Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Аналіз методів моніторингу та вимірювання продуктивності високонавантажених back-end систем
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-4

Залазаєв А.М
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Голян Віра Володимирівна
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Залазаєву Андрію Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Аналіз методів моніторингу та вимірювання продуктивності високонавантажених back-end систем»

Затверджена наказом по університету від 29.03. 2024р. № 250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19.06.2024

3. Вихідні дані до роботи Інформація щодо сучасних back-end систем, A/B тестування, методу лінійної згортки

4. Перелік питань, що потрібно опрацювати в роботі

мета роботи, аналіз предметної галузі і постановка задачі, огляд та аналіз літературних джерел з дослідження, дослідження теоретичне).

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	*виконано*
2	Аналіз та вибір API для дослідження	15.02 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 26.04.24	*виконано*
10	Підготовка презентації та доповіді	26.04 – 2.05.24	*виконано*
11	Нормоконтроль	3.06 – 08.06.24	*виконано*
12	Рецензування	08.06 – 14.06.24	*виконано*
13	Занесення диплома в електронний архів	15.06.2023	*виконано*
14	Попередній захист	15.06.2023	*виконано*
15	Допуск до захисту у зав. кафедри	17.05.2023	*виконано*

Дата видачі завдання 30 березня 2024р.

Студент (ка) _____ Залазаєв А.М
(підпис)

Керівник роботи _____ доц. Голян Віра Володимирівна
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 43с., 11 рис., 5 табл., 7 джерел.

ВИМІРЮВАННЯ, ДАНІ, МЕТОДИ МОНІТОРИНГУ, ОБЧИСЛЕННЯ, ПРОДУКТИВНІСТЬ, СИСТЕМИ, ШКАЛИ ЕФЕКТИВНОСТІ.

Об’єкт дослідження – високонавантажені back-end системи, що використовуються в інформаційних технологіях.

Мета роботи – аналіз та порівняння різних методів моніторингу та вимірювання продуктивності back-end систем для підвищення ефективності їх функціонування.

Результат роботи – розроблені рекомендації щодо вибору та впровадження оптимальних методів моніторингу та вимірювання продуктивності високонавантажених back-end систем.

Для досягнення поставленої мети використані такі методи дослідження: аналіз літератури, експериментальні вимірювання продуктивності, математичне моделювання.

Здійснено порівняльний аналіз методів, таких як моніторинг ресурсів, аналіз логів, тестування навантаження, та їх вплив на результативність back-end систем. Результати дослідження вказують на ефективність певних методів у різних умовах експлуатації.

Отримані дані можуть бути корисні для інженерів, адміністраторів систем, розробників програмного забезпечення, які працюють з високонавантаженими back-end системами, що вимагають постійного моніторингу та вдосконалення продуктивності.

MEASUREMENT, DATA, MONITORING METHODS, CALCULATION, PRODUCTIVITY, SYSTEMS, PERFORMANCE SCALES.

The object of the research is high-load back-end systems used in information technologies.

The aim of the work is to analyze and compare various methods of monitoring and measuring the productivity of back-end systems to enhance their operational efficiency.

The result of the work is the development of recommendations for the selection and implementation of optimal methods for monitoring and measuring the productivity of high-load back-end systems.

To achieve the set goal, the following research methods were used: literature analysis, experimental productivity measurements, mathematical modeling.

A comparative analysis of methods such as resource monitoring, log analysis, load testing, and their impact on the performance of back-end systems has been conducted. The research results indicate the effectiveness of certain methods in different operating conditions.

The obtained data can be valuable for engineers, system administrators, and software developers working with high-load back-end systems that require constant monitoring and performance improvement.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Залазаєв Андрій Миколайович, студент(ка) гр. ІПЗм-22-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Аналіз методів моніторингу та вимірювання продуктивності високонавантажених back-end систем», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Сучасний рівень наукового прогресу не тільки не залишається на місці, але й активно розвиває технології, випереджаючи їхній розвиток та впровадження новацій. Недавно для отримання необхідної інформації доводилося витрачати значний час на пошук відповідей на поставлені питання. В сучасному світі майже кожна компанія спрямована на вдосконалення процесу пошуку шляхом інтеграції систем, розроблених за аналогією з людським мозком, у свої продукти. Ці системи, відомі також як нейронні мережі, мають здатність працювати з різними типами даних, такими як зображення, звук, відео, текст та інше. Кожна з них відрізняється своєю архітектурою та типом пам’яті.

Об'єкт дослідження даного проекту – високонавантажені back-end системи, які широко використовуються в інформаційних технологіях. Мета дослідження – провести аналіз та порівняти різні методи моніторингу та вимірювання продуктивності цих back-end систем з метою підвищення їх ефективності.

Отримані дані можуть бути корисні для фахівців у галузі інженерії, адміністрування систем та розробників програмного забезпечення, які працюють з високонавантаженими back-end системами та потребують постійного моніторингу та вдосконалення їх продуктивності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз сфери back-end моніторингу

За допомогою back-end моніторингу можна відстежувати продуктивність інфраструктури веб-додатку. Сюди входять HTTP-сервер, проміжне програмне забезпечення, база даних, сторонні API-сервіси та багато інших. Кожен компонент може знаходитися в одному дата-центрі або в різних дата-центрах, залежно від характеру послуги, що надається.

Моніторинг back-end дозволяє швидко виявити та виправити будь-які проблеми, які можуть виникнути. Різні компоненти веб-додатку можуть стикатися з різними проблемами, серед них:

- труднощі з пам'яттю;
- помилки в коді;
- проблеми з програмним забезпеченням на системному рівні (проблеми з ОС, порушення безпеки);
- проблеми з комп'ютером (нестача пам'яті, місця на диску, збій диска, збій мережевої інтерфейсної карти (NIC)).

Такі проблеми, пов'язані з Інтернетом, є поширеними, але вони зазвичай мають більший вплив, ніж потрібно, через те, що залишаються непоміченими. Якщо ви використовуєте back-end моніторинг, ви будете негайно поінформовані про проблеми. Завдяки цьому їм не доведеться виявляти, що їхній веб-сайт зламався, через кілька годин після того, як вони відвідають його - або, що ще гірше, коли це виявлять їхні клієнти. Коли перебої виявляються раніше, їх можна вирішити швидше.

Внутрішній моніторинг та угоди про рівень обслуговування

Власники сайтів можуть використовувати back-end моніторинг для встановлення та моніторингу угод про рівень обслуговування (SLA) зі своїми постачальниками інфраструктури, на додаток до переваг, описаних вище.

