**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методики тестування продуктивності для аналізу стану сервера під навантаженням»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
*(код, найменування спеціальності)*
освітньо-професійної програми «Інженерія програмного забезпечення»
*(назва)*

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_______________ Іван СОБКО
*(підпис)*

Виконав: здобувач вищої освіти групи ПДМ-61

Іван СОБКО

Керівник: Наталія ТРІНТІНА
*к.т.н., доцент*

Рецензент: _______________
*науковий ступінь, вчене звання* Ім’я, ПРІЗВИЩЕ

**Київ 2024**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерії програмного забезпечення
_______________ Ірина ЗАМРІЙ
«_____» ___________ 2024 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Собку Івану Івановичу

1. Тема кваліфікаційної роботи: Розробка методики тестування продуктивності для аналізу стану сервера під навантаженням

керівник кваліфікаційної роботи Наталія ТРІНТІНА к.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: 3. Вихідні дані до роботи:
Матеріали переддипломної практики, методи навантаження, API запроси, цілі тестування продуктивності, виміри та аналіз під час навантаження сервера.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд предметної області.
2. Розробка моделей та методів.

3.Розробка програмного забезпечення моделей.
4.Проведення моделювання та аналіз отриманих результатів.

5. Перелік графічного матеріалу: *презентація*
1. Мета, об’єкта та предмет дослідження.
2. Приклад моделі роботи http запитів.
3. Методика аналізу шляху http запитів.
4. Математичний аналіз завантаження.

6. Дата видачі завдання «19» жовтня 2023 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз наявної науково-технічної літератури | 19.10-05.11.23 | |
| 2 | Огляд предметної області | 06.11-12.11.23 | |
| 3 | Аналіз методів та засобів тестування навантаження | 13.11-19.11.23 | |
| 4 | Розробка моделей та методів | 20.11-26.11.23 | |
| 5 | Розробка програмного забезпечення моделей | 27.11-03.12.23 | |
| 6 | Впровадження методу покращення опрацювання HTTP запитів | 04.12-10.12.23 | |
| 7 | Оформлення роботи: вступ, висновки, реферат | 11.12-20.12.23 | |
| 8 | Розробка демонстраційних матеріалів | 21.12-29.12.23 | |

Здобувач вищої освіти ______________ Іван СОБКО
*(підпис)*

Керівник
кваліфікаційної роботи ______________ Наталія ТРІНТІНА
*(підпис)*

# РЕФЕРАТ

Текстова частина магістерської роботи: 69с., 26 рис., 1 дод., 21 джерел.

МОДЕЛЬ, СИСТЕМА НАВАНТАЖЕННЯ СЕРВЕРА, ТЕСТУВАННЯ НАВАНТАЖЕННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, JAVA

Мета дослідження: збільшення кількості опрацьованих HTTP запитів за рахунок використання балансера навантаження

Об'єкт дослідження: тестування продуктивності сервера під навантаженням

Предмет дослідження: методи тестування продуктивності

Методи дослідження – методи тестування програмного забезпечення, методи тестування продуктивності, балансування навантаженості, методи проектування та розробки програмного забезпечення, технології об'єктно-орієнтованого програмування.

У роботі проведено аналіз сучасного використання методів навантаження та теорії тестування продуктивності для програмного продукту, виконано огляд існуючого програмного забезпечення та проведено аналіз використаних моделей, оглянуто Здійснено розробку моделі та алгоритмів навантаження з виводом статистики.

Розроблено програмне забезпечення для автоматизованого тестування продуктивності, яке дозволяє ефективно оцінювати роботу системи під навантаженням.

Проведено аналіз отриманих результатів, визначено закономірності поведінки програмних продуктів, що пов'язані з недостатньою потужністю сервера.

Ключові слова: Продуктивність, тестування, навантаження, середовище, аналіз.

# ABSTRACT

The text part of the master's thesis: 69 pages, 26 figures, 1 appendix, 21 sources.

MODEL, SERVER LOAD SYSTEM, LOAD TESTING, SOFTWARE, JAVA

The purpose of the work: increasing the number of processed HTTP requests due to the load balancer

Object of research - testing server performance under load

Subject of research - performance testing methods

Summary of the work: software testing methods, performance testing methods, load balancing, software design and development methods, object-oriented programming technologies.

The paper analyzes the modern use of loading methods and the theory of performance testing used for a software product, reviews existing software and analyzes models, and considers the development of loading models and algorithms with statistical output.

Software for automated performance testing has been developed, which allows you to effectively evaluate the performance of the system under load.

The analysis of the obtained results was carried out, the patterns of behavior of software products associated with insufficient server capacity were determined.



KEYWORDS: Performance, testing, load, environment, analysis.

# ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ........................................................................ 9
ВСТУП.................................................................................................................. 10
РОЗДІЛ 1АНАЛІЗ ПІДХОДІВ ТЕСТУВАННЯ ПРОДУКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІД НАВАНТАЖЕННЯМ........................... 12
1.1. Загальна класифікація стандартів до тестування продуктивності............12
1.2. Архітектура інструменту тестування...................................................... 17
1.3 Аналіз програмних засобів для тестування продуктивності......................21
РОЗДІЛ 2 РОЗРОБКА МЕТОДИКИ ТЕСТУВАННЯ З ВИКОРИСТАННЯМ НАВАНТАЖЕННЯ СЕРВЕРУ............................................ 30
2.1 Постановка задачі тестування продуктивності для аналізу стану сервера під навантаженням..................................................................................... 30
2.2 Розробка методу аналізу сервера під навантаженням............................. 31
2.3 Модель балансера навантаження............................................................ 38
2.4 Метод реплікації сервісу........................................................................ 55
РОЗДІЛ 3 ПРОВЕДЕННЯ МОДЕЛЮВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ.......................................................................................................81
ВИСНОВКИ........................................................................................................ 83
ПЕРЕЛІК ПОСИЛАНЬ........................................................................................ 84
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)............................................... 85

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| | | |
|---|---|---|
| IaaS | - | Infrastructure as a service |
| SaaS | - | Software as a service |
| PaaS | - | Platform as a service |
| XML | - | Extensible Markup Language |
| CSP | - | Cloud Service Provider |
| LB | - | Load Balancing |
| HTTP | | HyperText Transfer Protocol |

# ВСТУП

Сучасні програмні продукти та інформаційні системи надзвичайно складні та вимагають відповідної ефективності та стабільності під час роботи під великим навантаженням. Запити користувачів, обробка даних та функціонування систем у високонавантажених умовах є невід'ємною частиною сучасного інформаційного середовища. Тому проблема тестування продуктивності стає критично важливою в розробці програмного забезпечення та створенні інформаційних систем.

Метою даної роботи є вивчення методів та інструментів тестування продуктивності для аналізу стану програмного забезпечення та систем під впливом навантаження. Дослідження цієї теми є актуальним у зв'язку з постійним розвитком інформаційних технологій та зростанням вимог до якості та ефективності програмних продуктів.

У цьому контексті важливими аспектами є розробка та впровадження методів та інструментів тестування, які дозволяють забезпечити високу продуктивність та стабільність систем у реальних умовах експлуатації. Дослідження в цій області сприятиме покращенню якості та надійності програмних продуктів, що безпосередньо позитивно впливає на користувачів та бізнес-середовище загалом.

Ця робота спрямована на розгляд ключових аспектів тестування продуктивності, визначення оптимальних підходів та розробку рекомендацій для підвищення ефективності роботи програмного забезпечення та інформаційних систем у реальних умовах експлуатації.

Таким чином, завдання розробки методики тестування продуктивності є сучасним та актуальним.

*Мета дослідження:* збільшення кількості опрацьованих HTTP запитів за рахунок використання балансера навантаження

*Об’єкт дослідження:* тестування продуктивності сервера під навантаженням

*Предмет дослідження:* методи тестування продуктивності

*Методи дослідження* – методи тестування програмного забезпечення, методи тестування продуктивності, балансування навантаженості, методи

проектування та розробки програмного забезпечення, технології об’єктно-орієнтованого програмування.

*Практична значущість результатів* полягає в у можливості застосування отриманих даних та рекомендацій для покращення функціональності, надійності та ефективності програмного забезпечення та інформаційних систем у реальних умовах експлуатації.

Для досягнення мети вирішувалися наступні завдання.

1. Аналіз існуючих підходів до тестування продуктивності під навантаженням
2. Дослідження алгоритмів навантаження.
3. Розробка методу навантаження програмного продукту.
4. Розробка програмного забезпечення моделі.
5. Проведення моделювання поведінки системи та аналіз отриманих результатів.

# 1 АНАЛІЗ ПІДХОДІВ ТЕСТУВАННЯ ПРОДУКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІД НАВАНТАЖЕННЯМ

## 1.1. Загальна класифікація стандартів до тестування продуктивності

Тестування — це не просто перевірка задоволеності вимог, необхідних для розробки програмного забезпечення. Тестування – це процес перевірки та перевірки всіх бажаних вимог до продуктів, а також перевірки та перевірки будь-яких небажаних вимог до продуктів. Він також бачить будь-які приховані ефекти в продукті через ці вимоги. У тестуванні програмного забезпечення тестування масштабованості має визначити поведінку системи шляхом збільшення навантаження з певним коефіцієнтом масштабування. Для кожної точки масштабування мають бути визначені всі атрибути продуктивності. Крім того, слід визначити фактори, що впливають на здатність масштабування програми. Техніка перевірки вимірювання програмного забезпечення може застосовуватися для тестування масштабованості, щоб визначити поведінку системи шляхом збільшення навантаження з певним коефіцієнтом масштабування. Для кожної точки масштабування необхідно визначити всі атрибути продуктивності. Крім того, слід визначити фактори, що впливають на здатність масштабування додатків; документація щодо тестування програмного забезпечення, документація користувача програмного забезпечення, посібники з технічного обслуговування, процедури створення системи, процедури встановлення та примітки до випуску є можливими кандидатами на перевірку.

Пропускна здатність веб-додатку на основі запитів за секунду, одночасних користувачів або байтів переданих даних, а також вимірювання продуктивності. Для вимірювання налаштованого програмного забезпечення існує кілька методів, які можна застосувати під час тестування: специфікація вимог до програмного забезпечення, опис дизайну програмного забезпечення, документація щодо тестування програмного забезпечення, документація

користувача програмного забезпечення, посібники з технічного обслуговування, процедури побудови системи, процедури встановлення та примітки до випуску є можливими кандидатами на перевірку. Оглядові зустрічі слід запланувати в плані проекту або вони можуть проводитися за запитом, напр. за групою якості.

та сама програма, щоб визначити, чи може система обробляти навантаження без шкоди для функціональності чи продуктивності.

2- Об'ємне тестування:

Volume Testing визначає слабкі сторони системи для обробки великої кількості даних протягом коротких періодів часу.

3- Стрес-тестування:

Стрес-тестування визначає, чи здатна система обробляти велику кількість транзакцій обробки в періоди пікового навантаження. Потім ці дані повинні бути проаналізовані, щоб визначити загальний стан системи та виявити проблеми вузьких місць.

4- Функціональне тестування-

Тестування функціональності підтверджує, що програма відповідає своїм специфікаціям і відповідає очікуваним функціональним вимогам. Під час тестування функціональності створюється діапазон вхідних даних у вигляді тестових даних і виконуються тести, щоб перевірити, чи кожна функція відповідає вимогам.

5- Спайк тестування

Спикове тестування виконується шляхом раптового збільшення кількості користувачів або навантаження, створеного користувачами, на дуже велику кількість і спостереження за поведінкою системи. Мета полягає в тому, щоб визначити, чи постраждає продуктивність, чи система виникне збій, чи вона зможе впоратися із різкими змінами навантаження.

6- Тестування безпеки

Наші перевірки коду безпеки можуть бути автоматизованими або ручними, але переважно поєднанням обох.

Стандарти тестування продуктивності різноманітні та можуть включати різні аспекти вимог до тестування та оцінки продуктивності систем. Загальна класифікація стандартів тестування продуктивності включає наступні критерії:

- Більше 15 секунд

  Це виключає розмовну взаємодію. Для певних типів програм, певних типів користувачів може бути задоволено сидіти біля термінала більше 15 секунд, чекаючи відповіді на один простий запит. Однак для зайнятого оператора кол-центру або трейдера ф’ючерсів, затримки понад 15 секунд можуть здатися неприпустимими. Якщо такі затримки можуть виникнути, система має бути розроблений таким чином, щоб користувач міг звернутися до інших видів діяльності та запросити відповідь
  в якийсь пізніший час.

- Більше 4 секунд

  Ці затримки, як правило, надто тривалі для розмови, яку кінцевий користувач повинен зберегти інформацію в короткочасній пам’яті (пам’яті кінцевого користувача, а не комп’ютера. Такі затримки перешкоджатимуть вирішенню проблем і перешкоджатимуть введенню даних. Однак після завершення транзакції можна допускати затримки від 4 до 15 секунд.

- від 2 до 4 секунд

  Затримка довше 2 секунд може перешкоджати операціям, які вимагають високого рівня концентрація. Очікування від 2 до 4 секунд на терміналі може здатися напрочуд довгим, коли користувач поглинений і емоційно відданий виконанню поставленого завдання. Знову затримка у цьому діапазоні може бути прийнятним після незначного закриття. Може бути прийнятним зробити а покупець чекає від 2 до 4 секунд після введення своєї адреси та номера кредитної картки, але ні на ранньому етапі, коли вона порівнює різні характеристики продукту.

- Менше 2 секунд

  Коли користувач програми повинен запам’ятати інформацію протягом кількох відповідей, час відповіді має бути коротким. Чим детальнішу

- інформацію потрібно запам'ятати, тим більша потреба у відповідях менше 2 секунд. Таким чином, для комплексної діяльності таке як для перегляду продуктів камери, які відрізняються за кількома вимірами, 2 секунди представляють важливий ліміт часу відповіді.
- Час відгуку до секунди

  Певні види розумової роботи (наприклад, написання книги), особливо з програми, багаті графікою, вимагають дуже короткого часу відгуку, щоб підтримувати інтерес і увагу протягом тривалого часу. Художник перетягує зображення на інше місце розташування повинно мати можливість миттєво діяти на його наступну творчу думку.
- Час відгуку мілісекунд

  Реакцією на натискання клавіші є перегляд символу, що відображається на екрані, або клацання екранний об'єкт за допомогою миші має бути майже миттєвим: менш ніж через 0,1 секунди після дію. Багато комп'ютерних ігор вимагають надзвичайно швидкої взаємодії.

Як бачите, критичний бар'єр часу відповіді становить 2 секунди. Час відгуку більше ніж це має певний вплив на продуктивність середнього користувача, тому наша номінальна сторінка Час оновлення 8 секунд для Інтернет-додатків, безумовно, менш ніж ідеальний.

Сервісно-орієнтованими показниками є доступність і час відгуку. Ми можемо коротко визначити ці терміни так:

- Доступність

Період, коли програма доступна для кінцевого користувача, має велике значення. Недоступність може призвести до серйозних витрат у бізнесі, оскільки навіть невеликий перебіг може стати причиною значних фінансових втрат для багатьох додатків. У контексті тестування продуктивності, це означає повну неефективність програми для кінцевого користувача.

1. Час реакції

Час, необхідний програмі для відповіді на запит користувача, зазвичай вимірюється як час від запиту користувача до отримання повної відповіді від програми. У контексті тестування продуктивності, це вимірювання охоплює інтервал часу між відправленням запиту користувача до отриманням повної відповіді на робочу станцію користувача.

- Пропускна здатність

Швидкість, з якою відбуваються прикладні події. Хорошим прикладом може бути кількість відвідувань веб-сторінки за певний період часу.

- Утилізація

Відсоток теоретичної потужності ресурсу, який використовується. Приклади включають, скільки пропускної здатності мережі споживає трафік програми та обсяг пам'яті, що використовується на сервері, коли активні тисячі відвідувачів.

У сукупності ці показники можуть дати нам точне уявлення про продуктивність програми та її вплив, з точки зору потужності, на ландшафт програм.

Повертаючись до нашого обговорення поширених причин невдач: якщо ви не враховуєте продуктивність під час розробки програми, ви напрошуєтеся на проблеми. Хороший дизайн забезпечує високу продуктивність або, принаймні, гнучкість змінювати або переналаштовувати програму, щоб справлятися з неочікуваними проблемами продуктивності. Проблеми продуктивності, пов'язані з дизайном, які залишаються непоміченими до кінця життєвого циклу, важко повністю подолати, і зробити це іноді неможливо без значної переробки програми.

Більшість програм створено з програмних компонентів, які можна тестувати окремо та можуть добре працювати окремо, але не менш важливо розглядати програму в цілому. Ці компоненти повинні взаємодіяти ефективним чином, щоб досягти гарної продуктивності.

Тут тестування продуктивності проводиться безпосередньо перед розгортанням, при цьому мало уваги приділяється кількості необхідного часу або наслідкам відмови. Незважаючи на те, що цей режим кращий, ніж

«протипожежний», він усе ще несе значний ступінь ризику того, що ви не виявите серйозні дефекти продуктивності, які з’являються у виробництві, або не дасте достатньо часу для усунення проблем, виявлених перед розгортанням.

Одним із типових результатів є затримка розгортання програми, поки проблеми не вирішено. Додаток, який розгорнуто зі значними проблемами продуктивності, потребуватиме коштовних і трудомістких коригувальних робіт після розгортання. Що ще гірше, програму, можливо, доведеться повністю вилучити з обігу, оскільки вона втратила форму.

Усі ці результати мають надзвичайно негативний вплив на бізнес і на довіру тих, хто, як очікується, буде використовувати додаток. Вам потрібно якомога раніше перевірити наявність проблем з продуктивністю, а не відкладати це на останню хвилину.

Певні технології, які зазвичай використовуються для створення програм, погано працювали з першим і навіть другим поколінням автоматизованих інструментів тестування. Це стало значно слабшим виправданням для того, щоб не проводити жодного тестування продуктивності, оскільки переважна більшість додатків тепер певною мірою підтримуються Інтернетом. Веб-технологія, як правило, добре підтримується поточним набором рішень для автоматизованого тестування.

Розробка та розгортання наразі викристалізувалися на (відносно) кількох основних технологіях. Відповідно, більшість постачальників автоматизованих інструментів наслідували їхній приклад із підтримкою, яку надають їхні продукти.

## 1.2. Архітектура інструменту тестування

Інструменти автоматичного тестування продуктивності зазвичай мають такі компоненти.

- Модуль сценаріїв

  Дозволяє записувати дії кінцевого користувача та може підтримувати багато різних протоколів проміжного програмного забезпечення. Дозволяє модифікувати записані сценарії, щоб зв'язати внутрішні/зовнішні дані та налаштувати деталізацію вимірювання часу відгуку.

- Модуль управління тестами

Дозволяє створювати та виконувати сеанси навантажувального тестування або сценарії, які представляють різні суміші дій кінцевого користувача. У цих сеансах використовуються призначені сценарії та один або кілька інжекторів навантаження.

- Інжектор навантаження

Створює навантаження — як правило, з кількох робочих станцій або серверів, залежно від необхідного обсягу навантаження.

- Модуль аналізу

Надає можливість аналізувати дані, зібрані під час виконання кожного тесту. Ці дані зазвичай є сумішшю автоматично створених звітів і графічних або табличних звітів. Також може бути «експертна» функція, яка забезпечує автоматичний аналіз результатів і висвітлює проблемні області.

Доповненням до цих компонентів можуть бути додаткові модулі, які відстежують продуктивність сервера та мережі під час виконання навантажувального тесту. На малюнку 2-1 показано типове розгортання інструменту автоматизованого тестування продуктивності. Користувачів програми було замінено групою серверів або робочих станцій, які використовуватимуться для завантаження програми шляхом створення «віртуальних користувачів».

