАКТИЧНИХ РЕКОМЕНДАЦІЙ

Популярні веб-сайти мають величезну кількість клієнтів, які обслуговуються кожну секунду. Для підтримки обслуговування на високому рівні, для серверів повинно бути виділено один або декілька балансувальника навантаження. В даному розділі буде порівняно порівняв ефективність двох методів балансування навантаження, Round Robin та Least Connection. Розробка системи відмово-стійкості виникає коли система перестає буди доступною при певній кількості одночасних запитів і в вирішенні цієї проблеми допомагає балансувальник навантаження Haproxy. При проведенні експерименту буде порівняно результати використання Least Connection та Weighted Round Robin методів балансувальника навантаження Haproxy.

3.1 Конфігурування серверів за допомогою Azure Services

Для проведення експерименту найперше, що потрібно зробити це налаштувати середовище, в якому і буде проходити сам експеримент. Середовище для реалізації трьох веб-серверів а також балансувальника навантаження можна побачити в таблиці 3.1. Конфігурація веб серверів в Azure Services зображено на рисунку 3.1.

Сервер може реплікувати базу даних з інших серверів та вміти проводити синхронізацію з двома іншими серверами. Система балансування може розприділяти трафік на три сервери. Конфігурація балансувальника навантаження зображена на рисунку 3.2.

Таблиця 3.1 – Конфігурація середовища, в якому буде проходити експеримент

Компонент	Сервер	
	Балансувальник навантаження	Сервер веб-сайту, який буде тестуватись
Тип процесора	Intel Pentium CPU G620 2.60 GHz	Intel Pentium CPU E2200 2.20 GHz
Тип материнської плати	ECS H61H2-M12	HP FT917AA-AR6
Обсяг оперативної пам’яті	4 GB	1.75 GB
Обсяг жорсткого диску	500 GB	50 GB
Стандарт передачі даних в комп’ютерній мережі	2 Fast Ethernet	1 Fast Ethernet

Крім того, використовується один блочний перемикач (Cisco Catalyst 2960) для з’єднання комп’ютерів, одного ноутбука (Asus A43SD), який використовується як клієнт, з якого посилатимуться запити.

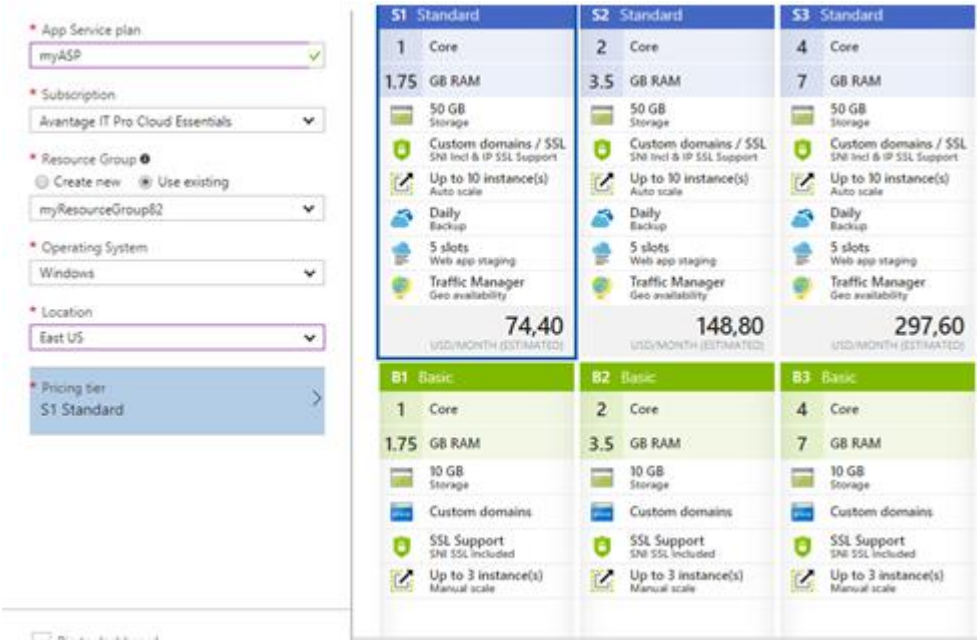

Рисунок 3.1 – Конфігурація веб-серверів з використанням Azure Services

Балансувальники навантаження Haproxy та Heartbeat так само як і три

інших сервери запущені на операційній системі Ubuntu server 16.04 LTS x64.

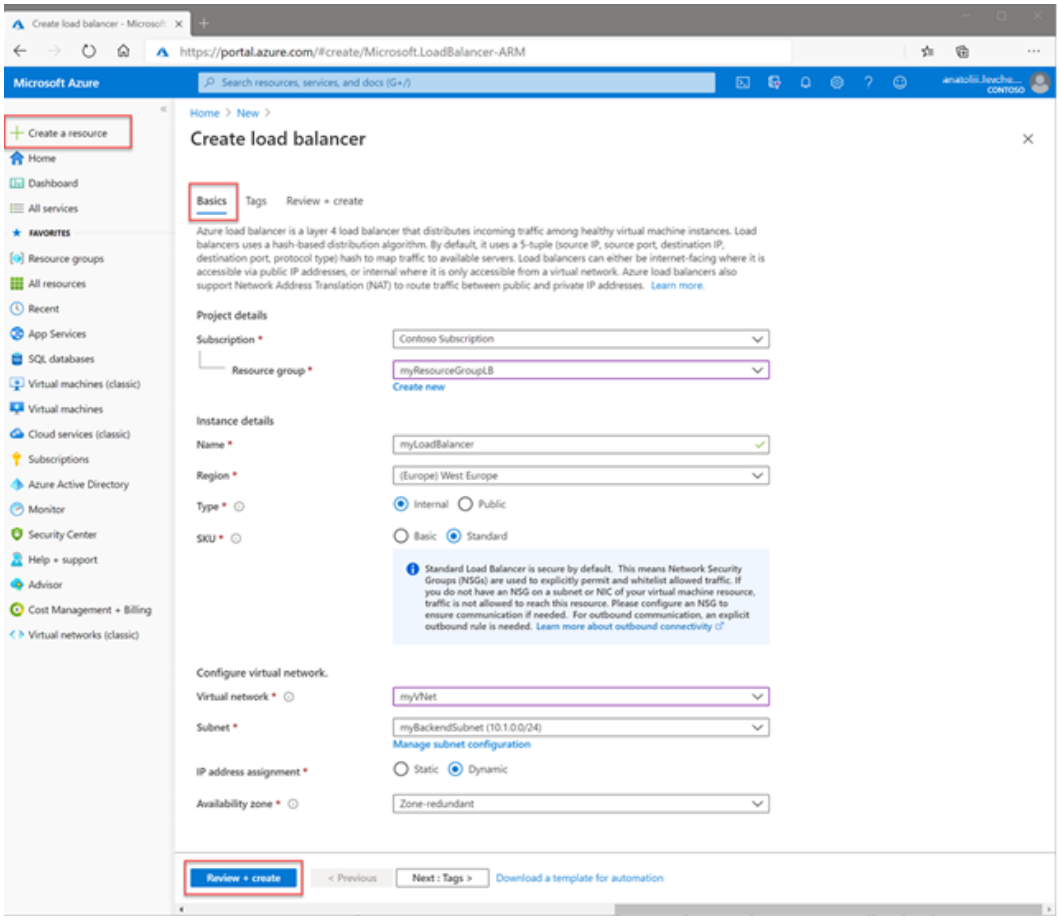

Рисунок 3.2 – Конфігурація балансувальника навантаження з допомогою Azure Service

База даних підтримується MySQL 5.7 та встановлена на трьох серверах Apache. Процес створення та конфігурації серверу бази даних зображено на рисунку 3.3.

Unison – це робоча програма, що використовується для підтримки файлової синхронізації на веб-серверах. Вона реплікує (реплікація – це механізм розподілу даних за вузлами, що дозволяє зберігати копії тих самих даних на різних вузлах мережі з метою прискорення пошуку і підвищення стійкості до відмов. Відношення чи фрагмент є реплікований, якщо його копії зберігаються на двох або більше вузлах) файли та каталоги відразу як вони змінюються та може зберігати реплікацію на різних хостах або різних жорстких дисках.

