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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика підвищення ефективності навантажувального тестування високонавантажених систем»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Богдан СТУДЕНТ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Богдан СТУДЕНТ

Керівник: Віталій ЗАЛИВА
доктор філософії, старший викладач кафедри

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2025

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

Студенту Богдану Михайловичу

1. Тема кваліфікаційної роботи: «Методика підвищення ефективності навантажувального тестування високонавантажених систем»

керівник кваліфікаційної роботи Віталій ЗАЛИВА, доктор філософії

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, параметри мультистратегічного методу, вимоги до навантажувального тестування високонавантаженої системи.

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

1. Дослідження навантажувального тестування.
2. Аналіз технологій навантажувального тестування високонавантажених систем.

3. Розробка методики підвищення ефективності навантажувального тестування високонавантажених систем.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкт та предмет дослідження.
2. Аналіз методів навантажувального тестування.
3. Алгоритм розподілу «віртуальних користувачів» у мультистратегічному методі.
4. Діаграма класів розподілу «віртуальних користувачів».
5. Математична модель мультистратегічного методу.
6. Процес розподілу навантаження.
7. Порівняння результатів тестування.
8. Висновки.
9. Публікації та апробація роботи.

6. Дата видачі завдання «16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-23.10.2024	
2	Вивчення матеріалів для аналізу розвитку технологій навантажувального тестування	24.10-27.10.2024	
3	Дослідження методів навантажувального тестування	28.10-31.10.2024	
4	Аналіз особливостей методів навантажувального тестування високонавантажених систем	01.11-03.11.2024	
5	Дослідження технологій тестування високонавантажених систем	04.11-09.11.2024	
6	Застосування навантажувального тестування у високонавантажених системах	10.11-13.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	14.11-19.11.2024	
8	Розробка демонстраційних матеріалів	20.11-24.11.2024	
9	Попередній захист роботи	29.11.2024	

Здобувач вищої освіти _____ Богдан СТУДЕНТ
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 82 стор., 8 табл., 19 рис., 37 джерел.

Мета роботи – підвищення ефективності виконання навантажувального тестування для високонавантажених систем за рахунок використання мультистратегічного методу тестування.

Об'єкт дослідження – процес навантажувального тестування високонавантажених систем.

Предмет дослідження – методи навантажувального тестування високонавантажених систем.

У кваліфікаційній роботі досліджено методи підвищення ефективності навантажувального тестування високонавантажених систем. Проведено аналіз сучасних підходів та інструментів тестування. Розроблено мультистратегічний метод навантажувального тестування, що поєднує просту, фіксовану та змінну стратегії навантаження, що дозволяє адаптивно оптимізувати розподіл потоків залежно від стану системи. Запропоновано математичну модель оцінки ефективності системи, проведено експериментальну перевірку розробленого методу. Робота сприяє підвищенню точності, стабільності та продуктивності систем під час пікових навантажень.

КЛЮЧОВІ СЛОВА: НАВАНТАЖУВАЛЬНЕ ТЕСТУВАННЯ, ВИСОКОНАВАНТАЖЕНІ СИСТЕМИ, МЕТОДИ ТЕСТУВАННЯ, СТРАТЕГІЇ ТЕСТУВАННЯ.

ABSTRACT

Text part of the master's qualification work: 82 pages, 19 pictures, 8 table, 37 sources.

The purpose of the work – to improve the efficiency of load testing for high-load systems by implementing a multi-strategic testing method.

Object of research – the process of load testing high-load systems.

Subject of research – methods of load testing high-load systems.

Summary of the work: The qualification thesis investigates methods to enhance the efficiency of load testing for high-load systems. An analysis of modern testing approaches and tools was conducted. A multi-strategic load testing method was developed, combining simple, fixed, and variable load strategies, enabling adaptive optimization of thread distribution depending on the system's state. A mathematical model for evaluating system efficiency was proposed, and the developed method was experimentally validated. The work contributes to improving the accuracy, stability, and performance of systems during peak loads.

KEYWORDS: LOAD TESTING, HIGH-LOAD SYSTEMS, TESTING METHODS, TESTING STRATEGIES.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

TPS - Transactions Per Second

RAT - Success Rate

Err - Error Count

BPS - Bytes Per Second

CI/CD - Continuous Integration/Continuous Delivery

TTFB - Time to First Byte

API - Application Programming Interface

ВСТУП

Сучасний розвиток інформаційних технологій та зростаючі потреби користувачів у стабільних і високопродуктивних системах зумовлюють необхідність ефективного навантажувального тестування. Високонавантажені системи, такі як фінансові платформи, стрімінгові сервіси, онлайн-магазини, щоденно обробляють мільйони запитів, що вимагає ретельного тестування для забезпечення їхньої надійності, продуктивності та масштабованості.

Однією з ключових проблем у навантажувальному тестуванні є адаптація до змінних умов роботи системи, таких як пікові навантаження, динаміка запитів та обмеженість ресурсів. Існуючі методи тестування часто не враховують ці аспекти, що призводить до недостатньої точності отриманих результатів і ризику виникнення збоїв у критичних умовах.

У сучасних умовах високотехнологічних систем та додатків, які обробляють великі обсяги даних і обслуговують тисячі користувачів одночасно, забезпечення їхньої стабільності та продуктивності є дуже важливим. Високонавантажені системи, такі як фінансові платформи, стрімінгові сервіси та онлайн-магазини, стикаються з викликами, пов’язаними зі зростаючими вимогами до продуктивності, масштабованості та надійності.

Існуючі стратегії навантажувального тестування часто не забезпечують достатньої точності й ефективності в умовах, особливо при пікових навантаженнях, що може призводити до некоректних результатів і значних ризиків для бізнесу. Використання поєднання різних стратегій у межах одного тесту дозволяє адаптивно змінювати навантаження та враховувати динамічні зміни у поведінці системи, що сприяє підвищенню якості тестування.

Метою роботи є розробка мультистратегічного методу навантажувального тестування високонавантажених систем, який об’єднує просту, фіксовану та змінну стратегії для оптимального розподілу потоків і забезпечення точності тестування.

Об'єкт дослідження: процес навантажувального тестування високонавантажених систем.
Предмет дослідження: методи навантажувального тестування високонавантажених систем.

Наукова новизна роботи полягає у запропонуванні мультистратегічного підходу, що дозволяє адаптивно змінювати вагу стратегій залежно від стану системи. Це сприяє покращенню стабільності, зниженню часу відповіді та зменшенню кількості помилок під час тестування.

Для досягнення мети було вирішено наступні завдання:

- Проведено аналіз існуючих методів навантажувального тестування.
- Розроблено мультистратегічний метод, що адаптується до змінних умов роботи системи.
- Сформульовано математичну модель оцінки ефективності тестування.
- Реалізовано алгоритм розподілу потоків між різними стратегіями.
- Проведено експериментальне тестування запропонованого методу та аналіз отриманих результатів.

Практична значущість результатів полягає у можливості впровадження розробленого методу для тестування реальних високонавантажених систем, що забезпечить підвищення їхньої надійності та продуктивності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Призначення та мета навантажувального тестування

Навантажувальне тестування - це процес імітації реального використання системи для оцінки поведінки програмного забезпечення, веб-сайтів, веб-додатків або API під великим навантаженням. Це необхідний крок для виявлення таких проблем, як затримки в обробці запитів, зниження продуктивності або обмеження масштабованості.

Приклади навантажувального тестування можуть відрізнятися залежно від варіанту використання. Наприклад, тестування великої кількості одночасних покупок під час розпродажу, завантаження великих файлів або перевірка здатності системи обробляти одночасні спроби входу після масштабного оновлення. Цей тип тестування дозволяє змоделювати реальні ситуації, з якими можуть зіткнутися користувачі, і оцінити, як система справляється з піковими навантаженнями.

Метою навантажувального тестування є не тільки перевірка стабільності роботи системи, а й отримання точних даних про ефективність її роботи в різних умовах. Зокрема, воно дозволяє виміряти час відгуку, пропускну здатність і використання ресурсів. Ці показники допомагають виявити межу стабільності системи і з'ясувати, коли вона може вийти з ладу через перевантаження. Важливо, щоб навантаження було нижчим за пікове, адже основна мета - виявити точки, де може відбутися деградація продуктивності ще до досягнення максимально можливого навантаження.

Використання інструментів навантажувального тестування не тільки дозволяє отримати відповідь на питання, чи витримає система очікувані навантаження, але і допомагає визначити:

- Яка максимальна кількість користувачів або запитів може бути оброблена системою без її збою?

- Як збільшення кількості одночасних користувачів впливає на швидкість роботи системи?
- Які компоненти або етапи системи найбільш вразливі до перевантаження?
- Скільки транзакцій може обробити система за певний проміжок часу без зниження якості обслуговування?
- Які існують можливості для оптимізації та підвищення продуктивності системи в умовах високих навантажень?

Рис 1.1. Навантажувальне тестування в процесі розробки ПЗ

Таким чином, навантажувальне тестування є важливою частиною процесу забезпечення надійності та ефективності високонавантажених систем. Воно дозволяє не тільки уникнути можливих проблем, а й своєчасно впровадити необхідні зміни для підвищення стабільності та масштабованості системи в умовах реальних експлуатаційних навантажень.

Щодня все більше користувачів отримують доступ до веб-додатків, а тестування навантаження допомагає знизити ризик збою програмного забезпечення і гарантує, що користувачі не зіткнуться з розчаруванням через проблеми з продуктивністю. Важливо перевірити, чи здатна система витримати

реалістичні сценарії навантаження з реальними користувачами. Навантажувальне тестування дозволяє виявити та діагностувати слабкі місця в системі, щоб швидко їх виправити. Це не тільки дозволяє виявити і виправити проблеми з продуктивністю до того, як програмне забезпечення потрапить у виробниче середовище, але також допомагає заощадити час розробки, що, в свою чергу, знижує витрати на проект.

Ціна помилок може бути набагато вищою, ніж незначні незручності. Згідно з дослідженням CISQ, неякісне програмне забезпечення коштувало економіці США 2,08 трильйона доларів у 2020 році. А зі зростанням використання програмного забезпечення в сучасному цифровому світі ці цифри можуть значно зрости. Помилки в програмному забезпеченні можуть мати серйозні фінансові наслідки, спричиняючи кіберзлочини, витоки даних та фінансові крадіжки, кількість яких зростає з кожним роком. Завдяки навантажувальному тестуванню ви можете виявити проблеми на ранніх стадіях і вирішити їх, щоб не запускати погано працюючі веб-сайти і додатки у виробництво.

Раннє виявлення таких проблем не тільки дозволяє уникнути негативних наслідків, але й підтримує репутацію компанії, оскільки користувачі, які стикаються з низькою продуктивністю, часто переходять до конкурентів. Таким чином, інвестуючи в навантажувальне тестування, компанії можуть забезпечити стабільність і високу якість своїх систем, що є ключовим у сучасному конкурентному середовищі.

Погана робота веб-сайтів і додатків може мати серйозні негативні наслідки для компаній, і навіть кілька секунд простою можуть суттєво вплинути на їхні фінансові результати. Згідно з дослідженням Gartner, середня вартість простою становить 5600 доларів за хвилину. Наприклад, у березні 2019 року 14-годинний простій Facebook коштував компанії приблизно 90 мільйонів доларів. Оцінки вартості простою можуть варіюватися від $100 000 на годину до понад $540 000 на годину, залежно від типу бізнесу. Іншим яскравим прикладом є ситуація з Target під час «чорної п'ятниці» 2019 року, коли їхній веб-сайт вийшов з ладу через

нездатність впоратися з великим трафіком, що призвело до втрати продажів і погіршення користувацького досвіду.

Подібні інциденти та низька продуктивність додатків є фінансовими жнивами і можуть бути токсичними для довіри та лояльності клієнтів. Вони не лише спричиняють фінансові збитки, але й впливають на репутацію компанії, що, в свою чергу, може призвести до втрати клієнтів та зниження конкурентоспроможності. Тому компаніям важливо запобігати таким ситуаціям, регулярно тестуючи свої системи, щоб переконатися, що вони можуть витримувати високі навантаження і залишатися стабільними навіть у найнапруженіші моменти.

1.2 Етапи проведення навантажувального тестування

Високонавантажені системи, такі як онлайн-банкінг, інтернет-магазини під час великих розпродажів або соціальні мережі, які обробляють мільйони одночасних запитів, вимагають особливого підходу до тестування, оскільки кожен збій може мати серйозні наслідки для бізнесу. Навантажувальне тестування таких систем допомагає оцінити їхню здатність витримувати великі обсяги трафіку і запитів, зберігаючи при цьому стабільну роботу. Наприклад, інтернет-магазин під час «чорної п'ятниці» повинен бути готовий до тисяч одночасних транзакцій, а платформа онлайн-банкінгу - до мільйонів запитів за короткий проміжок часу.

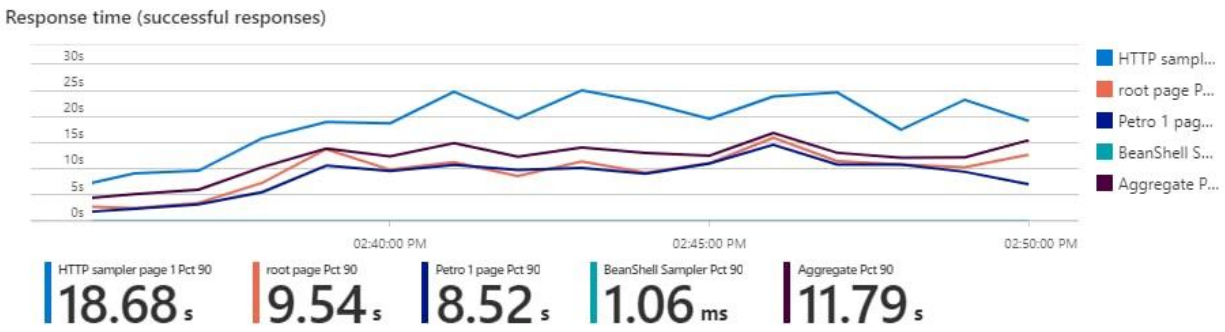


Рис 1.2. Приклад автоматизації зібрання тестових результатів

Щоб переконатися, що система витримає такі навантаження, необхідно правильно налаштувати процес тестування, який складається з декількох етапів.