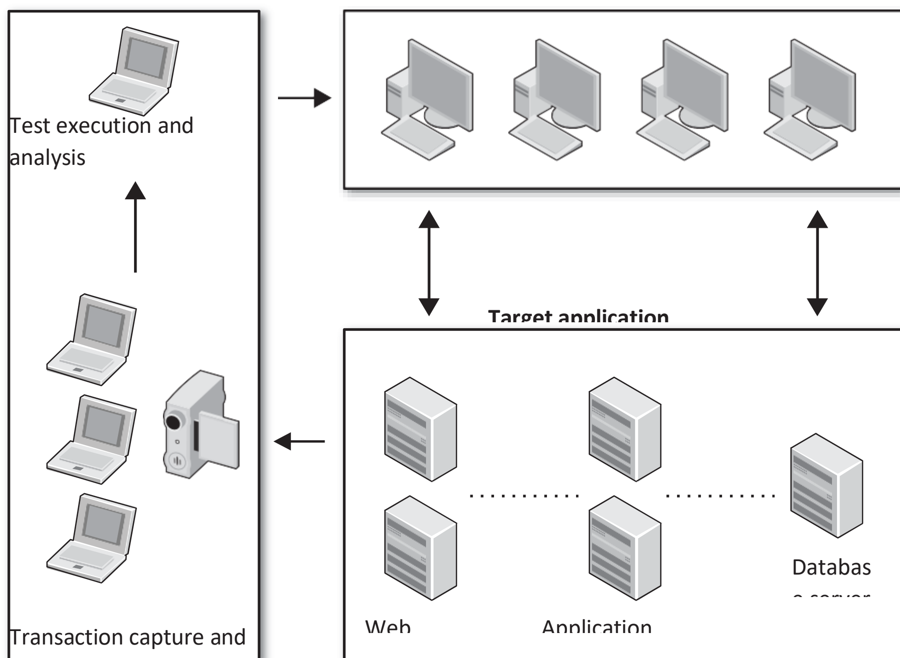



Рис. 1.1 Типове розгортання інструменту продуктивності

достатньо зазначити, що оцінка інструменту тестування продуктивності повинна включати наступні кроки.

Підтримка постачальників інструментів

Переконайтеся, що постачальник інструментів тестування продуктивності «офіційно» підтримує технологію вашої програми, зокрема те, як клієнт програми спілкується з наступним рівнем програми. Наприклад, типовий клієнт на основі браузера в більшості випадків використовуватиме HTTP або HTTPS як основний протокол зв’язку.

- Модель ліцензування

Більшість інструментів підвищення ефективності пропонують модель ліцензування на основі двох компонентів.

Це може бути організовано в перервах або групах, можливо, пропонуючи початковий рівень до 100 віртуальних користувачів з додатковою оплатою за більшу кількість. Ви також можете знайти постачальників інструментів, які пропонують необмежену кількість віртуальних користувачів за (вищу) фіксовану вартість.

Ви можете придбати інструмент із підтримкою протоколу HTTP, але пізніший запит на підтримку, наприклад, Citrix призведе до додаткових витрат. Технологія програми може бути або не бути заблокованою для віртуальних користувачів. Деякі постачальники також пропонують модель термінової ліцензії, за якою ви можете тимчасово продовжити свою ліцензію для одноразових або випадкових потреб для тестування значно більшої кількості віртуальних користувачів.

- Доказ концепції (POC)

Вам потрібно випробувати цей інструмент у своїй програмі, тому наполягайте на POC. Визначте принаймні дві транзакції для запису: одну, яку можна вважати «тільки для читання» (наприклад, простий пошук, який повертає набір результатів з одного або кількох записів); і той, який вставляє або робить оновлення бази даних програми. Це дозволить вам перевірити, чи записана транзакція дійсно відтворюється правильно. Якщо ваша програма «лише для читання», обов’язково перевіряйте журнали відтворення для кожної транзакції, щоб переконатися, що немає помилок, які вказують на помилку відтворення.

- Робота зі сценарієм

Багато постачальників інструментів підвищення продуктивності стверджують, що немає необхідності вносити зміни вручну до сценаріїв, які створюють їхні інструменти. Це може бути правдою для простих транзакцій браузера, які переміщуються навколо кількох сторінок, але реальність така, що вам доведеться заглибитися в код у певний момент під час сценарію транзакцій.

У результаті ви можете виявити, що для успішного відтворення записаної транзакції необхідно внести багато вручну. Якщо кожен набір змін потребує кількох годин на виконання сценарію одним інструментом, але здається, що вони автоматично обробляються іншим, тоді вам потрібно врахувати потенційну економію коштів і вплив додаткової роботи на ваш проект тестування продуктивності та членів команди тестування. .

Зверніть також увагу на рівень кваліфікації вашої команди тестування. Якщо вони здебільшого займаються розробкою, то забруднити руки в кодуванні не повинно бути великої проблеми. Однак якщо вони відносно не знайомі з кодуванням, можливо, краще розглянути інструмент, який надає багато функцій, керованих майстром, щоб допомогти у створенні та підготовці сценарію.

- Рішення проти інструменту тестування навантаження

Деякі постачальники пропонують лише інструмент для тестування навантаження, тоді як інші пропонують рішення для тестування продуктивності. Пропозиції рішень неминуче коштуватимуть дорожче, але загалом забезпечують набагато більший ступінь деталізації аналізу, який вони надають. На додаток до навантажувального тестування вони можуть включати будь-яке або все з наведеного нижче: автоматизоване керування вимогами, автоматичне створення та керування даними, налаштування й оптимізацію додатків перед тестуванням продуктивності, прогнозування часу відгуку та моделювання потужності, аналіз сервера додатків до рівня компонентів, і інтеграція з моніторингом досвіду кінцевого користувача (EUE) після розгортання.

## 1.3 Аналіз програмних засобів для тестування продуктивності

Тестування програмного забезпечення використовується для виявлення помилок у програмному продукті. Тестування програмного забезпечення визначає повноту, правильність продукту, а також використовується для покращення якості продукту. Тестування не гарантує безпомилкового програмного забезпечення. Натомість тестування допомагає виправити помилку в програмному забезпеченні. Існують різні підходи до тестування, які залежать від вимог до програмного забезпечення, категорії програмного забезпечення та доступних ресурсів. Простими словами, тестуванняце «перевірити продукт і оцінити це”.

Термін перевірки означає те, що тестувальник хоче узгодити вимоги з фактичним продуктом і реакцією продукту на бездіяльність поведінки на аналіз тестера. Хоча більшість інтелектуальних властивостей ідентичні перевірці вимог, словесне тестування стосується динамічного аналізу продукту. Тестування допомагає нам покращити якість програмного забезпечення. Якість програмного забезпечення можна покращити шляхом залучення нефункціональних атрибутів відповідно до стандарту ISO-9126. Тестування стосується перевірки та підтвердження програмного забезпечення, чи воно працює так, як було задумано. Це передбачає тестування продукту за допомогою статичних і динамічних методологій через людські помилки, ручний дизайн. Отже, якість програмного забезпечення може бути досягнута шляхом виконання заходів із забезпечення якості. Зазвичай розробник витрачає 40% вартості програмного забезпечення на тестування. Наприклад, монітор банківських транзакцій, контроль може коштувати в 3-5 разів дорожче, ніж усі інші дії разом, через антагоністичний характер тестування, розробник не враховує нотацій у своїй розробці програмного забезпечення.

У літературі було проведено значну роботу з порівняльного аналізу HP LoadRunner і Apache JMeter, але обмежена робота щодо BlazeMeter. Обмеження статті, запропонованої V.Chandel [1], полягає в тому, що вона лише описує Apache JMeter HP LoadRunner, але не відображає результати в реальному часі.

Інструменти автоматизованого тестування веб-сервісів, представлені в дослідженні [2], дуже цікаві, оскільки детально описують Apache JMeter, SoapUI та Storm, але вони не виправдовують, який із них кращий. У статті [3] M.С.Шарміла. та ін. Обговорили Apache JMeterin добре виховано, але фактори для порівняння з іншими інструментами не зосереджені. У дослідженні [4] K. Tirghoda детально проілюстрував Apachetool, але жодного сценарію для веб-сервісів, які тестуються за допомогою Apache tool, не створено. Садік та ін. [5] використовує час відгуку, пропускну здатність, затримку, масштабованість і використання ресурсів для Apache JMeter, але не окреслює проблеми безпеки, пов’язані з Apache JMeter. B. Patel та ін. [6] порівнював два інструменти тестування продуктивності, наприклад

LoadRunner і JMeter. Вони порівнювали такі параметри, як потужність генерування навантаження, інсталяція, звітність про результати кваліфікації завантаження, вартість, технічну якість програмного забезпечення та надійність. Порівняльний аналіз між HPLoadRunner і Apache JMeter виконано Р. Б. Ханом [7], але автор націлює лише веб-сайти LOANcalculator і BMI calculator через недостатній трафік на цих веб-сайтах. Автори в[8] висвітлюють два інструменти тестування навантаження: Sikuli та комерційний інструмент для приймального тестування. Вони порівнюють статичні властивості та систему управління промисловим трафіком, але між цими інструментами немає статистичної різниці; обидва виконали те саме автоматизоване тестування. У дослідженні [9] емпіричний аналіз інструментів тестування веб-сервісів виконується з технічними характеристиками, а порівняння завершується лише на основі продуктивності. У нещодавньому дослідженні [10] проведено порівняльний аналіз Selenium, Soap UI, HP QTP/UFT і Test Complete на основі різних функцій. Автори використовують 3-бальну шкалу, тобто добре, середньо і погано в порівнянні. Результати представлені у вигляді графіків на основі розрахункових значень для вибраних засобів, і мило вважається найкращим засобом серед них.

Apache JMeter [11] — це програмне забезпечення Apache з відкритим вихідним кодом, чиста програма Java, розроблена для виконання функціональної поведінки та тестування продуктивності спеціально для веб-програм, які далі розширюються в інших програмах як на статичних, так і на динамічних ресурсах (веб-служби SAOP/RESET), веб-динамічні мови PHP, Java і Asp.netfiles, але JMeter не виконує Javascript, знайдений на сторінках html, і не відтворює сторінки html, як це робить браузер. Його також можна використовувати для графічного аналізу продуктивності або для перевірки поведінки сервера\скрипту\об'єкта під високим одночасним навантаженням. На малюнку 1 показано, що коли запит, надісланий користувачем, безпосередньо підтверджується сервером, сервер відповідає запит користувача, тоді JMeter збирає дані та обробляє статистичну інформацію, щоб подальше завдання було виконано та відображено результати.

На малюнку 2 показано план тестування для початку тестування facebook. План тестування вимагає чотирьох елементів: http defaultrequest, http, менеджер файлів cookie, слухач і результат графіка На малюнку 3 описано кількість користувачів разом із часом початку та завершення тесту з підрахунком циклів у кожному потоці під час створення користувачів. На малюнку 4

чорні мітки показують дані Facebook, синя лінія вказує на незначну зміну середньої кількості користувачів, рожева лінія вказує на відсутність змін у середньому під час тестування. Проте червона лінія показує, що зміна відхилень відбувається постійно, тоді як вихід зеленої лінії збільшується зі збільшенням кількості користувачів. На осі у максимальний час становить 63068 мілісекунд, це час, необхідний для завершення тесту уніфікованого визначення ресурсів Facebook. Тут результат графіка відображає візуальну модель зразків Facebook, які можна читати та записувати з файлу.

Рис. 1.2 Механізм роботи Apache JMeter [3]

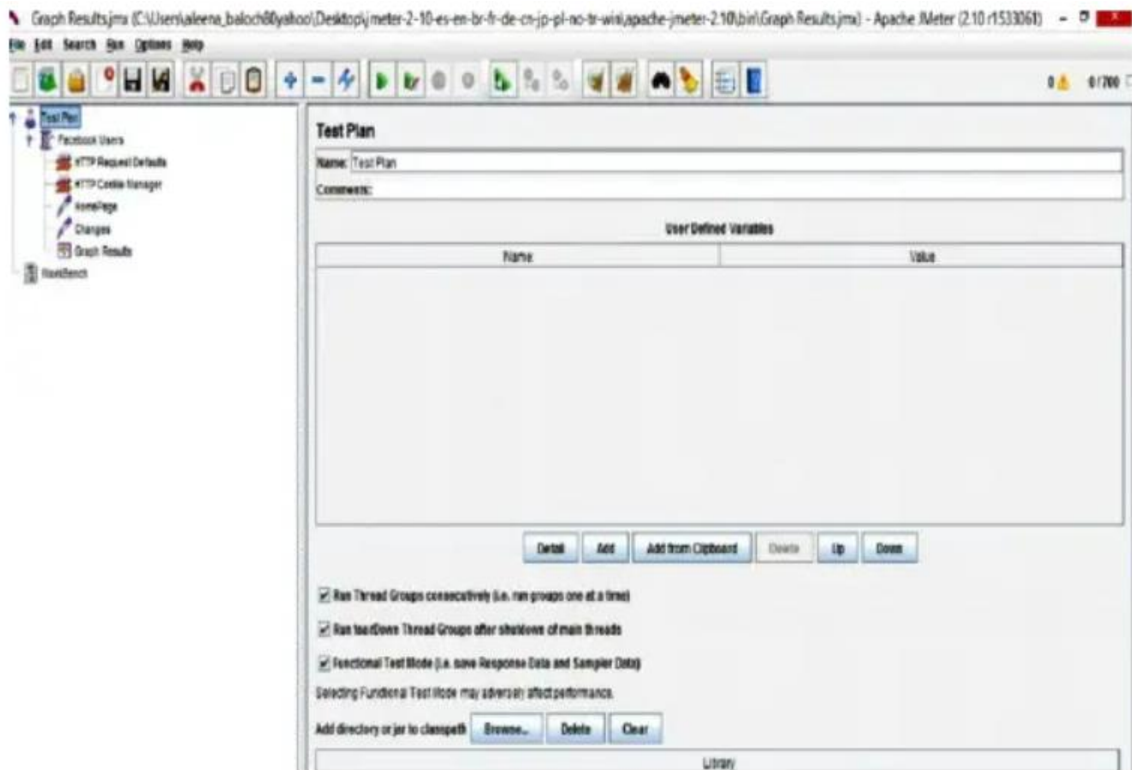



Рис. 1.3  Перевірка URL-адреси Facebook за допомогою Apache JMeter

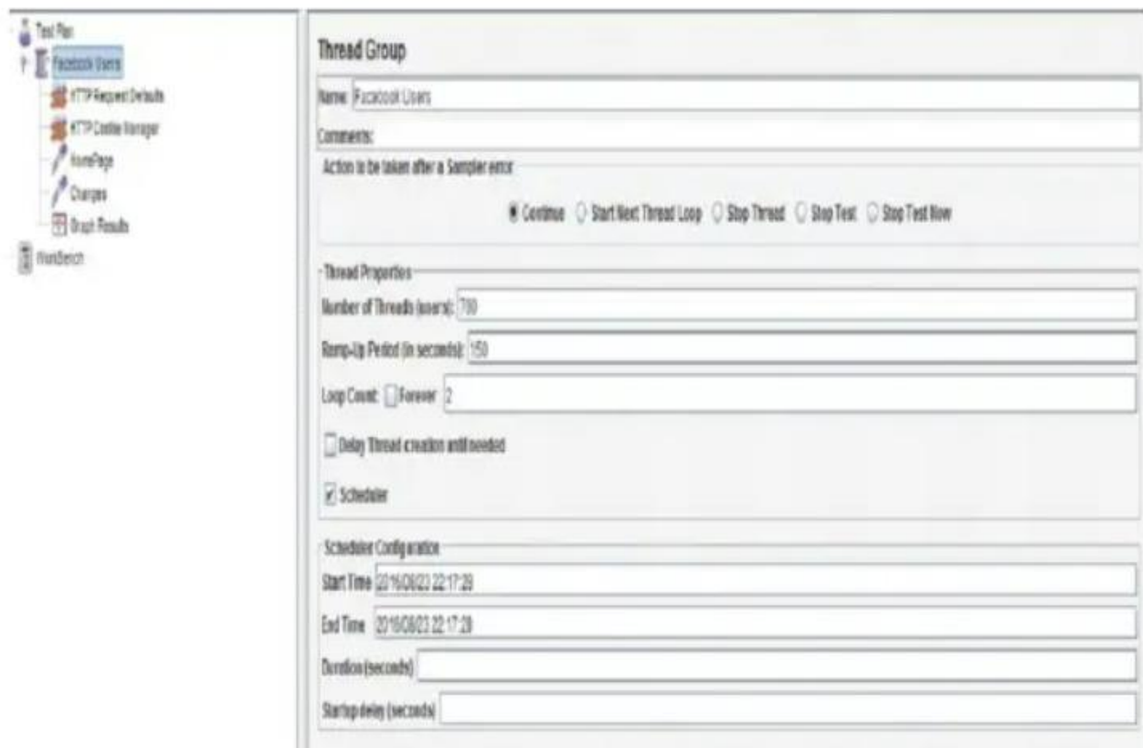



Рис. 1.4 Група потоків перевірки URL-адреси facebook за допомогою Apache JMeter.

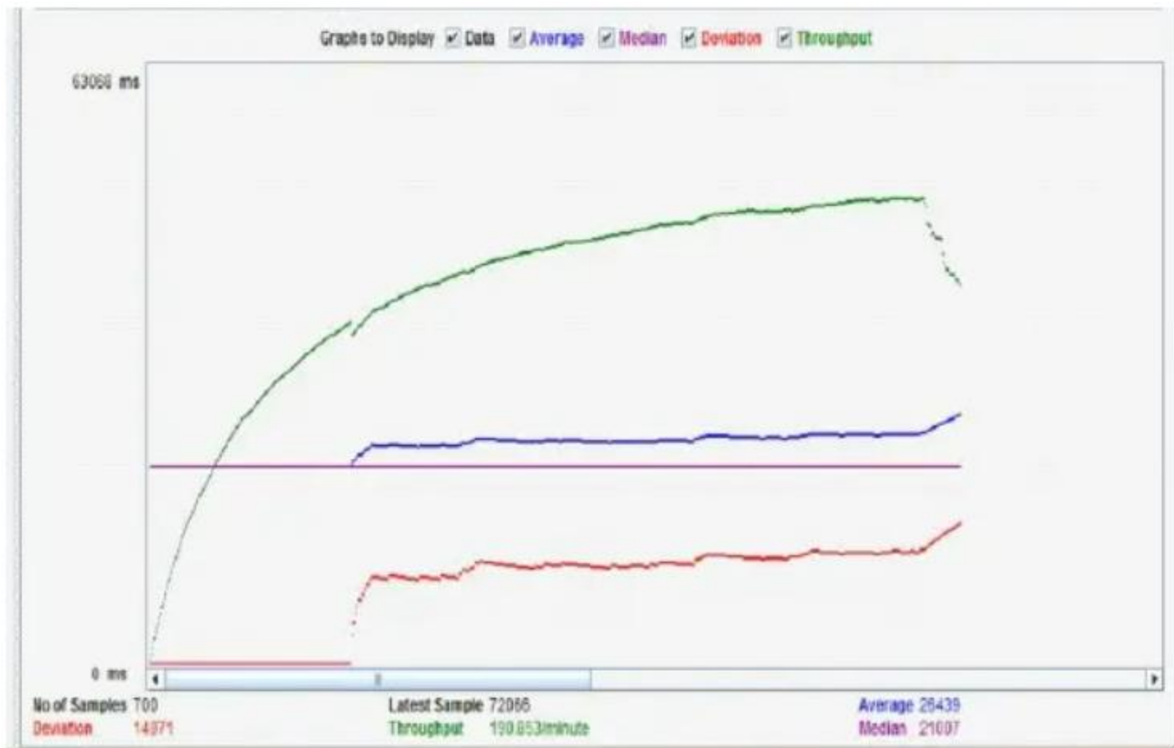



Рис. 1.5 Графічне зображення перевірки URL-адреси Facebook.