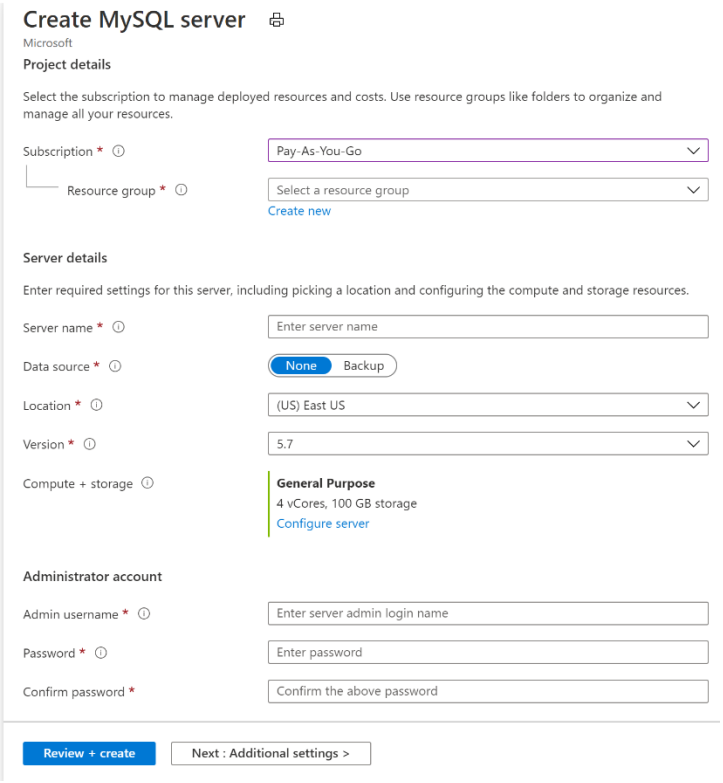

Рисунок 3.3 – Створення серверу бази даних MySQL

Haproxy використовується в основному як балансувальник навантаження та рідше використовується як зворотній проксі, в той час як Heartbeat використовується для відмово стійких цілей для забезпечення високої доступності мережі.

В експерименті беруть участь 3 веб-сервери, на яких розгорнуто систему, 2 балансувальника навантаження, 1 світч, та клієнт який здійснює запити. В експерименті веб серверами виступає зконфігуровані Azure Web Services, клієнтом виступає програма JMeter. Детальна фізична топологія цього експерименту зображена на рисунку 3.4 та пов’язана з нею логічна топологія зображена на рисунку 3.5.

Система реплікації в MySQL використовується як «master-master»: і перший, і другий сервери виступають як «master» і «slave», отже їх два «master» та два «slave». Схема реплікації така проводиться послідовно від сервера №1 до сервера №2 до сервер №3 і назад до сервера №1. «Master – master» також використовує синхронізацію.

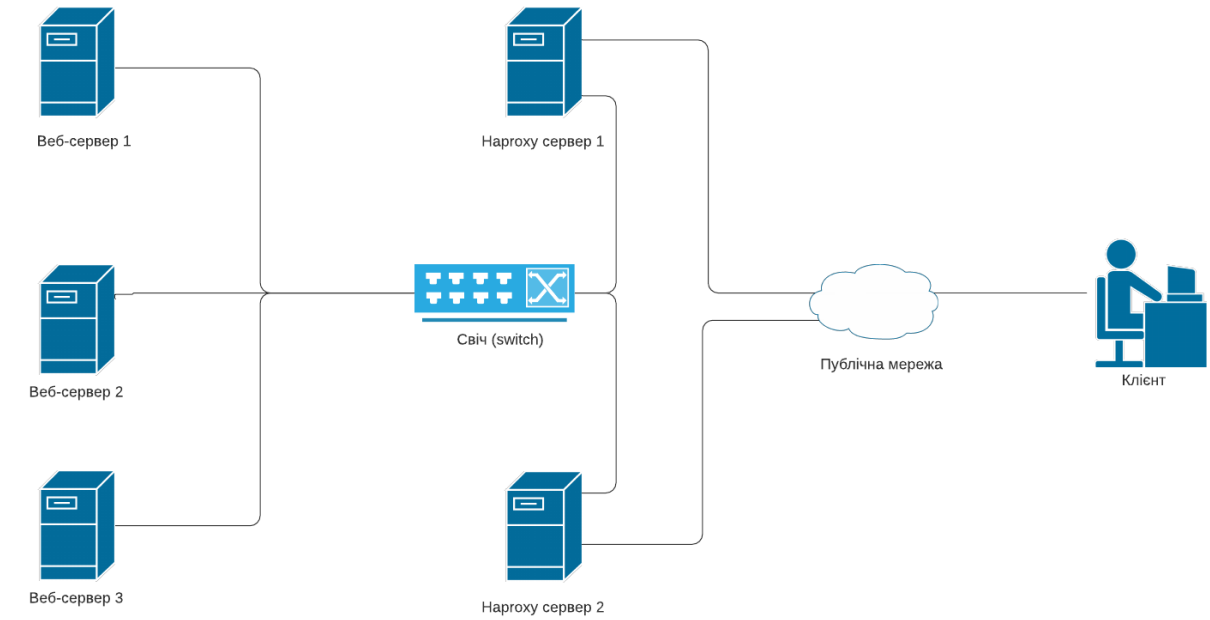

Рисунок 3.4 – Фізична топологія мережі, використана в експерименті

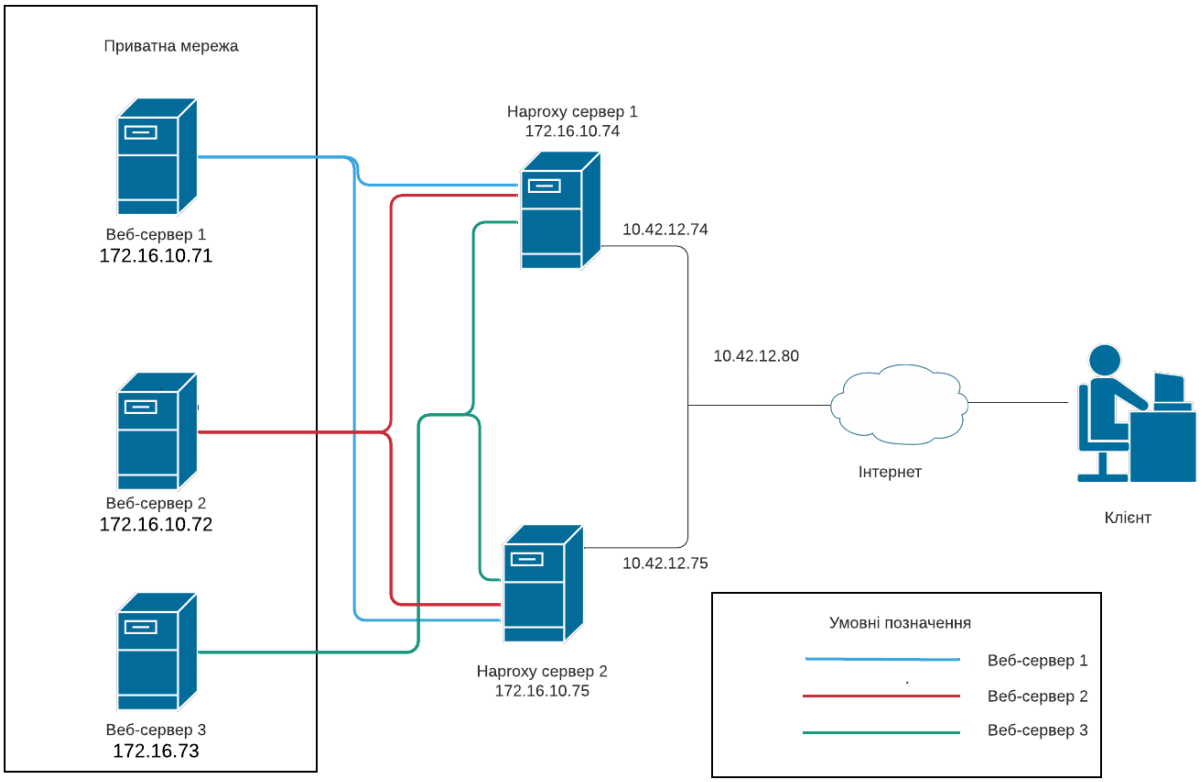

Рисунок 3.5 – Логічна топологія мережі, використана в експерименті

Unison використовує зірку топологія, де 1 сервер виконує роль концентратора, а решта виступають як дві спиці. Для відмовного дизайну передбачено два балансування навантаження сервери, які будуть використовуватися як активні – в режимі очікування, тому коли є один із цих двох серверів виходить з ладу, інший бере на себе обслуговування запитів. Час простою в це дослідження обмежується лише 10 мс, або відключення каналу зв'язку або вихід з ладу будь-якого із серверів балансу навантаження.

3.2 Тестування навантаженням за допомогою програми JMeter

Було проведено 2 сценарії для вимірювання продуктивності системи з допомогою програми JMeter:

– тестування навантаженням без використання будь-якого балансувальника навантаження (залежить від кількості з’єднань, швидкості з’єднання, рейта та timeouts (максимальний виділений час на обробку запитів)) (1);

– тестування навантаженням використовуючи балансувальник навантаженням з безлімітними підключеннями (Haproxy з використанням методів Least Connection (метод найменших з’єднань) та Round Robin) (2).

Для моделювання одночасної кількості користувачів, що посилають запити, створив 12 тестових груп. Приклад створення однієї зображено на рисунках 3.6 – 3.7. Кінцевий вигляд тестового сценарію зображений на рисунку 3.8.

Проводячи сценарій (1) було отримано результати у вигляді таблиці з програми JMeter Apache (приклад виконання однією частини з тесту зображено на рисунку 3.8). Всі зібрані дані показані в таблиці 3.2 та для візуалізації побудована гістограма (рис. 3.9). Зображено максимальну кількість підключень для першого сценарія (сценарій без використання балансувальника навантаженням). Без використання балансувальника навантаження запити для кожного з серверів різні. В цьому сценарії лише веб-сервер 1 обробляв майже всі

запити та, не справляючись з трафіком, повертав помилковий результат.