Щоб почати тестування високонавантаженої системи, спочатку потрібно визначити обсяг і цілі тестування, а також вибрати інструмент, який найкраще відповідає вимогам. Раніше навантажувальне тестування проводилося в кінці розробки, але з розвитком технологій і появою таких інструментів, як LoadView, ви можете почати тестування з самого початку без шкоди для якості, отримуючи точні результати для подальшої оптимізації системи. Ось покрокове керівництво про те, як правильно проводити навантажувальне тестування для високонавантажених систем:

- Визначення бізнес-цілей і завдань

На початку тестування важливо зібрати всі вимоги і чітко визначити цілі. Це включає в себе визначення того, що саме потрібно покращити або протестувати: час відгуку, пропускну здатність, використання ресурсів або максимальний рівень навантаження, який може витримати система. Наприклад, для інтернет-магазину це може бути максимальна кількість одночасних покупок, яку система може обробити без погіршення продуктивності. Для фінансової платформи важливо визначити, скільки одночасних транзакцій вона може обробити без перевантаження серверів. *Приклад*: Для інтернет-магазину основним завданням може бути перевірка здатності обробляти одночасні покупки під час «чорної п'ятниці», тоді як для фінансової платформи важливо оцінити, скільки одночасних транзакцій система може обробити без перевантаження серверів.

- Визначення шляхів користувачів

Важливо моделювати реальні сценарії використання системи. Потрібно створити шляхи, які проходитимуть користувачі при взаємодії з високонавантаженою системою, наприклад, процес здійснення покупки в інтернет-магазині або подання заявки на кредит через онлайн-банкінг. Для цього корисно використовувати метрики APM (Application Performance Management), щоб отримати точну картину того, як користувачі переміщаються по системі. Це дозволить будувати більш реалістичні сценарії для тестування та виявлення

потенційних проблем у найважливіших частинах системи. *Приклад*: У процесі тестування інтернет-магазину сценарії можуть включати навігацію по каталогу, вибір товару, додавання його в кошик, здійснення покупки. Для цього використовуються APM (Application Performance Management) метрики для створення реалістичних сценаріїв, які допоможуть виявити можливі проблеми в критичних точках системи.

- Встановлення контрольної бази

Для оцінки результатів тестування важливо встановити базову лінію для системи, яка дозволить порівняти продуктивність під час тесту з ідеальним станом. Наприклад, можна визначити час відгуку для декількох типових операцій (пошук товару в магазині, авторизація користувача), щоб порівняти ці значення з результатами після навантаження. Це дозволить виявити відхилення від контрольної бази і визначити слабкі місця в системі. *Приклад*: Можна встановити базовий час відгуку для операцій, таких як авторизація, пошук, обробка замовлення. Це допоможе зрозуміти відхилення від норми під час навантажувальних тестів.

- Автоматизація та ітерації

Завдяки автоматизації можливо регулярно проводити навантажувальне тестування в міру розширення і масштабування вашого бізнесу. Це особливо важливо для високонавантажених систем, оскільки навантаження на систему змінюється в залежності від часу доби, сезонних коливань або подій (наприклад, великих розпродажів або кампаній). Важливо, щоб тестування стало частиною процесу розробки і постійно повторювалося для виявлення нових проблем і підвищення стабільності системи.

- Вибір інструменту для навантажувального тестування

Для успішного тестування важливо вибрати інструмент, який підтримує великий обсяг навантаження, масштабованість і точну звітність. Наприклад, LoadView - це потужний інструмент, який дозволяє тестувати реальні сценарії з реальними браузерами, використовуючи понад 40 геолокацій для імітації глобального доступу користувачів. Зібрані дані дозволяють точно оцінити

продуктивність системи, виявити вузькі місця і вчасно внести корективи для забезпечення стабільної роботи під високим навантаженням. *Приклад*: Великі інтернет-магазини та платформи, такі як Amazon, використовують потужні інструменти для тестування, які дозволяють працювати з тисячами одночасних запитів та забезпечують стабільність під час розпродажів.

1.3 Інструменти для навантажувального тестування

Інструменти для навантажувального тестування є незамінними при перевірці стабільності та масштабованості високонавантажених систем. Вибір правильного інструменту для тестування залежить від типу системи, вимог до навантаження, масштабів проекту та наявних ресурсів.

За даними Gartner, у 2020 році понад 60% компаній, які займаються розробкою високонавантажених систем, вказали, що основним викликом для них є підтримка великого масштабу навантажувальних тестів при обмежених інфраструктурних ресурсах [1]. Більше того, згідно з опитуванням, проведеним TechValidate в 2021 році, близько 45% компаній зазначили, що основною проблемою є низька ефективність і висока вартість традиційних інструментів тестування при великих навантаженнях. Тому компанії змушені шукати рішення, які дозволяють знижувати витрати та час, необхідний для тестування, одночасно підвищуючи його точність.

Інструменти для навантажувального тестування зазвичай класифікуються за кількома параметрами:

Інструменти з відкритим кодом: Наприклад, Apache JMeter і k6. Вони є популярними завдяки своїй доступності (безкоштовність) і можливості легко налаштовувати різні сценарії тестування. Однак вони можуть мати обмеження по масштабуванню або підтримці специфічних протоколів для складних тестувань.

Комерційні інструменти: Такі як LoadRunner і WebLOAD, які мають більшу кількість можливостей для масштабування, аналізу та підтримки широкого спектра

протоколів. Проте їх вартість може бути значною, особливо для малих і середніх компаній.

Інструменти для хмарного тестування: Наприклад, BlazeMeter та LoadView, які дозволяють проводити масштабовані навантажувальні тести без необхідності створювати власну фізичну інфраструктуру. Це зменшує витрати і дозволяє швидко адаптувати тести під різні вимоги.

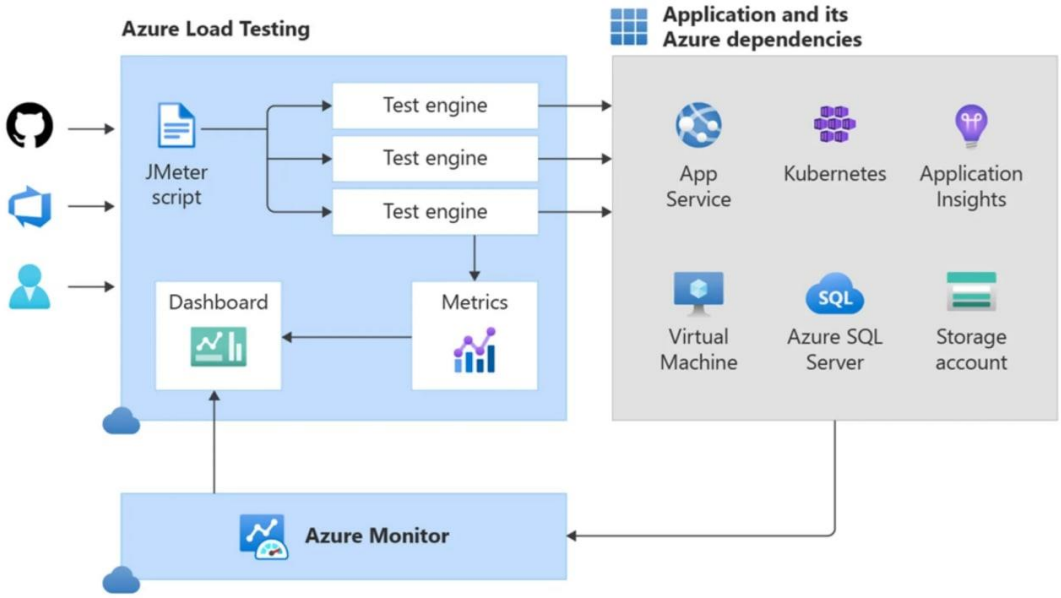


Рис 1.3. Схема реалізації навантажувального тестування за допомогою інфраструктури Microsoft Azure DevOps

Як показують статистичні дані, 50% компаній, що займаються тестуванням високонавантажених систем, заявили про підвищення ефективності тестів завдяки використанню хмарних рішень для моделювання масштабних навантажень [2]. Це дозволяє тестувати системи з глобальним доступом, моделюючи реальні умови, зокрема навантаження з різних частин світу.

Згідно з опитуванням TechRadar, 60% розробників вказали, що важливою особливістю при виборі інструменту є підтримка інтеграцій з CI/CD, що дозволяє автоматизувати тестування і включити його у процес безперервної інтеграції та доставки. Такі інструменти, як k6, стали популярними завдяки своїй інтеграції з популярними CI/CD платформами, такими як Jenkins та GitLab.

Вибір правильного інструменту також залежить від підтримки протоколів. Більшість інструментів підтримує основні веб-протоколи (HTTP, HTTPS, FTP), але для складніших систем, таких як банківські платформи чи платформи для електронної комерції, можуть знадобитися інструменти, які підтримують специфічні протоколи, як наприклад JMeter та LoadRunner, які можуть працювати з більш ніж 50 різними протоколами.

Зростання популярності мікросервісів і хмарних рішень значно змінило підхід до навантажувального тестування. Платформи, які підтримують моделювання трафіку з різних географічних точок, дозволяють компаніям імітувати реальні умови, тестуючи, як система буде працювати під високим навантаженням, що поступово поступає з різних куточків світу. Це стає можливим завдяки використанню таких інструментів, як LoadView, який надає можливість моделювати навантаження через більш ніж 40 геолокацій по всьому світу.

1.3.1 Критерії для вибору інструмент для навантажувального тестування високонавантажених систем

Вибір інструменту для навантажувального тестування високонавантажених систем є першочерговим етапом у забезпеченні стабільності та ефективності таких систем. Враховуючи масштаб і складність цих систем, правильний вибір інструменту може значно вплинути на точність і ефективність тестування.

Ключові фактори при виборі інструменту:

- Тип тестованої системи та протоколи
- Визначте, який тип системи ви тестуєте: вебсайт, API, мобільний додаток, мікросервіси або великі корпоративні платформи. Кожен тип системи має свої особливості, які потребують різних підходів до тестування.
- API: Для тестування API з високими навантаженнями інструменти, такі як k6, що підтримують написання тестів на JavaScript, є ідеальним вибором, оскільки вони оптимізовані для таких задач.
- Вебзастосунки та мобільні додатки: Для тестування вебзастосунків або мобільних додатків, що мають складну структуру, Gatling може бути

кращим вибором завдяки своїй здатності обробляти високі навантаження та створювати точні сценарії тестування, використовуючи Scala.

- Корпоративні платформи: Для великих корпоративних платформ, таких як фінансові установи, державні системи або глобальні інтернет-магазини, важливо, щоб інструмент підтримував не тільки HTTP/HTTPS, але й специфічні протоколи, такі як FTP, JDBC та SOAP. Для цього найкраще підходять JMeter і LoadRunner, які підтримують понад 50 різних протоколів [1].

Масштаб системи безпосередньо впливає на вибір інструменту. Якщо система обробляє мільйони одночасних користувачів або тисячі транзакцій в секунду, тестування повинно бути масштабованим.

LoadRunner є ідеальним для великих корпоративних платформ, оскільки він підтримує масштабування для обробки тисяч одночасних запитів.

Для середнього рівня навантаження на API або вебзастосунки більш підходять k6 або Gatling, які оптимізовані для високопродуктивних тестів без великих затрат ресурсів.

Статистика: Згідно з опитуванням TechValidate, 40% організацій, що тестують високонавантажені системи, вказують на те, що їм не вдається підтримати необхідний рівень масштабування при використанні традиційних інструментів, таких як JMeter. Для подолання цього обмеження компанії звертаються до хмарних рішень, які дозволяють масштабувати тестування без необхідності значних інвестицій в інфраструктуру [2].

У сьогоднішньому світі розробка програмного забезпечення часто включає безперервну інтеграцію та доставку (CI/CD), тому вибір інструменту для тестування має підтримувати автоматизовані процеси.

k6, Gatling і JMeter мають вбудовану підтримку для інтеграції з популярними CI/CD платформами, такими як Jenkins, GitLab або CircleCI, що дозволяє тестувати систему в умовах реального часу.

Статистика: Згідно з результатами опитування TechValidate, 50% компаній, що займаються навантажувальним тестуванням, активно використовують CI/CD

для інтеграції тестів у процеси розробки. Це дозволяє автоматизувати процес тестування і скоротити час між змінами коду та перевіркою його ефективності [3].

Вартість ліцензії є ще одним вагомим фактором при виборі інструменту для навантажувального тестування. Комерційні інструменти, як LoadRunner або WebLOAD, можуть бути дорогими, однак вони надають широкі можливості для масштабованого тестування і аналізу.

Для компаній з обмеженим бюджетом або для тих, хто тільки починає тестувати високонавантажені системи, інструменти з відкритим кодом, як-от JMeter або k6, можуть стати оптимальним рішенням.

Статистика: 70% малих та середніх компаній вказують, що вибір безкоштовних або відкритих інструментів для тестування є важливим фактором у їхньому рішенні через обмеження бюджету. Однак, при тестуванні великих або складних систем, інвестування в комерційні інструменти може бути виправдане за рахунок потужніших можливостей і більш точних результатів [4].

Вибір інструменту також залежить від підтримки необхідних протоколів. Якщо система використовує специфічні протоколи, важливо обрати інструмент, який підтримує ці протоколи.

JMeter і LoadRunner підтримують безліч протоколів, таких як HTTP, FTP, JDBC, SOAP, WebSocket, і це робить їх хорошим вибором для великих і комплексних систем.

Статистика: 60% компаній, що тестують високонавантажені системи, повідомляють про проблеми з підтримкою специфічних протоколів в інструментах тестування. Наприклад, для великих банківських платформ або систем інтернет-торгівлі важливо, щоб інструмент підтримував не тільки HTTP/HTTPS, але й специфічні протоколи, як-от FTP, JDBC та SOAP. Для цього найкраще підходять JMeter і LoadRunner [5].