BLAZEMETER [12] — це підприємство, повністю сумісне з Apache JMeter. Це забезпечує просту інтеграцію розробника інструмент у своє рідне середовище розвитку. Blaze Meter використовується для тестування мобільних веб-додатків; веб-сайт, веб-сервіси та тестування баз даних, які можуть імітувати тисячі користувачів. Мета цієї статті полягає в тому, щоб провести порівняльний аналіз соціального веб-сайту Facebook, щоб покращити продуктивність соціальних веб-сайтів, зробивши систему більш надійною за меншого часу відгуку.

Метою даної роботи є вдосконалення продуктивність соціального сайту «facebook» від виконання навантажувального тестування за допомогою BlazeMeter замість Apache JMeter. Оскільки ApacheJMeter дає збій, коли масштабованість продукту збільшується, а поведінка є негнучкою, це означає, що модифікації не можуть бути застосовані після виконання тестування, навіть якщо вони потрібні. Продуктивність інструментів тестування оцінюється на основі наступних факторів. Гнучкість: це нефункціональні атрибути якості розробки програмного забезпечення. Гнучкість означає, наскільки легко можна внести зміни в систему. Масштабованість: масштабованість означає, наскільки легко

систему можна розширити за рахунок збільшення кількості користувачів. Продуктивність: продуктивність — це здатність системи виконувати завдання. Контролер навантаження: пристрій, що використовується для регулювання кількість потужності, яку може споживати навантаження. Це може бути використано сторонньою енергетикою або комунальною службою для зниження потреб споживача в енергії в певний час. Надійність: здатність системи виконувати безвідмовну роботу протягом певного часу в певному середовищі.

Зведені звіти: дозволяють переглядати адміністративну інформацію для різних налаштувань і статусу. Час затримки: визначається як кількість часу, який потрібен повідомленню, щоб змінити систему або досягти призначення.

Таблиця 1.1.

Порівняння інструментів тестування на основі різних факторів.

| | **Фактор** | **BLAZEMETER** | **Jmeter** |
|---|---|---|---|
| **1** | Гнучкість | + | - |
| **2** | Масштабованість | + | - |
| **3** | Навантаження | - | + |
| **4** | Контролер завантаження | - | + |
| **5** | Надійність | - | + |
| **6** | Зведені звіти | + | - |
| **7** | Час затримки | + | + |

Рис. 1.6 Результат перевірки URL-адреси facebook за допомогою BlazeMeter на кількість звернень.середнє

Рис. 1.7 Результат тесту URL-адреси facebook BlazeMeter для затримки за допомогою BlazeMeter для затримки

На рисунку 6 показано, що в кожній точці яєчко передається віртуальними користувачами з максимальною пропускною здатністю та середнім часом відповіді 157 мілісекунд.

У рисунку 1.7 стовпець 2 описує кількість зразків із значенням 19612, стовпець 3 показує середній час відповіді на значення 187,3 мілісекунди, стовпець 4 відображає середню кількість звернень, стовпець 5 визначає час, у мілісекундах, стовпець 6 ілюструє мінімальний час відгуку, стовпець 7 середня пропускна здатність, а колонка 8 означає помилку 0% означає успішне проходження тесту.

На рисунку 6 показано, що в кожній точці яєчко передається віртуальними користувачами з максимальною пропускною здатністю та середнім часом відповіді 157 мілісекунд.

У таблиці 3 стовпець 2 описує кількість зразків із значенням 19612, стовпець 3 показує середній час відповіді на значення 187,3 мілісекунди, стовпець 4 відображає середню кількість звернень, стовпець 5 визначає час, у мілісекундах, стовпець 6 ілюструє мінімальний час відгуку, стовпець 7 середня пропускна здатність, а колонка 8 означає помилку 0% означає успішне проходження тесту.

# 2 РОЗРОБКА МЕТОДИКИ ТЕСТУВАННЯ З ВИКОРИСТАННЯМ НАВАНТАЖЕННЯ СЕРВЕРУ

## 2.1 Постановка задачі тестування продуктивності для аналізу стану сервера під навантаженням

У цьому документі, коли ми говоримо про тестування продуктивності програмного забезпечення, ми матимемо на увазі всі дії, пов'язані з оцінкою очікуваної роботи системи в польових умовах. Це оцінюється з точки зору користувача та зазвичай оцінюється з точки зору пропускної здатності, часу відповіді стимулу або деякої комбінації двох. Крім того, для оцінки рівня доступності системи можна використовувати тестування продуктивності. Наприклад, для багатьох телекомунікаційних або медичних застосувань система завжди доступна. У цій структурі існує кілька різних цілей, які можна поставити перед тестуванням продуктивності.

1. Вони включають: розробку алгоритмів вибору тестового випадку або генерації, спеціально призначених для перевірки критеріїв продуктивності, а не критеріїв функціональної коректності.

2. Визначення показників для оцінки повноти алгоритму вибору тестового випадку для даної програми.

3. Визначення показників для порівняння ефективності різних стратегій тестування продуктивності для даної програми.

4. Визначення зв'язків для порівняння відносної ефективності різних стратегій тестування продуктивності в цілому.

Це вимагає, щоб ми могли якось конкретно сказати, що означає, що ця стратегія тестування продуктивності є кращою за ту. Порівняння різних апаратних платформ або архітектур для певної програми. Існує також ряд речей, які необхідно виміряти під час оцінки продуктивності програмної системи. Серед них: використання ресурсів, пропускна спроможність, час

реакції стимулу та довжина черги з детальною інформацією про середню або максимальну кількість завдань, які очікують на обслуговування вибраними ресурсами. Типові ресурси, які необхідно враховувати, включають вимоги до пропускної здатності мережі, цикли ЦП, дисковий простір, операції доступу до диска та використання пам’яті [4]. Інші ресурси, які можуть бути важливими для конкретних проектів, включають використання комутатора (для телекомунікаційних проектів) або швидкість доступу до бази даних.

Іноді неможливо задовольнити всі запити на ресурси одночасно. Якщо загальні «вимоги» до дискового простору різними процесами перевищують доступний простір, цю проблему слід виявити якомога раніше на етапі вимог. У деяких випадках систему доведеться перебудовувати. Це може бути дуже дорогий процес із серйозними наслідками. В інших випадках процесам просто доведеться скоротити потребу в ресурсах, можливо, шляхом розробки кращих алгоритмів або зменшення функціональності. Іншою альтернативою може бути замовлення додаткового обладнання для задоволення вимог. У будь-якому випадку, чим раніше буде усвідомлено потенційні проблеми з продуктивністю, тим більша ймовірність того, що можна буде розробити прийнятне рішення, і тим економніше можна буде виконати будь-яку необхідну роботу з перебудови архітектури. Незважаючи на те, що тестування продуктивності може бути дорогим і трудомістким, ми вважаємо, що, тим не менш, воно буде економічно ефективним з причин, описаних вище.

## 2.2 Розробка методу аналізу сервера під навантаженням

Одним із традиційних способів тестування продуктивності є розробка еталонного тесту: робочого навантаження, яке є репрезентативним для того, як система буде використовуватися в польових умовах, а потім запуск системи на цих еталонних тестах. Поведінка системи на еталонних тестах вважається показником того, як система поводитиметься в полі. Звичайно, поняття репрезентативного навантаження саме по собі проблематично з кількох причин.

Перша проблема полягає в тому, звідки беруться ці дані. Це одне з найлегших для нас завдань. У нашому (телекомунікаційному) середовищі багато проектів регулярно відстежують системний трафік, таким чином надаючи тестувальнику ресурси для розробки так званого робочого профілю, який описує, як система історично використовувалася в польових умовах і, отже, ймовірно, буде використана в майбутнє. Операційний профіль — це розподіл ймовірностей, що описує частоту, з якою виконуються вибрані «важливі» операції. Зазвичай для цього потрібно, щоб фахівець із домену вирішував, які операції насправді є центральними для певної програми. Це також вимагає, щоб хтось приймав рішення щодо вікна спостереження, з якого будуть отримані дані. Якщо попередня версія цієї системи недоступна або дані про історичне використання не були зібрані для системи, часто існує подібна система, яка може надати вказівки для розробки еталонного тесту. В якості альтернативи, якщо існує система, вихідні дані якої стають вхідними даними для системи, що розглядається, і ця стара система контролюється, необхідні дані можуть бути доступними. Загальний прикладом аналізу навантаження може бути алгоритм, що описано укрупненою блок схемою та наведено на рисунку 2.1.

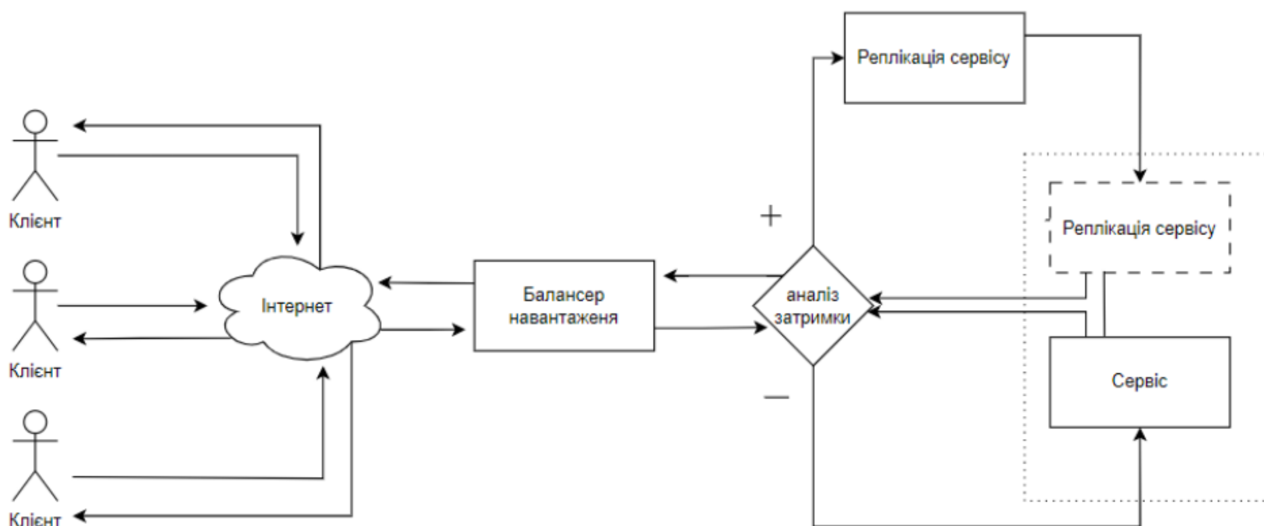



Рис. 2.1 Блок-схема роботи балансера навантаження

Друге питання, яке необхідно розглянути, полягає в тому, чи повинен еталон відображати середнє робоче навантаження чи дуже велике чи стресове навантаження. У будь-якому випадку знову необхідно розглянути вікно спостереження, з якого будуть взяті середні або напружені навантаження. Для багатьох телекомунікаційних систем зазвичай вважається, що період повторюваності становить один тиждень. Таким чином, за винятком особливих випадків, таких як свята чи природні (або неприродні) катаклізми, використання системи в певний день тижня виглядатиме майже ідентично тому самому дню тижня протягом наступного тижня. У [2] дані системного трафіку були надані для 24-годинного робочого дня протягом п'яти послідовних тижнів. Дослідження цих графіків узгоджується з цим спостереженням. П’ять графіків дуже схожі, мають однакову форму та однакову «годину зайнятості» кожного тижня

Ще одне питання, яке слід розглянути, полягає в тому, чи буде використовуватися середня швидкість надходження подій, чи буде використовуватися більш точний розподіл ймовірностей, що описує зміну часу надходження. Хоча цей розподіл ймовірностей може мати те саме середнє значення, він представлятиме значно точнішу картину поведінки системи. Однак це буде набагато дорожче визначити, а математика, залучена до цього, буде складнішою.

Для того, щоб проводити тестування продуктивності програмного забезпечення значущим чином, також необхідно мати вимоги до продуктивності, надані конкретним способом, який можна перевірити. Це має бути чітко включено до вимог або специфікаційного документа та може бути надано у термінах пропускної здатності або часу реакції стимулу, а також може включати вимоги доступності системи. Часто, на жаль, такі вимоги не надаються, що означає, що немає точного способу визначити, чи є продуктивність прийнятною. Крім того, одна з найсерйозніших проблем із тестуванням продуктивності полягає в тому, щоб переконатися, що заявлені

вимоги дійсно можна перевірити, щоб побачити, чи виконуються вони чи ні. Подібно до того, як не дуже корисно вибирати вхідні дані, для яких неможливо визначити, чи є кінцевий вихід правильним, під час тестування програмної системи на правильну функціональність, під час тестування продуктивності так само важливо написати вимоги, які перевіряється. Досить легко написати вимоги до продуктивності для компілятора, такі як: кожен модуль повинен бути скомпільований менш ніж за одну секунду. Хоча можна було б показати, що ця вимога не задовольняється шляхом компіляції модуля, компіляція якого займає більше однієї секунди, той факт, що компілятор був протестований на багатьох модулях, усі з яких правильно скомпільовані менш ніж за одну секунду, робить не гарантує, що вимога була задоволена. Таким чином, навіть правдоподібні вимоги до продуктивності можуть бути неперевіреними.

Більш задовільний тип вимог до продуктивності вказує на те, що система повинна мати коефіцієнт використання ЦП, який не перевищує 50%, коли система працює із середнім навантаженням. Якщо припустити, що було створено еталонний показник, який точно відображає середнє робоче навантаження, можна перевірити, чи було задоволено цю вимогу. Іншим прикладом вимог до продуктивності, які можна перевірити, може бути те, що коефіцієнт використання процесора не повинен перевищувати 90%, навіть якщо система працює з навантаженням, що дорівнює найвищому навантаженню, яке коли-небудь зустрічалося протягом будь-якого періоду моніторингу. Подібним чином можна перевірити такі речі, як максимальна довжина черги або загальна пропускна здатність, необхідна за певних навантажень. У цьому розділі описується, як  використовували наш підхід до тестування продуктивності для тестування системи шлюзу, яка є середнім рівнем трирівневої програми обробки транзакцій клієнт/сервер. Система приймає вхідні дані від абонента та повертає інформацію, яка знаходиться на хост-сервері. Вхідні дані можуть бути у формі двотональних багаточастотних сигналів (тони дотику) або мови з обмеженим словниковим запасом. Вихід має форму попередньо записаного оцифрованого або згенерованого комп'ютером мовлення. Клієнт або перший рівень цієї архітектури

складається з блоків голосового відповіді (VRU). Це комп'ютерні системи, які завершують дзвінок. Сервер або третій рівень складається з головних комп'ютерів мейнфреймів, які містять інформацію, яку запитують кінцеві користувачі. Середній рівень або другий рівень складається зі шлюзів: комп'ютерних систем, які дозволяють клієнтам взаємодіяти з серверами. Щоб мати можливість спілкуватися з різними хост-серверами, шлюз підтримує три різні мережеві протоколи: SNA/3270, TN3270 і TCP/IP. З архітектурної точки зору мета шлюзу полягає в тому, щоб забезпечити концентрацію мережевих з'єднань і звільнити частину обробки, яку в іншому випадку мав би виконувати кожен VRU. Шлюзи та VRU спільно використовують локальну мережу.

Хост-сервери є віддаленими системами. Один шлюз забезпечує обслуговування багатьох VRU. І навпаки, VRU може отримати доступ до послуг, які надаються кількома шлюзами. Один шлюз також може підключатися до кількох віддалених хостів. Архітектура системи показана на малюнку 1. Сценарій програми, що працює на VRU, взаємодіє з кінцевим користувачем. Коли програмі потрібно виконати транзакцію хоста, повідомлення між процесами зв'язку (IPC) надсилається до відповідного процесу сервера, який працює на шлюзі. Для наших цілей транзакція визначається як запит, надісланий хосту, і відповідь, отримана від хоста. Серверний процес на шлюзі форматує запит відповідно до вимог хоста та надсилає його хосту. Коли відповідь отримана, процес на шлюзі аналізує повернуті дані хоста, форматує повідомлення IPC і надсилає відповідь до програми на VRU. Архітектура програмного забезпечення високого рівня показана на малюнку 2. Існуючі системи шлюзів базуються на апаратному забезпеченні ПК (процесори Intel Pentium, шина ISA) під керуванням операційної системи UNIX. Щоб підвищити продуктивність, надійність і зменшити витрати на технічне обслуговування, було вирішено оновити апаратну платформу системами середнього рівня, які використовують кілька процесорів PA RISC, мають великий обсяг оперативної пам'яті та здатні підтримувати багато фізичних інтерфейсів вводу-виводу.

Коли нову платформу було придбано, єдина доступна інформація щодо її необхідної продуктивності надійшла з аналітичного дослідження, проведеного командою проекту. Це дослідження оцінило очікувану кількість транзакцій, які має бути оброблена новою платформою, коли вона підключена до хост-систем за протоколами SNA/3270 і TN3270. Було визначено, що 56 посилань, кожна з яких здатна Потрібна швидкість обробки 56 Кбіт/с, і що система повинна буде функціонувати з 80% активних посилань, кожна з яких оброблятиме транзакції в середньому 1620 байт. Хоча постачальник запевнив, що нова платформа зможе впоратися з необхідним робочим навантаженням, він не надав відповідних даних, щоб підтвердити ці заяви. Те, що вони надали, так це дані про продуктивність використання цих систем як серверів баз даних. Однак система шлюзу забезпечує функціональні можливості, які якісно відрізняються від функціональних можливостей, які надає сервер бази даних, і тому ми вважали, що дані про продуктивність, надані постачальником, не вказуватимуть на поведінку, яку ми можемо очікувати в нашому цільовому середовищі. Враховуючи те, що нова платформа містила нові програмні компоненти, продуктивність яких була здебільшого невідома, і враховуючи, що дані про продуктивність, надані постачальником, не були безпосередньо застосовні до системи, яка використовується як шлюз, команда проекту вирішила перевірити продуктивність шлюзу до до розгортання з конфігурацією, подібною до тієї, що використовуватиметься у виробництві. Важливим першим кроком було визначення цілей діяльності з тестування ефективності. Спочатку ми думали, що метою має бути перевірка вимог, встановлених аналітичним дослідженням. Тобто безпосередньо вирішити питання про те, чи зможе шлюз ефективно обробляти очікувану кількість транзакцій за секунду.

Як ми вже згадували раніше, команда проекту вирішила, що буде краще провести тестування продуктивності на конфігурації, подібній до тієї, що використовується у виробництві. Щоб досягти цього, нам довелося позичити обладнання для тестування у постачальника. Доступ до цих закладів був суворо

обмежений одним тижнем. У результаті тестові приклади потрібно було ретельно спланувати.

Було вирішено, що наше тестування продуктивності мало відповідати на запитання, перелічені в останньому розділі як для середнього, так і для пікового навантаження. Це означало, що ми мали визначити, якими були ці навантаження. Досвід підказує, що сценарій використання слід визначати з точки зору тих параметрів, які найбільш суттєво впливатимуть на загальну продуктивність системи. Тому нашим першочерговим завданням було визначення таких параметрів. ЩОБ зробити це, ми спочатку розглянули архітектуру програмного забезпечення шлюзу та його зв'язок із VRU та хост-системами клієнта. Ми відчули, що архітектура програмного забезпечення системи [3] може надати важливу інформацію, яка може бути корисною під час розробки нашого набору тестів продуктивності. Вивчаючи програмну архітектуру шлюзу, ми помітили, що його продуктивність тісно пов'язана з продуктивністю трьох процесів. Першим процесом був новий програмний компонент, якого не було на старій платформі. Існує один такий процес на шлюзі, і його мета полягає в спілкуванні на одному кінці з усіма хостами на основі 3270, а на іншому кінці з різними емуляторами SNA/3270. Другий процес був модифікованою версією старої платформи. Було внесено зміни, щоб цей процес міг працювати з новим процесом, згаданим вище. Жодної додаткової назви не включено. Існує один із цих процесів для кожного посилання на хост, і кожен із цих процесів може взаємодіяти з 25 різними емуляторами. Кожен емулятор підтримує максимум 10 логічних одиниць (LU). Третій процес безпосередньо взаємодіє з VRU. На шлюзі є один із цих процесів, і його було перенесено зі старої платформи без жодних змін. Оскільки перший процес стосується підключень до хост-систем клієнта та процесів, які спілкуються з кінцевим користувачем, а другий також має справу з тими процесами, які спілкуються з кінцевим користувачем, ми припустили, що кількість одночасних активних підключень до хосту клієнта двома параметрами мають бути системи та кількість одночасних абонентів. Додатковими параметрами, які помітно вплинули на загальну продуктивність шлюзу, були кількість транзакцій на виклик, розміри

транзакцій, час утримання викликів, частота між надходженнями викликів і суміш трафіку (тобто відсоток SNA проти TN3270 проти TCP/IP). Щоб визначити реалістичні значення цих параметрів, ми зібрали та проаналізували дані з поточних виробничих шлюзів. Ми використовували трасування для запису певних типів інформації для кожної транзакції. Це включало такі речі, як мітки часу, час відповіді та номер LU.