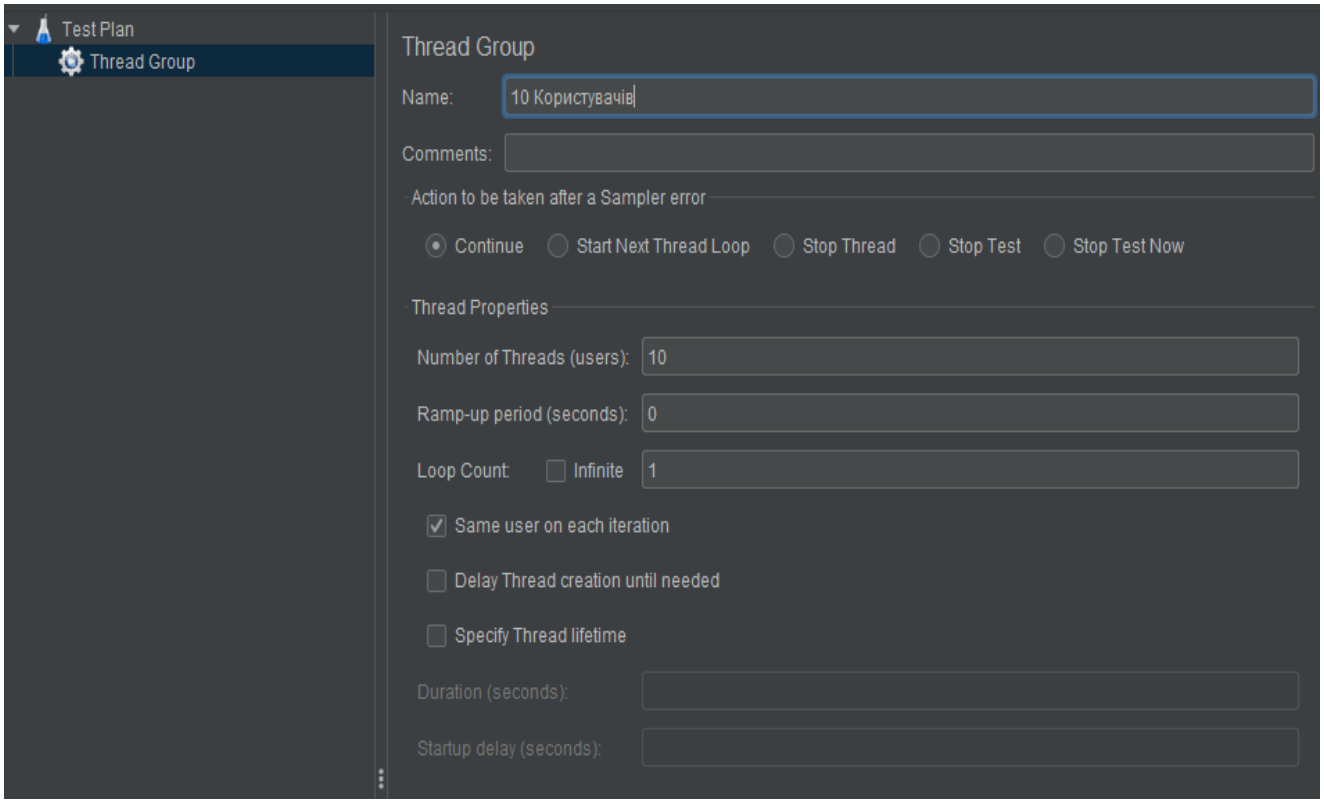

Рисунок 3.6 – Створення тестової групи для одночасного доступу 10 користувачів

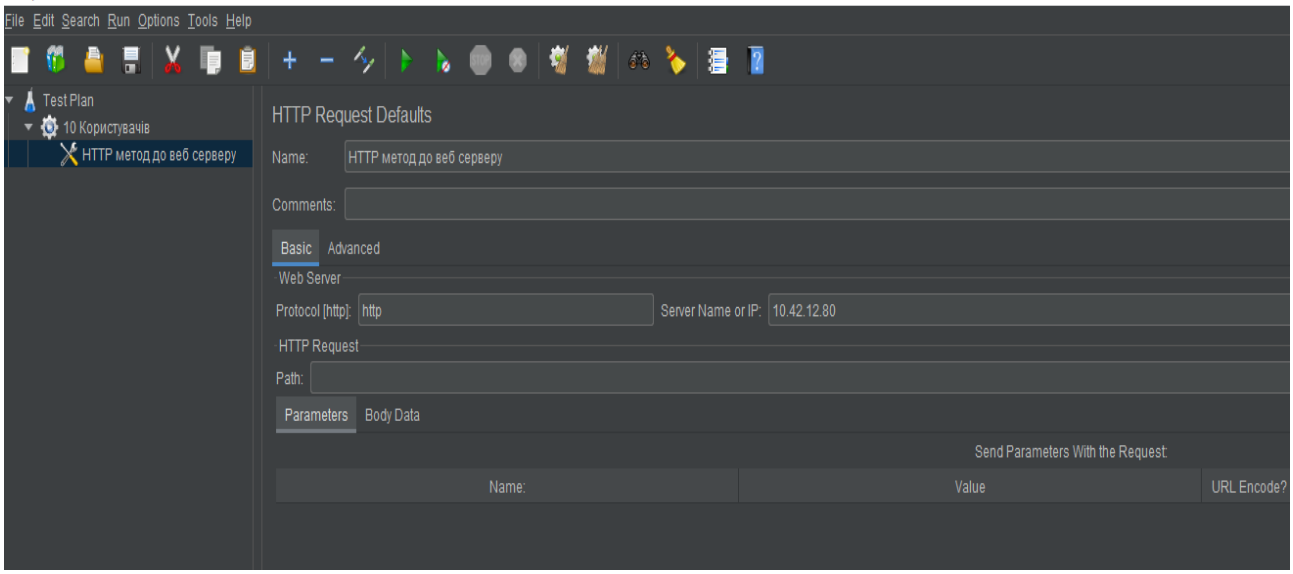

Рисунок 3.6 – Конфігурація відправки запитів в JMeter Apache

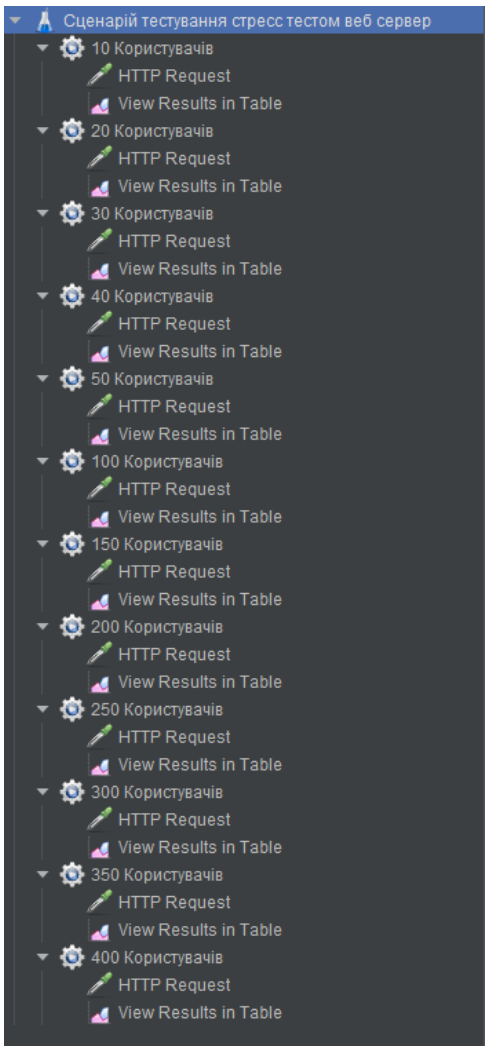

Рисунок 3.7 – Кінцевий вигляд тест сценарію в JMeter Apache

Трапляється несправність машини та потрібно уникати окремих точок відмови, коли це можливо. Це означає, що машини повинні мати копії, якщо існують репліки серверів, машинний збій не є повним відмовою вашої програми. Під час відмови машини клієнт повинен помічати якомога менше. Це проблема доступності: щоб уникнути відключень через відмову апаратного забезпечення, потрібно запустити кілька машин і мати можливість перенаправляти трафік з офлайн-систем якомога швидше. Існує горизонтальне масштабування для серверів – потрібно купувати найновішу та найкращу машину щороку, щоб не відставати від зростаючого попиту бази користувачів, а можна було придбати другу, щоб захистити себе від впевнених несправностей, але це стає дорогим. Є деякі випадки, коли масштабування по вертикалі є правильним вибором, але для переважної більшості навантажень

веб-додатків це не економічний вибір. Чим більше відносної потужності має машина за час, за який вона випущена, тим більша премія буде стягуватися за її потужність.

Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
10 Користувачів 1-6	HTTP Request	1760		122682	127	1741	55
10 Користувачів 1-9	HTTP Request	1826		122652	127	1810	58
10 Користувачів 1-2	HTTP Request	1848		122652	127	1820	56
10 Користувачів 1-7	HTTP Request	1875		122662	127	1851	60
10 Користувачів 1-1	HTTP Request	1925		122662	127	1900	60
10 Користувачів 1-5	HTTP Request	1997		122662	127	1969	67
10 Користувачів 1-4	HTTP Request	2003		122667	127	1924	59
10 Користувачів 1-10	HTTP Request	2011		122667	127	1992	58
10 Користувачів 1-3	HTTP Request	2038		122692	127	1947	59
10 Користувачів 1-8	HTTP Request	2074		122677	127	1969	58

Рисунок 3.8 – Приклад отриманих даних з сценарія (1)

Ці негаразди породили необхідність розподілу робочого навантаження на декількох машинах. Усі користувачі хочуть, щоб ваші послуги були швидкими та надійними, а потрібно надати їм якісну послугу з найвищою рентабельністю інвестицій. Балансувальники навантаження допомагають вирішити проблеми з продуктивністю, економією та доступністю.

Таблиця 3.2 – Отримані дані в ході виконання сценарія (1)

Кількість запитів	Веб-сервер 1	Веб-сервер 2	Веб-сервер 3
100	0	0	0
200	0	0	0
300	0	0	0
400	7	0	0
500	43	0	0
1000	91	0	0
1500	135	0	0
2000	177	0	0
2500	223	0	0
3000	252	0	0
3500	333	5	2
4000	374	48	46

Рисунок 3.9 – Візуалізація даних отриманих в ході виконання сценарія (1)

Для сценарія (2) було проведено тест, який показав швидкість обробки запитів. Кількість запитів збільшується до того моменту, де веб-сервіс перестає обробляти ці запити (деякі запити повертається з тайм-аутом).

Таблиця 3.3 – Максимальний рейтинг з’єднань для кожного з методу

Кількість запитів	Round Robin	LeastConn
100	10	10
200	19	20
300	20	30
400	20	39
500	20	50
1000	20	85
1500	25	50
2000	35	38
2500	41	50
3000	50	50
3500	58	58
4000	65	65

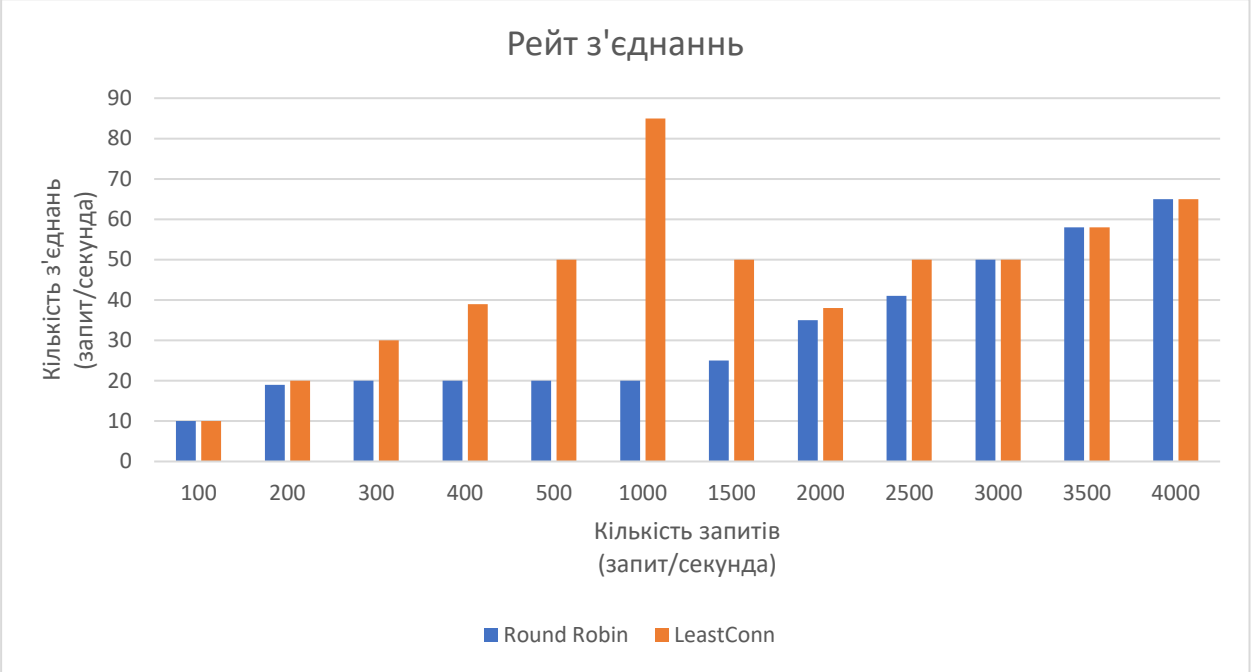

Рисунок 3.10 – Максимальний рейтинг з’єднань для кожного з алгоритму

Проведений тест показує рейтинг з’єднання для кожного з алгоритму. Максимальна загрузка серверу для кожного з алгоритму – це 65 з’єднань на секунду.

Тест напрямлений на визначення часу відгуку запиту для кожного з алгоритму зображений на рисунку 3.11. Алгоритм Round Robin показав найкращі результати при обробці трафіку, але при великих кількостях запитів. При малих кількостях показав кращий результат метод балансування навантаження LeastConn – метод найменших з’єднань.

При тестуванні пропускної здатності серверів з використанням двох методів балансування навантаження були показані результати, зображені на рисунку 3.11.

Таблиця 3.4 – Пропускна здатність серверу з використанням алгоритмів балансування навантаження

Кількість запитів	Round Robin	LeastConn
100	100	100
200	195	200
300	300	350

Продовження таблиці 3.4

Кількість запитів	Round Robin	LeastConn
400	210	400
500	200	500
1000	200	850
1500	210	450
2000	275	385
2500	325	395
3000	395	425
3500	425	500
4000	500	565

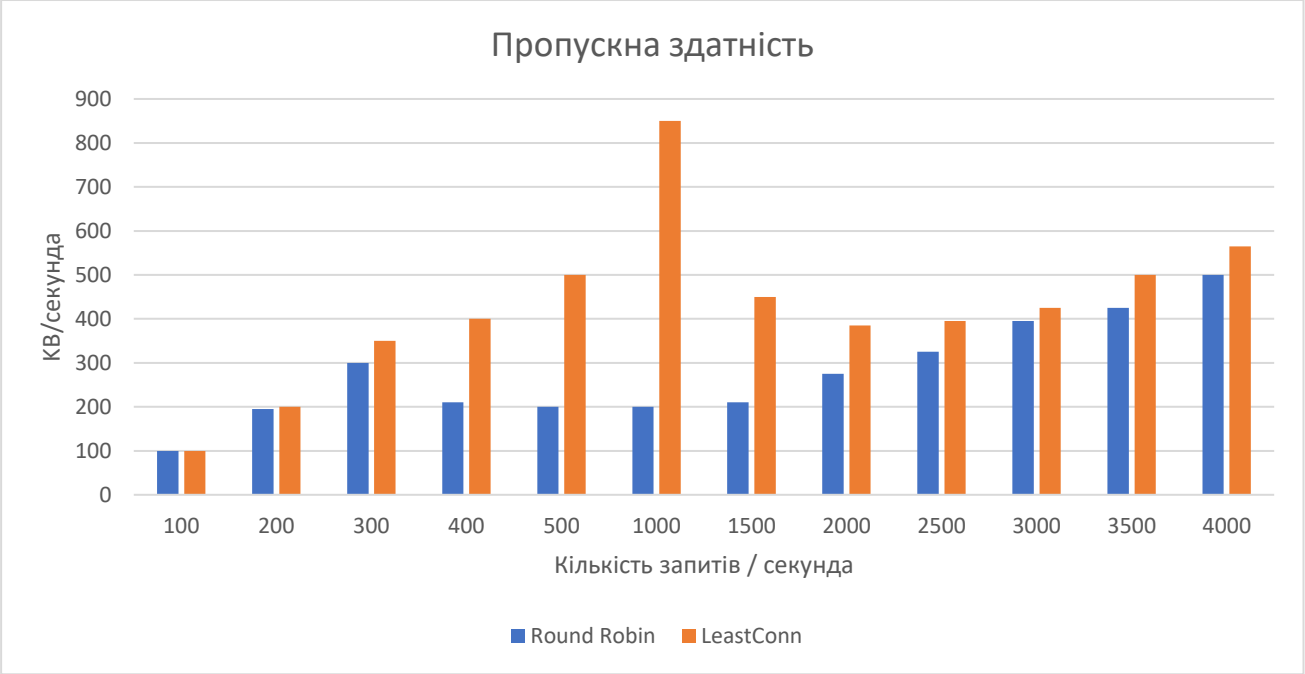

Рисунок 3.11 – Пропускна здатність серверу з використання алгоритмів Round Robin та Least Connection

На рисунку 3.12 зображено графік, який показує кількість «невдалих» (failed) запитів в сценарії (2). Алгоритми Least Connection та Round Robin показали, що обидва алгоритми працюють майже на одному рівні, але метод Least Connection продуктивніший за інший.