Моніторинг продуктивності вашого онлайн-додатку з точки зору користувача, включаючи весь сторонній контент, забезпечується за допомогою

front-end моніторингу. "Користувацький досвід" є синонімом фронтенд-моніторингу. Альтернативно, відомий як моніторинг "кінцевого користувача", він надає моментальний знімок того, що бачать відвідувачі вашого веб-сайту, коли вони його запитують. Фронтенд може кардинально відрізнятися залежно від географічного розташування користувача, його браузера та інших факторів. Ці розбіжності можна виявити завдяки фронтенд-моніторингу.

З іншого боку, фронтенд-моніторинг виявляє проблеми з продуктивністю, які не обов'язково пов'язані з виходом з ладу компонентів. Серед них:

- проблеми з контентом (наприклад, нове зображення на сайті занадто важке, що знижує продуктивність; скрипт, що обслуговується таким сайтом, як Facebook, не працює, тим самим сповільнюючи роботу вашого сайту);
- проблеми з сумісністю браузерів (наприклад, користувачі Firefox 13 завантажують сторінки набагато повільніше, ніж користувачі інших браузерів);
- проблеми, пов'язані з вашим місцезнаходженням. Як наслідок, користувачі в Азії бачать значно повільнішу роботу сайту, ніж користувачі в інших регіонах;
- проблеми з мережею на мобільних пристроях з'єднання є основною перешкодою.

1.2 Аналіз сучасних підходів до моніторингу back-end

Існують усталені рекомендації з інженерії надійності сайтів (Site Reliability Engineering, SRE). Найкраща практика - це стежити за чотирма золотими сигналами:

- затримка: час, необхідний для обслуговування запиту;
- трафік: кількість запитів за секунду/хвилину;
- помилки: кількість невдалих запитів;
- насиченість: навантаження на вашу мережу та сервери;

– визначаючи пріоритетність цих чотирьох сигналів, ми зосереджуємося на проблемах кінцевих користувачів. Ці показники показують вам, що щось не так: ваш сервіс відповідає повільно, з помилками або ви раптом отримуєте менше запитів від користувачів, ніж зазвичай (що вказує на проблему з додатком, який вам телефонує).

Для виміру існує два основних підходи: моніторинг у режимі реального часу та періодичний моніторинг.

Моніторинг у режимі реального часу передбачає постійне спостереження за системою та отримання сповіщень, як тільки виникає проблема. З іншого боку, періодичний моніторинг передбачає перевірку системи через регулярні проміжки часу, наприклад, щогодини або щодня. Обидва підходи мають свої переваги та недоліки, і вибір правильного залежить від ваших конкретних потреб і вимог.

1.3 Моніторинг у режимі реального часу

Моніторинг у режимі реального часу надає найактуальнішу інформацію про продуктивність вашої системи. За допомогою моніторингу в режимі реального часу ви можете швидко виявити проблеми і вжити заходів до того, як вони стануть критичними. Наприклад, якщо на вашому веб-сайті спостерігається раптовий сплеск трафіку, моніторинг у режимі реального часу допоможе вам швидко виявити причину і вжити заходів для запобігання простою. Моніторинг у режимі реального часу також корисний для виявлення аномалій, які можуть свідчити про загрози безпеці.

Переваги моніторингу в режимі реального часу:

– негайне сповіщення: моніторинг в реальному часі надає негайні сповіщення при виникненні проблеми, що дозволяє швидко реагувати і запобігати простоям;
– точні дані: моніторинг в режимі реального часу надає точні дані про продуктивність вашої системи, що дозволяє приймати обґрунтовані рішення;

– раннє виявлення: моніторинг в режимі реального часу може виявити проблеми на ранній стадії, до того, як вони стануть критичними.

Недоліки моніторингу в режимі реального часу:

– висока вартість: моніторинг у режимі реального часу може бути дорогим, оскільки вимагає спеціалізованого програмного та апаратного забезпечення;
– підвищене робоче навантаження: моніторинг у режимі реального часу вимагає постійної уваги, що може збільшити робоче навантаження та стрес.

1.4 Періодичний моніторинг

Періодичний моніторинг передбачає перевірку системи через регулярні проміжки часу, наприклад, щогодини або щодня. Цей підхід менш ресурсоємний, ніж моніторинг у режимі реального часу, оскільки вимагає менше апаратного та програмного забезпечення. Однак періодичний моніторинг може не надавати найактуальнішої інформації про продуктивність вашої системи.

Переваги періодичного моніторингу:

– нижча вартість: Періодичний моніторинг дешевший, ніж моніторинг у реальному часі, оскільки вимагає менше апаратного та програмного забезпечення;
– зменшення робочого навантаження: Періодичний моніторинг вимагає менше уваги, ніж моніторинг у режимі реального часу, що зменшує робоче навантаження та стрес;
– заплановані перевірки: Періодичний моніторинг можна запланувати на певний час, що полегшує управління ним.

Недоліки періодичного моніторингу:

– повільний час реагування: Періодичний моніторинг може не виявляти проблеми так швидко, як моніторинг в режимі реального часу, що може призвести до простоїв;

– неточні дані: Періодичний моніторинг може не надавати найактуальніші дані про продуктивність вашої системи, що може призвести до неточних рішень;
– пропущені аномалії: Періодичний моніторинг може пропустити аномалії, які виникають між перевірками, що може свідчити про загрози безпеці.

1.5 Вибір правильного підходу

Вибір правильного підходу залежить від конкретних потреб і вимог. Моніторинг у режимі реального часу ідеально підходить для систем, які потребують негайної уваги, таких як веб-сайти з високим трафіком або критична інфраструктура.

Періодичний моніторинг підходить для систем, які можуть витримати певний час простою, наприклад, внутрішніх мереж або некритичних додатків[1].

Також можна використовувати комбінацію моніторингу в режимі реального часу та періодичного моніторингу, залежно від вимог системи. Приклад такої системи наведено на рисунку 1.1.