### 2.3 Модель балансера навантаження

З метою ефективного виконання завдань багатошляхової маршрутизації в мережах зв'язку з неоднорідною архітектурою потрібен підхід, заснований на мінімізації високого рівня мережевого трафіку. Як фізичні характеристики побудови мережі [1-4], так і функціональна асиметрія, яка проявляється, наприклад, у розподілі пропускної здатності каналів зв'язку, що в перспективі може призвести до утворення «вузьких місць» у комунікаційних мереж і таким чином визначають максимальні значення навантаження на елементи мережі, можуть сприяти архітектурній неоднорідності. Це негативно впливає на ефективність балансування навантаження в частині забезпечення екстремальних значень показників QoS [5, 6], наприклад, середньої наскрізної затримки пакетів. Таким чином, це дослідження спрямоване на вдосконалення процесу балансування навантаження для вирішення проблеми багатошляхової маршрутизації в CN з різнорідною архітектурою [3] та адаптацію цих рішень для реалізації в бездротових мережах на основі SDN.

У цьому контексті в роботі пропонується удосконалити математичну модель балансування навантаження в КН [1-2], яка повністю задовольняє вимоги концепції інженерної трафіку. Це вдосконалення передбачало переоцінку критерію оптимальності для існуючих рішень маршрутизації. Наступним кроком пропонується лінійний квадратичний критерій, який має на меті мінімізувати максимальне навантаження трафіку на канали зв'язку в мережі, одночасно

враховуючи інші фактори, що впливають на ефективність використання каналу. В результаті процес балансування навантаження CN був краще організований, а критичний показник QoS мережі, середня наскрізна затримка пакетів, була зменшена. Порівняння отриманих результатів з розрахунками на основі інших критеріїв оптимальності, лінійної функції кількості використаних мережевих каналів зв'язку, було виконано в рамках кількісного аналізу переваг покращення.

У той же час проводився аналіз мережевих топологій, які відрізнялися розміром і ступенем неоднорідності. Середня наскрізна затримка пакетів вимірювалася в мілісекундах при трьох різних параметрах Херста. Кожне налаштування було для іншого типу мережевого трафіку. За допомогою цього дослідження було визначено, що середня наскрізна затримка пакетів може бути зменшена за допомогою запропонованого критерію для організації балансування навантаження в CN з гетерогенною архітектурою. Це пов'язано з тим, що запропонована модель переважно лінійна, що означає, що вона не ускладнить алгоритмічну підтримку та програмне забезпечення сучасних маршрутизаторів [1-4].

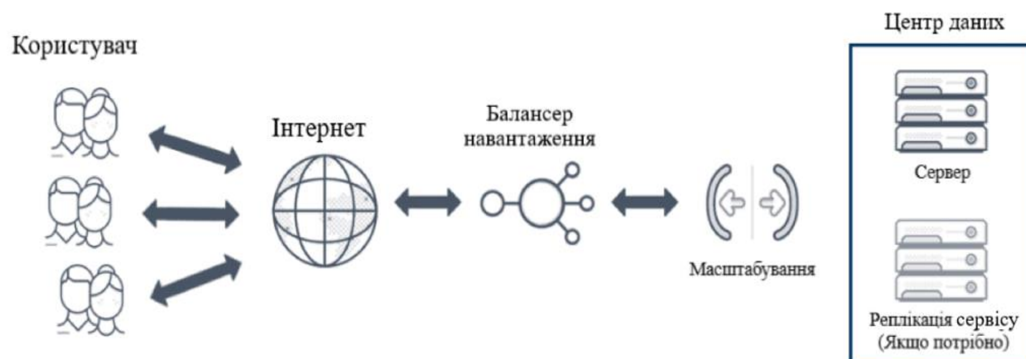



Рис. 2.2 Модель роботи балансера навантаження

Реалізація рекомендацій щодо практичного застосування отриманих моделей та методу балансування навантаження при вирішенні завдань маршрутизації та резервування мережевих ресурсів у КН спрямована

насамперед на створення сприятливих умов та визначення відповідної сфери їх застосування. Це пов’язано з необхідністю системи важливих організаційно-технічних заходів, які впливають на принципи структурно-топологічної та алгоритмічно-програмної реалізації. По-перше, як показали результати проведеного аналізу, запропоноване рішення забезпечує найвищу ефективність у разі використання мережевих структур з високою зв’язністю маршрутизаторів CN [1-4]. Це обумовлено тим, що можна розрахувати набір шляхів і ефективно розподілити між ними навантаження користувачів. В іншому випадку впровадження багатошляхової маршрутизації не покращить якість обслуговування [7-9].

По-друге, запропонована модель балансування навантаження покращить QoS при використанні в CN з неоднорідною архітектурою, тобто коли в мережі є розрізи, які мають набагато меншу пропускну здатність, ніж в інших розрізах. Тому при зборі інформації про стан мережі необхідно оновлювати дані про кількість та з’єднання маршрутизаторів CN та відстежувати стан більш складних фрагментів мережі, а саме її розрізів. Це призведе до деякого розширення набору задач, які вирішуються на маршрутизаторах при визначенні набору оптимальних маршрутів і порядку розподілу навантаження між ними.

Визначення порогових значень є ключовим етапом в процесі моніторингу та автоматизованого управління ресурсами в системах балансера. Цей процес полягає у встановленні конкретних значень для певних метрик, які вказують на необхідність втручання чи зміни в системі.

Для різних параметрів, таких як використання процесора, обсяг пам'яті, мережевий трафік та інші, встановлюються порогові значення, які відображають критичні точки, коли потрібно вжити заходи. Це може бути у вигляді встановлення верхніх меж для використання ресурсів, наприклад, якщо використання CPU перевищило 90% протягом певного часу, або встановлення нижніх меж для попередження про можливі проблеми, наприклад, коли обсяг доступної пам'яті опустився нижче 20%.

Визначення цих порогів потребує аналізу попередніх даних, досвіду роботи системи та встановлення оптимальних значень. Вони можуть бути налаштовані на основі специфіки системи, поточних потреб та можливостей інфраструктури. Крім того, порогові значення можуть бути регулярно переглянуті та вдосконалені для відповідності змінюючимся потребам системи.

Завдяки правильно налаштованим пороговим значенням системи балансера можуть ефективно реагувати на зміни у навантаженні, запобігати перевантаженням та забезпечувати оптимальне використання ресурсів. Вони є основою для автоматичного прийняття рішень щодо масштабування системи та підтримки її працездатності в умовах змінюючихся обставин. Таким чином, визначення порогових значень є ключовим елементом в системах моніторингу та автоматичного масштабування, що сприяє стабільності та ефективності роботи інфраструктури.

По-третє, організація балансування навантаження та резервування ресурсів у КН за запропонованою моделлю більш доцільна для середніх та високих навантажень мережі. У цьому випадку є необхідний запас мережевих ресурсів, які можна перерозподілити та використовувати збалансовано. При низькому навантаженні навіть неоптимальні рішення, такі як одноканальна маршрутизація, зможуть забезпечити необхідний рівень QoS, хоча це призведе до неефективного використання мережевого ресурсу. При критичному навантаженні мережі, тобто при стані КН, близькому до перевантаження, вільного ресурсу не буде, тому використання запропонованих рішень у цьому випадку також не є доцільним [1-4, 9].

По-четверте, практична реалізація запропонованих моделей і методу визначає необхідність підвищення рівня централізації рішень. Крім того, отримані

рішення можуть бути адаптовані до архітектури програмно-визначених мереж. Тут запропоновані моделі та метод балансування навантаження можуть бути реалізовані на контролерах SDN як основа відповідних механізмів маршрутизації та управління трафіком, що доповнюють останні рішення в цій галузі. На практиці можуть бути призначені функції центрального контролера SDN, який збирає інформацію про стан мережі та розрахунок маршрутів. Потенціал SDN для вирішення проблем бездротових технологій, таких як бездротові сенсорні мережі (WSN) на основі SDN, є значним.

У цьому розділі алгоритми балансування навантаження класифікуються на основі різних критеріїв. Пропонується підхід «зверху вниз», який дотримується в процесі класифікації. Обмеження існуючих оглядових робіт полягає в тому, що не існує належної та суттєвої ієрархічної таксономічної класифікації алгоритмів балансування навантаження, що ускладнює визначення того, де той чи інший алгоритм займає своє місце в таксономії. Різноманітні критерії, які використовуються для цілей класифікації, включають «природу алгоритму», «стан алгоритму», «ознаку, що використовується для балансування навантаження», «тип балансування навантаження» та «техніку, що використовується для балансування навантаження». У цій роботі вперше в літературі проведено поглиблений аналіз алгоритмів LB, чого не було в попередніх дослідженнях. Залежно від природи алгоритму, алгоритми балансування навантаження є проактивними або реактивними. Це перша широка категорія, яку ми розмістили в таксономії і яка до цього часу не була показана в жодному з літературних досліджень. Залежно від стану системи алгоритми LB бувають статичними, динамічними або гібридними. На основі ознаки, яка використовується для балансування навантаження, алгоритми LB класифікуються як алгоритми планування та розподілу. На основі типу алгоритмів LB вони згруповані як алгоритми VM LB, алгоритми CPU LB, алгоритми Task LB, алгоритми Server LB, алгоритми Network LB і алгоритми нормального балансування хмарного навантаження. На основі функціональних можливостей методи балансування навантаження згруповані як апаратне балансування

навантаження; еластичне балансування навантаження, а останнє далі групується в балансування навантаження мережі, балансування навантаження програми та класичне балансування навантаження. Виходячи з техніки, алгоритми балансування навантаження класифікуються як; Машинне навчання, еволюційні, натхненні природою, математичні алгоритми та методи на основі роїв.

Перша класифікація алгоритмів балансування навантаження в цій роботі була зроблена на основі природи алгоритму. На основі цієї класифікації алгоритми LB класифікуються як проактивні підходи та реактивні підходи. Однак в інших галузях технології, зокрема в комунікації та мережах для мобільних adhoc мереж (MANETS), природа протоколів маршрутизації зв'язку була широко вивчена за цими двома варіантами [ 36 ].

Алгоритмічна техніка LB на основі проактивності — це підхід до розробки алгоритму, який враховує дії, викликаючи зміни, а не лише реагуючи на ці зміни, коли вони відбуваються. Це має на меті отримати хороший результат, щоб уникнути проблеми заздалегідь, а не чекати, поки проблема виникне. Проактивна поведінка спрямована на виявлення й використання можливостей, а також на вжиття попереджувальних дій проти потенційних проблем і загроз. Обмеження існуючих підходів полягає в тому, що було використано обмежену кількість проактивних підходів, і це теж у традиційний спосіб без нових концепцій. У таблиці 1 зображено проактивні підходи в існуючих підходах LB. Polepally та ін. [ 37 ] запропонував оптимізацію бабки та підхід LB на основі вимірювання обмежень у хмарних обчисленнях шляхом розподілу рівномірного навантаження між віртуальними машинами з мінімальним енергоспоживанням. Сяо та ін. [ 38 ] запропонував заснований на теорії ігор алгоритм LB з урахуванням справедливості для мінімізації очікуваного часу відповіді при збереженні справедливості. Точка рівноваги Неша гри відповідає балансуванню навантаження на оптимальному рівні.

Реактивні підходи діють у відповідь на ситуацію, а не контролюють її. У реактивному підході балансування навантаження проблема дисбалансування навантаження вирішується по мірі її виникнення, після чого видно наслідки. Більшість алгоритмів балансування навантаження підпадають під цю категорію. Основний недолік, який було проаналізовано в літературі з існуючих робіт з балансування навантаження, полягає в тому, що проблема дисбалансування навантаження залишається реалізованою, а потім дослідники пропонують деякі підходи для вирішення цієї проблеми шляхом оптимізації деяких параметрів балансування навантаження [ 32 ], як показано на рис. 2. У таблиці 2 обговорюються реактивні підходи в існуючих підходах LB, наприклад Adhikari et al. [ 39 ] запропонував евристичний алгоритм планування та балансування навантаження для хмари IaaS, щоб мінімізувати час виконання завдання, проміжок часу, час очікування та збільшити використання ресурсів. Проактивні підходи є більш ефективними, ніж реактивні підходи, оскільки перший намагається уникнути проблеми заздалегідь, тоді як останній пропонує рішення після виникнення проблеми.

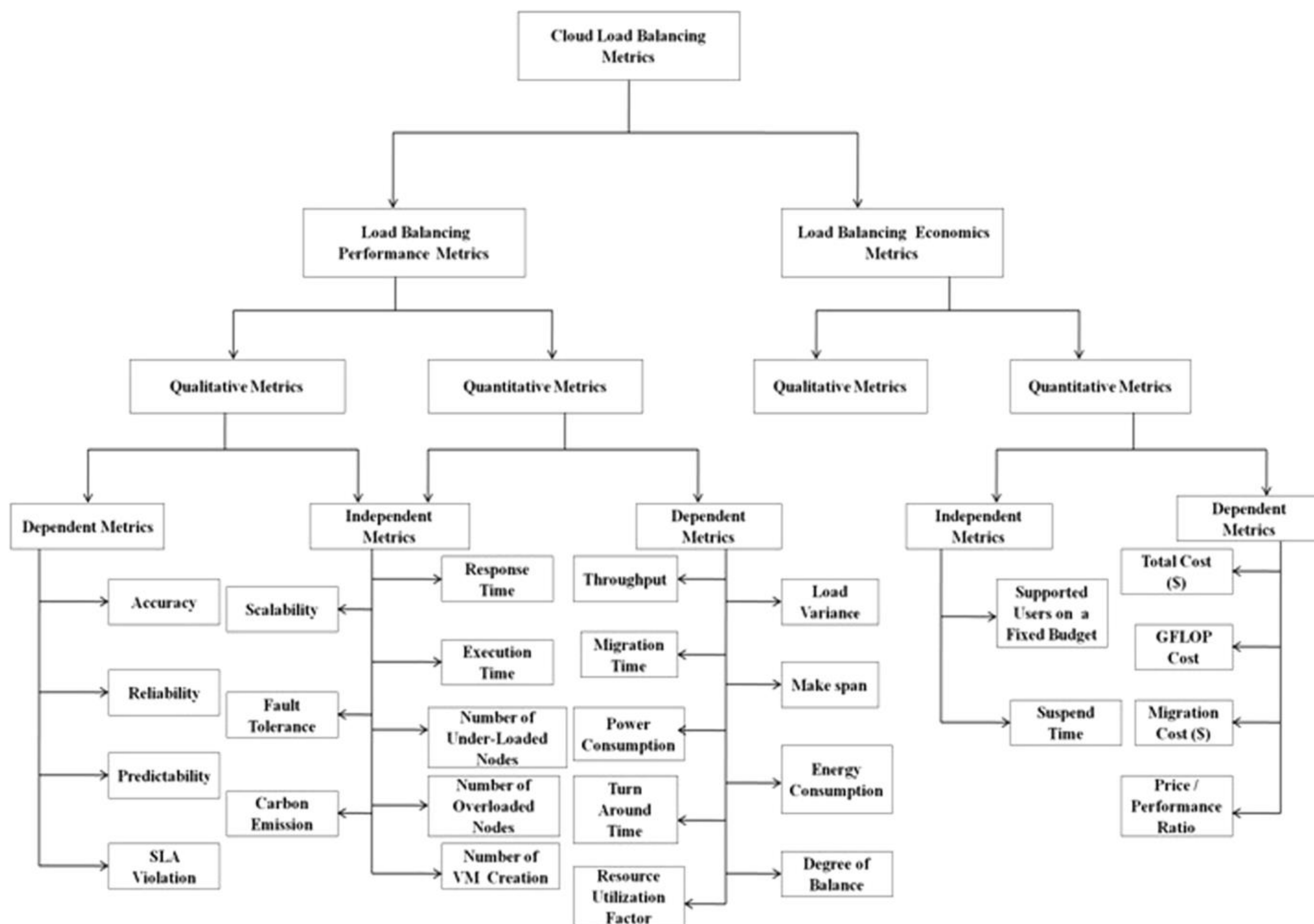



Рис. 2.3 Схема хмарного балансера навантаження

На основі інформації про стан системи, на яку спирається алгоритм, алгоритми LB широко класифікуються як статичні, динамічні та гібридні. З огляду існуючої літератури очевидно, що це найбільш широко використовувана система класифікації для алгоритмів LB. Більшість робіт із порівняльних досліджень балансування навантаження починаються з алгоритмічної таксономії, розміщуючи цю категорію на вершині таксономії. При статичному балансуванні навантаження трафік рівномірно розподіляється між серверами. Це робиться за допомогою алгоритму, який має попередні знання про системні ресурси та вимоги до завдань. Статичний алгоритм LB планує завдання для віртуальної машини для виконання під час компіляції. Перевага статичного алгоритму полягає в його меншій складності, але вони страждають від фатального вузького місця, оскільки не можуть перемістити завдання під час виконання на іншу машину для балансування навантаження. Статичні алгоритми не враховують поточний стан системи та вимагають попередніх знань про машини та завдання, такі як вимоги до ресурсів

завдання, час зв'язку, обчислювальна потужність вузлів, пам'ять, сховище, пропускна здатність тощо. Основним недоліком статичного алгоритму LB є те, що процес міграції неможливий під час виконання завдань і, отже, не підходить для розподіленої системи, як-от хмара, де стан системи змінюється динамічно.

Крім того, на основі режиму виконання завдань динамічні алгоритми згруповані в автономний режим, також званий пакетним режимом, і онлайн-режим або живий режим, як показано на рис. 3 . У пакетному режимі завдання виділяється лише в деяких попередньо визначених випадках, коли, як і в режимі онлайн, завдання користувача зіставляється з віртуальною машиною, щойно воно потрапляє в планувальник. Алгоритми динамічного балансування навантаження є порівняно складними алгоритмами на відміну від своїх аналогів, які обробляють вхідний потік трафіку під час виконання та можуть змінювати стан запущеного завдання в будь-який момент часу. Динамічне балансування навантаження враховує поточний стан системи та здатне справлятися з непередбачуваним навантаженням обробки. Перевага динамічного балансування навантаження полягає в тому, що завдання можуть динамічно переміщатися від перевантаженої машини до недостатньо завантаженої, але є значно складнішими за своєю природою та набагато складнішими для розробки порівняно зі статичними алгоритмами LB.

Однак динамічні алгоритми LB дуже ефективні з точки зору продуктивності, точності та функціональності. Алгоритми статичного балансування навантаження працюють плавно, якщо вузли мають невеликі варіації навантаження, але не можуть працювати в середовищах зі змінним навантаженням. На малюнку 3 показано таксономію балансування навантаження на основі природи та стану алгоритму.