Таблиця 3.4 – Кількість «невдалих» запитів для кожного з алгоритму

Кількість запитів	Round Robin	LeastConn
100	100	100
200	195	200
300	300	350
400	210	400
500	200	500
1000	200	850
1500	210	450
2000	275	385
2500	325	395
3000	395	425
3500	425	500
4000	500	565

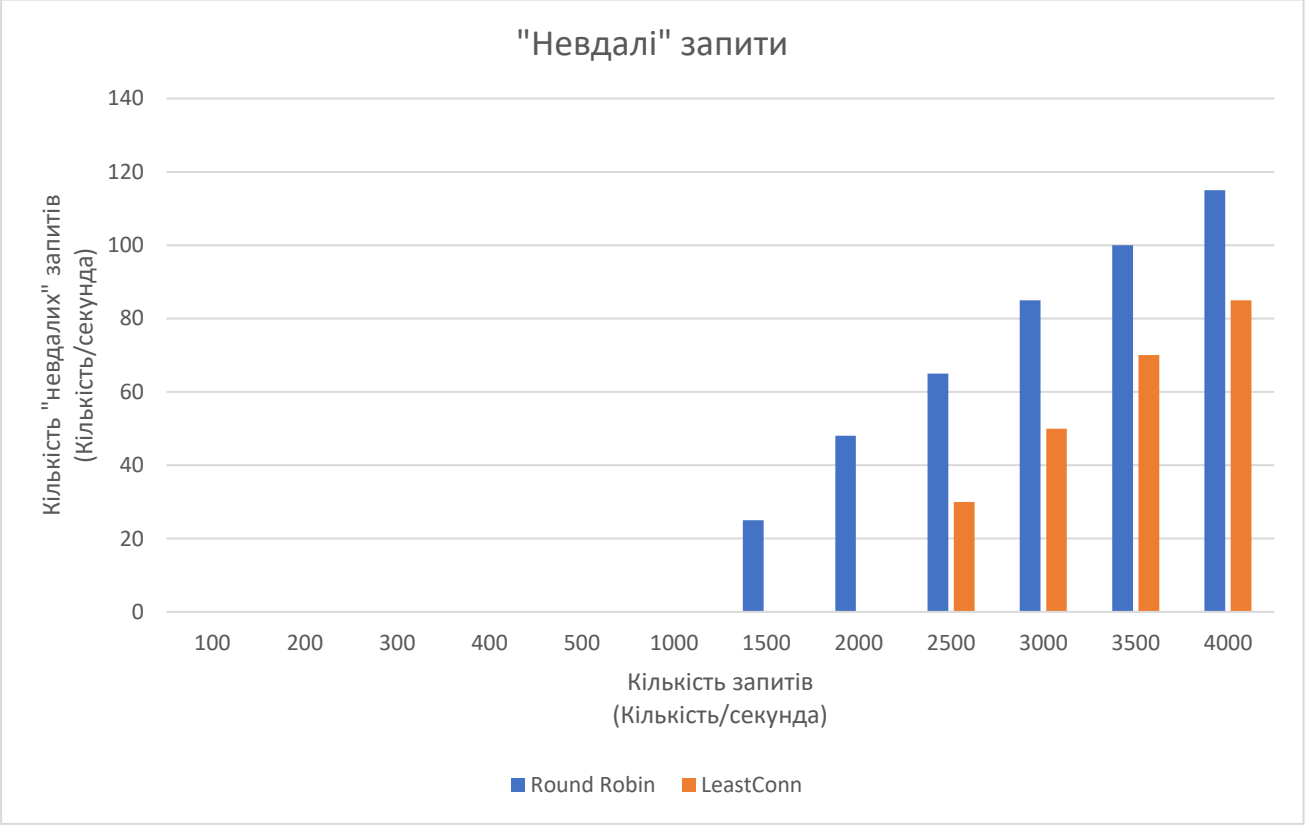

Рисунок 3.12 – Кількість «невдалих» запитів для кожного з алгоритму

3.3 Аналіз результатів експерименту

Встановлена система балансування навантаження працює нормально і здатний розподіляти навантаження та витримати відмову серверу. Процес реплікації на три вузли, що використовують односторонню кільцеву топологію, є успішним підходом до побудови комп'ютерної мережі. Доступність служби цієї системи становить 10 мілісекунд для обробки активних навантажень. Алгоритм Least Connection – це алгоритм, який перевершує інший (Round Robin) з точки зору швидкості з'єднання, часу відгуку, пропускної здатності та кількості невдалих з'єднань.

Метод Least Connection розподіляє нові підключення між доступними учасниками на основі поточної кількості підключень між системою BIG-IP та сервером. Це не враховує зв'язки серверів з іншими системами. Цей режим є хорошим методом розподілу загального призначення для більшості навантажень, але він може бути особливо корисним при підтримці довгожителів: такі з'єднання, як FTP та TELNET. З часом зв'язки повинні розподілятися відносно рівномірно по пулу. Якщо всі пристрої в даний час мають однакову кількість завантаженість, трафік розподіляється по них круговим способом – тобто по черзі.

Реалізація Least Connection HAProxy вже набагато краща за Round Robin і відповідає найвищому рівню точності, який може дати теоретична, найкраща версія сценарію алгоритму. Це дозволяє уникнути чергового трафіку на одному сервері і, отже, ідеально підходить як для одиночного, так і для розподіленого балансування навантаження.

Незважаючи на те, що це відмінне рішення, особливо як альтернатива Round Robin, тому щоразу, коли є вибір використання того чи іншого алгоритму балансування навантаження, в більшості випадків потрібно віддавати перевагу Least Connection, оскільки це, як довели результати тестування навантаження на сервери, демонструє кращий час відгуку сервера та зменшує витрати на завантаження та обробну потужність сервера.

Використання методу Least Connection – це більш динамічний підхід. При реалізації цього методу балансувальник навантаження повинен визначити, скільки активних з'єднань на даний момент має кожен сервер. Він призначить новий запит серверу з найменшою кількістю з'єднань. Якщо сервери мають неоднакову ємність, тоді кількість з'єднань буде розділено на відносну потужність сервера, щоб вирішити, який із них має найменше навантаження. Це забезпечує найкращий постійний баланс за рахунок додаткової складності. Балансир навантаження є критично важливим компонентом служби, оскільки, якщо він виходить з ладу, всі сервери, які він балансує, вимикаються. Щоб гарантувати доступність, у разі необхідності повинен бути додатковий доступний балансир навантажувача для відмови.

3.4 Висновки з розділу 3

Проведено порівняльний експеримент двох методів балансування навантаження Least Connection та Round Robin. В ході проведення було:

– проведено конфігурацію 3 серверів за допомогою Azure Services, на яких розвернута система, яка тестується великою кількістю одночасних користувачів;

– проведено конфігурацію бази даних MySQL також за допомогою Azure Services, яка використовується для моделювання реальної обробки запитів;

– проведено конфігурація балансувальників навантаження, методи яких порівнюються в експериментів;

– створено два сценарії проведення експерименту. Перший заключається в проведенні тестування серверів без використання балансувальників, інший з використанням;

– виконано чотири тестування навантаженням на сервери за допомогою програми JMeter Apache, з якої по завершенні кожного з тестування було отримано дані про час обробки запиту сервером (час від надсилання запиту клієнтом, до моменту отримання відповіді з серверу), статус код запиту, який показував успішність обробки запиту, пропускну здатність сервера, затримку та

кількість одночасних з’єднань, які сервер міг обробити за одиницю часу – секунду;

– проаналізовано по отриманим даним результати тестування 2 методів балансування навантаження: Least Connection та Round Robin. В ході аналізу було побудовано 4 таблиці та 4 гістограми на основі табличних даних;

– створено практичні рекомендації щодо використання методу балансування навантаження Least Connection, що базується на основі реальних досліджень в порівнянні з методом балансування Round Robin.

Таким чином, було вирішено третє та четверте завдання дипломної роботи – проведено експериментальне дослідження, проаналізовано результати експерименту та сформульовано практичні рекомендації щодо вирішення задачі вибору методу балансувальника навантаження.

ВИСНОВКИ

Одержані результати в сукупності вирішують важливе практичне завдання – порівняння двох найпоширеніших методів балансування навантаження: Least Connection та Round Robin при оптимізації розвантаження трафіку на додаткові сервери та здійснення підходу відмово стійкості серверів, що обробляють запити. На підставі проведеного дослідження можна зробити наступні висновки.

Проведений аналіз сучасних високонавантажених систем в світі типу Google, Facebook, YouTube показав, що впродовж останніх років навантаження на веб-сайти, веб-додатки та інші типи програмного забезпечення, що використовується в мережі Інтернет невпинно зростає. Як показує тенденція [37] у всіх регіонах світу щороку все більше людей перебувають у мережі. Швидкість, з якою світ змінюється, неймовірно висока. Будь-якого дня за останні 5 років вперше в мережі було в середньому 640 000 людей. Це було 27 000 щогодини. Тим, хто більшість днів у мережі, легко забути, наскільки молодий Інтернет. Графік під діаграмою нагадує про те, як нещодавно стали доступними веб-сайти та технології, інтегровані у повсякденне життя мільйонів: У 1990-х роках не було Вікіпедії, Twitter був запущений в 2006 році, а "Нашому світові в даних" лише 4 роки скільки людей приєдналося з тих пір4). І хоча багато з нас не уявляють свого життя без послуг, які надає Інтернет, головне повідомлення для мене з цього огляду світової історії Інтернету полягає в тому, що ми все ще перебуваємо на дуже ранніх стадіях Інтернету. Лише в 2017 році половина світового населення була в мережі; і тому у 2018 році все ще буває так, що близько половини світового населення не користується Інтернетом.