1.3.2 Порівняння сучасних інструментів навантажувального тестування

Навантажувальне тестування є критичним етапом для забезпечення ефективності і стабільності високонавантажених систем. Вибір правильного

інструменту дозволяє не тільки скоротити час, витрачений на тестування, але й отримати точні результати, які допоможуть виявити вузькі місця до того, як система потрапить в експлуатацію. Згідно з дослідженням Gartner, 60% організацій заявляють, що вибір правильного інструменту безпосередньо впливає на успішність тестування високонавантажених систем [1].

Інструменти для навантажувального тестування можуть варіюватися залежно від типу тестованої системи, її масштабів і вимог до продуктивності. Для високонавантажених систем, таких як інтернет-магазини, фінансові платформи або хмарні сервіси, важливо, щоб інструмент підтримував не тільки базові функції тестування, а й можливість масштабування під величезні навантаження.

Порівняння основних інструментів для навантажувального тестування високонавантажених систем показує їхні переваги та недоліки, що допоможе вибрати найбільш підходящий інструмент для конкретного випадку.

Таблиця 1.1

Переваги та недоліки інструментів

Інструмент	Переваги	Недоліки
JMeter	- Відкритий код. - Підтримка різних протоколів. - Велика кількість плагінів.	- Велике споживання пам'яті при великих навантаженнях.
Gatling	- Висока продуктивність. - Легкий для налаштування тестів.	- Потребує знань Scala для написання сценаріїв. - Обмежена підтримка протоколів порівняно з іншими інструментами.
k6	- Швидкість і легкість використання. - Підтримка JavaScript для написання сценаріїв. - Інтеграція з CI/CD.	- Обмежена підтримка деяких протоколів. - Обмежені інструменти для аналізу результатів.
LoadRunner	- Підтримка понад 50 протоколів. - Потужний аналіз і звітність. - Підходить для великих підприємств.	- Висока вартість. - Складність в освоєнні для новачків.

Продовження таблиці 1.1

Переваги та недоліки інструментів

WebLOAD	- Розширена підтримка протоколів. - Можливість масштабування для великих тестів.	- Висока вартість ліцензії. - Може бути складним для нових користувачів.

JMeter — популярний інструмент для тестування вебсайтів, API і середніх за навантаженням систем. Він має великий набір плагінів і можливостей для тестування, однак при великому навантаженні його ефективність може знижуватись через високе споживання пам'яті, що може призвести до проблем з масштабуванням. Згідно з дослідженням Open Source Testing, 40% компаній, що використовують JMeter, стикаються з обмеженнями по продуктивності при обробці дуже великих обсягів даних [2].

Gatling ідеально підходить для вебзастосунків та API, де важлива висока швидкість та продуктивність. Цей інструмент дозволяє створювати складні тести за допомогою мови Scala і має високий рівень інтеграції з CI/CD, що робить його популярним серед розробників. Проте, для новачків цей інструмент може бути складним, оскільки він потребує знань Scala для налаштування тестів. TechValidate вказує, що 35% компаній з великими обсягами навантаження вибирають Gatling за високу продуктивність та здатність обробляти складні сценарії тестування [3].

k6 є чудовим вибором для API та мікросервісів, оскільки підтримує автоматизоване тестування та інтеграцію з CI/CD процесами. Це дозволяє знизити час на тестування та автоматизувати процеси, що є важливо для компаній, які активно використовують DevOps-підхід. Однак для систем з більш складними протоколами k6 може бути обмеженим у порівнянні з іншими інструментами, такими як LoadRunner. BlazeMeter, компанія, яка підтримує k6, повідомляє, що 45% компаній застосовують його для тестування API та мікросервісів [4].

LoadRunner — потужний комерційний інструмент, який добре підходить для великих корпоративних систем, що мають складну архітектуру і високі вимоги до підтримки протоколів. Його основна перевага — здатність справлятися з дуже великими навантаженнями та підтримка більше 50 протоколів. Однак його висока вартість і складність в освоєнні можуть стати бар'єром для невеликих команд. TechRadar відзначає, що 60% підприємств, що використовують LoadRunner, зосереджуються на масштабуванні тестування для глобальних додатків [5].

WebLOAD підходить для тестування великих систем, зокрема для електронної комерції та банківських платформ, де важливою є можливість масштабування тестів. Однак висока вартість ліцензії може бути значною перешкодою для малих компаній. За результатами дослідження Load Testing Solutions, 50% великих підприємств використовують WebLOAD для перевірки складних сценаріїв навантаження [6].

1.4 Проблеми та виклики навантажувального тестування високонавантажених систем

Тестування стикається з низкою складних проблем, які виникають через складність самих систем, величезний обсяг даних і великий масштаб взаємодії між компонентами. Однією з основних проблем є моделювання реальних умов навантаження. Високонавантажені системи часто мають складну архітектуру з численними мікросервісами, інтеграціями з зовнішніми API та різними базами даних. Це робить відтворення реальних умов складним, оскільки для точного тестування необхідно враховувати різні сценарії використання, включаючи не лише звичайні операції, але й пікові навантаження, коли система обробляє величезну кількість запитів одночасно. Наприклад, під час святкових розпродажів або великих подій, таких як Чорна п’ятниця, онлайн-платформи, такі як Amazon або eBay, стикаються з мільйонами одночасних запитів, що ставить перед системами серйозні виклики у забезпеченні стабільності.

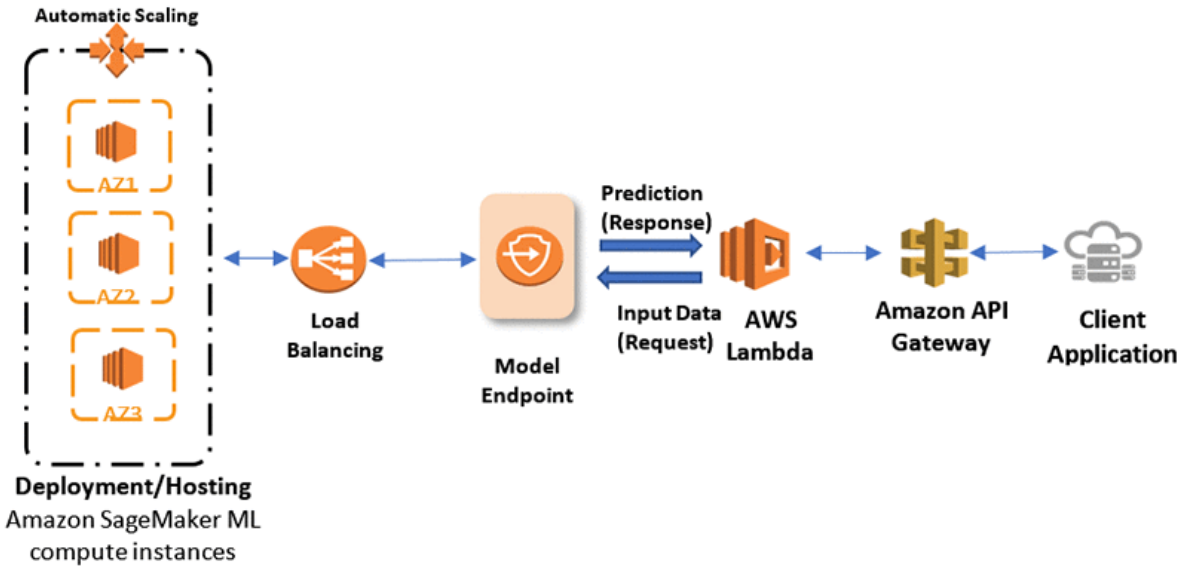


Рис 1.4. Тестування автоматичного виділення інстансів за допомогою Load Balancing та Lambda при збільшенні навантаження

Крім цього варто відзначити, обмеженість ресурсів при проведенні масштабованих тестів. Багато компаній, які тестують високонавантажені системи, стикаються з проблемою відсутності достатніх фізичних ресурсів для тестування з високими навантаженнями. Наприклад, моделювання сотень тисяч або мільйонів одночасних користувачів вимагає значних серверних потужностей. Це часто призводить до потреби у хмарних рішеннях, які можуть значно знизити витрати на інфраструктуру. Як зазначено в дослідженні, лише 25% компаній використовують повний потенціал своїх тестових інфраструктур, а більшість з них стикається з обмеженнями по кількості одночасних користувачів під час тестів [2].

Ще однією проблемою є складність інтеграції з реальними даними та іншими системами. Високонавантажені системи часто складаються з численних мікросервісів, інтегрованих із зовнішніми API, базами даних і іншими програмними компонентами. Для точного тестування необхідно використовувати реальні дані або їх еквіваленти, що може бути складно через складну інтеграцію з іншими платформами або доступ до тестових даних. Однак навіть коли компанії намагаються використовувати заміщення даних (наприклад, генератори тестових

даних), це може не відображати реальні умови експлуатації, що призводить до неточних результатів тестування.

Однією з основних статистичних проблем є визначення критеріїв успіху та показників ефективності під час тестування високонавантажених систем. Визначення критеріїв успіху для тестів, таких як максимальна кількість одночасних користувачів, допустимі затримки або пропускна здатність, може бути досить складним. Наприклад, навіть мінімальні затримки, не помітні при звичайних умовах, можуть спричиняти значні проблеми під великим навантаженням, особливо у випадку висококонкурентних онлайн-ресурсів. Це стає ще складніше при тестуванні додатків, що містять складні етапи обробки даних, такі як транзакції в реальному часі або обробка фінансових операцій.

1.4.1 Способи вирішення проблем

Для подолання викликів та проблем, що виникають при тестуванні високонавантажених систем, існують різні технології, підходи та методи, які активно застосовуються як у великих компаніях, так і серед малих і середніх організацій. Ці способи дозволяють вирішувати проблеми масштабованості, інтеграції, ефективності і ресурсоємності, значно знижуючи час і витрати на тестування.

Використання хмарних технологій для масштабування тестів
Одним із найефективніших способів вирішення проблеми масштабування тестування є використання хмарних платформ. Технології на кшталт Amazon Web Services (AWS), Microsoft Azure та Google Cloud дозволяють компаніям швидко масштабувати навантажувальні тести, моделюючи реальні умови роботи системи під високими навантаженнями. За допомогою хмарних інструментів можна створити тестове середовище з великою кількістю одночасних запитів, не маючи необхідності в дорогих фізичних ресурсах.

Це дозволяє знизити витрати на інфраструктуру і забезпечити швидке проведення тестів, адаптуючи їх до змінюваних умов і кількості одночасних користувачів. Як показує дослідження TechValidate, 65% компаній, які

використовують хмарні технології для навантажувального тестування, відзначають значне зниження витрат на інфраструктуру та підвищення гнучкості тестування [1]. Автоматизація є одним з ключових підходів для скорочення часу тестування та підвищення ефективності процесу. Використання автоматизованих інструментів, таких як JMeter, Gatling, k6 або LoadRunner, дозволяє значно прискорити процес виконання тестів, а також дає змогу запускати тести кілька разів для перевірки результатів при різних умовах.

Автоматизація також дозволяє інтегрувати тестування в CI/CD пайплайни, що автоматизує процеси тестування під час безперервної інтеграції і доставки. Це дозволяє кожен новий код або зміну в системі тестувати автоматично, без потреби в ручному втручанні. 50% компаній, що активно тестують високонавантажені системи, використовують автоматизацію як основний інструмент для прискорення тестування і зменшення витрат [2].

Інтеграція AI та машинного навчання в тестування
Штучний інтелект (AI) та машинне навчання є перспективними технологіями для вирішення проблем тестування високонавантажених систем. Використання AI дозволяє оптимізувати тестові сценарії, автоматично виявляти потенційні вузькі місця та робити прогнози щодо ефективності системи при різних умовах навантаження. Інструменти, що інтегрують AI та машинне навчання, можуть автоматично адаптувати тести на основі результатів попередніх тестів, що знижує витрати на ручне налаштування тестових параметрів.

AI також допомагає в аналізі великих обсягів даних, надаючи більш точні результати та допомагаючи виявити нетипові сценарії, які можуть бути упущені за допомогою традиційних методів тестування. Наприклад, Google та Microsoft активно використовують AI для оптимізації тестування своїх великих платформ, що дозволяє знижувати час на тестування і покращувати точність результатів [3].

Інтеграція з реальними даними для тестування. Для досягнення більш точних результатів важливо інтегрувати тестування з реальними даними або їх заміщенням. Використання генераторів тестових даних, а також зберігання та обробка реальних даних у тестовому середовищі дозволяє отримати більш точні

результати, що відповідають реальним умовам експлуатації. Проте це завдання є складним через можливі юридичні та безпекові обмеження на використання реальних даних.

У разі неможливості використання реальних даних можна використовувати анонімізовані або згенеровані дані, які найбільше наближені до реальних умов. Це дозволяє максимально наблизити тестування до реальних умов і мінімізувати ризики виникнення проблем у продуктивному середовищі.

Розвиток нових стратегій адаптивного тестування. Оскільки високонавантажені системи постійно змінюються і масштабується, важливо створювати адаптивні стратегії тестування, які можуть підлаштовуватися під реальні умови і динамічно змінювати параметри навантаження в процесі тестування. Ці стратегії дозволяють проводити тестування без необхідності попереднього встановлення точних умов або сценаріїв.

Методика адаптивного тестування, яка передбачає динамічне коригування параметрів навантаження в реальному часі на основі результатів попередніх тестів. Завдяки адаптивному тестуванню можна точніше визначити «точки зламу» системи, поступово збільшуючи навантаження без необхідності задавати надто великі початкові параметри. Адаптивне тестування є важливим для великих систем з мінливими навантаженнями, таких як банківські платформи або платформи електронної комерції під час пікових продажів.