Алгоритми цієї категорії класифікуються як алгоритми планування та розподілу. Алгоритми розподілу та планування в хмарі класифікуються на основі поточного стану ВМ і відповідно можуть бути статичними або

динамічними. Політики розподілу та планування відіграють важливу роль у управлінні ресурсами та моніторингу продуктивності хмари, що, у свою чергу, добре впливає на доставку QoS користувачу. Політики планування розкладаються на три наступні дії: планування завдань, планування ресурсів і планування віртуальної машини; так само політики розподілу розкладаються на розподіл завдань, розподіл ресурсів і розподіл віртуальної машини відповідно.

Планування завдань — це метод призначення завдань користувача відповідним обчислювальним ресурсам для виконання, тоді як планування ресурсів — це процес планування, керування та моніторингу обчислювальних ресурсів для виконання завдань. Планування віртуальних машин — це процес створення, знищення віртуальних машин і керування ними на фізичному хості, окрім керування віртуальними машинами під час процесу міграції між хостами. Розподіл завдання — це акт розподілу завдання на ресурс, на якому воно має виконуватися. Розподіл ресурсу — це акт виділення ресурсу для завдання для його виконання. Розподіл завдань і розподіл ресурсів є інверсією одне одного. Розподіл віртуальної машини – це розподіл віртуальної машини для користувача або набору користувачів. На рисунку  4 показано алгоритми балансування навантаження на основі використовуваної ознаки.

На основі функціональних можливостей балансувальники навантаження класифікуються як апаратний балансувальник навантаження та еластичний балансувальник навантаження, як показано на рис. 5. Апаратні засоби балансування навантаження стосуються розподілу робочого навантаження на апаратному рівні, тобто пам’яті, пам’яті та ЦП. Еластична балансування навантаження автоматично розподіляє вхідний трафік додатків між кількома цілями, такими як екземпляри Amazon EC2, контейнери та IP-адреси. Він може обробляти різне навантаження трафіку програми користувача в одній зоні доступності або в кількох зонах доступності. Elastic Load Balancing пропонує три типи балансувальників навантаження, які мають високу доступність, автоматичне масштабування та надійну безпеку, необхідну для забезпечення відмовостійких

програм користувача. Application Load Balancer працює на рівні запиту (рівень 7), маршрутизуючи трафік до цілей – примірників EC2, контейнерів та IP-адрес на основі вмісту запиту. Ідеально підходить для розширеного балансування навантаження трафіку HTTP і HTTPS, Application Load Balancer забезпечує розширену маршрутизацію запитів, націлену на доставку сучасних архітектур додатків, включаючи мікросервіси та програми на основі контейнерів. Application Load Balancer спрощує та покращує безпеку вашої програми, забезпечуючи постійне використання найновіших шифрів і протоколів SSL/TLS. Балансувальники мережевого навантаження реалізовані на транспортному рівні моделі OSI. Він здатний обробляти мільйони запитів за секунду. Балансування мережевого навантаження широко використовується Microsoft azure та AWS у моделі розгортання. Функція балансування мережевого навантаження дозволяє розподіляти трафік між серверами за допомогою інтернет-протоколу TCP/IP. Класичний балансувальник навантаження забезпечує базове балансування навантаження в кількох екземплярах Amazon EC2 і працює як на рівні запиту, так і на рівні з’єднання. Classic Load Balancer призначений для додатків, створених у мережі EC2-Classic.

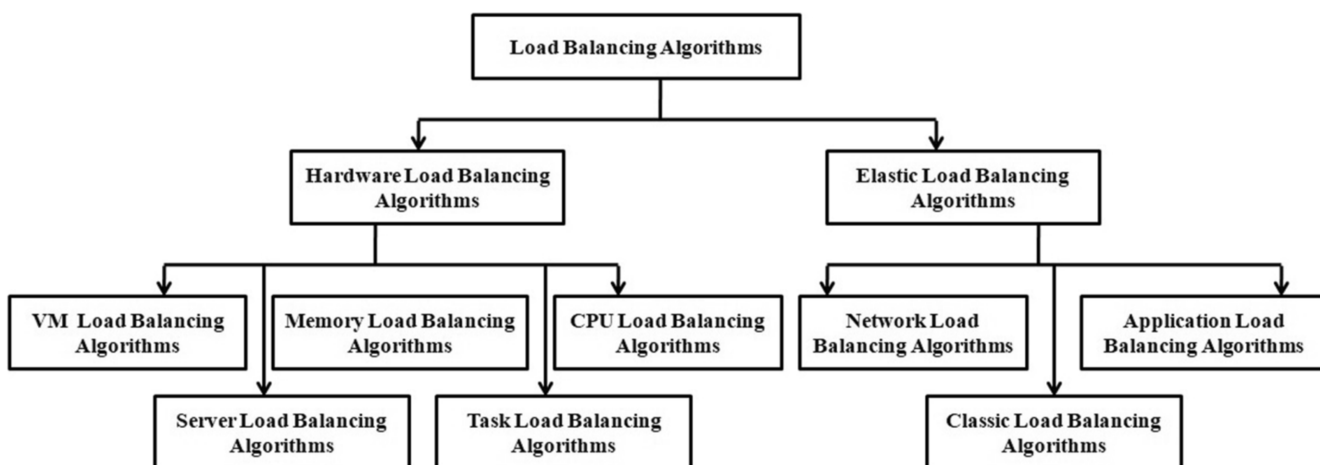


Рис. 2.4 Схема алгоритму балансера навантаження

На основі типу алгоритми LB класифікуються як VM LB, CPU LB, завдання LB, серверний LB, мережевий LB і звичайний хмарний LB, як показано на рис. 5 . Балансування навантаження віртуальних машин — це процес перерозподілу

віртуальних машин від перевантажених вузлів до недостатньо завантажених вузлів, і вперше було представлено як нову функцію папки «Вхідні» у Windows Server 2016, яка дозволяє оптимізувати використання вузлів у кластері збоїв. Балансування навантаження на віртуальну машину визначає надзакріплені вузли та перерозподіляє віртуальні машини з цих вузлів на підзакріплені вузли. Віртуальні машини переносяться в реальному часі з вузла, який перевищує порогове значення, до новододаного вузла в кластері збоїв. Балансування навантаження віртуальної машини досягається за допомогою процесу міграції віртуальної машини. Балансування навантаження ЦП — це процес обмеження навантаження на ЦП у межах його порогового значення. Балансування навантаження — це акт розподілу завдань між віртуальними машинами від перевантажених машин до недостатньо завантажених. LB сервера — це правильний розподіл загального вхідного навантаження в центрі обробки даних або фермі серверів між серверами. Мережевий LB займається управлінням вхідним трафіком без використання складних протоколів.

На основі використовуваної техніки алгоритми балансування навантаження класифікуються як евристичні та метаевристичні методи, а також методи оптимізації.

Евристичний підхід — це підхід до вирішення проблем бухгалтерського обліку. Практичний метод або методологія гарантовано не є оптимальними, досконалими, логічними чи обґрунтованими, але достатніми для досягнення негайної мети. Пошук оптимального рішення може бути неможливим або недоцільним, особливо для балансування навантаження, яке є складною проблемою NP, і евристика відіграє важливу роль у прискоренні процесу пошуку гідного рішення. Евристичні методи розроблені відповідно до стратегій, отриманих із попереднього досвіду вирішення подібних наборів проблем. Евристика відіграє вирішальну роль у процесі балансування навантаження для вирішення різних проблем, з якими стикаються CSP. Було проведено багато дослідницьких робіт із застосуванням евристичних і метаевристичних підходів до балансування хмарного

навантаження, і тому ми класифікували евристичні та метаевристичні методи на природничі алгоритми та класичні алгоритми. Природні алгоритми поділяються на еволюційні алгоритми та алгоритми на основі роїв.

Методи оптимізації використовуються для пошуку оптимальних рішень проблеми. Методи оптимізації в балансуванні хмарного навантаження в цілому класифікуються як класичні та некласичні методи оптимізації. Ці алгоритми можуть бути як стохастичними, так і детермінованими. Подальша класифікація класифікує методи оптимізації на алгоритми, що базуються на обмеженнях і без них, і це може бути або оптимізація за однією ціллю, або за багатьма критеріями. Багатокритеріальна оптимізація також класифікується як багатоатрибутна та багатоцільова оптимізація. Багатоцільові алгоритми можуть базуватися на машинному навчанні, на основі природи, на основі роїв або алгоритмів балансування навантаження на основі математичних даних. На малюнку 6 показано алгоритми балансування навантаження на основі використовуваної техніки.

Моніторинг метрик є критично важливою складовою будь-якої інфраструктури, що дозволяє виявляти, збирати та аналізувати дані про різноманітні параметри працездатності системи, її ресурсів та навантаження. Цей процес забезпечує детальний уявлення про стан інфраструктури та дозволяє приймати інформовані рішення щодо її оптимізації, попередження неполадок та виявлення проблем у реальному часі.

Під час моніторингу метрик відстежуються ключові параметри, такі як використання CPU, обсяг вільної пам'яті, мережевий трафік, завантаження диска, кількість запитів та інші характеристики системи. Ці метрики відображають стан ресурсів та роботу системи в цілому. Наприклад, висока кількість запитів на сервер може свідчити про збільшення навантаження на систему, що може призвести до затримок у відповіді.

Збір метрик проводиться за допомогою балансера навантаження. Цей інструмент дозволяє налаштовувати моніторинг різних параметрів, встановлювати сповіщення про виявлення аномалій чи перевищення порогових значень.

Моніторинг метрик є важливим етапом у системах автоматичного балансування, де вимірювання навантаження та реакція на зміни у використанні ресурсів відбувається автоматично на підставі метрик, зібраних системою моніторингу.

Отримані дані під час моніторингу метрик використовуються для прийняття рішень про балансування, оптимізацію роботи системи, виявлення проблем та їх подальшого усунення. Моніторинг метрик є невід'ємною складовою для забезпечення ефективності, стабільності та доступності інфраструктури в сучасних системах інформаційних технологій.

Система спостерігає за різними метриками, такими як CPU використання, загальне навантаження на сервер, кількість запитів тощо. Ці метрики допомагають визначити поточний стан системи.

Налаштовуються порогові значення для цих метрик, що вказують, коли система повинна змінювати кількість ресурсів.

На основі аналізу метрик і порогових значень система балансування приймає рішення щодо потреби у зміні кількості ресурсів.

Після прийняття рішення система виконує дії зі зміною кількості ресурсів. Наприклад, вона може автоматично додавати нові сервіси або зменшувати їх кількість в залежності від поточного навантаження.

Після внесення змін система балансування перевіряє, чи були здійснені зміни вірно, і забезпечує, що ресурси збалансовані та працюють належним чином.

Процес постійно повторюється, система постійно моніторить метрики, аналізує їх і приймає рішення щодо зміни розміру ресурсів відповідно до поточної ситуації.

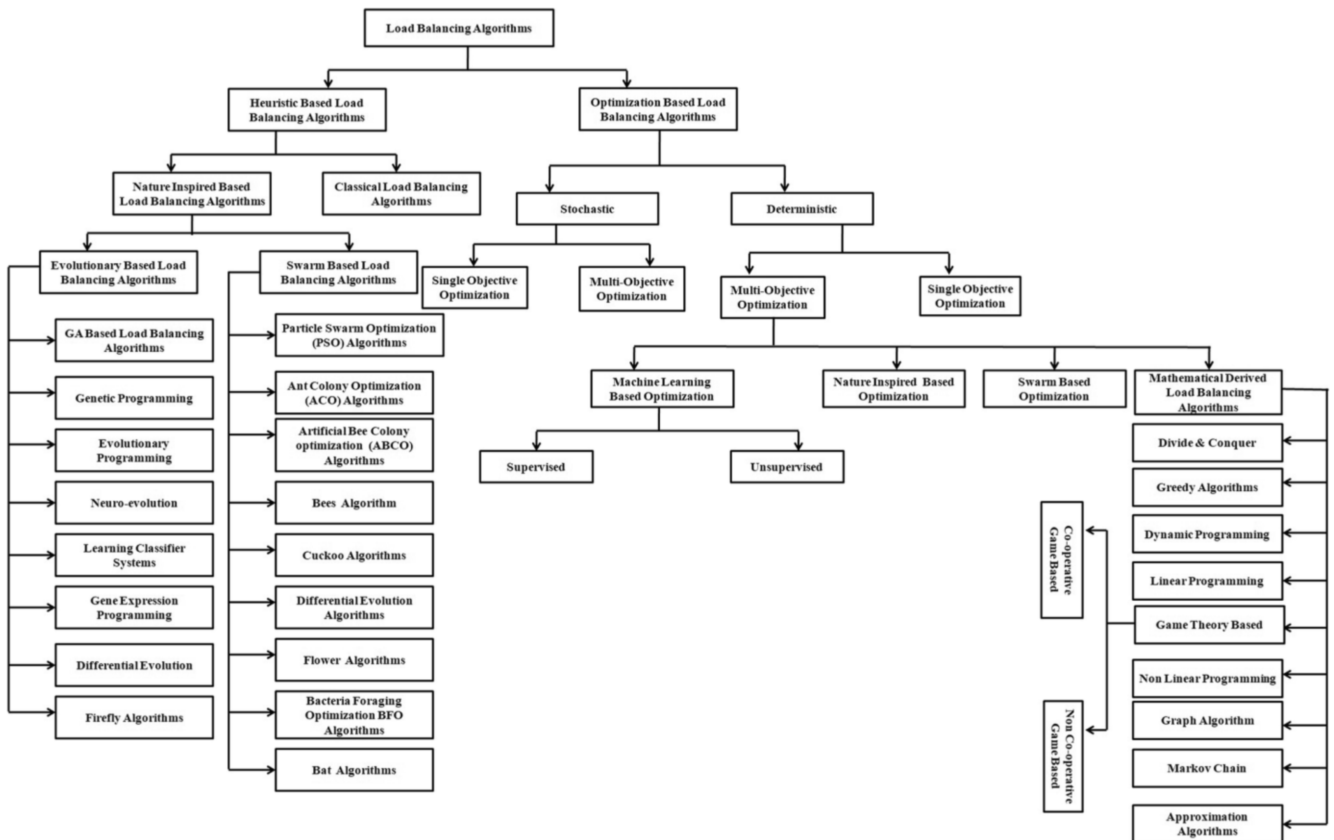



Рис. 2.5 Розширений алгоритм балансера

Таблиця 1 і Таблиця 2 відповідно перераховують різні характеристики проактивних і реактивних підходів у відповідній літературі за різними параметрами. Сильні та слабкі сторони кожного підходу також відображено в таблиці 1 і таблиці 2 . Таблиця 3 зображує різні досліджувані підходи як одноцільові та багатоцільові. У таблиці 3 також висвітлено платформу реалізації, інструмент і середовище моделювання, в яких конкретний підхід вивчався та досліджувався. Нарешті, у таблиці 4 представлені основні показники балансування навантаження, проаналізовані в існуючих підходах.

У цьому розділі викладено результати, отримані в результаті порівняльного аналізу різних підходів до балансування навантаження в хмарних обчисленнях. На малюнку 7(a) показано відсоток різних типів планування в підходах до балансування навантаження на основі проактивного. Зрозуміло, що планування завдань і планування ресурсів, кожне з яких має 45,45% внеску, частіше розглядається в проактивних підходах з меншою увагою до планування віртуальної машини, який сприяє 9,09%. З рис. 7(b) видно, що більшість

реактивних підходів у існуючій літературі вивчалися під час планування завдань, яке становить 51,85%, за яким слідує планування віртуальної машини, яке сприяє 25,93% та планування ресурсів, яке сприяє 22,22% відповідно. Рис. 8 описує відсоток дослідницьких статей про балансування хмарного навантаження, що визначає складність алгоритму. Підраховано, що 80% дослідницьких статей не розглядали алгоритмічну складність у своїй роботі, тоді як лише 20% визначають її у своїй роботі. З рис. 9(a) проаналізовано , що проактивні підходи завжди мають динамічний характер, тоді як рис. 9(b) показує, що більшість реактивних підходів підпадають під динамічний стан алгоритму, який сприяє 68%, за яким слідує статичний алгоритм, який сприяє 20% і гібридний алгоритм, який становить 12%. З рис. 10(a) видно, що 60% проактивних підходів є багатоцільовими, тоді як 40% є підходами з однією ціллю. Подібним чином 56% реактивних підходів є багатоцільовими, тоді як 44% є підходами з єдиною ціллю, як показано на рис. 10(b) . На рисунку 11 показано середовище тестування, у якому було запущено певний підхід для оцінки показників продуктивності. Зрозуміло, що симулятор CloudSim широко використовується для проведення симуляційного експерименту, становлячи 33,33% експериментального впровадження, за яким слід симулятор Cloud Analyst з 19,44% експериментального впровадження. C і C++, реалізація підходів балансування навантаження в Matlab становить 11,11% кожен відповідно, тоді як інші складають 19,44%. Реалізація підходів балансування навантаження хмари в режимі реального часу є дуже низькою і становить лише 5,56%. На рисунку 12 зображено відсоток метрик LB в існуючих підходах, де найбільш широко обговорюються час відгуку, час виконання, використання ресурсів, робочий діапазон, масштабованість і вартість виконання, кожна з яких становить 13,39%, 11,81%, 11,02%, 9,45%, 9,45% і 8,66 % відповідно.

## 2.4 Метод реплікації сервісу

Відповідно до загальноприйнятого розуміння, реплікація - це повторення процедури дослідження та спостереження, чи повторюється попередній результат. Це визначення є інтуїтивно зрозумілим, простим у застосуванні та невірним. Ми пропонуємо, щоб реплікація була дослідженням, для якого будь-який результат вважатиметься діагностичним доказом твердження з попереднього дослідження. Це визначення зменшує акцент на оперативних характеристиках дослідження та посилює акцент на інтерпретації можливих результатів. Метою реплікації є просування теорії шляхом зіткнення існуючого розуміння з новими доказами. За іронією долі, цінність реплікації може бути найсильнішою, коли наявне розуміння найслабше. Успішне повторення надає докази можливості узагальнення в умовах, які неминуче відрізняються від оригінального дослідження; Невдала реплікація вказує на те, що надійність знахідки може бути більш обмеженою, ніж вважалося раніше. Визначення реплікації як протистояння поточним теоретичним очікуванням прояснює її важливу, захоплюючу та генеративну роль у науковому прогресі.

Розробка системи для розрахунку часу відгуку є важливим етапом в створенні програм та сервісів, де швидкість реакції на запити користувачів має вирішальне значення. Використання мови програмування Java для цієї мети відображається у використанні різноманітних методів та бібліотек, що дозволяють точно вимірювати час відгуку системи.

Один з методів розрахунку часу відгуку - використання класу System у Java. Метод currentTimeMillis() цього класу дозволяє отримати час в мілісекундах від початку епохи Unix. За допомогою цього методу можна визначити час початку та завершення виконання певного фрагмента коду чи операції, обчислити різницю між цими часами та визначити час відгуку.

Java також надає інші класи та бібліотеки для точного вимірювання часу відгуку. Наприклад, клас Instant з пакету java.time в Java 8 та пізніших версіях, який надає можливість працювати з часовими мітками з високою точністю.

Додатково, в Java є багато сторонніх бібліотек, таких як Apache Commons або Google Guava, які мають інструменти для вимірювання часу відгуку, створення таймерів та проведення бенчмарків.

Крім простого вимірювання часу відгуку, розробники можуть використовувати різні підходи для покращення часу реакції системи, такі як кешування, оптимізація запитів до баз даних, розпаралелювання або використання асинхронних операцій.