Проаналізовано головні проблеми масштабування високонавантажених систем на прикладі Google, Facebook, YouTube. Виявлено найпоширеніші підходи в вирішенні цих проблем: відкладені обчислення, попередні обчислення, фонові обчислення, кешування з використанням бібліотеки Memcached,

розпаралелювання задач та балансування навантаження з використанням балансувальника навантаження типу Haproxy.

Сформульовано мету експериментального дослідження і завдання для її реалізації. Визначено відгуки та фактори експерименту. Відгуками експерименту є: час з моменту посилання запиту до його отримання користувачем, пропускна здатність серверу з використанням різного методу балансування навантаження, кількість одночасно встановлених з'єднань та кількість невдалих з'єднань, продуктивність системи при одночасному зверненні сотні користувачів. Факторами, що впливають на результати, є: модель процесора сервера, на якому розгорнута система, модель процесора балансувальника навантаження, тип материнської плати, обсяг оперативної пам'яті та обсяг пам'яті жорсткого диску, тип бази даних та операційної системи, кількість одночасних користувачів, що здійснюють запити до системи.

Визначено технічні засоби конфігурації веб-серверів та балансувальників навантаження. Розглянуто програмні засоби реалізації, що будуть використані в процесі проведення експерименту. Ними стали хмарні сервіси Azure Services для конфігурації середовища розробки: веб-сервера, балансувальники навантаження, сервер бази даних; Apache JMeter, який може бути використаний як засіб тестування навантаження для аналізу та вимірювання продуктивності різноманітних служб з акцентом на веб-додатках; Microsoft Excel для побудови гістограм на основі даних отриманих в результаті проведення експерименту.

Проведено порівняльний експеримент 2 методів балансування навантаження на різному наборі одночасної кількості користувачів. На основі отриманих даних побудовано 4 гістограми, що демонструють кращу роботу одного з методів, зроблено висновок:

– реалізація Least Connection HAProxy вже набагато краща за Round Robin і відповідає найвищому рівню точності, який може дати теоретична, найкраща версія сценарію алгоритму. Це дозволяє уникнути чергового трафіку на одному сервері і, отже, ідеально підходить як для одиночного, так і для розподіленого балансування навантаження;

– щоразу, коли є вибір використання того чи іншого алгоритму балансування навантаження, в більшості випадків потрібно віддавати перевагу Least Connection, оскільки це, як довели результати тестування навантаження на сервери, демонструє кращий час відгуку сервера та зменшує витрати на завантаження та обробну потужність сервера.

Створено практичні рекомендації до вибору методу балансування навантаження при оптимізації трафіку на веб-сервери.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. API [Электронный ресурс]/ Режим доступа: https://ru.wikipedia.org/wiki/API - 08.09.2018 г

2. IMDG [Электронный ресурс]/ Режим доступа: https://habr.com/post/160517/

3. GFS [Электронный ресурс]/ Режим доступа: bourabai.kz/dbt/google/GFS.htm

4. Nginx [Электронный ресурс]/ Режим доступа: https://ru.wikipedia.org/wiki/Nginx

5. NoSQL [Электронный ресурс]/ Режим доступа: https://habr.com/post/152477

6. Конференция разработчиков высоконагруженных систем HighLoad++ 2018 [Электронный ресурс]/ Режим доступа: http://www.highload.ru/moscow/2018

7. Лекции Технопарка. 3 семестр. Проектирование высоконагруженных систем [Электронный ресурс]/ Режим доступа: https://habr.com/company/mailru/blog/254843/

8. HighLoad++ Junior [Электронный ресурс]/ Режим доступа: http://highload.guide/blog/highload-for-beginners.html

9. Отличительные особенности высоконагруженных систем [Электронный ресурс]/ Режим доступа: http://hawkhouse.ru/blog/chem-standartnaya-arhitektura-otlichaetsya-ot-arhitektury-vysokonagruzhennyh-prilozhenij/

10. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / Martin Kleppman, 2016 / [Электронный ресурс]. — Режим доступа: https://arxiv.org/abs/1512.05616.

11. Проектирование и внедрение высоконагруженных систем [Электронный ресурс]/ Режим доступа: http://dotrunet.ru/hiload.html

12. Волкова В.Н. Системный анализ информационных комплексов. Учебное пособие. М.: Лань, 2017. 336 с.

13. Назаров С.В. Архитектура и проектирование программных систем. М.: Инфра-М, 2016. 376 с.

14. Грекул В.И., Коровкина Н.Л., Левочкина Г.А. Проектирование информационных систем. Учебник и практикум. М.: Юрайт, 2017. 386 с.

15. Зараменских Е.П. Управление жизненным циклом информационных систем. Учебник и практикум. М.: Юрайт, 2017. 432 с.

16. Григорьев М. В., Григорьева И.И. Проектирование информационных систем. Учебное пособие. М.: Юрайт, 2016. 320 с.

17. Перез Чернов А. Высоконагруженные Системы: курс лекций / Александр Перез Чернов. – Минск: Кислоград, 2017. – 105 с.

18.

19. Кэширование данных [Электронный ресурс]/ Режим доступа: https://ruhighload.com/Кэширование+данных

20. What is Memcaced? [Электронный ресурс]/ Режим доступа: http://memcached.org

21. Архитектурные особенности высоконагруженных систем [Электронный ресурс]/ Режим доступа: http://www.highload.ru/2017/abstracts/1076.html

22. Что такое балансировка нагрузки [Электронный ресурс]/ Режим доступа: https://www.8host.com/blog/chto-takoe-balansirovka-nagruzki/

23. Введение в современную балансировку сетевой нагрузки [Электронный ресурс]/ Режим доступа: https://medium.com/southbridge/ introduction-to-modern-network-load-balancing-and-52e8ca36adde

24. Архитектура Google [Электронный ресурс]/ Режим доступа: http://www.insight-it.ru/masshtabiruemost/arkhitektura-google/

25. Boykin O., Ritchie S., O’Connell I., Lin J. Summingbird: A Framework for Integrating Batch and Online MapReduce Computations // 40th International Conference on Very Large Data Bases (VLDB), September 2014 [Электронный ресурс]. — Режим доступа: http://www.vldb.org/pvldb/vol7/p1441-boykin.pdf.

26. Development of a High Load Project Like Coogle [Электронный ресурс].

— Режим доступа: https://stfalcon.com/en/blog/post/development-of-high-load-project-like-google/

27. Архитектура больших проектов: Facebook [Электронный ресурс]/ Режим доступа: http://tokarchuk.ru/архитектура-больших-проектов-facebook/

28. Goodman E. P., Powles J. Facebook and Google: Most Powerful and Secretive Empires We've Ever Known. September 28, 2016 [Электронный ресурс]/ Режим доступа: https://www.theguardian.com/technology/2016/sep/28/googlefacebook-powerful-secretive-empire-transparency

29. Все о социальных сетях. Технологии [Электронный ресурс]/ Режим доступа: https://intellect.ml/vse-o-sotsialnykh-setyakh-tekhnologii-2052

30. Как устроен Facebook [Электронный ресурс]/ Режим доступа: https://masterpro.ws/forum/18-obo-vsem-kurilka/616-kak-ustroeny-facebook

31. Архитектура YouTube [Электронный ресурс]/ Режим доступа: http://www.insight-it. ru/masshtabiruemost/arkhitektura-youtube/

32. Последовательность организации эксперимента [Електронний ресурс] – Режим доступу до ресурсу: http://topknowledge.ru/nauchnye-issledovaniya/3041-posledovatelnost-organizatsii-eksperimenta.html.

33. Любченко Е.А., Чуднова О.А. Планирование и организация эксперимента: учебное пособие. Часть 1. – Владивосток: Изд-во ТГЭУ, 2010. – 156 с.

34. О.П. Дишлевий, Підбір метрик для властивостей програмного забезпечення., Проблеми програмування, 2010.

35. Коэффициент корреляции Пирсона – Методы математической статистики [http://psystat.at.ua/] от 15/11/12.

36. Коэффициент корреляции рангов Спирмена [http://cito-web.yspu.org/link1/metod/met125/node36.html] от 15/11/12.

37. The internet's history has just begun [Електронний ресурс] – Режим доступу до ресурсу: https://ourworldindata.org/internet-history-just-begun

ДОДАТОК А

Макет статті

Національний аерокосмічний університет ім. Н.Е. Жуковського «ХАИ»

***Анотація* — Стаття розкриває особливості проектування програмного забезпечення високонавантажених систем. Проаналізовані джерела недавніх років, розглянутий міжнародний підхід створення комп'ютерних систем, перераховані найбільш поширені практики та технології. На основі ряду джерел по даній тематиці було виявлено, що конкретні рішення при розробці інфраструктури підбираються індивідуально, в залежності від особливості проекту та поданої задачі)**

Ключові слова — програмне забезпечення, інформаційні системи, комп'ютерні системи, високе навантаження, архітектура систем, патерни проектування.