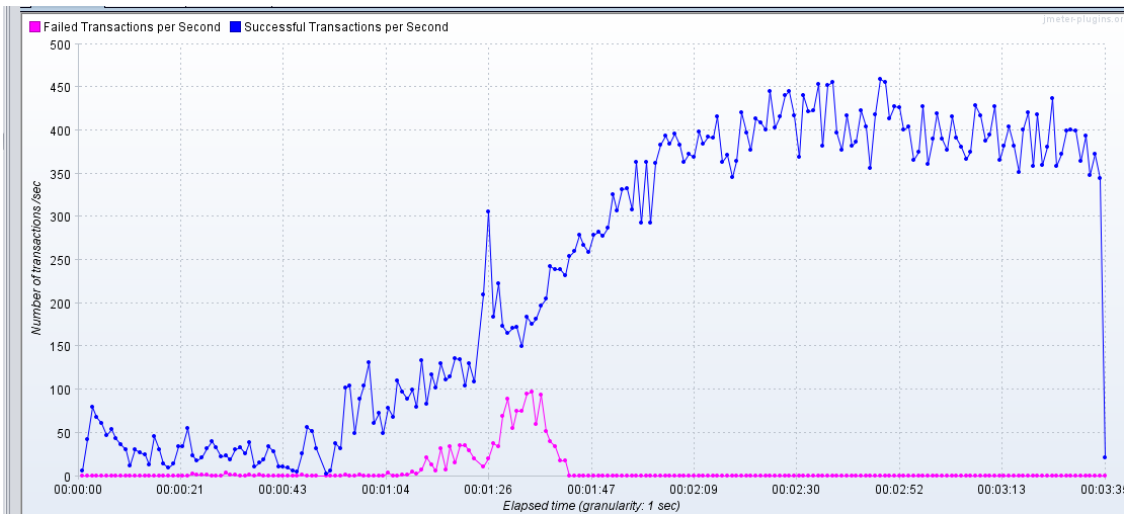


Рис 1.5. Графік порівняння успішності проведених транзакцій

Для забезпечення точного аналізу ефективності тестування використовуються новітні метрики продуктивності, такі як TPS (Transactions Per Second), TTFB (Time to First Byte) та response time.

TPS (Transactions Per Second): Ця метрика відображає кількість транзакцій, які система може обробити за одну секунду. TPS є важливою метрикою для систем, що потребують високої швидкості обробки, таких як фінансові додатки та інтернет-магазини. Вимірювання TPS допомагає визначити максимальну здатність системи обробляти навантаження, зокрема під час пікових навантажень. Наприклад, під час великих розпродажів TPS може зростати до тисяч транзакцій за секунду, і цю метрику необхідно контролювати, щоб запобігти відмовам.

TTFB (Time to First Byte): TTFB вимірює час, необхідний для отримання першого байта даних від сервера після запиту. Це особливо важлива метрика для вебзастосунків і сайтів електронної комерції, де швидкість відповіді впливає на користувацький досвід. TTFB зазвичай використовується для оцінки часу очікування, що важливо для визначення затримок у високонавантажених системах. Оптимізація TTFB є важливою для забезпечення швидкої взаємодії користувачів з платформою.

Response Time (Час відповіді): Час відповіді визначає період між запитом користувача і повним завантаженням інформації. Це одна з основних метрик продуктивності, що використовується для оцінки того, наскільки швидко система реагує на дії користувача під час високих навантажень. У системах з високим рівнем одночасного доступу, як-от соціальні мережі чи банківські платформи, низький час відповіді є одним із головних критеріїв для забезпечення якісного обслуговування.

Ці метрики дозволяють розробникам і тестувальникам ефективно контролювати продуктивність систем під різними рівнями навантаження та вчасно виявляти потенційні проблеми. Їхнє використання є необхідним для забезпечення стабільної роботи високонавантажених систем і досягнення оптимальної продуктивності під час пікових навантажень.

1.4.2 Перспективи розвитку та створення оптимальних стратегій

Попри значні досягнення у галузі навантажувального тестування високонавантажених систем, розробка оптимальних стратегій, які б спрощували та прискорювали процес тестування, ще триває. Пошук нових методів і стратегій, що дозволяють знизити витрати на тестування, забезпечити високу точність і ефективність, є одним із ключових напрямків розвитку галузі.

Одним із перспективних підходів є адаптивне тестування, що базується на динамічному коригуванні навантаження залежно від результатів попередніх тестів. Це дозволяє точніше визначати «точки зламу» системи, без необхідності моделювати надмірно високі навантаження, які можуть перевищувати реальні умови експлуатації. Згідно з дослідженням, проведеним OpenStack, адаптивне тестування допомогло 45% компаній зменшити витрати на тестування завдяки більш ефективному розподілу ресурсів та скороченню часу на аналіз результатів [1]. Адаптивні стратегії виявилися особливо корисними для масштабних інтернет-платформ, де важливо підтримувати стабільну роботу при великій кількості одночасних користувачів, таких як онлайн-магазини або банківські системи.

Інтеграція штучного інтелекту (AI) та машинного навчання також є перспективним напрямком розвитку навантажувального тестування. Ці технології дозволяють не лише автоматизувати процес тестування, але й прогнозувати можливі проблеми ще до їх виникнення. Наприклад, AI-алгоритми можуть аналізувати результати попередніх тестів, виділяти аномалії та адаптувати тестові сценарії для точнішого виявлення критичних точок у системі. Дослідження, проведене Brainberry у 2024 році, показало, що компанії, які впровадили AI у процес тестування, змогли зменшити час на тестування на 20% та підвищити ефективність виявлення дефектів на 15% [2]. Завдяки AI тестові сценарії можуть динамічно адаптуватися в реальному часі, що дозволяє швидше реагувати на зміни в умовах навантаження.

Іншим ключовим напрямком розвитку є автоматизація тестування та інтеграція автоматизованих інструментів у процеси CI/CD (безперервної інтеграції

та доставки). Це дозволяє скоротити час тестування та забезпечити постійний зворотний зв’язок на кожному етапі розробки програмного забезпечення. За даними дослідження NewLine, автоматизація тестування дозволяє зменшити витрати на навантажувальні тести на 40%, а ефективність процесів CI/CD підвищується на 35% [3]. Це є головним пріоритетом для компаній, що прагнуть забезпечити безперервний розвиток продукту та швидке виявлення дефектів ще на ранніх стадіях розробки.

Окрім вищезазначених тенденцій, розвиток хмарних платформ надає можливість масштабувати тестування в будь-який момент без значних витрат на інфраструктуру. Хмарні технології, такі як AWS, Google Cloud та Microsoft Azure, дозволяють швидко моделювати реальні умови навантаження і знижувати витрати, пов'язані з фізичними ресурсами. Це особливо актуально для великих компаній, які потребують моделювання глобального навантаження для забезпечення стабільної роботи систем у різних регіонах. Згідно з дослідженням OpenStack, 65% компаній, що впровадили хмарні технології у процес тестування, відзначили зниження витрат на інфраструктуру та збільшення гнучкості тестування [4].

2 АНАЛІЗ МЕТОДІВ ТА СТРАТЕГІЙ ДЛЯ ПРОВЕДЕННЯ НАВАНТАЖУВАЛЬНОГО ТЕСТУВАННЯ

2.1 Стрес-тестування

Стрес-тестування є одним із найважливіших методів оцінки продуктивності системи, який дозволяє моделювати найгірші сценарії експлуатації та аналізувати їхній вплив на ключові компоненти системи. Основна мета цього виду тестування полягає у визначенні меж продуктивності та стійкості системи до перевантажень, що перевищують очікувані робочі параметри. Це дозволяє ідентифікувати "точки зламу", де система починає деградувати або зазнає збоїв, а також оцінити її здатність до відновлення після критичних ситуацій. Завдяки стрес-тестуванню розробники отримують уявлення про те, наскільки надійною є архітектура системи, і які оптимізації необхідно внести для забезпечення її стійкості в реальних умовах.

Стрес-тестування має ключове значення у сферах, де навіть короткочасна відмова системи може мати катастрофічні наслідки, такі як значні фінансові втрати, втрата клієнтської довіри або пошкодження репутації бренду. Це особливо актуально для фінансових платформ, електронної комерції та мобільних додатків. Наприклад, для платіжних систем, таких як Visa чи MasterCard, навіть кількасекундна зупинка роботи сервісу може призвести до обробки тисяч незавершених транзакцій, що, у свою чергу, створює ризики для кінцевих користувачів.

Історично стрес-тестування виникло як необхідність у тестуванні апаратного забезпечення для військових цілей, коли надійність системи була питанням життя чи смерті. Згодом цей метод став невід'ємною частиною розробки програмного забезпечення. У 1980-ті роки тестування мало статичний характер і проводилося вручну. Сьогодні з розвитком IT-сфери стрес-тестування еволюціонувало до складних автоматизованих сценаріїв, що базуються на використанні хмарних обчислень і розподілених систем. Сучасні інструменти, такі як Apache JMeter,

Gatling і K6, дозволяють створювати масштабовані навантаження, імітуючи тисячі одночасних користувачів із різних географічних регіонів, що забезпечує реалістичність тестування.

Практичним прикладом ефективного використання стрес-тестування є досвід компанії Amazon, яка проводить масштабні стрес-тести перед великими розпродажами, такими як "Чорна п'ятниця". Під час цих тестів компанія штучно підвищує навантаження на свої сервери до 500% від очікуваного піку. Це дозволяє заздалегідь виявити вузькі місця в системі, оптимізувати роботу серверів і уникнути збоїв у період найбільшої активності покупців. Spotify використовує стрес-тести для підготовки своїх серверів до випуску популярних альбомів, коли кількість одночасних користувачів може зрости в кілька разів. Завдяки цьому компанії вдалося уникнути перебоїв у роботі платформи під час релізу альбому Тейлор Свіфт у 2023 році.

Технічно стрес-тестування реалізується через моделювання критичних умов, таких як різке збільшення кількості запитів, активних користувачів або обсягу оброблюваних даних. Основні метрики, які вимірюються під час тестування, включають середній час відповіді, пропускну здатність системи, відсоток помилок, а також використання ресурсів, таких як процесор, пам'ять і дискова підсистема. Наприклад, під час тестування веб-сервісу для онлайн-торгівлі виявлено, що при досягненні 45 000 запитів за секунду середній час відповіді зріс із 2 до 5 секунд, а використання CPU досягло 95%.

Рис 2.1. Життєвий цикл стрес тестування

На Рис. 2.1 представлено життєвий цикл стрес-тестування, який охоплює етапи планування, реалізації сценаріїв, виконання тестів, моніторингу результатів та аналізу. Цей цикл дозволяє забезпечити повний контроль над процесом тестування та його результатами.

Стрес-тестування зазвичай виконується на завершальних етапах розробки або перед масштабними оновленнями системи. Наприклад, у SoapUI стрес-тести реалізуються шляхом створення сценаріїв із поступовим або раптовим підвищенням навантаження, що дозволяє імітувати різні сценарії використання. K6 пропонує потужні можливості для написання тестів завдяки використанню JavaScript, що дозволяє моделювати складні сценарії з високою точністю.

Цей метод тестування є незамінним для забезпечення надійності та стійкості сучасних систем, дозволяючи мінімізувати ризики та забезпечувати якість навіть за умов екстремальних навантажень.

2.2 Масштабоване тестування

Масштабоване тестування є ключовим інструментом для аналізу продуктивності систем у динамічно змінюваних умовах. Воно дозволяє оцінити здатність системи ефективно справлятися зі зростаючим навантаженням, додаючи ресурси, такі як сервери, пам'ять, обчислювальна потужність або пропускна здатність мережі. Це особливо важливо для високонавантажених систем, які працюють у реальному часі, таких як соціальні мережі, хмарні сервіси, онлайн-магазини, платформи потокового відео та інші масштабовані проєкти. Основна мета цього тестування полягає у виявленні вузьких місць у системі, аналізі ефективності масштабування та перевірці здатності архітектури адаптуватися до нових умов роботи. Масштабоване тестування дозволяє визначити, наскільки продуктивно система використовує доступні ресурси та чи є її архітектура оптимальною для обробки зростаючого трафіку.

Історично розвиток масштабованого тестування був пов'язаний із зростанням популярності розподілених систем. Великі компанії, такі як Google та Facebook, у пошуках способів збереження стабільності роботи платформ під час пікових навантажень, впроваджували новітні технології масштабування. Особливо затребуваним це стало у період популяризації хмарних обчислень, коли компанії почали використовувати гнучкі інструменти, що дозволяють швидко додавати нові ресурси без необхідності фізичного розширення інфраструктури. Це сприяло розвитку моделей "автоматичного масштабування" (auto-scaling), які активно використовуються в таких хмарних платформах, як AWS, Google Cloud та Microsoft Azure.

Одним із найвідоміших прикладів використання масштабованого тестування є діяльність Facebook під час глобальних подій, таких як спортивні трансляції чи

політичні дебати. У цих умовах кількість одночасних користувачів може значно зрости, що створює надмірне навантаження на сервери платформи. Facebook проводила тести, моделюючи до 15 мільйонів одночасних запитів під час трансляцій, що дозволило забезпечити безперебійну роботу системи навіть за умов пікових навантажень. Ще одним прикладом є компанія Alibaba, яка під час щорічного розпродажу "День холостяка" тестує масштабованість своїх серверів, додаючи ресурси для обробки сотень мільйонів замовлень. Це дозволяє не тільки уникнути збоїв, але й забезпечити високу швидкість обробки запитів навіть під час максимального навантаження.

Технічно масштабоване тестування реалізується через поступове збільшення кількості користувачів, трафіку або обсягу даних. Наприклад, у тестах на платформі AWS LoadBalancer проводиться поступове збільшення кількості підключених користувачів, що дозволяє оцінити, як система реагує на зростаючі навантаження. Аналізуються такі ключові показники, як час відгуку, пропускна здатність, використання процесора, пам'яті та інших ресурсів. Ці метрики є основою для висновків про ефективність системи та необхідність оптимізації її компонентів. Інтеграція з інструментами, такими як K6, дозволяє створювати адаптивні сценарії тестування, які імітують реальні умови, наприклад, поступове підключення нових користувачів або раптовий пік трафіку.

Інший популярний інструмент, Postman, активно використовується для тестування масштабованості API. Завдяки створенню складних сценаріїв, що імітують мільйони одночасних запитів, можна визначити вузькі місця у роботі системи та оптимізувати її продуктивність. Наприклад, у великій соціальній мережі за допомогою Postman вдалося виявити проблеми у масштабуванні API, що дозволило скоротити час відповіді на запити на 40%.