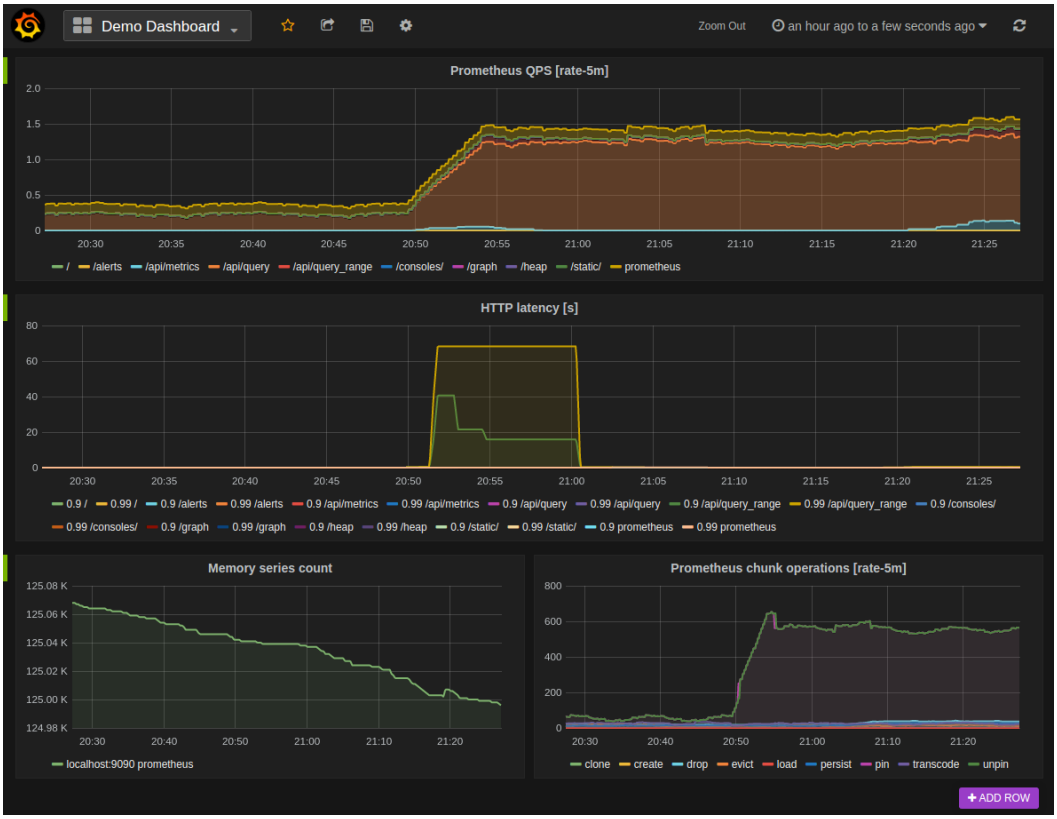

Рисунок 1.1 – інтерфейс Protheus

Наприклад, ви можете використовувати моніторинг у режимі реального часу для критично важливих компонентів і періодичний моніторинг для некритичних компонентів.

Вибір правильного підходу залежить від конкретних потреб і вимог. Моніторинг у режимі реального часу надає найактуальнішу інформацію про продуктивність вашої системи, але він може бути дорогим і вимагає постійної уваги. Періодичний моніторинг дешевший, але може не надавати найактуальніші дані.

Поєднання обох підходів може бути найкращим варіантом для систем, які потребують балансу між негайною увагою та економічною ефективністю

1.6 Аналіз та інтерпретація даних про продуктивність

Аналіз та інтерпретація даних про продуктивність є важливою частиною будь-якого back-end-плану. Він допомагає виявити потенційні проблеми та сфери для вдосконалення, що в кінцевому підсумку може призвести до підвищення продуктивності та покращення користувацького досвіду. Однак не завжди легко зрозуміти, з чого почати або як зрозуміти сенс даних.

показники:

Перш ніж почати аналізувати дані про ефективність, необхідно знати, що саме шукати.

Визначити показники, які є найбільш важливими для додатку або веб-сайту.

До найпоширеніших показників належать час завантаження сторінки, час відповіді сервера та частота помилок.

Необхідно чітко розумієти, що вимірює кожен показник і як він пов'язаний із загальною продуктивністю.

В цьому можуть допомгти інструменти візуалізації: дані про продуктивність можуть бути приголомшливими, з великими наборами даних. Інструменти візуалізації, такі як графіки та діаграми, можуть допомогти зрозуміти сенс даних і виявити тенденції або закономірності.

Наприклад, лінійний графік може показати, як час завантаження сторінки змінюється з часом, а діаграма розсіювання допоможе виявити кореляцію між різними показниками.

Порівняння різних періодів часу допомогає не тільки аналізувати дані про продуктивність у теперішньому часі, але й порівнювати їх з історичними даними. Це допоможе виявити тенденції та закономірності з плином часу і побачити, чи покращується продуктивність, чи погіршується[2].

Аналіз викидів надасть розуміння проблем у продуктивності. Викиди - це точки даних, які значно відрізняються від решти даних. Це можуть бути індикатори потенційних проблем або областей для покращення.

Наприклад, якщо час відгуку сервера зазвичай становить близько 100 мс, але час від часу збільшується до 500 мс, можливо, варто з'ясувати, що спричиняє такі стрибки.

Під час аналізу необхідно враховувати контекст бо дані про продуктивність не існують у вакуумі.

Важливо враховувати контекст, в якому були зібрані дані. Наприклад, якщо час завантаження сторінки на мобільних пристроях повільніший, ніж на десктопі, це може бути пов'язано з різницею у швидкості мережі або можливостями пристрою.

Розуміння контексту може допомогти приймати більш обґрунтовані рішення про те, як підвищити продуктивність.

Використання A/B-тестування: A/B-тестування передбачає порівняння двох версій веб-сайту або додатку, щоб побачити, яка з них працює краще.

Метод передбачає порівняння двох або більше варіантів системи або її компонентів, щоб визначити, який із них має кращий вплив на продуктивність.

Основні етапи A/B тестування:

– визначте цілі та показники: заздалегідь визначте цілі тесту та показники, які ви плануєте вимірювати. Це може бути скорочення часу відповіді, покращення часу завантаження сторінки або інші ключові показники ефективності;

– розробка варіантів: створюйте різні версії серверного компонента або параметра для порівняння. Наприклад, ви можете протестувати різні алгоритми обробки запитів або налаштування сервера;
– розподіл трафіку: розподіл трафіку між початковим (A) і новим (B) параметрами. Щоб отримати достовірні результати, користувачі повинні бути випадковим чином призначені для одного з варіантів.

Швидкий збір даних A/B-тестування може допомогти швидко зібрати реальні дані про продуктивність. Задля візуалізації результатів можна використовувати ПЗ WMPU, приклад наведено на рисунку 1.2.