Використання Java для розрахунку часу відгуку дозволяє розробникам ефективно контролювати та вимірювати швидкодію системи. Працюючи з різноманітними інструментами та бібліотеками, вони можуть забезпечити точність та високу продуктивність у вимірюванні часу реакції програм та сервісів.

```
public class ResponseTimeExample {
    no usages
    public static void main(String[] args) {
        long startTime = System.currentTimeMillis();

        // Операції чи функції, що вимірюються щодо часу відгуку
        // Наприклад, виклик функції чи операції:
        // someOperation();

        long endTime = System.currentTimeMillis();
        long responseTime = endTime - startTime;
        System.out.println("Час відгуку: " + responseTime + " мс");
    }
}
```

Рис. 2.6 Використання System.currentTimeMillis()

```java
public class ResponseTimeExample {
    no usages
    public static void main(String[] args) {
        Instant startTime = Instant.now();

        // Операції чи функції, що вимірюються щодо часу відгуку
        // Наприклад, виклик функції чи операції:
        // someOperation();

        Instant endTime = Instant.now();
        Duration duration = Duration.between(startTime, endTime);
        System.out.println("Час відгуку: " + duration.toMillis() + " мс");
    }
}
```

Рис. 2.7 Використання класу Instant з пакету java.time

Розробка системи реплікації даних є важливою складовою створення високодоступних та надійних програмних продуктів. Використання мови програмування Java для реалізації реплікації дозволяє розробникам створювати ефективні, масштабовані та надійні системи для зберігання та управління даними.

```
// Клас, який представляє реплікаційний механізм
2 usages
class ReplicationManager {
    2 usages
    private List<ReplicatedObject> replicas = new ArrayList<>();

    1 usage
    public void replicate(ReplicatedObject object) {
        ReplicatedObject copy = new ReplicatedObject(object.getData());
        replicas.add(copy);
    }

    1 usage
    public List<ReplicatedObject> getReplicas() {
        return replicas;
    }
}

no usages
public class ReplicationExample {
    no usages
    public static void main(String[] args) {
        ReplicationManager manager = new ReplicationManager();
        ReplicatedObject original = new ReplicatedObject( data: "Initial data");

        manager.replicate(original); // Реплікація об'єкту

        List<ReplicatedObject> replicatedObjects = manager.getReplicas();
        for (ReplicatedObject replica : replicatedObjects) {
            System.out.println("Replicated data: " + replica.getData());
        }
```

Рис. 2.8 Приклад реалізації системи реплікації

Цей приклад демонструє простий механізм реплікації об'єктів за допомогою класу ReplicationManager та ReplicatedObject

Останнім часом хмарні обчислення [1, 2, 3] як нова корпоративна модель стали популярними для надання послуг на вимогу користувачам за потреби. Хмарні обчислення — це, по суті, потужна обчислювальна парадигма для надання послуг через мережу. Модель хмарних обчислень була розділена на

інфраструктуру як послугу (IaaS), платформу як послугу (Paas) і програмне забезпечення як послугу (SaaS) [4].

У середовищі хмарних обчислень технологія віртуалізації [5, 6, 7] є імпортною роллю для забезпечення фізичних ресурсів, таких як процесори, дискове сховище та широкосмугова мережа. Віртуалізація стосується насамперед віртуалізації платформи або абстракції фізичних ресурсів для користувачів. У Could ці фізичні ресурси розглядаються як пул ресурсів, тому ці ресурси можуть бути розподілені за вимогою. Обчислення в масштабі хмарної системи дозволяють користувачам отримувати доступ до величезних і еластичних ресурсів за вимогою. Однак попит користувачів на ресурси може бути різним у різний час, підтримання ресурсів, достатніх для задоволення пікових потреб у ресурсах, може бути дорогим. Навпаки, якщо користувач підтримує лише мінімальні обчислювальні ресурси, ресурсу недостатньо для виконання пікових вимог.

Тому динамічна масштабованість є ключовим моментом успіху середовища хмарних обчислень. Динамічне змінення розміру — це функція, яка дозволяє серверу змінювати розмір віртуальної машини, щоб задовольнити нові потреби в ресурсах [7]. Коли віртуальна машина недостатньо або надмірно забезпечена, для подолання цієї проблеми можна використати динамічну зміну розміру. Однак він не підходить для хмарної бізнес-моделі, як Amazon EC2. В Amazon EC2 [8] віртуальна машина є базовою одиницею, і користувач може орендувати одиницю або більше одиниць, а не одиницю зі спеціальними ресурсами. Динамічне змінення розміру не може подолати балансування навантаження, оскільки воно лише збільшує ресурси на певній віртуальній машині. Trieu C. Chieu та ін. [9] запропонував корисну архітектуру для динамічного масштабування веб-додатку у віртуалізованому середовищі хмарних обчислень. Їхня архітектура включає зовнішній балансувальник навантаження, систему моніторингу віртуального кластера та систему автоматичного надання, і ефективно працює для загальних веб-додатків у віртуальних машинах. Однак користувач може орендувати багато віртуальних машин для виконання своїх програм, які не є лише веб-програмами. У той же час з міркувань безпеки віртуальні машини для конкретного користувача

можна розглядати як групу, і користувач не може отримати доступ до ресурсів з інших груп.

У цій статті ми представимо сценарій динамічного масштабування з новим дизайном програм, розгорнутих на віртуальних машинах у віртуальному кластері. Решта паперу організована таким чином. Розділ 2 ілюструє архітектуру віртуального кластера у віртуалізованому середовищі хмарних обчислень. Розділ 3 описує наш алгоритм автоматичного масштабування. Нарешті, розділ 5 завершує документ.

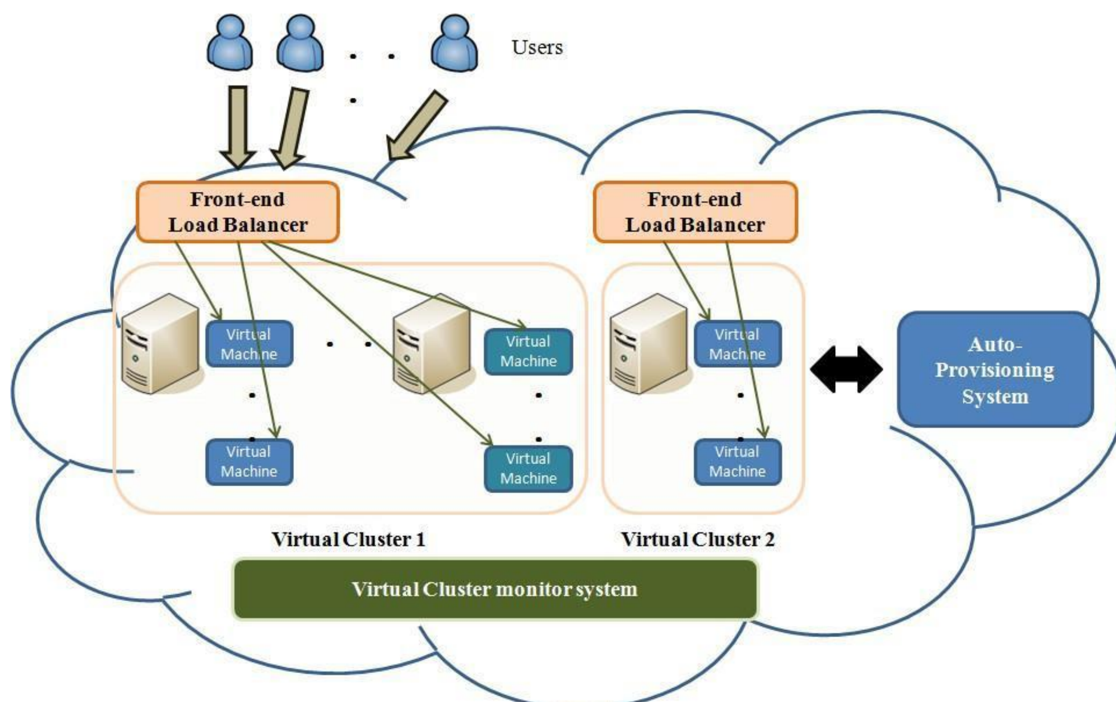



Рис. 2.9 Схема віртуального кластера

Ми розглядаємо два сценарії щодо веб-сервісу та програми паралельної обробки в середовищі хмарних обчислень. Веб-сервіс має бути доступним у будь-який час і забезпечувати найшвидший час відповіді незалежно від кількості обслуговуваних користувачів. Таким чином, система хмарного сервісу повинна динамічно масштабувати сервіс; розширення та скорочення кількості веб-серверів і компонентів веб-служб для великих і малих вимог. Хмарні обчислення також дозволяють користувачам отримати доступ до

обчислювальної потужності на рівні суперкомп'ютера, розподіляючи завдання на велику кількість обчислювальних вузлів (віртуальних машин). Користувачі можуть не знати точної кількості обчислювальних вузлів, які слід використовувати для ефективного виконання своїх завдань. Таким чином, часто трапляється недостатнє та надлишкове забезпечення. Система хмарних обчислень повинна масштабувати обчислювальні вузли відповідно до робочого навантаження. Масштабований Архітектура, яка ефективно обробляє ці два сценарії, проілюстрована на малюнку 1. Ця архітектура включає три основні компоненти: зовнішній балансувальник навантаження, систему моніторингу віртуального кластера та систему автоматичного надання з алгоритмом автоматичного масштабування. Внутрішній балансувальник навантаження може балансувати запити між різними віртуальними машинами, які виконують ту саму програму у віртуальному кластері. Система моніторингу віртуального кластера збирає дані про використання ресурсів усіх віртуальних машин для кожного віртуального кластера, які працюють у хмарі. Система автоматичного надання може горизонтально розширювати/зменшувати кількість віртуальних машин відповідно до робочого навантаження віртуального кластера, до якого належать віртуальні машини. Якщо програма, що працює у віртуальному кластері, споживає більшість ресурсів, автоматичне масштабування може створити нову віртуальну машину, яка виконує ту саму програму та зовнішній балансувальник навантаження, а потім балансує запити між віртуальними машинами в одному віртуальному кластері.

У нашій архітектурі для балансування навантаження веб-програми використовується зовнішній балансувальник навантаження. Apache HTTP Load-Balancer використовується як зовнішній балансувальник навантаження, щоб дозволити маршрутизувати вхідний HTTP-запит на веб-сервери, які виконують веб-програму. Оскільки конфігурацію Apache HTTP Load-Balancer можна оновлювати динамічно, це дозволяє хмарній системі автоматично та динамічно додавати нові веб-сервери для віртуального кластера. Додаткові веб-сервери

дозволяють масштабувати віртуальний кластер і таким чином забезпечують кращий час відповіді на вхідні запити HTTP.

```
Auto-Scaling Algorithm1
Input:  n: number of Clusters
        VC: an Virtual Cluster consists of VMs that run the same
computational system
        VM_ns: number of the active sessions in a virtual machine
        S_iMax: maximum sessions for a virtual machine of i-th Cluster
        S_upper_bound: session upper-threshold
        S_low_bound: session low-threshold
        E_below: a virtual machine set records virtual machines that
exceed the session upper-threshold
Output: Front Load-Balancer Set FLB

1.  For i = 1 to n
2.    For each VM ∈ VC_i
3.      If (VM_ns/S_iMax ≥ S_upper_bound) then
4.        e = e + 1
5.      If (VM_ns/S_iMax ≤ S_low_bound) Then
6.        b = b + 1
7.        Record VM to E_below
8.    If (e == | VC_i|) then
9.      Provision and start a new VM that runs the same system as VC_i
10.     Add new VM to FLB //Front Load-Balancer Set
11.   If (b ≥ 2) then
12.     For each VM in E_below
13.       If (VM_ns == 0) then
14.         Remove VM from FLB //Front Load-Balancer Set
15.         Destroy VM
16.   Empty E_below
```

Рис. 2.10 Алгоритм балансування навантаження

Для веб-додатків система моніторингу віртуального кластера може визначити, чи кількість активних HTTP-сеансів перевищує порогове значення у віртуальному кластері. Для завдання розподіленого обчислення система моніторингу віртуального кластера здатна визначити, чи кількість віртуальних машин перевищує поріг використання фізичних ресурсів у віртуальному кластері. Як показано на малюнку 1, алгоритм автоматичного масштабування реалізовано в системі автоматичного надання, а система моніторингу віртуального кластера використовується для керування та ініціювання збільшення та зменшення масштабу в системі автоматичного надання кількості екземплярів віртуальної машини. на основі статистики індикатора масштабування.

Встановлення програмного пристрою на віртуальну машину створює віртуальний пристрій. Образ віртуального пристрою — це набір віртуальних пристроїв на віртуальній машині. Образ віртуального пристрою включає гостьову ОС і додатки, і він може усунути витрати на встановлення, конфігурацію та обслуговування, пов’язані із запуском складних стеків програмного забезпечення. Таким чином, щоб спростити процес надання, шаблон образу пристрою віртуальної машини, який включає веб-програму та її відповідний веб-сервер, зберігається в сховищі зображень хмарної системи. Нові віртуальні машини веб-серверів і веб-додатків можна швидко створити та надати віртуальному кластеру за допомогою відповідного шаблону образу пристрою.

Для програми веб-сервісу пропускна здатність мережі та кількість сеансів є найважливішими показниками для угоди про рівень обслуговування (SLA) і якості обслуговування (QoS). Рис. 2 ілюструє наш алгоритм автоматичного масштабування для програми веб-служби. Алгоритм спочатку визначає поточні віртуальні машини з пропускною спроможністю мережі та активними сеансами вище або нижче заданого порогового значення, відповідно. Якщо пропускна здатність мережі та активні сеанси всіх віртуальних машин перевищують заданий верхній поріг, нова віртуальна машина буде підготовлена, запущена, а потім додана до зовнішнього балансувальника навантаження відповідно. Якщо є віртуальні машини з пропускною спроможністю мережі та активними сеансами, нижчими за вказане нижнє порогове значення, і принаймні з однією віртуальною машиною, яка не має мережевого трафіку чи активних сеансів, неактивну віртуальну машину буде видалено з зовнішнього балансувальника навантаження та припинено з системи. .

Для розподілених обчислювальних завдань фізичні обчислювальні ресурси, такі як використання центрального процесора та пам’яті, є найважливішими показниками розвитку робочого навантаження віртуального кластера. Рис. 3 ілюструє наш алгоритм автоматичного масштабування для

розподілених обчислювальних завдань. Алгоритм спочатку визначає поточні віртуальні машини з використанням фізичних ресурсів вище або нижче заданого порогового значення. Якщо використання ресурсів усіх віртуальних машин перевищує заданий верхній поріг, нова віртуальна машина буде створена, підготовлена, запущена, а потім виконає ті самі обчислювальні завдання у віртуальному кластері. Якщо використання ресурсів деяких віртуальних машин є нижчим за заданий нижній поріг і принаймні одна віртуальна машина не має обчислювального завдання, неактивна віртуальна машина буде припинена з віртуального кластера.

У цьому документі ми представили два сценарії масштабування для автоматичної масштабованості веб-додатків і розподілених обчислювальних завдань у віртуальному кластері у віртуалізованому середовищі хмарних обчислень. Запропонована архітектура хмарних обчислень складається з переднього балансувальника навантаження, системи моніторингу віртуального кластера та системи автоматичного забезпечення. Внутрішній балансир навантаження використовується для маршрутизації та балансування запитів користувачів до хмарних служб, розгорнутих у віртуальному кластері. Система моніторингу віртуального кластера використовується для збору даних про використання фізичних ресурсів кожної віртуальної машини у віртуальному кластері. Система автоматичного надання використовується для динамічного надання віртуальних машин на основі кількості активних сеансів або використання ресурсів у віртуальному кластері. Крім того, неактивні віртуальні машини знищуються, а потім ресурси можуть бути звільнені. З іншого боку, витрати на енергію можна зменшити, якщо видалити неактивні віртуальні машини. Наша робота показала, що запропонований алгоритм здатний справлятися з раптовими вимогами до навантаження, підтримуючи високе використання ресурсів і знижуючи витрати на енергію. Механізм автоматичного масштабування хмарної системи є важливим елементом у покращенні використання ресурсів, таким чином зменшуючи витрати на інфраструктуру та управління.

Достовірність наукових тверджень підтверджується доказами їх відтворюваності з використанням нових даних [ 1 ]. Це відрізняється від повторного тестування заяви з використанням тих самих аналізів і тих самих даних (зазвичай це називається відтворюваністю або обчислювальною відтворюваністю ) і використання тих самих даних з різними аналізами (зазвичай це називається надійністю ). Недавні спроби систематично відтворити опубліковані твердження свідчать про напрочуд низькі показники успіху. Наприклад, у 6 нещодавніх спробах тиражування 190 тверджень у соціальних і поведінкових науках 90 (47%) були успішно відтворені відповідно до основного критерію успіху кожного дослідження [ 2 ]. Подібним чином, огляд великої вибірки 18 гіпотез щодо генів-кандидатів або гіпотез про взаємодію генів-кандидатів для депресії не знайшов підтвердження жодній із них [ 3 ], що є особливо приголомшливим результатом, враховуючи, що їх вплив досліджували понад 1000 статей. Проблеми реплікації породили ініціативи щодо підвищення точності та прозорості досліджень, таких як попередня реєстрація та відкриті дані, матеріали та код [ 4–6 ] . Водночас невдачі в повторенні викликали дебати про значення реплікації та її наслідки для довіри до дослідження. Копії неминуче відрізняються від оригінальних досліджень. Як ми вирішуємо, чи є щось реплікацією? Відповідь змінює концепцію реплікації з нудної, некреативної господарської діяльності на захоплюючу, генеративну, життєво важливу роль у прогресі досліджень.

Відповідно до загальноприйнятого розуміння, реплікація – це повторення процедури дослідження та спостереження, чи повторюється попередній результат [ 7 ]. Це визначення реплікації є інтуїтивно зрозумілим, простим у застосуванні та неправильним.

Проблема полягає в тому, що це визначення наголошує на повторенні технічних методів — процедури, протоколу або маніпульованих і виміряних подій. Чому це проблема? Уявіть собі, що оригінальне поведінкове дослідження було проведено в Сполучених Штатах англійською мовою. Що робити, якщо тиражування буде виконано на Філіппінах із зразком, який розмовляє тагальською

мовою? Щоб матеріали були реплікацією, чи мають бути подані англійською мовою? Без перегляду культурного контексту? Якщо допускаються незначні зміни, то що вважається незначним, щоб кваліфікуватись як повторення процедури? У більш широкому плані неможливо відтворити землетрус, спалах наднової зірки, плейстоцен або вибори. Якщо реплікація вимагає повторення маніпуляційних або виміряних подій дослідження, тоді неможливо провести реплікації в спостережних дослідженнях або дослідженнях минулих подій.

```
Auto-Scaling Algorithm2
Input:  n: number of Clusters
        VC: an Virtual Cluster consists of VMs that run the same
computational system
        VM_R: The use of resources in a virtual machine
        R_upper_bound: The upper-threshold of use of physical resources
        R_low_bound: The low-threshold of use of physical resources
        V_below: a virtual machine set records virtual machines that

1. For i = 1 to n
2.   For each VM ∈ VC_i
3.     If (VM_R ≥ R_upper_bound) then
4.       e = e + 1
5.     If (VM_R ≤ R_low_bound) Then
6.       b = b + 1
7.       Record VM to V_below
8.   If (e == | VC_i|) then
9.     Provision and start a new VM that runs the same computing
   tasks as VC_i
10.  If (b ≥ 2) then
11.    For each VM in V_below
12.      If (VM is idle) then
13.        Destroy VM
14.  Empty V_below
```

Рис. 2.11 Алгоритм балансування 2

Повторення процедур дослідження є привабливим визначенням реплікації, оскільки воно часто відповідає тому, що роблять дослідники під час реплікації, тобто якомога точніше дотримуються оригінальних методів і процедур. Але причина цього полягає не в тому, що повторення процедур визначає реплікацію. Реплікації часто повторюють процедури, тому що теорії занадто розпливчасті, а методи занадто погано зрозумілі, щоб продуктивно проводити реплікації та просувати теоретичне розуміння інакше [ 8 ].

Попередні коментатори проводили відмінності між типами реплікації, такими як «пряма» проти «концептуальної» реплікації, і стверджували на користь

оцінки одного над іншим (наприклад, [ 9 , 10 ]). Навпаки, ми стверджуємо, що відмінності між «прямим» і «концептуальним» є щонайменше нерелевантними і, можливо, контрпродуктивними для розуміння реплікації та її ролі в просуванні знань. Процедурні визначення реплікації є масками для недостатньо розроблених теоретичних очікувань, а «концептуальні реплікації», як вони ідентифікуються на практиці, часто не відповідають критеріям, які ми тут розробляємо та вважаємо важливими для того, щоб тест кваліфікувався як реплікація.