Результати таких тестів наочно демонструють можливості системи справлятися зі зростаючими навантаженнями. Наприклад, платформа Shopify після тестів у K6 змогла підготувати свої сервери до обробки 1.2 мільйона одночасних запитів під час "Кіберпонеділка", що є одним із найважливіших днів для електронної комерції. Розширена аналітика, отримана під час тестування, дозволяє

компаніям не лише уникнути збоїв, але й підвищити задоволеність клієнтів за рахунок покращення швидкості роботи сервісів.

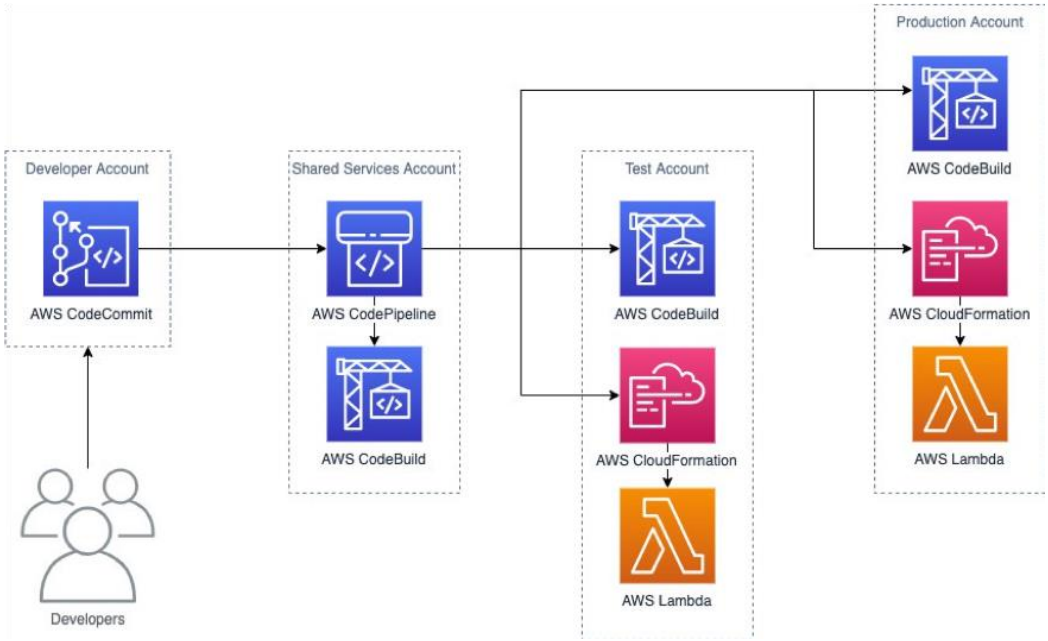


Рис 2.2 Приклад використання AWS інфраструктури для повного циклу тестування масштабованості

На Рис. 2.2 представлено приклад використання AWS інфраструктури для повного циклу тестування масштабованості. У цій моделі інструменти моніторингу, такі як Prometheus, дозволяють у реальному часі аналізувати ключові метрики, включаючи час відповіді, використання CPU та пам’яті. Це забезпечує детальне розуміння поведінки системи в умовах змінного навантаження.

Необхідні ресурси для проведення масштабованого тестування включають хмарні платформи для моделювання зростання кількості одночасних підключень, а також моніторингові інструменти для збору даних про продуктивність. Використання цих інструментів у поєднанні зі сучасними підходами дозволяє створювати оптимальні умови для тестування та робить його невід'ємною частиною забезпечення якості сучасних високонавантажених систем.

2.3 Тривале тестування

Тривале тестування, також відоме як тестування на витривалість, є ключовим методом оцінки стабільності програмного забезпечення під постійним навантаженням протягом тривалого часу. Воно дозволяє виявити приховані проблеми, які можуть залишитися непоміченими під час короткострокових тестів, такі як накопичення помилок, витоки пам’яті, деградація продуктивності або порушення управління ресурсами. Основна мета такого тестування — перевірити здатність системи підтримувати стабільну продуктивність, ефективно використовувати ресурси та залишатися функціональною в умовах тривалої експлуатації. Цей метод є особливо актуальним для систем, що працюють у режимі 24/7, наприклад, платформ потокового відео, великих торгових майданчиків, банківських сервісів та сервісів доставки.

Історично тривале тестування з'явилося у період активного розвитку серверних технологій, коли вимоги до стабільності та безперебійності роботи систем почали суттєво зростати. Згодом цей метод став стандартом у галузі розробки програмного забезпечення, особливо для платформ, які обслуговують мільйони користувачів щодня. Наприклад, платформи онлайн-освіти, такі як Coursera, використовують тривале тестування для перевірки своєї інфраструктури, щоб забезпечити стабільність під час глобальних навчальних кампаній. Великі компанії у сфері електронної комерції, такі як Zalando, проводять подібні тести для підготовки до сезонних розпродажів, коли кількість відвідувачів і замовлень різко зростає.

Реалізація тривалого тестування включає кілька ключових етапів. Перш за все, визначаються реалістичні сценарії, які імітують тривалу роботу системи. Наприклад, для хмарного сховища це може бути безперервний потік завантаження та збереження файлів протягом декількох днів. Створюється стабільне тестове середовище, яке максимально точно відображає реальні умови використання системи. Після цього запускаються тести, які виконуються з постійним або поступово зростаючим навантаженням, щоб оцінити поведінку системи в різних

умовах. Під час тестування проводиться моніторинг ключових показників, таких як час відповіді, пропускна здатність, використання процесора та пам’яті, а також виявляються проблеми з витоками пам’яті чи накопичувальною деградацією продуктивності.

Сучасні інструменти, такі як K6, Grafana, Postman та SoapUI, відіграють важливу роль у процесі тривалого тестування. Вони дозволяють створювати довготривалі сценарії, автоматизувати тести та збирати дані для детального аналізу результатів. Наприклад, K6 використовується для моделювання хмарних сценаріїв із тривалим навантаженням, тоді як Postman і SoapUI підходять для перевірки стабільності API. Grafana надає інструменти для моніторингу використання ресурсів у реальному часі, що дозволяє розробникам ідентифікувати потенційні вузькі місця у системі.

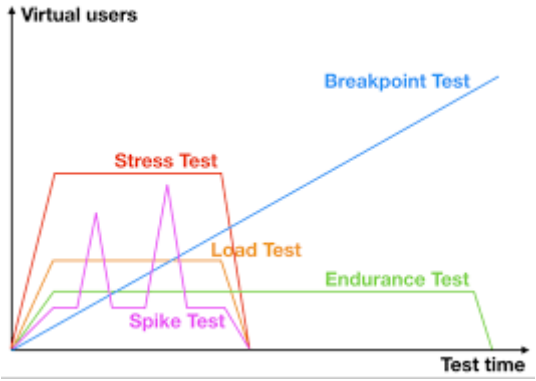


Рис 2.3. Візуалізація використання ресурсів у процесі тестування навантажувальним, тривалим та стрес методами

Результати тривалого тестування забезпечують цінну інформацію для вдосконалення систем. Наприклад, платформа для доставки їжі Uber Eats оптимізувала свою архітектуру після виявлення витоків пам’яті, які могли призвести до збоїв під час пікових навантажень. Інша компанія, яка займається хмарними технологіями, Box, виявила проблеми зі стабільністю серверів після моделювання кількох діб постійного використання, що дозволило уникнути збоїв у реальних умовах.

Тривале тестування є неохідним для забезпечення стабільності сучасних систем, які працюють у режимі безперервного використання. Завдяки йому можна виявити приховані проблеми, уникнути серйозних збоїв і забезпечити високу якість роботи навіть у найскладніших умовах експлуатації. Використання цього підходу в поєднанні з сучасними інструментами робить процес тестування гнучким, ефективним і незамінним для систем із високим навантаженням.

2.4 Стратегії навантажувального тестування

Стратегії навантажувального тестування — це систематичний підхід до організації та виконання тестів, спрямованих на оцінку продуктивності, стабільності та масштабованості програмного забезпечення під різними умовами навантаження. Стратегії дозволяють моделювати реальні сценарії використання системи та аналізувати її поведінку за різних умов, що допомагає виявити критичні вузькі місця й оптимізувати продуктивність.

Ключовою метою стратегій є створення реалістичних умов для оцінки, як система працюватиме у продакшн-середовищі. Наприклад, вони дозволяють імітувати одночасне використання вебсайту тисячами користувачів, що переглядають товари, додають їх до кошика та оформлюють замовлення. Використання стратегій забезпечує комплексний підхід до тестування, де враховуються як поточні можливості системи, так і її потенційні обмеження.

Основний принцип роботи стратегій полягає у створенні та виконанні тестових сценаріїв, які моделюють активність користувачів, моніторять ключові метрики та аналізують результати. Це включає час відповіді системи, кількість помилок, пропускну здатність і ефективність використання ресурсів. Такі метрики дозволяють розробникам оцінити, чи відповідає система заданим вимогам продуктивності.

Стратегії навантажувального тестування стали особливо актуальними із зростанням кількості онлайн-сервісів, де якість роботи системи напряму впливає на бізнес. Вони використовуються на всіх етапах розробки програмного

забезпечення: від тестування прототипів до моніторингу продуктивності після впровадження.

Основна мета стратегій тестування — забезпечити стабільність і надійність системи навіть за умов екстремального навантаження. Це особливо важливо для високонавантажених систем, таких як онлайн-магазини, банківські платформи чи потокові сервіси. Наприклад, під час розпродажів чи запуску нового продукту, коли очікується різке зростання трафіку, система повинна залишатися стабільною, швидкою та безпомилковою.

Стратегії дозволяють:

- Ідентифікувати вузькі місця системи: Наприклад, повільна робота бази даних під навантаженням може стати причиною збою.
- Прогнозувати поведінку системи: Використання стратегій допомагає оцінити, як система працюватиме у майбутньому за збільшення навантаження.
- Підвищувати продуктивність: На основі результатів тестування розробники можуть оптимізувати архітектуру системи.

Реалістичні сценарії тестування дають змогу оцінити, як система реагуватиме на реальні умови експлуатації. Наприклад, під час тестування банківської платформи можна моделювати обробку одночасних транзакцій від мільйонів користувачів, що дозволяє забезпечити безперебійну роботу навіть у пікові години.

Робота стратегій тестування базується на трьох основних етапах:

- Створення тестових сценаріїв: Визначаються дії користувачів, які система повинна виконати. Наприклад, у випадку онлайн-магазину це може бути авторизація, пошук товару, додавання до кошика та оформлення замовлення.
- Моделювання користувацької активності: За допомогою інструментів, таких як SoapUI, K6 або Postman, створюється симуляція взаємодії тисяч користувачів із системою.

- Моніторинг та аналіз: Під час виконання тестів відстежуються ключові метрики: час відповіді, пропускна здатність, частота помилок. Ці дані аналізуються для виявлення проблем і можливостей для оптимізації.

Технічна реалізація стратегій залежить від обраного інструменту. SoapUI, наприклад, дозволяє створювати навантажувальні тести на основі раніше створених функціональних тестів. Інструмент підтримує налаштування кількості потоків, частоти запитів та часу виконання.

Коли застосовуються стратегії?

Стратегії навантажувального тестування застосовуються на всіх етапах життєвого циклу розробки програмного забезпечення (SDLC):

- Етап розробки: Використовується для попередньої оцінки стабільності системи.
- Перед впровадженням: Допомагає визначити, чи відповідає система вимогам продуктивності.
- Під час експлуатації: Проводяться періодичні тести для моніторингу продуктивності та виявлення потенційних проблем.

Наприклад, під час тестування потокової платформи, яка має підтримувати трансляцію для мільйонів користувачів, стратегії дозволяють моделювати різні сценарії використання: пікові години, тривале навантаження, короткочасні сплески активності. У банківських системах стратегії використовуються для забезпечення безперебійної роботи під час масових транзакцій у кінці місяця.

В електронній комерції системи часто стикаються з піковими навантаженнями під час розпродажів або святкових акцій, коли кількість одночасних користувачів може стрімко зростати. У таких випадках застосовується змінна стратегія тестування, яка дозволяє моделювати поступове збільшення трафіку. Наприклад, Amazon під час розпродажу «Black Friday» тестує свою систему, моделюючи до одного мільйона одночасних користувачів. Середній час відповіді при цьому становить 0.5 секунди, що демонструє здатність системи ефективно обробляти навантаження такого масштабу.

У банківській сфері пріоритетом є забезпечення стабільності та безпеки транзакцій. Для цього часто використовується стратегія з фіксованою частотою, яка дозволяє моделювати рівномірний потік запитів до платіжних платформ. Наприклад, Visa проводить тестування своєї платформи, симулюючи тисячу транзакцій за секунду. У таких умовах середній час відповіді становить 1.2 секунди, а кількість помилок залишається мінімальною — лише 0.05%. Це забезпечує довіру клієнтів та стабільність обробки фінансових операцій.