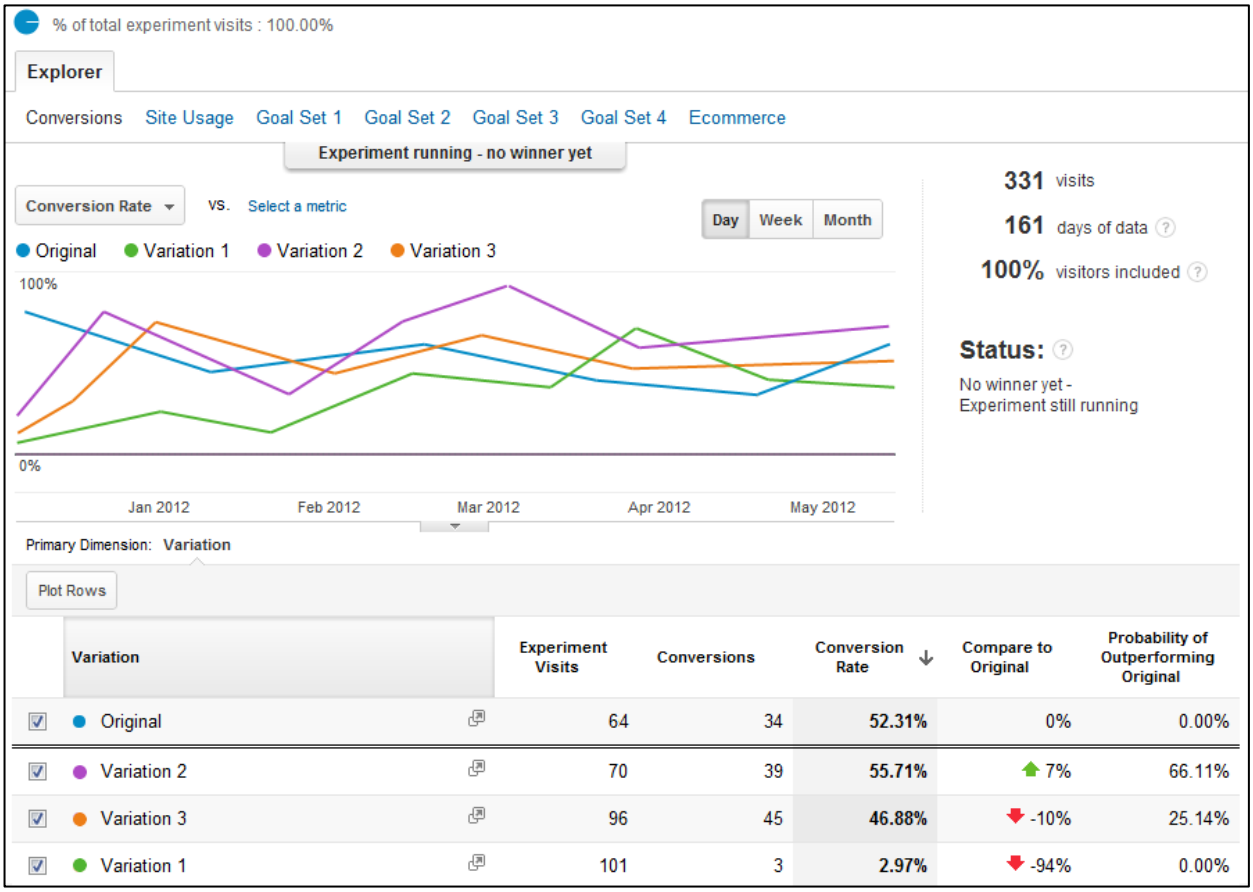

Рисунок 1.2 – WMPU A/B тестування

Тестування можна використовувати для оцінки різних системних параметрів, таких як налаштування сервера, алгоритми оптимізації, обробка даних тощо.

Тестування A/B є потужним інструментом для оцінки та порівняння різних аспектів продуктивності серверної системи.

Використання інформативних метрик є важливою частиною вимірів. Непередбачувані умови у програмі викликатимуть найбільш критичні сповіщення. Однак, деякі сигнали будуть засновані на метриках.

Наприклад, щоб дізнатися, що back-end має стабільну продуктивність і користувачі мають хороший досвід збереження елементів у додатку.

Перша метрика, яка спадає на думку, – це середній час відгуку кінцевої точки збереження. Ймовірно, щоб він був, менше 200 мс щохвилини, так середній час очікування завантаження 150 мс[3]. Це оповіщення досить просто налаштувати в більшості систем моніторингу.

Якщо оповіщення ніколи не спрацьовувало, то, все має бути гаразд. Але, якщо протягом якоїсь випадкової хвилини кінцева точка обробила 100 запитів. Дев'яносто з них зайняли 50 мілісекунд, але в якийсь момент черга накопичилася, і п'ять запитів зайняли 500 мс, а останні п'ять – десь між секундою і двома секундами. Середня затримка все ще була б у діапазоні 120-170 мс, але 10% запитів були поза цим діапазоном, а 5% користувачів мали не найкращий досвід.

Метрики на основі процентилів були б набагато кращим вибором для цього оповіщення. Треба мати на увазі, що показники 1% і 5% часто будуть набагато вищими за середні.

Рекомендується використовувати для цього оповіщення правило 5% затримки > 200 мс. Створення іншого тригера з затримкою 1% > 1 секунди дозволить відстежувати граничні випадки, коли back-end стає дуже перевантаженим, але при цьому уникнути надмірної кількості сповіщень.

Використання показників на основі процентилів, зокрема 95-го та 99-го процентилів, може виявити, як деякі екстремальні значення впливають на загальну продуктивність. Наприклад, якщо 95% користувачів мають час відповіді менше 200 мс, але 99% користувачів мають час відповіді менше 500 мс, це може вказувати на одиничний випадок зниження продуктивності системи.

Застосування правила процентилів, щоб краще контролювати крайові випадки та уникати надмірного сповіщення.

Наприклад, налаштування сповіщення про затримку 5% понад 200 мілісекунд і тригер затримки 1% понад 1 секунду, щоб ви могли виявити критичні умови, коли серверна система перевантажена, не перевантажуючи команди сповіщень.

Загальна мета використання процентилів полягає в тому, щоб продуктивність системи була стабільною та придатною не лише для середнього користувача, але й для тих, хто може зіткнутися з екстремальними умовами використання.

1.7 Визначення ключових показників ефективності (KPI)

Перш ніж почати аналізувати дані про продуктивність внутрішньої системи, важливо визначити та використовувати ключові показники ефективності (KPI).

KPI – це конкретні показники, які дозволяють виміряти успішність вашої системи та визначити, чи відповідає вона вашим цілям і вимогам.

Найважливіші KPI для серверних систем включають:

- час відповіді сервера: час, потрібний серверу для обробки запиту. Короткий час відповіді свідчить про ефективність сервера;
- час завантаження сторінки: час, необхідний для завантаження сторінки або функції. Швидке завантаження сприяє позитивному досвіду користувача;
- частота помилок: кількість помилок, що виникають під час роботи системи. Низький рівень помилок свідчить про стабільність системи.
- пропускна здатність: кількість даних, яку система може обробити за одиницю часу. Висока пропускна здатність важлива для обробки великої кількості запитів;
- використання ресурсів: ефективність використання ресурсів, таких як процесор і пам’ять. Ефективне використання ресурсів забезпечує стабільність і продуктивність системи;
- скасовані або невдалі запити: кількість скасованих або невдалих запитів. Це може свідчити про проблему зі зв’язком або інший технічний аспект.