I. Визначення поняття «високонавантажені системи»

На сьогоднішній день існує величезна кількість проектів, що управляють високонавантаженими даними (data-intensive system). Технології, на яких побудовані подібні проекти, важливі в багатьох прикладних областях: від прогнозуючої аналітики до моніторингу навколишнього середовища, від країни в смартфоні до розумних будинків [1]. Окрім того, наукові області (біо-інформатика, дослідження клімату, геологія, радіоастрономія, астрофізика та інші) також зустрічаються з проблемами, що зв'язані з ефективним маніпулюванням величезними об'ємами даних, включаючи збір, генерацію, поширення складних, неоднорідних даних, допомога в прийнятті рішень, моделювання, прогнозування, візуалізації даних та іншими.

На початкових етапах функціонування високонавантажених сайт або додаток здійснює з'єднання користувачів з віддаленими ресурсами за допомогою інтернету (доступ і, можливо, зміна даних). Але в міру розвитку виникає все більша необхідність в технологіях, спрямованих на оптимізацію як апаратної, так і програмної боку системи.

Апаратна сторона (якщо мова про невеликі додатках) може містити один веб-сервер і базу даних; при малих обсягах даних у нас немає необхідності замислюватися про обладнання як про компонент нашого застосування, але в міру того, як проект масштабується, обладнання стає все більш важливою частиною загального дизайну. При проектуванні програмної сторони додатки необхідно сформулювати, як ми зберігаємо дані, як отримуємо до них доступ і модифікуємо (бізнес-логіка) і як представляємо дані користувачам (логіка взаємодії).

Реалізація подібного функціоналу може здатися тривіальною, проте необхідно спроектувати сервіс з високим ступенем настройки і оптимізації під конкретні потреби, щоб змусити цей функціонал працювати для мільйонів користувачів, які не витрачаючи занадто багато коштів на обладнання.

II. Вимоги до високонавантажених систем

Високонавантажені системи обробляють великі обсяги даних і формують за рахунок цього свою високу цінність для бізнесу, тому збої та інші проблеми з якістю результату обходяться компаніям вкрай дорого.

Ключовим джерелом проблем високонавантажених додатків є обсяг даних, їх складність і швидкість зміни. Тому важливо, щоб загальна архітектура великого додатка розроблялася як з точки зору програмних компонентів, так і апаратної частини, на якій вони функціонують. Більш того, при проектуванні інформаційної системи необхідно чітко розуміти, які встановлені терміни поставки, законодавчі обмеження, досвід фахівців, що беруть участь в конструюванні, усвідомлювати супутні ризики і їх прийнятність для компанії.

При розробці великомасштабних інформаційних систем (в першу чергу в сфері веб-технологій) важливо врахувати ряд принципів.

A. Доступність

Час безвідмовної роботи прямо корелює з репутацією і функціонуванням багатьох компаній.

B. Продуктивність

Швидкість веб-сайту впливає на задоволеність користувачів сервісом, а також на рейтинг в пошукових ведучих (що відбивається на відвідуваності

C. Надійність

Питати щоразу повинен повертати користувачам одні й ті ж дані, щоб користувачі були впевнені - якщо якісь дані записані / внесені в систему, при подальшому витяганні можна розраховувати на їх незмінність і збереження.

D. Масштабованість

Може йтися про різні параметри системи: скільки додаткового трафіку вона може обробляти, наскільки просто збільшити ємність сховища даних, скільки транзакцій може бути оброблено понад поточних можливостей.

E. Керованість

Розробка системи, яка проста в експлуатації, вкрай важлива на більш пізніх етапах розвитку проекту (простота діагностики і розуміння суті проблем, коли вони виникають, легкість оновлень або модифікацій).

F. Вартість

Має на увазі витрати на апаратне і програмне забезпечення. Важливо врахувати й інші аспекти, необхідні для розгортання та обслуговування системи: кількість часу, що витрачається розробниками на збірку системи, обсяг зусиль, необхідних для запуску системи, підготовку, навчання кадрів і т. д.

Перераховані вище принципи можуть йти врозріз один одному, і досягнення однієї мети буде здійснюватися за рахунок іншого. Наприклад, підвищення кількості серверів (масштабованість) досягається ціною керованості (вам доведеться працювати з додатковими серверами) і вартістю (витрата на придбання серверів) [2].

Коли мова заходить про великі центри обробки та зберігання даних, відомо, що апаратні збої (будь то відключення живлення, збій вінчестерів або ОЗУ і т. П.) Відбуваються постійно. Одним із шляхів вирішення проблеми є створення архітектури без загального доступу. Завдяки такій архітектурі відсутній центральний сервер, керуючий і координуючий дії інших вузлів, і, відповідно, кожен вузол системи може працювати незалежно один від одного. У подібних системах немає єдиної точки відмови, тому вони набагато більш стійкі до збоїв. Іншим методом запобігання збоїв є підвищення надмірності (redundancy) окремих компонентів системи, спрямоване на зниження частоти збоїв (резервне електроживлення, RAID - надлишковий масив дисків і т. П.). Коли один з компонентів виходить з ладу, запасний компонент бере на себе його функціонал. Подібним чином можна повністю уникнути збою, однак варіант цілком прийнятний в більшості випадків, т. К. Є можливість відновити систему з резервної копії в короткі терміни.

Якщо говорити про глобальне резервування, то всі правила взаємодії між серверами поширюються і на дата-центри - у нас повинен бути запас міцності (ємність, обчислювальні потужності і т. П.), Щоб продовжити працювати при втраті одного дата-центру без відчутної утрати якості послуг, що надаються.

Для коректної обробки помилок надмірність повинна поширюватися і на дані, і на надані послуги. Наприклад, якщо сервер зберігає тільки одну копію файлу, його відключення призводить до обмеження доступу до файлу. Саме тому практикується створення декількох надлишкових копій одних і тих же даних або сервісів.

Програмні збої, породжувані помилкою в системі, в силу проблемності запобігання, а також впливу на декілька вузлів відразу, викликають набагато більше відмов системи, ніж апаратні. На думку багатьох фахівців забезпечення якості в процесі розробки ПЗ для високонавантажених додатків все ще перебуває в зародковому стані.

Частими є збої, пов'язані з граничним навантаженням одного з ресурсів системи (простору в сховище, CPU, оперативної пам'яті і т. П.), Збої в роботі сервісу, що обумовлює функціонал системи (затримка або припинення відповідей і т.п.), програмні помилки , що виникають при «поганих» вступних даних, а також невеликі збої в одному компоненті, які провокують серію послідовних збоїв по ланцюгу (каскадні збої). Трапляються перебої, обумовлені якістю наданих провайдерами послуг - провайдери на своїх серверах запускають застарілі сервіси, налаштовані для конкретного типу сховища, формату даних і використовуваної політики управління. Для подолання виникає розриву в подібних випадках в архітектурі інформаційної системи необхідна реалізація проміжного програмного забезпечення, яке, з одного боку, займається доставкою даних і метаданих, відповідають загальному в рамках системи API, а з іншого взаємодіє з існуючими внутрішніми серверами і додатками.

Хоча на початковому етапі придбання обладнання здається витратним кроком, у міру розвитку проекту вартість розробки ПО стає незрівнянно більше. Саме тому в сучасних системах прагнуть розвивати архітектуру так, щоб масштабування практично не вимагало розробки ПО - придбання, встановлення та налаштування обладнання обходиться набагато дешевше.

III. Обладнання, Апаратна сторона системи

На початкових етапах апаратна сторона додатка становить більшу частину всіх витрат на запуск системи. Витрати на розробку програмної сторони (зарплати розробникам, фахівцям, обслуговуючим систему і т. д.) значно вище (як було зазначено вище), але за обладнання доводиться платити на самому початку і відразу всією сумою. Мова йде не про якісь конкретні типи процесорів, марках обладнання або архітектурі, а, в цілому, про сукупності складових елементів (апаратне і програмне забезпечення) додатки. Обладнання може бути представлено одним типом обчислювальних машин з однаковими поставленими операційними системами (ОС) і наборами утиліт, однак найчастіше це парк з різних класів обладнання з різними ОС.

При зберіганні великих обсягів даних використовуються спеціалізовані підходи [3]:

- організація зберігання файлів на основі метаданих (поділ по якимось певним атрибутам);
- реплікація серверів управління файлами для поділу загального сховища даних і їх доступу, пошуку, обробки і т. п.;
- відбір технологій поширення даних з урахуванням усіх обмежень і переваг;
- додаткові виникаючі завдання пошуку і обробки даних.

Перелічимо найбільш поширені методи розміщення обладнання додатків.

A. Спільно використовуване обладнання (shared hardware)

Як правило, орендується у великих провайдерів (хостинги і ін.), Ресурси серверів експлуатуються спільно з іншими користувачами. Користувач не має можливості конфігурації сервера під себе і установки додаткових модулів. Подібні платформи підходять для етапу розробки, розміщення прототипу і раннього запуску додатка.

B. Виділене обладнання (dedicated hardware)

Обладнання, необхідне для запуску програми, ви орендуєте у постачальника. Він зі свого боку володіє і займається обслуговуванням даного обладнання, тому вам не доведеться міняти виходять з ладу сервера, диски зберігання даних, стійки і т. П. - тільки вхід в систему і обмін даними по захищеному каналу. Виділене обладнання за можливостями відрізняється від некерованого (ми можемо тільки логін користувача) до повністю керованого нами (отримуємо віддалений доступ до консолі для подальшої автоматичної настройки). Плюсом є відсутність необхідності наявності системних адміністраторів в штаті.