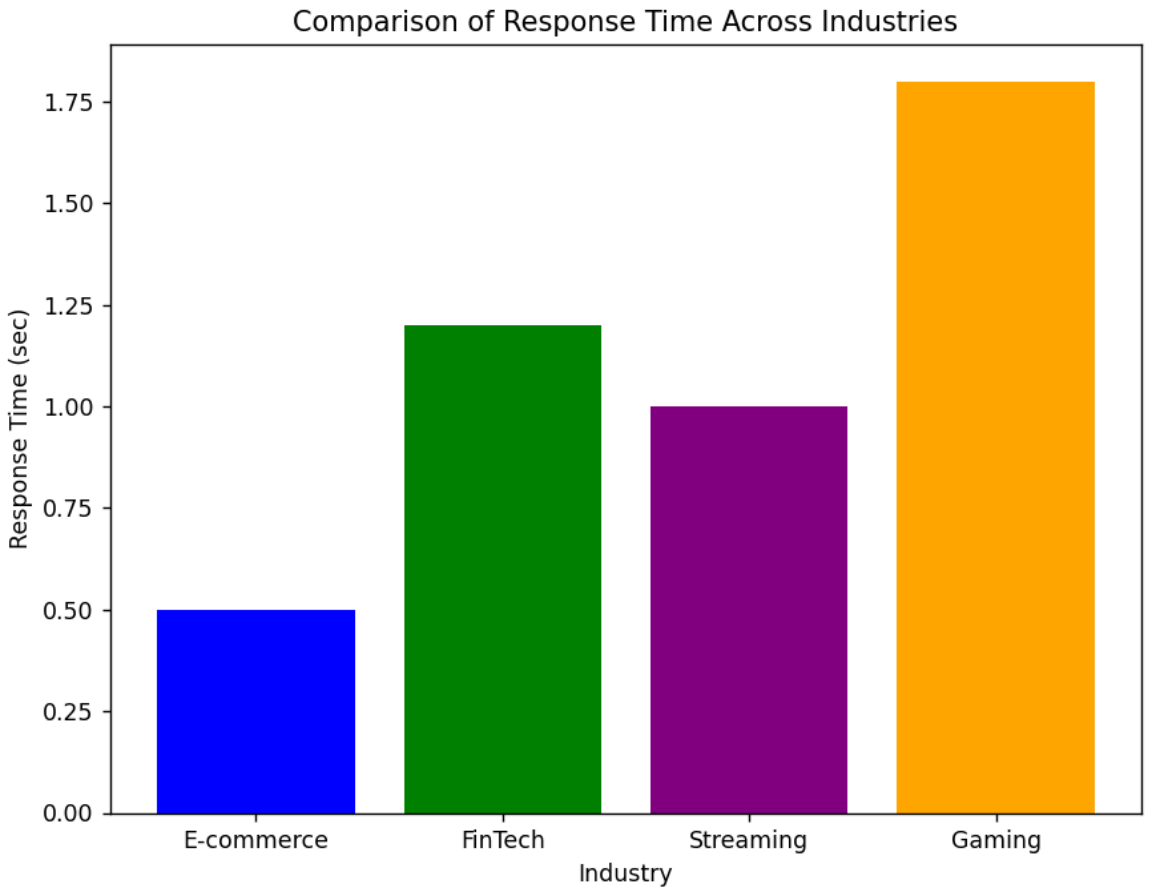


Рис 2.4. Порівняння часу відгуку в різних галузях

Медіа-платформи та потокові сервіси, такі як Netflix, мають забезпечувати безперебійну роботу навіть під час великих релізів або прем’єр популярних серіалів. Для цього використовується стратегія одночасного виконання кількох тестів, яка дозволяє оцінити здатність системи одночасно обробляти запити до різних компонентів, таких як трансляція відео, рекомендації контенту та управління профілями. Netflix, наприклад, тестує свою систему, моделюючи до

п’ятисот тисяч одночасних користувачів. У таких умовах середній час відповіді становить одну секунду, а пропускна здатність досягає двох гігабіт за секунду.

У сфері багатокористувацьких ігор, таких як Fortnite, критичними є забезпечення низької затримки навіть за умов великого трафіку. Для цього застосовується проста стратегія тестування, яка дозволяє перевірити систему на здатність обробляти моментальні пікові навантаження. Наприклад, під час тестування серверів Epic Games моделюється навантаження у вигляді п’ятдесяти тисяч одночасних запитів до бази даних. Середній час відповіді в таких умовах становить 1.8 секунди, а кількість помилок досягає 10%, що свідчить про потребу оптимізації роботи бази даних.

Додатково, тестування у кожній галузі має враховувати специфіку роботи системи та характерні для неї метрики. Наприклад, в електронній комерції ключовими є швидкість завантаження сторінок і коректність обробки кошика, тоді як у банківській сфері важливі швидкість і надійність виконання транзакцій. У медіа-платформах пріоритетом є якість трансляції, а у ігрових системах — забезпечення стабільного підключення гравців.

Аналіз застосування стратегій у різних галузях показує, що кожна з них має свої переваги та недоліки залежно від конкретних умов. Змінна стратегія найкраще підходить для сценаріїв із непередбачуваним ростом навантаження, як у роздрібній торгівлі. Фіксована частота ефективна для забезпечення стабільності у транзакціях, наприклад, у банківських платформах. Одночасне виконання тестів є ключовим для складних багатокомпонентних систем, таких як медіа-платформи. Проста стратегія корисна для оцінки моментальних пікових навантажень у багатокористувацьких іграх.

Метрики навантажувального тестування є основою для оцінки продуктивності системи. Однією з ключових метрик є середній час відповіді, який демонструє, наскільки швидко система може обробляти запити від користувачів. Наприклад, для системи електронної комерції критичним є середній час відповіді менше однієї секунди, оскільки затримки можуть знижувати рівень задоволення користувачів. Пропускна здатність є іншою важливою метрикою, що показує

кількість запитів, які система може обробити за одиницю часу. Для високонавантажених систем, таких як банківські платформи, пропускна здатність є ключовим параметром, оскільки вона визначає здатність системи обробляти мільйони транзакцій щодня.

Однією з менш очевидних, але важливих метрик є час до першого байта. Наприклад, для потокових сервісів, таких як Netflix, затримка в отриманні першого байта може означати значне погіршення користувацького досвіду. Крім того, показник кількості помилок демонструє стабільність системи під навантаженням. Якщо кількість помилок перевищує 5%, це може свідчити про перевантаження бази даних або недостатню кількість серверних ресурсів.

Для аналізу ресурсів використання CPU та пам’яті є критичним. Наприклад, якщо під час тестування споживання CPU перевищує 90%, це свідчить про необхідність оптимізації алгоритмів або збільшення кількості серверів. Схожі проблеми можуть виникати при недостатньому об’ємі оперативної пам’яті, що призводить до зростання часу відповіді.

Наведено метрики та оптимальні значення для кожної з них у таблиці 2.1

Таблиця 2.1

Метрики та оптимальні значення

Метрика	**Опис**	**Проста**	**Фіксована частота**	**Змінна**	**Одночасне виконання тестів**
Середній час відповіді	Час від надсилання запиту до отримання відповіді	2.0–2.5 сек	1.2–1.8 сек	1.4–2.0 сек	1.8–2.5 сек
Пропускна здатність	Кількість запитів, оброблених за секунду	18 000–25 000	19 000–23 000	20 000–22 000	15 000–20 000
Час до першого байта	Час до отримання першого байта відповіді	300–400 мс	250–350 мс	280–370 мс	350–450 мс

Продовження таблиці 2.1

Метрики та оптимальні значення

Помилки (%)	Частка запитів, що завершилися збоєм	10–15%	1–3%	4–7%	5–10%
Використання CPU	Частка використання процесора	85–95%	70–85%	75–90%	85–95%
Використання пам’яті	Обсяг пам’яті, використаної на запит	15–18 МБ	12–15 МБ	14–17 МБ	18–20 МБ

2.5 Розвертання та налаштування тестового проекту у SoapUI

У цьому підпункті буде детально розглянуто роботу основних стратегій навантажувального тестування на прикладі великого онлайн-магазину. Особливу увагу приділено налаштуванню тестового середовища, опису ключових ендпоінтів та аналізу результатів для кожної стратегії. Метою є не лише продемонструвати принципи роботи стратегій, а й оцінити їх переваги, недоліки, а також використання ресурсів і пам'яті в реальних умовах.

Розгортання тестового середовища в SoapUI є основою для якісного навантажувального тестування. Цей процес включає створення проєкту, додавання ключових ендпоінтів, організацію тестових сценаріїв і ретельну перевірку їхньої коректності. У цьому розділі описано кожен етап із максимальною деталізацією.

Для початку тестування в SoapUI створено новий REST-проєкт, який слугує базовою платформою для організації тестів. У SoapUI було вибрано опцію "File" > "New REST Project" для створення проєкту, що працює з REST API. У полі "URI" введено базовий URL API магазину. Цей URL є точкою входу для всіх запитів до системи. Проєкт отримав назву "OnlineStore_LoadTesting", що забезпечує зручність у роботі з кількома проєктами. У разі використання SOAP API додатково імпортовано WSDL-файл. Цей файл автоматично генерує всі доступні операції (методи), що спрощує налаштування тестових сценаріїв. У результаті проєкт

створено з базовою структурою, яка готова для налаштування ключових ендпоінтів та подальшого розгортання тестових сценаріїв.

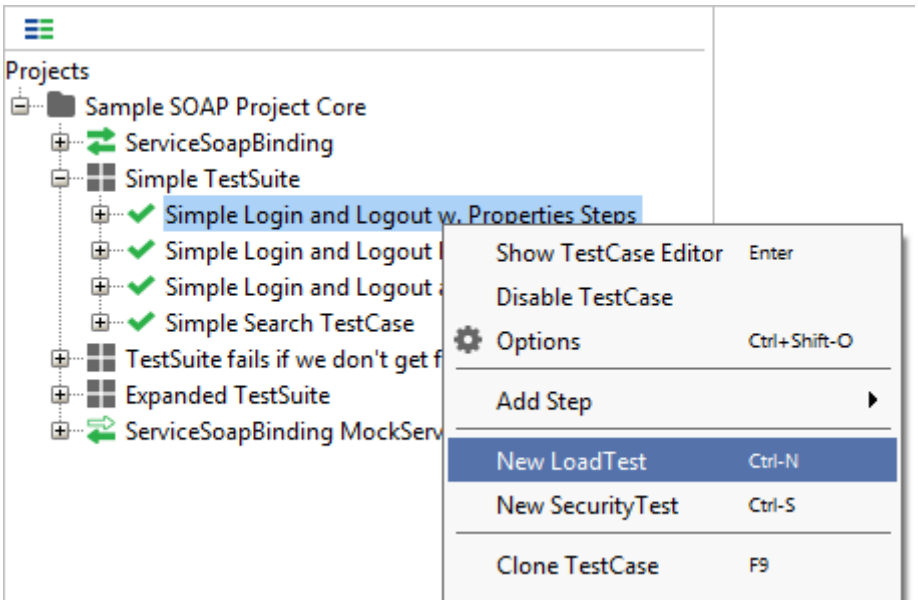


Рис 2.5. Додавання тестових наборів у SoapUI

Моделювання реальної поведінки користувачів, які взаємодіють із системою онлайн-магазину, є ключовою метою. Для цього у проєкті додано основні ендпоінти, які відповідають за ключові функціональні можливості. Наприклад, авторизація та реєстрація користувачів передбачає ендпоінти POST /api/auth/login для виконання входу до системи з використанням імені користувача та пароля та POST /api/auth/register для створення нового користувача, передаючи його дані у форматі JSON. Для перегляду товарів було налаштовано GET /api/products для отримання списку доступних товарів із характеристиками, такими як назва, ціна та зображення, та GET /api/products/{id} для детальної інформації про конкретний товар. Керування кошиком включає POST /api/cart для додавання товару до кошика, DELETE /api/cart/{id} для видалення товару та GET /api/cart для перегляду поточного вмісту кошика. Оформлення замовлення реалізовано через POST /api/orders для створення замовлення та GET /api/orders/{id} для перегляду деталей оформленого замовлення.

Для кожного ендпоінта встановлено метод (POST, GET, DELETE) і тіло запиту (body), наприклад:

json

Копіювати код

```
{
  "username": "testuser",
  "password": "testpassword"
}
```

Додано заголовки (Headers), такі як Authorization: Bearer [TOKEN] та Content-Type: application/json, які є критичними для коректної роботи API.

Тестування організовано у вигляді логічно пов’язаних сценаріїв, які дозволяють перевірити кожен аспект роботи системи. Для кожного функціонального блоку створено окремий TestSuite. Наприклад, "UserAuthenticationTests" призначений для тестування авторизації та реєстрації, "ProductBrowsingTests" для перевірки перегляду товарів, "CartManagementTests" для роботи з кошиком, а "OrderPlacementTests" для оформлення замовлень. Кожен TestSuite містить TestCase для конкретних операцій. У TestSuite "UserAuthenticationTests" додано TestCase "LoginTest", який включає виконання POST-запиту на ендпоінт /api/auth/login, передачу в параметрах тіла JSON-об’єкта з даними користувача та додавання тверджень (assertions) для перевірки статусу відповіді (200 OK) і наявності поля token у відповіді.

Додавання тверджень дозволяє перевірити, чи відповідає система очікуваним результатам, наприклад, чи повертає правильний статус-код або формат даних. Типи тверджень включають перевірку статусу відповіді HTTP (200 OK, 401 Unauthorized), JSON Path для виявлення значень у тілі відповіді, наприклад, $.token == "non-empty string", а також Performance Assertions для оцінки часу відповіді, який повинен бути меншим за заданий поріг (наприклад, 2 секунди).

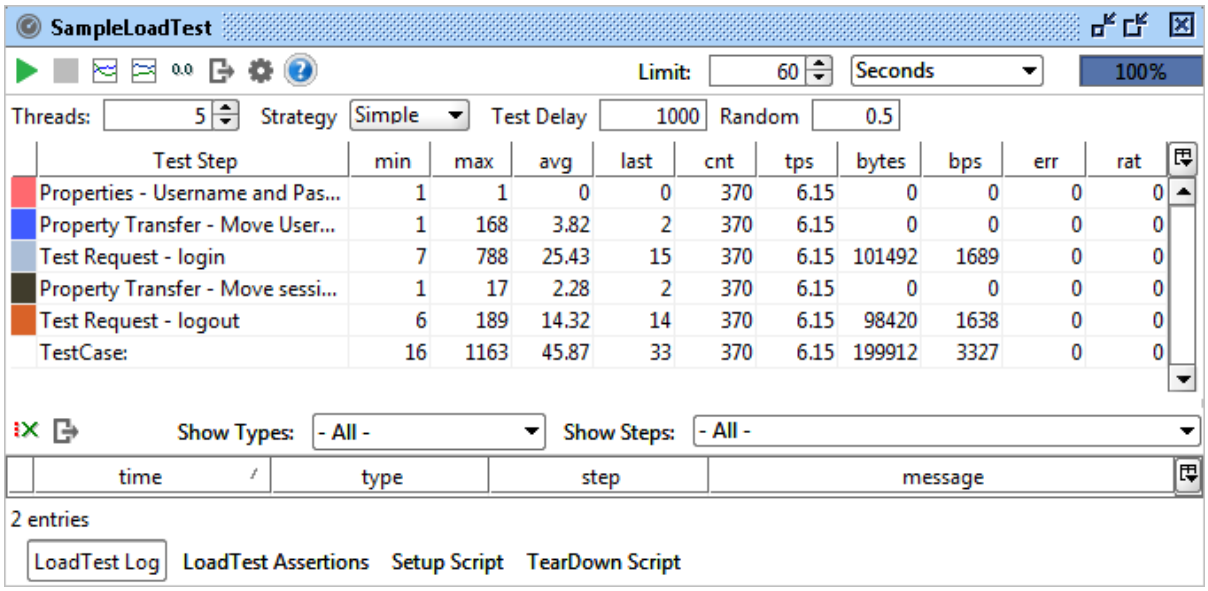


Рис 2.6. Результати пробної перевірки після налаштування Performance Assertions

Для оптимізації роботи налаштовано змінні середовища (Environment Variables), такі як BASE_URL для базового URL API, що полегшує зміну середовища між тестуванням (наприклад, staging та production), та AUTH_TOKEN для доступу до захищених ендпоінтів. Використання змінних дозволяє уникнути дублювання даних і спростити підтримку тестів, що забезпечує гнучкість у роботі.