Використання KPI допомагає зосередитися на конкретних аспектах продуктивності та своєчасно реагувати на проблеми, допомагаючи забезпечити оптимальну роботу серверної системи.

1.8 Порівняння між різними періодами

Порівняння різних періодів часу є важливим кроком у аналізі продуктивності серверної системи. Цей підхід дозволяє отримати уявлення про те, як продуктивність змінюється з часом, і визначити тенденції, які мають вирішальне значення для прийняття управлінських рішень і покращення продуктивності системи.

Аналіз змін – переглядаючи дані за різні періоди часу, ви можете визначити зміни продуктивності та визначити, чи пов’язані вони з конкретними подіями чи змінами в системі. Наприклад, ви можете відчути підвищення або зниження продуктивності після впровадження оновлення або модифікації.

Застосування трендового методу – використання діаграм і методів аналізу тенденцій дозволяє визначити поточні зміни в продуктивності. Наприклад, лінійна діаграма може показати, як змінюється час завантаження сторінки або час відповіді сервера протягом тривалого періоду часу.

Аналіз тенденцій – порівнюючи дані за різні періоди, ви можете виявити тенденції та побачити, чи є систематичні зміни. Наприклад, підвищення продуктивності може бути виявлено в періоди низького трафіку, і навпаки, зниження продуктивності під час пікових навантажень.

Оцінка реакцій на зміни – шляхом порівняння даних до і після змін у серверній системі можна оцінити ефективність запроваджених інновацій та їх вплив на продуктивність. Це дає змогу зробити об’єктивні висновки щодо ефективності внесених змін.

Врахування сезонності – аналізуючи дані за різні періоди, важливо враховувати сезонні коливання, які можуть вплинути на продуктивність. Наприклад, збільшення трафіку під час свят може вплинути на час відповіді сервера.

Порівняння різних періодів часу може допомогти відстежувати розвиток внутрішніх систем і вживати заходів для забезпечення стабільної та оптимальної продуктивності протягом життєвого циклу системи.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Аналіз методів роботи з метриками

Створення програм для тестування навантаження – це складний процес, оскільки він пов'язаний з багатьма аспектами, такими як масштабованість, точність вимірювань, можливість налаштування, підтримка протоколів тощо.

Під час дослідження ми порівняємо різні ПЗ для тестування навантаження back-end систем.

Для цього скористаємося лінійною адаптивною згорткою з ваговими коефіцієнтами.

У якості критеріїв для порівняння, було обрано наступні параметри:

- масштабованість – здатність ПЗ швидко адаптуватися до навантажень у середовищі. Де 1 – ПЗ придатний тільки для навчальних цілей, а 10 – ідеальна масштабованість на будь-які навантаження;
- точність вимірювань – параметр, що оцінює якість наданих програмою звітів та аналітики. Де 1 – дані мають похибку в вимірах і часі та не можуть бути використані у реальних проектах, а 10 – дані в реальному часі з ідеальною точністю;
- налаштування – параметр, що описує гнучкість налаштування системи. Де 1 – систему треба дороблювати під вимоги і не можна змінювати конфігурацію під час роботи, а 10 – гнучка конфігурація, у декількох форматах з можливістю зміни під час роботи;
- протоколи – параметр, що описує зручність спілкування з системою. Де 1 – система використовує застарілі способи спілкування і не надає альтернатив, а 10 – підтримка HTTP, WebDAV, SOAP, XMPP і тд. Одночасно;
- зручність використання – параметр, що відповідає за UX. Де 1 – наявність консольного інтерфейсу без віддаленого управління, а 10 – сучасний інтерфейс доступний у мережі та на декількох пристроях.

Вибір буде проводитись серед наступних ПЗ.

Apache JMeter.

Масштабованість: Підтримує велику кількість одночасних користувачів.

Точність вимірювань: Забезпечує докладні звіти та графіки результатів тестування.

Налаштування: Має розширювану архітектуру, дозволяючи користувачам додавати власні компоненти[4].

Протоколи: Підтримує багато різних протоколів, таких як HTTP, JDBC, SOAP, REST та інші.

Зручність використання: Інтерфейс користувача досить зручний та інтуїтивно зрозумілий.

Gatling.

Масштабованість: Добре підходить для тестування великої кількості користувачів.

Точність вимірювань: Надає докладні статистичні дані та графіки про продуктивність.

Налаштування: Можливість програмування тестів на Scala, що дозволяє вам створювати складні сценарії.

Протоколи: Підтримує HTTP та WebSocket, існує підтримка для інших протоколів за допомогою додаткових плагінів.

Зручність використання: Спеціально розроблений для високої продуктивності та швидкості виконання.

Vegeta.

Масштабованість: Ефективно взаємодіє з великою кількістю запитів за допомогою паралельного виконання.

Точність вимірювань: Надає звіт з ключовими метриками, такими як середній час відповіді, розмах і т. д.

Налаштування: Простий у використанні конфігураційний файл для визначення навантаження.

Протоколи: Головним чином орієнтований на HTTP.

Зручність використання: Легкий та зручний у використанні для базових тестів навантаження.

Locust.

Масштабованість: Добре пристосований для тестування великої кількості користувачів.

Точність вимірювань: Надає детальні статистичні дані, такі як середній час відповіді, кількість запитів на секунду і інше.

Налаштування: Простий у використанні Python-код для визначення сценаріїв та налаштування.

Протоколи: Зазвичай використовується для тестування HTTP-протоколу.

Зручність використання: Інтуїтивний веб-інтерфейс для створення та запуску тестів.

Tsung.

Масштабованість: Дуже масштабований та може витримати велику кількість одночасних користувачів.

Точність вимірювань: Надає розширені звіти та графіки про продуктивність.

Налаштування: Використовує XML для опису сценаріїв, можливість налаштовувати різні параметри тестів.

Протоколи: Підтримує різні протоколи, включаючи HTTP, WebDAV, SOAP, XMPP, та інші.

Зручність використання: Має консольний інтерфейс та можливість віддаленого управління.

Тепер знаючи елементи та критерії вибору можемо перейти до процесу описання даних елементів за обраними критеріями.

Для цього найкраще підійде векторний опис у вигляді таблиці, що надасть розуміння загального вигляду.

Дану таблицю значень критеріїв для кожного з елементів вибору можна побачити у таблиці 2.1 нижче.