пропонуємо альтернативне визначення реплікації, яке більш охоплює всі дослідження та більше відповідає ролі реплікації в просуванні знань. Реплікація – це дослідження, будь-який результат якого вважатиметься діагностичним доказом твердження з попереднього дослідження. Це визначення зменшує акцент на оперативних характеристиках дослідження та посилює акцент на інтерпретації можливих результатів.

Щоб бути реплікацією, мають бути правдивими 2 речі: результати, що відповідають попереднім заявам, підвищать довіру до претензій, а результати, несумісні з попередніми претензіями, зменшать довіру до претензій. Симетрія сприяє реплікації як механізму протиставлення попередніх заяв новим доказам. Тому заява про те, що дослідження є повторенням, є теоретичним зобов’язанням. Реплікація дає можливість перевірити, чи здатні існуючі теорії, гіпотези чи моделі передбачити результати, які ще не спостерігалися. Успішні копії підвищують довіру до цих моделей; невдалі повторення знижують впевненість і стимулюють теоретичні інновації для вдосконалення або відмови від моделі. Це не означає, що масштаб зміни переконань є симетричним для «успіхів» і «невдач». Попередні та існуючі докази інформують про те, якою мірою результати відтворення змінюють переконання. Проте, як теоретичне зобов’язання, реплікація передбачає попереднє зобов’язання серйозно ставитися до всіх результатів.

Оскільки повторення визначається на основі теоретичних очікувань, не всі погодяться, що одне дослідження є повторенням іншого. Крім того, не завжди можливо зробити попередні зобов’язання щодо діагностики дослідження як

реплікації, часто з тієї простої причини, що результати дослідження вже відомі. Рішення про те, чи є дослідження реплікацією після спостереження за результатами, може використати упередження пост-гок міркувань, щоб відкинути «невдачі» як неповторення, а «успіхи» як діагностичні тести тверджень, або навпаки, якщо спостерігач бажає дискредитувати твердження. Це може непродуктивно сповільнювати прогрес дослідження, відкидаючи контрдокази реплікації. Одночасно реплікації можуть не досягти запланованих діагностичних цілей через помилку або несправність у процедурі, яку можна ідентифікувати лише постфактум. Коли існує невизначеність щодо статусу претензій і якості методів, немає простого рішення розрізнити вмотивовані та принципові міркування щодо доказів. Найефективніше рішення науки — повторити.

У найкращому вигляді наука мінімізує вплив ідеологічних зобов'язань і упереджень, будучи відкритим соціальним підприємством. Щоб досягти цього, дослідники повинні отримувати винагороду за чітке та апріорне формулювання своїх теорій, щоб вони могли продуктивно зіткнутися з доказами [ 4 , 6 ]. Кращими є ті теорії, які чітко пояснюють, як вони можуть бути підтверджені та оскаржені реплікацією. Повторне копіювання часто є необхідним для вирішення впевненості в заяві, і, незмінно, дослідникам буде багато про що сперечатися, навіть якщо реплікація та попереднє зобов'язання є нормативними практиками.

Метою реплікації є просування теорії шляхом зіткнення існуючого розуміння з новими доказами. За іронією долі, цінність реплікації може бути найсильнішою, коли наявне розуміння найслабше. Теорія розвивається поступово, починаючи з концептуальних стрибків, несподіваних спостережень і безлічі доказів. Це нормально; це нечітко на кордонах знань. Діалог між теорією та доказами полегшує ідентифікацію контурів, обмежень та очікувань щодо досліджуваних явищ. Відтворювані докази є якорями для цього ітераційного процесу. Якщо доказ можна відтворити, то теорія повинна врешті-решт пояснити його, навіть якщо лише відкинути його як нерелевантний через недійсність методів. Наприклад, твердження про те, що в заможних країнах більше людей із ожирінням у

середньому порівняно з біднішими країнами, і про те, що люди в заможніших країнах у середньому живуть довше, ніж люди з бідніших країн, можуть бути добре відтворені. Усі теоретичні точки зору щодо зв'язку між багатством, ожирінням і довголіттям повинні були б враховувати ці повторювані твердження.

Точної копії не існує. Ми не можемо відтворити землетрус, епоху чи вибори, але тиражування не означає повторення історичних подій. Реплікація стосується визначення умов, достатніх для оцінки попередніх претензій. Відтворення може відбуватися під час спостережних досліджень, коли умови, які вважаються необхідними для спостереження за доказами, повторюються, наприклад, коли нова сейсмічна подія має характеристики, які вважаються необхідними та достатніми для спостереження результату, передбаченого попередньою теорією, або коли новий метод для переоцінки скам'янілостей пропонує незалежний тест існуючих тверджень щодо цієї скам'янілості. Навіть в експериментальних дослідженнях оригінальні та реплікаційні дослідження неминуче відрізняються в деяких аспектах вибірки або одиниць, з яких збираються дані, лікування, яке вводиться, результати, які вимірюються, і умови, в яких проводяться дослідження [ 11 ].

Індивідуальні дослідження не надають вичерпних або остаточних доказів щодо всіх умов для спостереження доказів щодо претензій. Прогалини заповнені теорією. В одному дослідженні вивчається лише підмножина одиниць, методів лікування, результатів і налаштувань. Дослідження проводилося в певному кліматі, в певний час доби, в певний момент історії, з використанням конкретного методу вимірювання, з використанням конкретних оцінок, з конкретним зразком. Рідко дослідники обмежують свої висновки саме цими умовами. Якби це було так, наукові твердження були б історичними твердженнями, оскільки ці точні умови ніколи не повторяться. Якщо вважається, що твердження розкриває закономірність світу, то воно неминуче узагальнює ситуації, які ще не спостерігалися. Фундаментальне питання: з незліченних варіацій одиниць, методів лікування, результатів і налаштувань, які з них мають значення? Можна очікувати, що час

доби для збору даних буде нерелевантним для твердження про особистість і батьківство або критичним для твердження про циркадні ритми та гальмування.

Коли теорії занадто незрілі, щоб робити чіткі прогнози, повторення початкових процедур стає дуже корисним. Використання одних і тих самих процедур є проміжним рішенням у разі відсутності чіткої теоретичної специфікації того, що необхідно для отримання доказів щодо позову. Крім того, використання тих самих процедур зменшує невизначеність щодо того, що кваліфікується як доказ, який «відповідає» попереднім заявам. Реплікація не стосується процедур як таких, але використання подібних процедур зменшує невизначеність у всесвіті можливих одиниць, методів лікування, результатів і параметрів, які можуть бути важливими для заяви.

Оскільки немає точного повторення, кожен тест повторення оцінює можливість узагальнення для унікальних умов нового дослідження. Однак кожен тест на можливість узагальнення не є повторенням.рис 1Ліва панель ілюструє відкриття та умови навколо нього, на які воно потенційно може бути узагальнено. Простір узагальнення великий через теоретичну незрілість; існує багато умов, за яких претензія може бути підтверджена, але помилки не дискредитують оригінальну претензію.рис 1Права панель ілюструє дорослішання розуміння претензії. Простір узагальнення скоротився, оскільки деякі тести ідентифікували граничні умови (сірі тести), а простір відтворюваності збільшився, оскільки успішні реплікації та узагальнення (кольорові тести) покращили теоретичну специфікацію щодо того, коли очікується відтворюваність.

Успішне повторення надає докази можливості узагальнення в умовах, які неминуче відрізняються від оригінального дослідження; невдала реплікація вказує на те, що надійність знахідки може бути більш обмеженою, ніж вважалося раніше. Багаторазове тестування відтворюваності та можливості узагальнення для підрозділів, методів лікування, результатів і налаштувань сприяє покращенню теоретичної специфічності та прогнозу на майбутнє.Теоретичне дозрівання проілюстровано врис 2. Прогресивна дослідницька програма (лівий шлях) успішно

відтворює висновки в умовах, які вважаються нерелевантними, а також розвиває теоретичне уявлення, щоб більш чітко розрізняти умови, для яких явище буде спостерігатися або не спостерігатися. Це проілюстровано скороченням простору узагальнення, в якому теорія не робить чітких передбачень. Дегенеративна дослідницька програма (правильний шлях) постійно не в змозі відтворити результати та поступово звужує всесвіт умов, до яких може бути застосовано твердження. Це проілюстровано скороченням узагальнюваності та простору відтворюваності, оскільки теорія повинна бути обмежена умовами, що постійно звужуються [ 12 ].

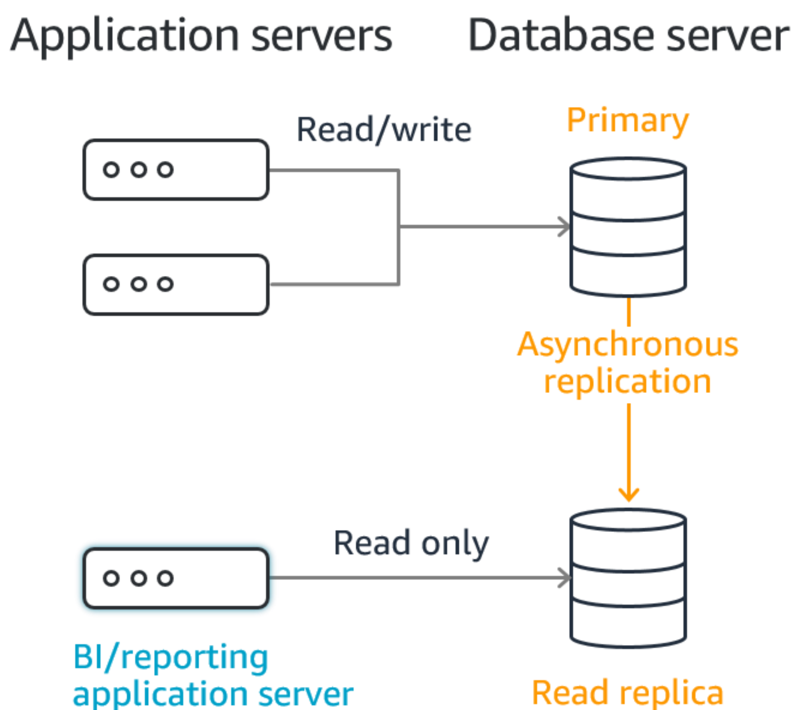



Рис. 2.12 Узагальнений алгоритм читання репліки

Неоднозначність між спростуванням оригінального твердження чи визначенням граничної умови також означає, що розуміння того, чи є дослідження копією, може змінитися через накопичення знань. Наприклад, відомий експеримент Отто Леві (Нобелівська премія з фізіології та медицини 1936 р.) показав, що гальмівний фактор «вагустов», який згодом був визначений як ацетилхолін, вивільнявся з блукаючого нерва жаб, що свідчить про те, що передача нейронів є хімічним процесом. Набагато пізніше, після того, як йому та іншим не

вдалося відтворити його оригінальне твердження, важливе теоретичне розуміння виявило, що пора року, в яку Лоуї проводив свій експеримент, була критичною для його успіху [ 13 ]. Початкове дослідження проводилося з так званими зимовими жабами. Спроби реплікації, виконані з літніми жабами, провалилися через сезонну чутливість серця жаби до нерозпізнаного ацетилхоліну, що ускладнює демонстрацію ефектів вагусної стимуляції. Завдяки подальшим тестам, які надали підтверджуючі докази, розуміння заяви покращилося. Те, що сприймалося як реплікація, більше не було, тому що нові докази показали, що вони не вивчали те саме. Теоретичне розуміння розвинулося, і подальші копії підтвердили переглянуті твердження. Це не проблема, це прогрес.

Термін «концептуальна реплікація» застосовувався до досліджень, які використовують різні методи для перевірки того самого питання, що й у попередньому дослідженні. Це корисна дослідницька діяльність для покращення розуміння, але багато досліджень із такою міткою не є копіями за нашим визначенням. Пам’ятайте, що «щоб бути реплікацією, мають бути правдивими 2 речі: результати, які відповідають попередній заяві, підвищать довіру до заяви, а результати, несумісні з попередньою заявою, зменшать довіру до заяви». Багато «концептуальних копій» відповідають першій Критерій і не відповідають другому. Тобто, вони не розроблені таким чином, що невдача повторення призвела б до перегляду впевненості в оригінальному твердженні. Натомість «концептуальні повторення» часто є тестами на узагальнення. Невдачі трактуються, щонайбільше, як визначення граничних умов. Відповіддю є самооцінка того, чи перевіряється відтворюваність чи можливість узагальнення: результат, який не відповідає попереднім висновкам, змусить мене втратити довіру до теоретичних тверджень? Якщо ні, то це тест на узагальненість.

VNF можна розгортати на основі віртуалізації гіпервізора або віртуалізації контейнера. Порівняно з віртуальними машинами (ВМ) контейнери легкі, споживають менше ЦП/пам’яті/енергоресурсів і мають значно менший час запуску. посилання [4] порівнює віртуалізацію на основі гіпервізора (Xen, KVM тощо) і

віртуалізацію на основі контейнерів (Docker, LXC тощо) з точки зору енергоспоживання. Контейнери потребують набагато менше енергії, і це значно впливає на експлуатаційні витрати. Вибір технології віртуалізації також впливає на QoS. посилання [5] обговорює час запуску різних технологій віртуалізації та показує, як після рішення про масштабування створення нової віртуальної машини/контейнера займає більше/коротше часу залежно від технології віртуалізації.

Попередні дослідження автоматичного масштабування можна розділити на дві групи: на основі порогових значень і на основі прогнозів (аналіз часових рядів, ML тощо). Підходи на основі порогових значень використовувалися власниками DC [3] для масштабування обчислювальних ресурсів. Підходи на основі статичних порогів [6] [7] [8] використовують попередньо визначені верхній і нижній пороги для масштабування, що не є практичним рішенням у сценарії динамічного попиту. Покращення були запропоновані з використанням динамічних порогових підходів [9] [10] [11]. Що стосується підходів на основі прогнозування, попередні дослідження використовували авторегресію (AR) [12], ковзне середнє (MA) [13] і авторегресійне ковзне середнє (ARMA) [14] для прогнозування майбутнього робочого навантаження для автоматичного масштабування .

Недавні дослідження також досліджували масштабування VNF у телекомунікаційних хмарних мережах. посилання [15] пропонує алгоритм автоматичного масштабування для мереж мобільного зв’язку 5G, обмежений термінами та бюджетом. Тут VNF динамічно масштабуються на основі кількості поточних завдань у системі за допомогою евристичного алгоритму. Будучи реактивним за своєю природою, цей метод дасть неоптимальні результати в мережі з коливанням трафіку. посилання [16] пропонує ресурсоефективний підхід до автоматичного масштабування телекомунікаційної хмари з використанням підкріпленого навчання та підкреслює, що автоматичне масштабування VNF має вирішальне значення як для гарантії QoS, так і для управління витратами. Навчання з підкріпленням виконується без будь-яких знань про попередні моделі трафіку чи

рішення про масштабування. Таким чином, підходи автоматичного масштабування в [15] [16] не користуються історичними даними трафіку та рішеннями щодо масштабування.

посилання [17] охоплює автоматичне масштабування з точки зору DevOps. Це дослідження розглядає масштабування ресурсів площини даних як функцію навантаження трафіку, але також у цьому випадку використовується реактивний підхід, який не використовує історичні дані. Крім того, порівняно з [15]–[17], наше дослідження досліджує переваги автоматичного масштабування також з точки зору орендодавця.

Незважаючи на те, що автоматичне масштабування є надзвичайно цінною функцією для постачальників програмного забезпечення, визначення політики автоматичного масштабування, яка може гарантувати відсутність порушень продуктивності, є надзвичайно складним завданням і «приреченим на провал» [2], якщо не вжити значних заходів. Крім того, для того, щоб політика автоматичного масштабування на основі правил була правильно налаштована, потрібен глибокий рівень знань і високий рівень досвіду, що не обов’язково відповідає дійсності на практиці [3, 1].

Останнім часом постачальники публічних хмар, як-от Amazon EC2 і Microsoft Azure, збільшили гнучкість, пропоновану користувачам під час визначення політик автоматичного масштабування, дозволивши комбінації правил автоматичного масштабування для широкого діапазону показників. Однак ця «свобода» можливості вказувати кілька правил автоматичного масштабування коштує надзвичайно великого простору конфігурації. Насправді він є експоненціальним за кількістю показників ефективності та предикатів, що робить практично неможливим знайти оптимальні значення для змінних автоматичного масштабування [4].

Крім того, політика автоматичного масштабування складається не лише з порогових значень показників продуктивності, а й із тимчасових параметрів, якими часто нехтують, незважаючи на їхню важливість у налаштуванні правильної

політики автоматичного масштабування. Ці параметри включають часовий інтервал, протягом якого механізм автоматичного масштабування перевіряє, щоб визначити, чи виконувати дію автоматичного масштабування, і тривалість, протягом якої йому заборонено ініціювати дії автоматичного масштабування після успішного запиту на автоматичне масштабування (круто). - період падіння). Оскільки обидва ці параметри має вказувати людина-оператор, стає складним завданням зрозуміти вплив цих параметрів на показники продуктивності програми, що працює в хмарі. Саме цей вплив ми хочемо проаналізувати кількісно.

Як зазначено в [5], політикам автоматичного масштабування «як правило, бракує гарантій правильності». Можливість визначати політики автоматичного масштабування, які можуть забезпечити гарантії продуктивності та зменшити кількість порушень угод про рівень обслуговування (SLA), є важливою для більш надійних і підзвітних хмарних операцій. Однак це складне завдання через: (i) великий простір конфігурації умов і параметрів, які необхідно визначити; (ii) непередбачуваність хмари як робочого середовища через її спільну, еластичну природу та природу на вимогу; і (iii) неоднорідність надання хмарних ресурсів, що ускладнює визначення надійної та універсальної політики автоматичного масштабування. Наприклад, дивлячись на постачальників загальнодоступних хмар, можна помітити, що немає гарантій щодо часу, який знадобиться для обслуговування запиту на автоматичне масштабування, а також того, чи отримає запит на автоматичне масштабування успішну відповідь чи ні. Нашою ймовірнісною моделлю є ланцюг Маркова з дискретним часом (DTMC), який ми вказуємо мовою моделювання інструменту PRISM. Він враховує стохастичність запитів автоматичного масштабування шляхом переходу між різними часами очікування з імовірністю p, яка може бути задана користувачем апріорі або залишена як вільний параметр для імовірнісної перевірки політики автоматичного масштабування за різні значення p. Ми демонструємо правильність і корисність цього підходу через широку перевірку, розглядаючи сценарії «Інфраструктура як послуга» (IaaS) і «Платформа як послуга» (PaaS), що працюють у хмарі Amazon

EC2 і Microsoft Azure відповідно. Ми оприлюднили дані, використані для перевірки нашої моделі [8].

Щоб підтвердити точність нашої моделі, ми виконуємо аналіз робочих характеристик приймача (ROC), який широко використовується в машинному навчанні та інтелектуальному аналізі даних [9]. У певному сенсі ми дотримуємося подібного підходу до перевірки моделей класифікації, розглядаючи ймовірність як міру ранжирування, яка визначає ймовірність події, що цікавить, у нашому випадку ймовірність порушення QoS. ROC можна використовувати, щоб допомогти особі, яка приймає рішення, вибрати відповідний поріг класифікації шляхом кількісного визначення компромісу між чутливістю та специфічністю. Крім того, його можна використовувати для перевірки точності моделі класифікації шляхом обчислення показника AUC (площа під кривою), який є одним із найбільш часто використовуваних підсумкових індексів [10].