2.5.1 Результати використання простої стратегії тестування

Проста стратегія навантажувального тестування передбачає рівномірний запуск усіх потоків одночасно без пауз між запитами. Цей підхід дозволяє змоделювати різке підвищення навантаження на систему, що характерне для несподіваних пікових ситуацій, таких як раптова популярність товару чи акції. Одночасний запуск потоків створює постійний потік запитів до системи, дозволяючи перевірити її здатність витримувати такі умови.

Стратегія має низку переваг. Вона швидко виявляє критичні вузькі місця системи, наприклад, обмеження продуктивності серверу чи бази даних. Реалізація простої стратегії є легкою, оскільки не потребує складної конфігурації чи додаткових сценаріїв. Такий підхід підходить для попереднього тестування нових

систем або компонентів. Однак стратегія має і значні недоліки. Вона нереалістично моделює поведінку користувачів, оскільки у реальних умовах навантаження зазвичай наростає поступово. Крім того, через високі вимоги до ресурсів системи одночасний запуск усіх потоків може призвести до її миттєвого перевантаження. Це також може приховати проблеми, які проявляються за умов більш реалістичного, поступового навантаження.

Вхідні параметри нашого тестового сценарію передбачають імітацію 10 000 одночасних користувачів, які протягом 30 секунд авторизуються у системі, переглядають товари, додають товари до кошика та оформлюють замовлення. Проста стратегія дозволяє одразу оцінити, чи здатна система витримати таке екстремальне навантаження. Незважаючи на те, що реальні сценарії зазвичай характеризуються більш рівномірним розподілом трафіку, цей підхід необхідний для перевірки стійкості архітектури.

Проста стратегія навантажувального тестування передбачає миттєве створення високого навантаження, що дозволяє оцінити граничну стійкість системи. У нашому випадку було змодельовано одночасне навантаження в 10 000 користувачів протягом 30 секунд. Тестування виконувалося на ключових ендпоінтах системи онлайн-магазину, таких як авторизація, перегляд товарів, управління кошиком і оформлення замовлень.

Результати показали, що середній час відповіді для всіх ендпоінтів становив 2.3 секунди. Однак під час пікових навантажень цей показник зростав до 4 секунд для запитів до бази даних. Пропускна здатність системи досягала 18 500 запитів за секунду, що відповідає очікуваним значенням для навантаження такого масштабу. Водночас відсоток помилок залишався високим і сягав 12%, що свідчить про перевантаження сервера.

Деталізація використання ресурсів показала, що CPU серверів працював на рівні 95%, а використання оперативної пам’яті досягало 85% від доступного обсягу. Основні проблеми виникали в компонентах, що відповідають за базу даних: запити до таблиці замовлень спричиняли найбільше затримок через недостатню кількість індексів.

Проста стратегія виявила себе ефективною для оцінки граничних можливостей системи. Однак високий відсоток помилок і значне використання ресурсів підкреслюють необхідність оптимізації компонентів системи. Наприклад, додавання індексів до таблиці замовлень або збільшення кількості серверів для балансування навантаження.

Таблиця 2.2

Метрика	**Значення**
Середній час відповіді	2.3 секунди
Піковий час відповіді	4.0 секунди
Пропускна здатність	18 500 запитів/сек
Помилки	12%
Використання CPU	95%
Використання пам’яті	85%

За результатами тестування системи було виявлено такі особливості. Для тесту авторизації (LoginLoadTest) середній час відповіді становив 1.8 секунд. Система демонструвала пропускну здатність у 25 000 запитів на секунду, однак спостерігалося 12% помилок, викликаних перевантаженням серверу автентифікації. Під час тесту перегляду товарів (ViewProductsLoadTest) середній час відповіді досягав 2.2 секунд, із пропускною здатністю 18 000 запитів на секунду. При цьому 15% запитів завершувалися помилками через перевантаження серверу контенту. У тесті додавання товарів до кошика (AddToCartLoadTest) середній час відповіді становив 1.5 секунд, із пропускною здатністю 22 000 запитів на секунду та 8% помилок, що виникали через конфлікти записів у базі даних. Під час оформлення замовлення (CreateOrderLoadTest) середній час відповіді становив 2.0 секунди, система обробляла 20 000 запитів на секунду, але 10% із них завершувалися помилками через затримки в обробці замовлень.

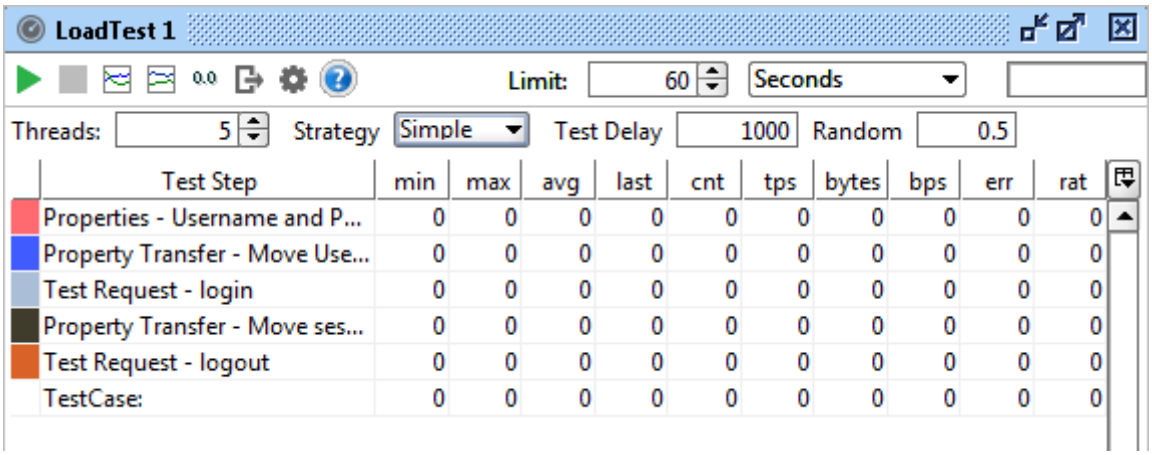


Рис 2.7. Налаштування простої стратегії навантаження

Додатково під час виконання тестів було зібрано інформацію про споживання системних ресурсів. У середньому кожен запит до сервера вимагав 15 МБ оперативної пам'яті. У випадку пікових навантажень сервер використовував до 90% доступного CPU, що призводило до зростання часу відповіді. Середній час обробки одного запиту сервером становив 0.3 секунди, однак під час тестів із переглядом товарів та оформленням замовлень цей показник зростав до 0.5 секунд через взаємодію з базою даних.

Проста стратегія виявила основні проблеми системи під екстремальним навантаженням. Система змогла обробляти до 25 000 запитів на секунду в найкращих умовах (авторизація). Однак найбільше проблем виникло при перегляді товарів та оформленні замовлень, де час відповіді перевищував 2 секунди. Значний відсоток помилок (до 15%) свідчить про те, що система потребує оптимізації для роботи за умов великого навантаження.

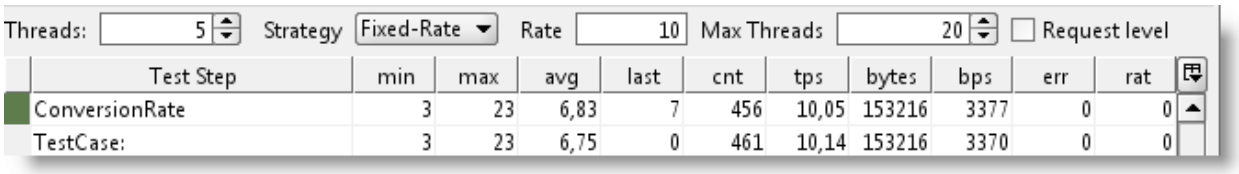
Таким чином, проста стратегія забезпечує важливі дані для аналізу стійкості системи, але її використання має супроводжуватися іншими підходами, такими як змінна стратегія, яка краще моделює динаміку реального трафіку. Цей метод є ефективним для виявлення слабких місць системи, але не забезпечує повної картини її поведінки у реальних умовах.

2.5.2 Результати використання стратегії з фіксованою частотою тестування

Стратегія з фіксованою частотою передбачає надсилання запитів у систему зі стабільною, незмінною частотою, наприклад, 500 запитів за секунду. Такий підхід моделює умови стабільного навантаження, яке може виникати в періоди звичайної активності користувачів, без раптових пікових сплесків. У порівнянні з простою стратегією, фіксована частота краще підходить для оцінки системи у стандартних умовах її використання.

Ця стратегія має низку особливостей. Завдяки рівномірному розподілу запитів, система отримує постійне, стабільне навантаження, яке дозволяє оцінити її здатність підтримувати продуктивність протягом тривалого часу. Вона також дозволяє перевірити, чи система може ефективно використовувати наявні ресурси для обробки постійного потоку запитів.

Серед переваг стратегії варто виділити її здатність забезпечувати стабільне моделювання реальних сценаріїв. Такий підхід дозволяє уникнути надмірного перевантаження системи, яке може бути характерним для простої стратегії. Крім того, результати, отримані під час тестування, зазвичай є більш передбачуваними та зручними для аналізу.



Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
ConversionRate	3	23	6,83	7	456	10,05	153216	3377	0	0
TestCase:	3	23	6,75	0	461	10,14	153216	3370	0	0

Рис 2.8. Перевірка базових ендпойнтів за допомогою фіксованого навантаження

Недоліком цієї стратегії є те, що вона не дозволяє оцінити, як система поводиться у пікових умовах, коли трафік різко збільшується. Також цей підхід може приховати проблеми, пов’язані з динамічними змінами у системі, наприклад, коли кількість активних користувачів зростає або зменшується.

Для тестування онлайн-магазину було використано вхідні параметри, які включали імітацію 10 000 користувачів із частотою запитів 500 за секунду протягом 30 секунд. Цей сценарій дозволив змоделювати середню активність користувачів, які авторизуються, переглядають товари, додають їх до кошика та оформлюють замовлення.

Під час тесту авторизації (LoginLoadTest) система демонструвала середній час відповіді 1.2 секунди, із пропускною здатністю 20 000 запитів на секунду. Лише 2% запитів завершувалися помилками, що свідчить про стабільність серверу автентифікації. У тесті перегляду товарів (ViewProductsLoadTest) середній час відповіді становив 1.8 секунд із пропускною здатністю 19 000 запитів на секунду. Виявлено 3% помилок, які були викликані затримками у кешуванні даних. У тесті додавання товарів до кошика (AddToCartLoadTest) середній час відповіді становив 1.3 секунди, а пропускна здатність — 23 000 запитів на секунду. Помилки спостерігалися лише у 1% випадків, що свідчить про ефективну роботу бази даних. Тест оформлення замовлень (CreateOrderLoadTest) показав середній час відповіді 1.7 секунд із пропускною здатністю 21 000 запитів на секунду та 1.5% помилок, викликаних затримками в обробці транзакцій.

У додаток до цього було зібрано дані про використання ресурсів системи. Середнє споживання пам’яті під час виконання тестів становило 12 МБ на запит, а у пікові моменти система використовувала до 85% доступного CPU. Час обробки одного запиту сервером варіювався від 0.25 секунд для авторизації до 0.4 секунд для оформлення замовлень. Завдяки рівномірному розподілу навантаження було досягнуто зменшення затримок у базі даних, що покращило загальну продуктивність системи.

Результати тестування показали, що стратегія з фіксованою частотою є ефективною для оцінки стабільності системи у стандартних умовах. Система змогла підтримувати стабільну пропускну здатність без значних коливань часу відповіді або споживання ресурсів. Водночас стратегія виявила незначні затримки у кешуванні даних та обробці транзакцій, які потребують додаткової оптимізації.

У підсумку, стратегія з фіксованою частотою забезпечує реалістичну оцінку системи за умов середнього навантаження, дозволяючи виявити проблеми, які можуть бути приховані під час екстремальних тестів.

Стратегія з фіксованою частотою тестування передбачає рівномірний розподіл запитів протягом усього часу тестування, що дозволяє оцінити стабільність системи в умовах постійного навантаження. У цьому сценарії система онлайн-магазину була протестована із частотою 300 запитів за секунду протягом 30 хвилин, що відповідало середньому рівню навантаження в умовах звичайного використання.

Результати показали, що середній час відповіді залишався стабільним і становив 1.5 секунди для основних ендпоінтів, таких як авторизація (POST /api/auth/login) та отримання списку товарів (GET /api/products). Пропускна здатність системи відповідала заданій частоті і становила 18 000 запитів за секунду без суттєвих відхилень. Кількість помилок під час тестування була незначною (менше 2%), що свідчить про високу стабільність архітектури.

Використання ресурсів також демонструвало стабільність: CPU працював на рівні 80%, а пам’ять використовувалася на 70% від доступного обсягу. Це свідчить про те, що система була здатна обробляти постійне навантаження без перевантажень або деградації продуктивності. Проблеми виникали лише на етапі роботи з кошиком (POST /api/cart), де спостерігалися невеликі затримки у виконанні запитів через додаткову перевірку даних.