Таблиця 2.1 – Опис альтернатив за обраними критеріями

Критерії / Інструменти	Apache JMeter	Gatling	Vegeta	Locust	Tsung
Масштабованість	8	9	7	8	10
Точність вимірювань	9	8	7	8	9
Налаштування	8	9	6	7	8
Підтримка протоколів	9	8	7	8	9
Зручність використання	8	7	8	8	7

Використаємо принцип Парето, щоб завчасно виключити деякі із версій перед порівнянням.

Оскільки Vegeta та Locust має гірші характеристики у порівнянні з Apache Jmeter, за принципом Парето їх можна виключити.

Внесемо відповідні зміни у таблицю 2.2.

Таблиця 2.2 – Опис альтернатив, після застосування принципу Парето

Критерії / Інструменти	Apache JMeter	Gatling	Tsung
Масштабованість	8	9	10
Точність вимірювань	9	8	9
Налаштування	8	9	8
Підтримка протоколів	9	8	9
Зручність використання	8	7	7

Перейдемо до нормування критеріїв.

Визначимо найкраще та найгірше значення кожного з критеріїв:

– масштабованість:

максимум – 10, мінімум – 8;

– точність вимірювань:

максимум – 9, мінімум – 8;

– налаштування:

максимум – 9, мінімум – 8;

– підтримка протоколів:

максимум – 9, мінімум – 8;

– зручність використання:

максимум – 8, мінімум – 7.

Проведемо розрахунки для нормування з урахуванням мінімуму та максимуму та запишемо їх у таблицю 2.3.

Таблиця 2.3 – Нормування з урахуванням мінімуму та максимуму

Критерії / Інструменти	Apache JMeter	Gatling	Tsung
Масштабованість	0	0,5	1
Точність вимірювань	1	0	1
Налаштування	0	1	0
Підтримка протоколів	1	0	1
Зручність використання	1	0	0

У записаній задачі вибору критерії мають різний пріоритет, та важливість.

Тому для вирішення задачі вибору, використаємо лінійну адаптивну згортку з ваговими коефіцієнтами, щоб мати можливість скорегувати пріоритет деяких критеріїв над іншими.

Визначимо коефіцієнти для загорткової моделі.

Критерії точності вимірювань та масштабованість є однаково критичними для дослідження.

Критерії налаштування та підтримки протоколів менш важливі, але можуть вплинути на результати.

Останнім критерієм є зручність використання. Коефіцієнт буде дорівнювати:

Внесемо коефіцієнти до таблиці 2.4.

Таблиця 2.4 – Таблиця коефіцієнтів

Критерії / Інструменти	Коефіцієнт	Apache JMeter	Gatling	Tsung
Масштабованість	3/10	0	0,5	1
Точність вимірювань	3/10	1	0	1
Налаштування	1/6	0	1	0
Підтримка протоколів	1/6	1	0	1
Зручність використання	1/15	1	0	0

На основі визначених коефіцієнтів проведемо розрахунки корисності за допомогою нормованих значень.

Для цього використаємо формулу лінійної адаптивної згортки з ваговими коефіцієнтами.

Внесемо розрахунки до таблиці 2.5.

Таблиця 2.5 – таблиця розрахунків

Критерії / Інструменти	Коефіцієнт	Apache JMeter	Gatling	Tsung
Масштабованість	3/10	0	0,5	1
Точність вимірювань	3/10	1	0	1
Налаштування	1/6	0	1	0
Підтримка протоколів	1/6	1	0	1
Зручність використання	1/15	1	0	0
Корисність		0,53	0,32	0,77

За результатами вирішення задачі вибору, для проведення дослідження, краще обрати Tsung, так як він має кращу маштабованість, точність вимірювань та підтримку протоколів.

2.2 Методи вирішення проблеми

Останнім часом високонавантажені back-end системи стали ключовим компонентом для забезпечення швидкодії та надійності різноманітних веб-сервісів та додатків. З цієї причини аналіз методів вимірювання їхньої продуктивності стає важливим завданням, спрямованим на забезпечення оптимальної ефективності та найвищого рівня обслуговування користувачів.

Високонавантажені back-end системи піддаються постійному навантаженню, що робить вимірювання їх продуктивності складною задачею.

Таким чином, важливо розглянути ефективні методи та метрики для аналізу їхньої роботи.

Зважаючи на важливість аналізу продуктивності високонавантажених back-end систем та потребу у розрахункових формулах для оцінки їхньої ефективності, було визначено необхідні метрики.

Час відповіді (Response Time).

Час відповіді можна обчислити, вимірявши різницю між часом надходження запиту і часом отримання відповіді (формула 1).

$$Rt = Et - St, \quad (1)$$

де Rt – час запиту;

Et – час закінчення обробки запиту;

St – дата початку запиту.

Пропускна здатність (Throughput).

Пропускна здатність може бути обчислена як кількість оброблених запитів протягом певного часу (формула 2).

$$Th = \frac{Tr}{Tt}, \quad (2)$$

де Th – пропускна здатність;

Tr – загальна кількість запитів;

Tt – загальний час.

Коефіцієнт використання ресурсів (Resource Utilization).

Для кожного ресурсу можна обчислити коефіцієнт використання, поділивши час, протягом якого цей ресурс був використаний, на загальний час спостереження (формула 3).

$$Ru=\frac{Rtu}{Tt} * 100\%, \quad (3)$$

де Ru – коефіцієнт використання ресурсів;

Rtu – час використання ресурсу;

Tt – загальний час.

Для порівняння метрик можна скористатися поняттям "агрегованої метрики" або "критерію прийняття рішення"[5], яка об'єднує різні метрики в одне значення, що відображає загальну ефективність системи. Один із широко використовуваних методів для цього - метод агрегації метрик за допомогою вагових коефіцієнтів.

Ми маємо три метрики: час відповіді (Rt), пропускна здатність (Th) і використання ресурсів (Ru). Нехай w1, w2 і w3 - це вагові коефіцієнти, що відображають важливість кожної метрики. Сума цих вагових коефіцієнтів повинна дорівнювати 1 (w1 + w2 + w3 = 1).

Агрегована метрика оцінки продуктивності може бути обчислена за формулою 4:

$$OP = w1 \times R + w2 \times T + w3 \times U \quad (4)$$

Коефіцієнти було обрано відповідно до рекомендацій [6] Rt – 40%, Th – 30%, Ru – 30%.

За допомогою вагових коефіцієнтів можна обчислити значення ОП для кожної системи і порівняти їх. Система, яка матиме більше значення ОП, вважатиметься кращою з точки зору загальної ефективності.

Наведений підхід було реалізовано в серверному застосунку для порівняння методів мониторингу back-end систем.