Наша перевірка починається з упорядкування ймовірностей, які були обчислені з нашої моделі PRISM для кожної політики автоматичного масштабування. Потім ми будуємо відповідну криву ROC, обчислюючи точки для відповідних порогів [0..1]. Завдяки цьому аналізу ми можемо знайти оптимальний поріг розрізнення між політиками автоматичного масштабування, які можуть призвести до порушення ЦП/часу відповіді. Наш критерій оптимальності – це точка, яка мінімізує евклідову відстань між кривою ROC, і точка (0,1), яку часто називають точкою «ідеальної класифікації». Крім того, це дає нам можливість уточнити наші початкові оцінки порушень після того, як ми побачимо реальні дані, і отримати глобальне порогове значення для розрізнення між політиками автоматичного масштабування. Після побудови кривої ROC ми обчислюємо AUC, який у нашому випадку можна інтерпретувати як кількість разів, коли наша модель може розрізнити порушення продуктивності/непорушення випадково вибраних політик автоматичного масштабування. Це «важлива статистична властивість» [9] AUC і одна з причин того, що вона так широко використовується для перевірки продуктивності класифікаторів.

Політика автоматичного масштабування [4] визначає умови, за яких ємність буде додана або видалена з хмарної системи, щоб задовольнити цілі власника програми. Автоматичне масштабування поділяється на методи масштабування вгору/зниження та масштабування/збільшення, причому ці два підходи також визначаються як вертикальний (додати більше оперативної пам'яті або ЦП до існуючих віртуальних машин) і горизонтальний (додати більше «дешевих» віртуальних машин). масштабування. У цій роботі ми зосереджуємось на масштабуванні та входженні, оскільки це широко використовуваний і економічно ефективний підхід.

Основний метод автоматичного масштабування, який надають постачальникам додатків усіма сьогодні постачальниками загальнодоступних хмар (наприклад, Amazon, Microsoft Azure, Google Cloud), заснований на правилах. Метод на основі правил є найпопулярнішим і вважається найсучаснішим у автоматичному масштабуванні програми в хмарі [13].

У підході, заснованому на правилах, постачальник програми має вказати верхню та/або нижню межу показника продуктивності (наприклад, використання ЦП) разом із бажаною зміною потужності для цієї ситуації. Наприклад, метод на основі правил, який ініціюватиме рішення про масштабування, коли використання ЦП перевищує 60%, може мати вигляд: якщо використання ЦП > 60%, додайте 1 екземпляр [14]. Показники продуктивності, якими зазвичай керуються постачальники публічних хмар, включають використання ЦП, пропускну здатність і довжину черги. Ми розглядаємо рішення щодо автоматичного масштабування на основі використання ЦП, оскільки це один із найважливіших показників у плануванні потужності, а також найбільш широко використовуваний у політиках автоматичного масштабування.

PRISM [7] — це засіб перевірки ймовірнісної моделі, який підтримує побудову та формальний кількісний аналіз різних імовірнісних моделей, включаючи ланцюги Маркова з дискретним часом (DTMC), ланцюги Маркова з неперервним часом і процеси прийняття рішень Маркова, які виражаються на мові

моделювання PRISM. Модель у PRISM розбивається на модулі, які представляють різні компоненти системи/процесу, що моделюється. Стан моделі містить значення для набору змінних, які є локальними для деякого модуля або глобальними для всієї моделі. Тут ми використовуємо DTMC, які добре підходять для моделювання систем, стани яких розвиваються імовірнісним шляхом, але без будь-якого недетермінізму чи зовнішнього контролю. Тому вони підходять тут, де ми хочемо перевірити політики автоматичного масштабування, результати яких є імовірнісними.

Крім того, PRISM підтримує числові властивості, такі як P=?[F fail], що означає «яка ймовірність того, що хмарна програма опиниться в стані збою в результаті прийнятих рішень щодо автоматичного масштабування?». Звичайно, те, що вважається невдалим станом, відрізнятиметься для власників хмарних програм залежно від відносної важливості, яку вони надають нефункціональним аспектам своєї програми. PRISM дозволяє визначити та проаналізувати широкий спектр таких властивостей.

Хоча для Amazon EC2 і Microsoft Azure було розроблено дві різні моделі, основні принципи, що лежать в їх основі, схожі. Наприклад, обидві моделі очікують на вході політику автоматичного масштабування, створення їхніх станів покладається на кластеризацію, а «філософія» того, як відбуваються переходи між станами, певною мірою однакова. Крім того, обидві моделі мають однакові цілі високого рівня, оскільки вони призначені для допомоги в процесі прийняття рішень автоматичного масштабування. У наступному параграфі ми описуємо важливі складові наших моделей, висвітлюючи їхні відмінності та пояснюючи причини цих відмінностей.

Розробка реплікації за іншою методологією вимагає розуміння теорії та методів, щоб будь-який результат вважався діагностичним доказом попередньої заяви. На практиці це означає, що тиражування часто обмежується відносно чітким дотриманням оригінальних методів для тем, у яких теорія та методологія незрілі — обставина, яку зазвичай називають «прямим» або «тісним» реплікацією, —

оскільки подібність методів служить підставою для для теоретичної точності та вимірювання. Фактично, повторення попередньої претензії з іншою методологією можна вважати важливою віхою для теоретичної та методологічної зрілості.

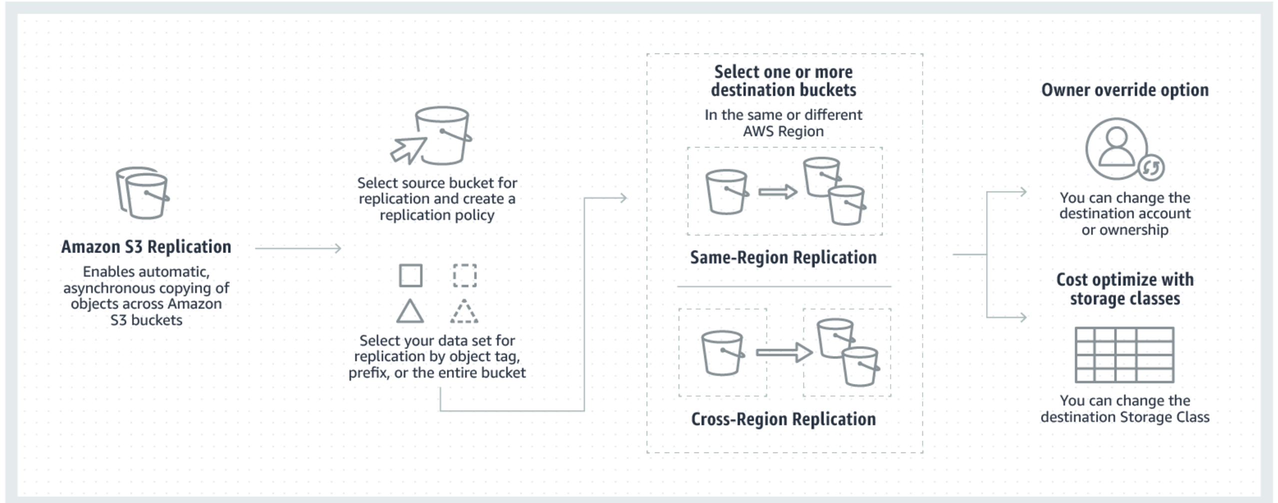



Рис. 2.13 Узагальнений алгоритм проведення моделювання

# 3 ПРОВЕДЕННЯ МОДЕЛЮВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Для отримання даних для аналізу виконано ряд POST запитів кількістю в 2000 на сервіс з метою перевірки продуктивності сервера під навантаженням

Отримаємо дані щодо залежності часу, необхідного опрацювання надісланих запитів.

На рисунку 3.1 наведено діаграму, де показано кількість надісланих HTTP запитів на сервер.

На рисунку 3.2 наведено графіки, час відгуку надісланих запитів без використання балансера навантаження.

На рисунку 3.3 Час відгуку надісланих запитів з використання балансера навантаження.

Аналізуючи отримані результати, можна зробити висновок:

Без балансера — при загальній кількості запитів в 2000 штук, успішно опрацьованих 1000 штук, що складає 50%, з відгуком на запит до 1 секунд.

З використанням балансера навантаження - при загальній кількості запитів в 2000 штук, успішно опрацьованих 1800 штук, що складає 90%, з відгуком на запит до 60 мілісекунд.

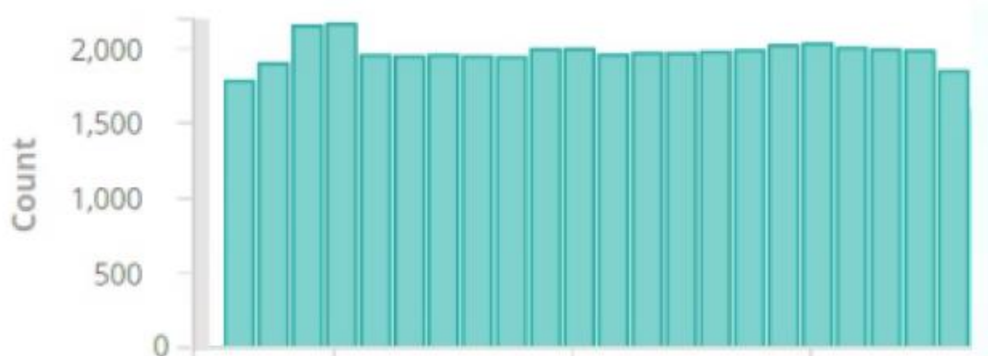



Рис. 3.1 Кількість надісланих HTTP запитів

Рис. 3.2 Час відгуку надісланих запитів без використання балансера

Рис. 3.3 Час відгуку надісланих запитів з використання балансера навантаження

# ВИСНОВКИ

В результаті виконання магістерської роботи було розроблено модель поведінки людей у приміщенні з використанням клітинних автоматів та розроблено програмне забезпечення для реалізації процесу моделювання та збору статистичних даних.

При виконанні роботи було виконано наступні задачі.

2. Проведено огляд підходів до тестування продуктивності.

3. Визначено критерії оцінювання продуктивності: масштабованість, пропускна здатність, використання ресурсів, коефіцієнт завантаження, швидкодія, коефіцієнт помилок, час відгуку.

4. Проведено аналіз недоліків продуктивності при навантаженні: при навантаженість на сервіс зростає, що призводить до збільшення часу відгуку.

5. Розроблено діаграму аналізу HTTP запитів.

6. Розроблено метод збільшення кількості опрацьованих HTTP запитів за рахунок використання балансера навантаження через визначення часу відгуку та коефіцієнту помилок.

7. Проведено порівняльну характеристику використання ресурсів сервера при навантажені:

8. Без балансера — при загальній кількості запитів в 2000 штук, успішно опрацьованих 1000 штук, що складає 50%, з відгуком на запит до 1 секунд

9. З використанням балансера навантаження - при загальній кількості запитів в 2000 штук, успішно опрацьованих 1800 штук, що складає 90%, з відгуком на запит до 60 мілісекунд.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. V. Chandel et al., "Comparative Study ofTesting Tools: Apache JMeter and LoadRunner," International Journal ofComputing and Corporate Research,2013.
2. G. Murawski, et al., "Evaluation of loadtesting tools," 2014.
3. Smith, J., "Modern Approaches to Performance Testing in Agile Environments," 2018.
4. Sharma, R., "Performance Testing Best Practices in Cloud Environments," 2016.
5. Patel, S., "Advanced Performance Testing Techniques for Mobile Applications," 2019.
6. Nguyen, T., "Machine Learning Applications in Performance Testing," 2017.
7. Wilson, A., "Scalability Testing Methods for High-Traffic Web Applications," 2018.
8. Garcia, M., "Performance Testing in DevOps: Challenges and Solutions," 2019.
9. Brown, K., "Big Data and Performance Testing: Strategies for Success," 2017.
10. Gonzalez, L., "Continuous Performance Testing in CI/CD Pipelines," 2018.
11. Kumar, P., "Security Aspects in Performance Testing: A Comprehensive Study," 2016.
12. Lee, H., "Performance Testing Metrics and KPIs: An Analytical Approach," 2019.
13. Robinson, D., "Performance Testing in Microservices Architecture," 2017.
14. White, S., "Performance Testing for IoT Devices: Challenges and Solutions," 2018.
15. Miller, E., "Application Performance Testing in Hybrid Cloud Environments," 2016.
16. Thompson, N., "Performance Testing Automation: Tools and Techniques," 2019.
17. Carter, M., "API Performance Testing: Methods and Strategies," 2017.
18. Adams, R., "Performance Testing in AI-Powered Applications," 2018.
19. Hall, G., "Performance Testing of Blockchain Systems," 2019.
20. Barnes, H., "Performance Testing in Containerized Environments," 2017.
21. Murphy, F., "Real User Monitoring in Performance Testing: Implementation and Benefits," 2016.

# ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

## (Презентація)

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

**МАГІСТЕРСЬКА РОБОТА**
**РОЗРОБКА МЕТОДИКИ ТЕСТУВАННЯ ПРОДУКТИВНОСТІ ДЛЯ АНАЛІЗУ СТАНУ СЕРВЕРА ПІД НАВАНТАЖЕННЯМ**

Виконав: Студент групи ПДМ-61 Собко Іван Іванович
Керівник: Трінтіна Наталія Альбертівна, доцент кафедри, кандидат технічних наук

Київ - 2024

### АКТУАЛЬНІСТЬ РОБОТИ

| | |
|---|---|
| **Цифрова трансформація бізнесу** | У зв'язку з швидко змінюючимся цифровим середовищем, організації постійно шукають способи оптимізувати свої процеси та використання технологій. |
| **Управління проектами** | У процесі реалізації проектів важливо визначити, які процеси та методи працюють найбільш ефективно. |
| **Оптимізація робочих процесів** | У всіх сферах діяльності люди шукають способи покращення робочих процесів. |
| **Технологічний розвиток та інновації** | Швидкий розвиток технологій створює нові можливості для вдосконалення продуктивності. |

## МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

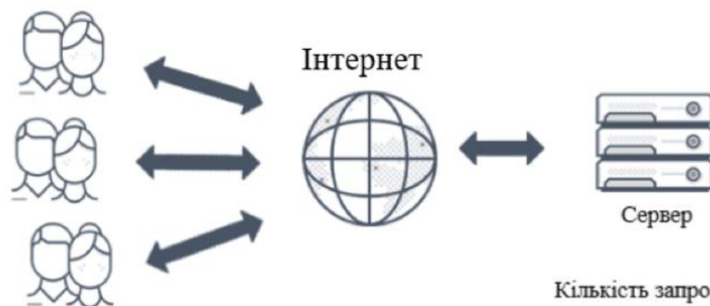
**Мета дослідження:** збільшення кількості опрацьованих HTTP запитів за рахунок використання балансера навантаження

**Об’єкт дослідження:** тестування продуктивності сервера під навантаженням

**Предмет дослідження:** методи тестування продуктивності



## ПРИКЛАД МОДЕЛІ РОБОТИ HTTP ЗАПИТІВ

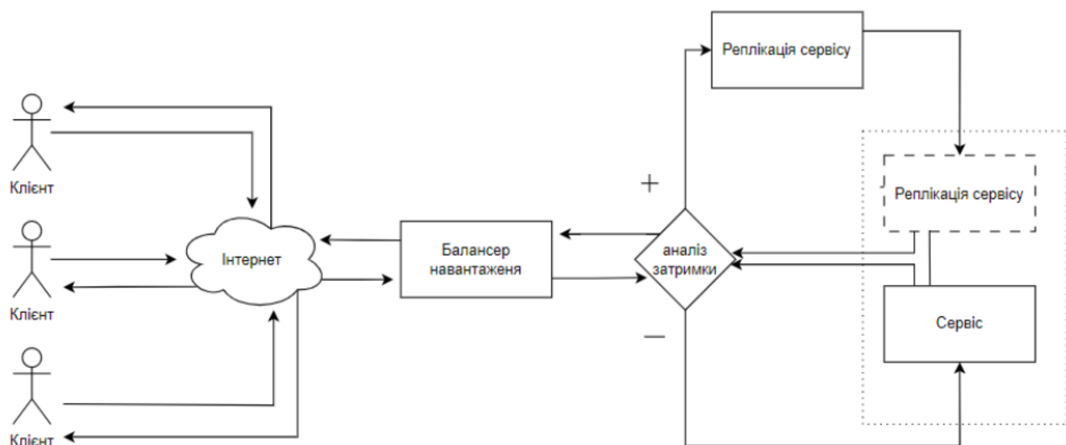


## КРИТЕРІЇ ОЦІНЮВАННЯ ПРОДУКТИВНОСТІ

| Метрика | Опис |
|---|---|
| Масштабованість | Вміння системи адекватно обробляти зростаюче навантаження, зберігаючи при цьому продуктивність. |
| Пропускна здатність | Кількість оброблених запитів чи транзакцій за певний час. |
| Використання ресурсів | Спостереження за використанням ресурсів. |
| Коефіцієнт завантаження | Час, який потрібний для завантаження системи або окремих її частин. |
| Швидкодія | Затримка між часом відправки запиту та часом отримання відповіді. |
| Коефіцієнт помилок | Відсоток помилок чи невдач у виконанні запитів. |
| Час відгуку | Час, який потрібно системі для обробки запиту користувача. |



## МЕТОДИКА АНАЛІЗУ ШЛЯХУ HTTP ЗАПИТІВ



## МАТЕМАТИЧНИЙ АНАЛІЗ ЗАВАНТАЖЕННЯ

Час відгуку: $t_{total} = t_1 + t_2 + t_3 + t_4$

$t_1$ - час затримки на стороні клієнта
$t_2$ - час передачі
$t_3$ - час обробки на сервері
$t_4$ - час передачі відповіді

Функція розрахунку середнього значення:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

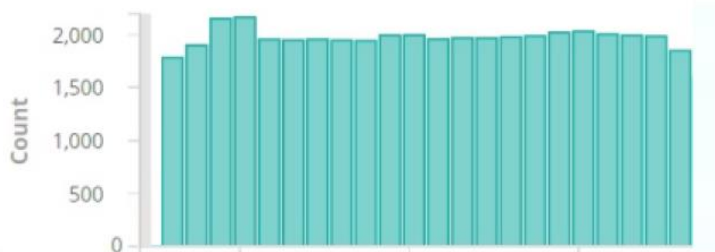
Коефіцієнт помилок: $k_{error} = \frac{N_{error}}{N_{total}} \times 100\%$

$N_{error}$ = кількість помилкових запитів
$N_{total}$ = загальна кількість запитів

За період 5хв $\begin{cases} \bar{t}_{total} > 1\text{сек} \\ k_{error} > 20\% \end{cases}$



## ПОРІВНЯННЯ ХАРАКТЕРИСТИК ПРОДУКТИВНОСТІ



# РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРОДУКТИВНОСТІ

| Показники | Без балансера навантаження | З балансером навантаження |
|---|---|---|
| Кількість запитів | 2000 | 2000 |
| Успішно оброблених запитів | 1000 | 1800 |
| Час відгуку | 1с | 60мс |
| Успішно оброблених запитів у відсотках | 50% | 90% |



# ВИСНОВКИ

1. Проведено огляд підходів до тестування продуктивності.
2. Визначено критерії оцінювання продуктивності: масштабованість, пропускна здатність, використання ресурсів, коефіцієнт завантаження, швидкодія, коефіцієнт помилок, час відгуку.
3. Проведено аналіз недоліків продуктивності при навантаженні: при навантаженість на сервіс зростає, що призводить до збільшення часу відгуку.
4. Розроблено діаграму аналізу HTTP запитів.
5. Розроблено метод збільшення кількості опрацьованих HTTP запитів за рахунок використання балансера навантаження через визначення часу відгуку та коефіцієнту помилок.
6. Проведено порівняльну характеристику використання ресурсів сервера при навантажені:

Без балансера — при загальній кількості запитів в 2000 штук, успішно опрацьованих 1000 штук, що складає 50%, з відгуком на запит до 1 секунд

З використанням балансера навантаження - при загальній кількості запитів в 2000 штук, успішно опрацьованих 1800 штук, що складає 90%, з відгуком на запит до 60 мілісекунд