C. Спільно розміщене обладнання (co-located hardware)

Ви надаєте своє обладнання і обслуговування системи, а постачальник спільного розміщення забезпечує вас майданчиком, пропускною спроможністю мережі і електрикою. Надаються такі функції, як моніторинг мережі, перезавантаження сервера в разі збою і т. П. Деякі постачальники надають послуги з моніторингу роботи серверів, діагностиці та оповіщення в разі збоїв, проте цей функціонал може бути відсутнім, або за нього буде братися окрема плата.

Самостійний хостинг (self-hosting) У випадку, якщо ваш проект розрісся до кількох тисяч серверів, виникає сенс створення власних дата-центрів. Однак треба бути готовим до того, що вам доведеться створити власний майданчик, вибудувати відповідно до потреб серверну архітектуру, найняти DevOps-фахівців, а також обслуговуючий персонал, забезпечити системою пожежної безпеки, резервними підключеннями до електромережі і інтернету, джерелом безперебійного живлення (UPS) і т.д.

Вузьким місцем веб-додатків є пропускна здатність бази даних, що викликано низькою швидкістю введення і виведення інформації на диску. Отримані обсяги даних після декількох етапів обробки необхідно переносити в підсумкову область зберігання даних, що вимагає високої пропускної здатності мережі, відповідної по швидкості генерації даних, і задіяння таких протоколів передачі даних, які в повній мірі використовують цю смугу пропускання. проблема відставання пропускної здатності мережі від швидкості генерації даних і зростання обсягів інформації, що зберігається актуальна і сьогодні, тому одним із завдань є розробка технологій, що сприяють максимальної швидкості передачі інформації, будь то створення декількох паралельних потоків даних, декомпозиція переданих файлів, настройка буфера і т.п.

Автор «Building Scalable Web Sites» К Хендерсон пише: "Планування потужностей - досить точна наука" [5]. Його рекомендації з приводу початкового дизайну апаратної частини програми відображаються так:

- купувати поширене, представлене на ринку обладнання.
- використовувати готові збірки ОС і перевірені дистрибутиви.
- використовувати по можливості готове програмне забезпечення.

IV. Патерни проектування, програмна сторона системи

При розробці високонавантаженого даними додатка (DIA) необхідно реалізувати можливості, що забезпечують регулярно потрібну функціональність. Наприклад, стандартними вважаються такі можливості: зберігання даних для подальшого використання їх поточним або зовнішнім додатком - база даних; пошук і фільтрація даних за ключовими словами та іншими способами - пошуковий індекс; збереження найбільш часто проведених операцій для прискорення повернення результату - кеш; переробка великих пакетів даних - пакетна обробка; відправка повідомлення між процесами для асинхронної обробки - потокова обробка.

При проектуванні слід скористатися таким алгоритмом: отримати детальну інформацію про своїх даних; побудувати за отриманою інформацією схему використання і руху даних; підібрати архітектурний патерн, розробити архітектуру під кожне конкретне використання; підібрати найбільш доречні інструменти та технології під кожну обрану архітектуру.

Якщо стоїть завдання отримати детальну інформацію про дані, то необхідно відповісти на наступні питання: які є дані в проекті, як вони пов'язані між собою, яка кількість даних кожного типу, які мінімальні, максимальні і середні розміри у даних кожного типу, яка кількість актуальних даних , яким чином визначається актуальність даних, якими темпами зростає обсяг даних, яка частота читання даних [10].

У випадку з підбором архітектурних патернів, необхідно вибрати найбільш актуальні зі списку рекомендованих до використання: сервісно-орієнтована архітектура; відкладені обчислення; асинхронна обробка; конвеєрна обробка; використання "товстого" клієнта; функціональний розподіл; шардінг (sharding); центральний диспетчер; реплікація; кластеризація (clustering); денормалізація (denormalization); введення надмірності; паралельне виконання. При виборі патернів необхідно ясно розуміти плюси і мінуси кожного з них, межі застосування, а також ознайомитися з уже реалізованими прикладами використання[4].

Початковим етапом в розробці більшості веб-проектів є поділ структури на три частини, кожна з яких відповідає за обробку або виконання певного класу задач: фронтенд (front-end, швидка обробка легких даних), бекенд (back-end, обчислення) і зберігання даних .

Першою ланкою виступає фронтенд, який призначений для обробки легких даних, як правило, статичного контенту сайту, звільняючи від цього масивний бекенд. Ключовим моментом у фронтенді є те, яка кількість ресурсів витрачається на обробку одного запиту, тому для цієї ланки використовуються легковагі сервера, наприклад, Nginx.

Друга ланка це бекенд. На ньому, як правило виконуються обчислення, реалізується

бізнес-логіка, тому легковагі запити обробляти важким, високопродуктивним бекенд неефективно.

Третя ланка - зберігання інформації. У найпростішому вигляді являє собою базу даних, але скоріше це сховище даних на серверах, розташованих в дата-центрах. У будь-якому випадку воно є ключовим елементом нашої системи - дані, які лежать в основі інших сервісів додатки.

Для ефективної роботи системи зі сховищем даних її розбивають на декілька рівнів, і там, де раніше був великий монолітний блок коду, тепер присутні такі елементи, як бізнес-логіка, логіка взаємодії і розмітка / уявлення. Т. к. Кожному з елементів системи доводиться взаємодіяти з іншими, розробникам необхідно виконати додаткову роботу по реалізації інтерфейсів на кордоні взаємодії рівнів.

Бізнес-логіка (business logic) Розташовується безпосередньо над сховищем, і в ній полягає ключовий функціонал додатка, то, чим воно відрізняється від інших подібних сервісів. Рівень бізнес-логіки - це єдиний шлях отримання доступу до сховища даних, т. К. Поведінка нашої системи безпосередньо визначається правилами, які регулюють доступ до даних, і особливостями їх маніпуляції (як зберігаються і обробляються).

Логіка взаємодії (interaction logic) Наступний після бізнес-логіки рівень, описує те, яким чином дані змінюються, що і як відображається користувачеві. Залежно від Ваших дій змінюється відображена інформація, однак основний функціонал додатка залишається без змін [7].

В інтернеті для розробки високонавантаженої системи часто використовуються такі технології як розпаралелювання, попередня обробка, конвеєр, відкладені обчислення, пошук, кешування різних типів, проксі-сервер, індекс, балансування, чергу.

Перераховані вище технології мають безліч готових програмних реалізацій, що володіють своїми особливостями, але вирішальних одні завдання. Вибір на користь того чи іншого рішення залежить конкретних ситуацій і потреб, що виникають при роботі над системою.

За результатами дослідження можна зробити наступні висновки: сфера високонавантажених систем знаходиться в стадії активного розвитку, і на ринку можна зустріти величезну кількість різних технологій і інструментів, які сприяють вирішенню конкретних завдань. Тому не можна говорити про якісь універсальних наборах технологій, їх варто підбирати для свого проекту в кожній ситуації індивідуально [6]. У той же час ще на стадії планування фахівцями застосовуються досить чітко опрацьовані, стандартні практики, які сприяють уникнути проблем з надійністю системи і складнощами в масштабуванні. Йдеться і про початковий збір інформації про вимоги до інформаційної системи, і про послідовність подальших дій, і класичних архітектурних патернах, використання яких в значній мірі і визначає вибір технологій для реалізації цих патернів. Тільки вивчивши цей базис, а також проаналізувавши поточну ситуацію в реалізованому проекті, можна переходити до вибору технологій і побудови структури програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] G. Eason, B. Noble, and I.N. Sneddon, “On certain integrals of Lipschitz- G. Casale, D. Ardagna, M. Artac, F. Barbier, E. D. Nitto, A. Henry, G. Iuhasz, C. Joubert, J. Merseguer, V. I. Munteanu, "Dice: quality-driven development of dataintensive cloud applications." in Proceedings of the Seventh International Workshop on Modeling in Software Engineering. IEEE Press, 2015, pp. 78-83.)

[2] C. A. Mattmann, D. J. Crichton, A. F. Hart, C. Goodale, J. S. Hughes, S. Kelly, L. Cinquini, T. H. Painter, J. Lazio, D. Waliser, "Architecting data-intensive software systems" in Handbook of Data Intensive Computing, Springer, 2011, pp. 25-57.I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[3] O. Hummel, H. Eichelberger, A. Giloj, D. Werle, K. Schmid, "A collection of software engineering challenges for big data system development", in 2018 44th Euromicro Conferenceon Software Engineering and Advanced Applications (SEAA), IEEE, 2018, pp. 362-369.

[4] B. Burns, "Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services", 2018

[5] C. Henderson, "Building Scalable Web Sites", 1st Edition, Sebastopol, O'Reilly Media, Inc, 2006.

[6] C. Lynch, "Big data: How do your data grow?", Nature, 2008, pp. 28-29.

I. C. Richard, "How Complex Systems Fail", Chicago, Cognitive technologies Laboratory, University of Chicago, 2000