Отже, цей підхід дозволяє зробити однозначний висновок про те, яка з двох систем краща, на основі комплексного аналізу різних метрик з урахуванням їх важливості.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Методи вирішення проблеми

У цьому розділі буде розглянуто налаштування середовища для роботи з PostgreSQL та створення трьох програмних рішень на базі Java. Ці рішення включають використання чистого JDBC, використання Hibernate для ORM (Object-Relational Mapping) та використання Spring Data JPA для спрощеного доступу до даних та Python скрипти для виклику необхідних методів.

Для проведення дослідження необхідно створити окремий проекти для перевірки кожного з обраних підходів. Розробка буде здійснюватися з використанням середовища розробки IntelliJ IDEA та PostgreSQL.

Програма повинна мати однаковий інтерфейс для проведення експериментів, тому повинна мати два контролери для роботи з кожною з колекцій: **BlobController** та **ItemsController** .Ці контролери реалізують RESTful API та дозволяють проводити тестування POJO(ItemsDto.java) об’єктів та файлів(BlobDto.java), код моделей наведено на рисунку 3.1. та 3.2.

```
public static class BlobDTO {  no usages
    private int id;  3 usages
    private byte[] data;  3 usages

    public BlobDTO(int id, byte[] data) {  no usages
        this.id = id;
        this.data = data;
    }

    public int getId() {  no usages
        return id;
    }

    public void setId(int id) {  no usages
        this.id = id;
    }

    public byte[] getData() {  no usages
        return data;
    }

    public void setData(byte[] data) {  no usages
        this.data = data;
    }
}
```

Рисунок 3.1 – Клас BlobDto.java в IntegrationResearch.proj (створено самостійно)

RESTful API – це архітектурний стиль інтерфейсу прикладної програми (API), який використовує запити HTTP для доступу та використання даних.

```java
public class ItemDTO {  no usages
    private int id;  3 usages
    private String name;  1 usage
    private double price;  1 usage

    public ItemDTO() {  no usages
    }

    public ItemDTO(int id, String name, double price) {  no usages
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public int getId() {  no usages
        return id;
    }

    public void setId(int id) {  no usages
        this.id = id;
    }
}
```

Рисунок 3.2 – Клас ItemDto.java в IntegrationResearch.proj (створено самостійно)

Проаналізуємо клас Main.java, що налаштовує і запускає веб-додаток (див. рис. 3.3). Він буде уніфікованим для всіх програм, оскільки ми винесли логіку підключення до бази даних у окремий метод, описаний раніше.

```java
public static void main(String[] args) {
    App.run(Sandbox.class);
    Database.configure(dataSource());
}

public static DataSource dataSource() {  1 usage
    DefaultFileTypeDetector DataSourceBuilder = null;
    return DataSourceBuilder.create()
            .url("jdbc:postgresql://localhost:5432/mydatabase")
            .username("myuser")
            .password("mypassword")
            .build();
}
```

Рисунок 3.3 – Клас Main.java в IntegrationResearch.proj (створено самостійно)

Методи для створення навантаження та виміру ефективності (дивись рисунок 3.4 для об’єктів та рисунок 3.5 для JSON).

Клас BlobTest використовується під час навантажувального тестування для заміру швидкості читання, запису, оновлення даних з залученням файлів великого розміру у різних форматах (див.рис.3.4).

Клас BlobTest використовується під час навантажувального тестування REST для заміру швидкості відповіді, часу на обробку запитів з використанням JSON об’єктів типу Item (див.рис.3.5).

```
public class BlobTest {  no usages
    private final Connection connection;  5 usages


    public void createBlob(int id, byte[] data) throws SQLException {  no usages
        String sql = "INSERT INTO blobs (id, data) VALUES (?, ?)";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt( parameterIndex: 1, id);
            statement.setBytes( parameterIndex: 2, data);
            statement.executeUpdate();
        }
    }

    public byte[] readBlob(int id) throws SQLException {  no usages
        String sql = "SELECT data FROM blobs WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt( parameterIndex: 1, id);
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                return resultSet.getBytes( columnLabel: "data");
            }
        }
        return null;
    }

    public void updateBlob(int id, byte[] data) throws SQLException {  no usages
        String sql = "UPDATE blobs SET data = ? WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setBytes( parameterIndex: 1, data);
            statement.setInt( parameterIndex: 2, id);
            statement.executeUpdate();
        }
    }

    public void deleteBlob(int id) throws SQLException {  no usages
        String sql = "DELETE FROM blobs WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt( parameterIndex: 1, id);
            statement.executeUpdate();
        }
```

Рисунок 3.4 – Клас BlobTest.java в IntegrationResearch.proj (створено самостійно)

```
public class ItemTest {  no usages
    public void createItem(int id, String name, double price) throws SQLException {  no usages
        String sql = "INSERT INTO items (id, name, price) VALUES (?, ?, ?)";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt( parameterIndex: 1, id);
            statement.setString( parameterIndex: 2, name);
            statement.setDouble( parameterIndex: 3, price);
            statement.executeUpdate();
        }
    }

    public String readItem(int id) throws SQLException {  no usages
        String sql = "SELECT name, price FROM items WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt( parameterIndex: 1, id);
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                return "Name: " + resultSet.getString( columnLabel: "name") + ", Price: " + resultSet.getDouble( columnLabel: "price");
            }
        }
        return null;
    }

    public void updateItem(int id, String name, double price) throws SQLException {  no usages
        String sql = "UPDATE items SET name = ?, price = ? WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString( parameterIndex: 1, name);
            statement.setDouble( parameterIndex: 2, price);
            statement.setInt( parameterIndex: 3, id);
            statement.executeUpdate();
        }
    }

    public void deleteItem(int id) throws SQLException {  no usages
        String sql = "DELETE FROM items WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt( parameterIndex: 1, id);
            statement.executeUpdate();
        }
    }
```

Рисунок 3.5 – Клас ItemTest.java в IntegrationResearch.proj (створено самостійно)

3.2 Створення сценаріїв навантаження

Для створення навантаження було розроблено скрипт на мові Python, що надсилає 100 тисяч запитів з файлами та 100 тисяч запитів з об’єктами одночасно, використовуючи 50 потоків процесора(дивись рисунок 3.6).

```python
import requests
import threading
import json
from concurrent.futures import ThreadPoolExecutor

def send_file_request():
    url = "http://localhost:8080/blobtest"
    files = {'file': ('example.txt', open('example.txt', 'rb'))}
    response = requests.post(url, files=files)
    print(f"File request response: {response.status_code}")

def send_json_request():
    url = "http://localhost:8080/itemtest"
    data = {"key": "value"}
    headers = {'Content-Type': 'application/json'}
    response = requests.post(url, data=json.dumps(data), headers=headers)
    print(f"JSON request response: {response.status_code}")

with ThreadPoolExecutor(max_workers=50) as executor:
    for _ in range(100000):
        executor.submit(send_file_request)
    for _ in range(100000):
        executor.submit(send_json_request)
```

Рисунок 3.6 – Файл LoadTest.py в IntegrationResearch.proj (створено самостійно)

З залученням скрипту та ізолюванням середовища для кожного з інструментів для тестування навантаження back-end серверів на виході було отримано дані для дослідження кожного підходу.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментальних досліджень

Для проведення дослідження трьох технологій Apache JMeter, Gatling та Tsung з метою оцінки швидкості виконання запитів та використання пам'яті було зроблено наступні кроки:

- підготовка середовища: налаштовано окремі сервери для кожного інструменту (JMeter, Gatling, Tsung), щоб уникнути взаємного впливу, забезпечена умова, щоб всі сервери мали однакові апаратні та програмні характеристики;
- створення сценаріїв навантаження: розроблено сценарій навантаження для кожного інструменту. Сценарій включає 100 тис. одночасних користувачів, які надсилають HTTP запити на певний сервер(PUT, POST, GET, DELETE), використано однакову цільову систему (сервер), щоб зібрати дані для порівняння;
- вимірювання показників: швидкість виконання запитів (мс) - заміряно середній час виконання запиту для кожного інструменту, витрати пам'яті (МБ) – заміряно середнє використання пам'яті програмою під час виконання запиту;
- збір даних: запущено кожен сценарій 10 разів і зібрано дані для кожного запуску.

4.2 Аналіз отриманих результатів

Було обчислено середні значення та стандартні відхилення для кожного показника та візуалізовано результати у вигляді таблиці для порівняння(дивись рисунки 4.1 та 4.2).

Для вимірів було використано виділений сервер від Amazon T4G Medium[7], що дозволило мати точні та незалежні результати для дослідження.

Інструмент	Швидкість виконання запиту для файлів (мс)	Стандартне відхилення (мс)	Використання пам'яті (МБ)	Стандартне відхилення (МБ)
JMeter	150	10	200	15
Gatling	130	8	180	12
Tsung	170	12	190	14

Рисунок 4.1 – результати тестування для файлів

Виходячи з результатів тестування Gatling є найкращими інструментом для проводження тестування з залученням великих файлів.

Інструмент	Швидкість виконання запиту для об'єктів (мс)	Стандартне відхилення (мс)	Використання пам'яті (МБ)	Стандартне відхилення (МБ)
JMeter	35	4	20	5
Gatling	60	2	16	3
Tsung	50	6	21	8

Рисунок 4.2 – результати тестування для об’єктів

Виходячи з результатів тестування для об’єктів немає очевидного вибору та обрання певного інструменту залежить від потреб, а саме JMeter, якщо існує потреба у швидкій відповіді але пам’ять не є критичною та Gatling за умов обмеженної оперативний пам’яті.

Загалом кожен з підходів має свої переваги та недоліки, а саме Tsung може стати у нагоді через свої переваги:

- дуже масштабований та може витримати велику кількість одночасних користувачів;
- точність вимірювань: Надає розширені звіти та графіки про продуктивність;
- налаштування: Використовує XML для опису сценаріїв, можливість налаштовувати різні параметри тестів;

- протоколи: Підтримує різні протоколи, включаючи HTTP, WebDAV, SOAP, XMPP, та інші;
- зручність використання: Має консольний інтерфейс та можливість віддаленого управління.

Але за умов коли потрібна швидкість та оптимальність використання ресурсів слід поглянути на наведені аналоги.

Якщо ми розглядаємо обсяг коду, варто зазначити, що для менших проектів вибір між різними підходами до роботи не допоможе значно зменшити обсяг коду для кожного з API. Такі переваги стають помітними в основному у великих проектах зі складною структурою логіки.

ВИСНОВКИ

Після завершення даної кваліфікаційної практики було досягнуто всіх поставлених цілей, відповідно до технічного завдання проекту. Конкретно:

- було розроблено технічне завдання для магістерського дослідження;
- проведено збір та опис теоретичних матеріалів з теми моніторингу back-end і напрямків досліджень;
- наведено та описано основну інформацію щодо засобів моніторингу та вимірювання;
- здійснено аналіз актуальних рішень на ринку;
- розроблено програмне забезпечення для порівняння методів моніторингу back-end систем.

Було створено план проведення експериментального дослідження та проведено його. Для оцінювання було обрано критерії порівняння програмних реалізацій були обрані швидкість виконання запитів (мс), кількість витраченної пам’яті на виконання запиту (byte) та кількість коду, який необхідно написати.

На основі результатів проведених експериментів було створено рекомендації з використання систем для тестування високонавантажених back-end застосунків.

Була спроектована модель даних та розроблена база даних. Було створено зібрано усі необхідні метрики для аналізу.

За результатами роботи були створені тези доповіді IX Міжнародна науково-технічна конференції «Поліграфічні, мультимедійні та web-технології» за темою «Аналіз методів моніторингу та вимірювання продуктивності високонавантажених back-end систем»

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Backend Monitoring vs. Frontend Monitoring | Mirbozorgi. Mirbozorgi. URL: https://mirbozorgi.com/backend-monitoring-vs-frontend-monitoring/ (дата звернення: 2.12.2023).

2. Staying Ahead: Performance Monitoring in Your Backend Plan - FasterCapital. FasterCapital. URL: https://fastercapital.com/content/Staying-Ahead--Performance-Monitoring-in-Your-Backend-Plan.html#Tools-and-Techniques (дата звернення: 3.12.2023).

3. Web Application Monitoring Best Practices. Software Development Company | LeanyLabs. URL: https://leanylabs.com/blog/web-monitoring-best-practices/#use-informative-metrics (дата звернення: 10.12.2023).

4. Tschense A. Java Monitoring Infrastructure in Sap Netweaver '04. SAP Press America, 2004. 91 p.

5. Критерії прийняття рішень. Studies. URL: https://studies.in.ua/teorija-pryjnjattja-sus-pol-rishen-shpargalky/4087-kriteryi-priynyattya-rshen.html (дата звернення: 24.03.2024).

6. Головна - Таврійський державний агротехнологічний університет імені Дмитра Моторного. URL: http://www.tsatu.edu.ua/et/wp-content/uploads/sites/33/prezentacija-do-temy-nevyznachenist.pdf (дата звернення: 21.03.2024).

7. Amazon EC2 instance types - Amazon Elastic Compute Cloud. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html (date of access: 02.06.2024).