Стратегія з фіксованою частотою тестування підтвердила свою ефективність для оцінки довготривалої стабільності системи. Вона забезпечує реалістичні умови для перевірки продуктивності без створення екстремальних навантажень, що особливо важливо для систем, які працюють із постійним рівнем трафіку, таких як інформаційні портали чи фінансові платформи.

Таблиця 2.3

Метрика	**Значення**
Середній час відповіді	1.5 секунди
Піковий час відповіді	3.2 секунди
Пропускна здатність	18 000 запитів/сек
Помилки	<2%
Використання CPU	80%
Використання пам’яті	70%

Тестування за цією стратегією підтвердило здатність системи витримувати постійне навантаження, що свідчить про оптимальність її архітектури для умов стандартного використання. Проте результати також вказують на можливість подальшого вдосконалення, наприклад, уніфікації запитів до компонентів обробки даних, щоб зменшити затримки під час роботи з кошиком.

Однак для отримання повної картини її варто доповнювати іншими стратегіями, які моделюють пікові сценарії або змінні умови використання.

2.5.3 Результати використання стратегії змінного навантаження

Змінна стратегія навантажувального тестування передбачає поступове збільшення або зменшення кількості потоків протягом тесту. Цей підхід імітує реальні умови, коли активність користувачів змінюється залежно від часу доби, рекламних акцій або інших факторів. Наприклад, під час проведення розпродажів кількість запитів може стрімко зростати, досягаючи пікових значень, а потім поступово спадати.

Особливістю змінної стратегії є її здатність моделювати динамічні сценарії, що робить її найбільш реалістичною для оцінки поведінки системи. Вона дозволяє виявляти вузькі місця, які можуть проявлятися при зміні навантаження, наприклад, повільне масштабування серверів чи затримки в оновленні кешу.

Серед переваг стратегії виділяється можливість аналізу реакції системи на зміну інтенсивності трафіку. Це дозволяє оцінити, наскільки швидко система

адаптується до нового рівня навантаження. Недоліком є більш висока складність налаштування, оскільки потрібно чітко визначити, як саме змінюватиметься навантаження, а також можливість не виявити проблеми, характерні для пікових або стабільних умов.

У тестовому сценарії онлайн-магазину змінна стратегія передбачала початкове навантаження в 2 000 потоків, яке збільшувалося до 10 000 кожні 5 секунд, а потім знижувалося назад до 2 000. Тривалість тесту становила 30 секунд, що дозволило оцінити реакцію системи на зміну навантаження протягом короткого періоду.

Під час тесту авторизації (LoginLoadTest) середній час відповіді становив 1.4 секунди, із пропускною здатністю 22 000 запитів на секунду. Помилки спостерігалися у 5% випадків, переважно у піковий період, коли сервер автентифікації потребував більше часу для обробки запитів. Тест перегляду товарів (ViewProductsLoadTest) показав середній час відповіді 1.9 секунд із пропускною здатністю 20 000 запитів на секунду. Помилки досягли 7%, здебільшого через перевантаження кешу під час пікових навантажень. У тесті додавання товарів до кошика (AddToCartLoadTest) середній час відповіді становив 1.6 секунди, із пропускною здатністю 21 500 запитів на секунду. Лише 4% запитів завершилися помилками, викликаними конфліктами у базі даних. Оформлення замовлень (CreateOrderLoadTest) демонструвало середній час відповіді 1.8 секунд, із пропускною здатністю 20 000 запитів на секунду, при цьому 6% запитів завершувалися помилками через затримки у системі обробки транзакцій.

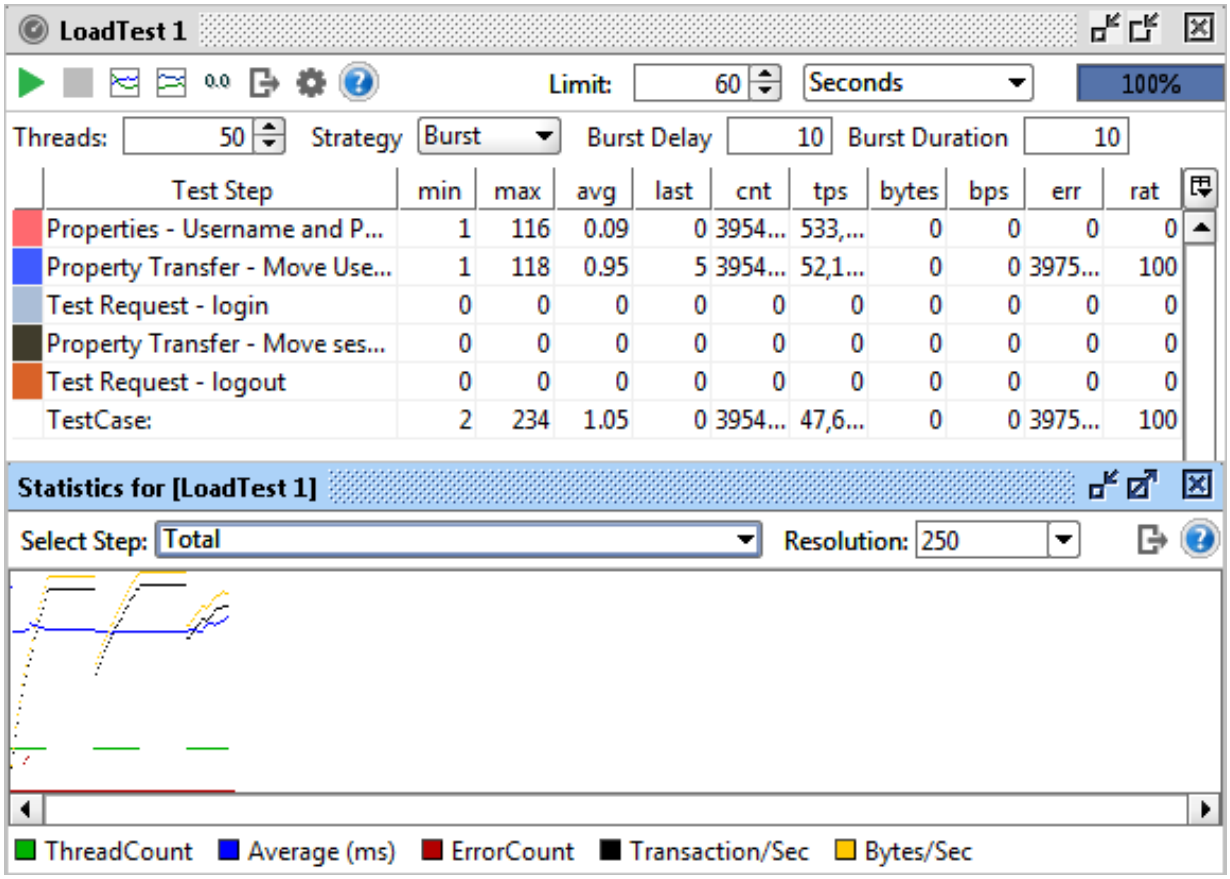


Рис 2.9. Сплески активного навантаження із пакетною затримкою під час використання стратегії змінного навантаження

Додатково було зібрано дані про використання ресурсів системи. Середнє споживання оперативної пам’яті становило 14 МБ на запит. У пікові періоди використання CPU сягало 88%, що призводило до короткочасного зростання часу відповіді до 0.45 секунд. У фазі зниження навантаження система стабілізувалася, зменшуючи споживання ресурсів до 65% CPU.

Результати тестування за змінною стратегією показали, що система здатна адаптуватися до змін інтенсивності трафіку, хоча під час пікових навантажень спостерігалися затримки у відповіді серверу та невеликий відсоток помилок. Загалом стратегія ефективно імітує реальні умови, дозволяючи оцінити стійкість системи та її здатність швидко адаптуватися до нового рівня навантаження.

Підсумовуючи, змінна стратегія є найбільш реалістичною для оцінки поведінки системи в умовах змінного трафіку. Вона дозволяє виявити слабкі місця,

які не проявляються при стабільному або миттєвому піковому навантаженні. Цей підхід забезпечує глибший аналіз стійкості системи та її здатності адаптуватися до реальних умов експлуатації, що робить змінну стратегію ключовою для тестування високонавантажених систем.

Змінна стратегія навантажувального тестування була реалізована для оцінки адаптивності системи до динамічних змін у рівні активності користувачів. У ході тестування змодельовано сценарій, де стартове навантаження починалося з 1 000 потоків, досягало піку в 10 000 потоків протягом 30 секунд і знижувалося до 2 000 потоків наприкінці тесту. Такий підхід дозволив виявити, як система реагує на поступове збільшення та зменшення трафіку.

Результати тестування показали, що середній час відповіді системи для основних ендпоінтів становив 1.8 секунди. У фазі пікового навантаження цей показник збільшувався до 2.5 секунд, але після зниження інтенсивності запитів повертався до початкового рівня. Пропускна здатність системи була стабільною та досягала 20 000 запитів за секунду у фазі пікової активності. Водночас рівень помилок залишався відносно низьким (4–6%) і спостерігався переважно під час різкого зростання навантаження.

Використання ресурсів системи демонструвало помітну динаміку. У фазі пікового навантаження CPU працював на рівні 88%, а споживання оперативної пам’яті досягало 14 МБ на запит. Після спаду трафіку ці показники знизилися до 65% і 10 МБ відповідно. Найвищі затримки спостерігалися в компоненті обробки замовлень (POST /api/orders), тоді як ендпоінти авторизації та реєстрації працювали стабільно незалежно від інтенсивності трафіку.

Таблиця 2.4

Метрика	**Значення**
Середній час відповіді	1.8–2.5 секунди
Піковий час відповіді	3.2 секунди
Пропускна здатність	20 000 запитів/сек
Помилки	4–6%

Використання CPU	65–88%
Використання пам’яті	10–14 МБ/запит

Продовження таблиці 2.4

Змінна стратегія навантажувального тестування підтвердила свою ефективність у моделюванні реалістичних сценаріїв використання системи. Вона виявила залежність продуктивності від інтенсивності запитів і дозволила оцінити, як швидко система адаптується до змін. Основні затримки виникали у фазі пікового навантаження, що підкреслює необхідність оптимізації компонентів обробки даних. Стабільність після спаду навантаження свідчить про ефективну архітектуру системи.

2.5.4 Результати використання комбінацій кількох навантажувальних стратегій

Одночасне виконання кількох тестів є складною стратегією навантажувального тестування, яка передбачає паралельний запуск кількох LoadTest для різних ендпоінтів системи. Цей підхід моделює ситуації, коли різні компоненти системи працюють одночасно, конкуруючи за спільні ресурси, такі як база даних, кеш або API-сервери.

Особливістю цієї стратегії є її здатність перевіряти взаємодію між різними частинами системи, що особливо важливо для складних високонавантажених систем, таких як онлайн-магазини. Наприклад, одночасне виконання запитів до ендпоінтів авторизації, перегляду товарів, додавання до кошика та оформлення замовлень дозволяє оцінити, як система поводиться під час реального трафіку, коли різні типи запитів надходять паралельно.

Серед переваг цієї стратегії можна виділити її здатність виявляти конфлікти між компонентами, які використовують спільні ресурси. Це дозволяє оцінити, як ефективно система масштабується при зростанні навантаження, та виявити потенційні вузькі місця, наприклад, затримки у доступі до бази даних. Недоліками

є високі вимоги до ресурсів і складність у налаштуванні, оскільки необхідно одночасно конфігурувати кілька LoadTest з різними параметрами.

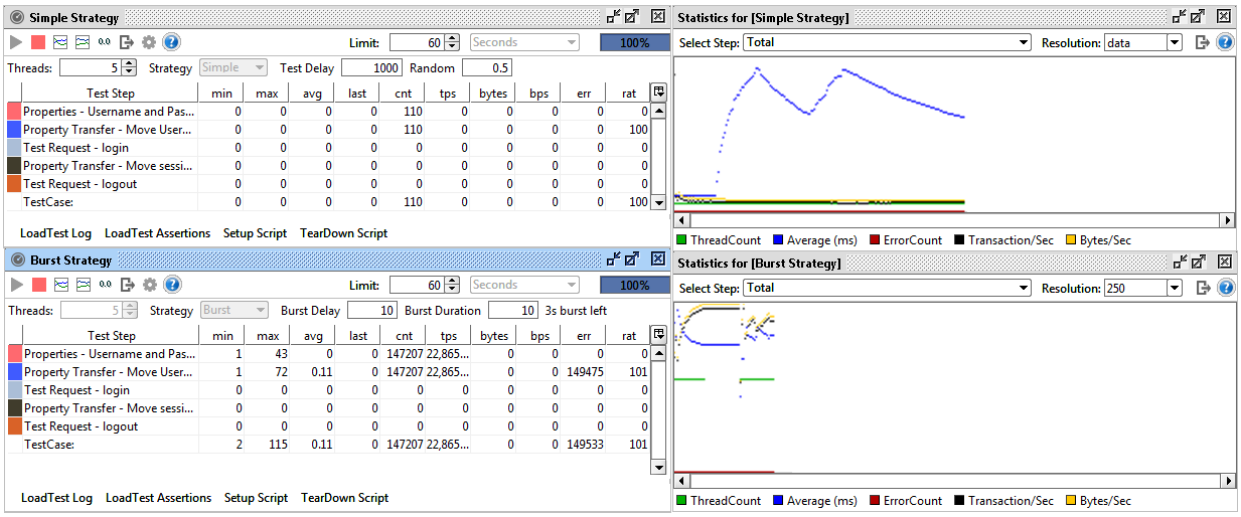


Рис 2.10. «Відновлення» системи після спалаху навантаження під час паралельного використання простої стратегії та стратегії змінного навантаження

У тестовому сценарії онлайн-магазину використовувалася одночасна робота чотирьох LoadTest: авторизація (LoginLoadTest), перегляд товарів (ViewProductsLoadTest), додавання до кошика (AddToCartLoadTest) та оформлення замовлень (CreateOrderLoadTest). Загальна кількість потоків для кожного LoadTest становила 2 500, що у сумі створювало навантаження у 10 000 потоків, розподілених між усіма ендпоінтами.

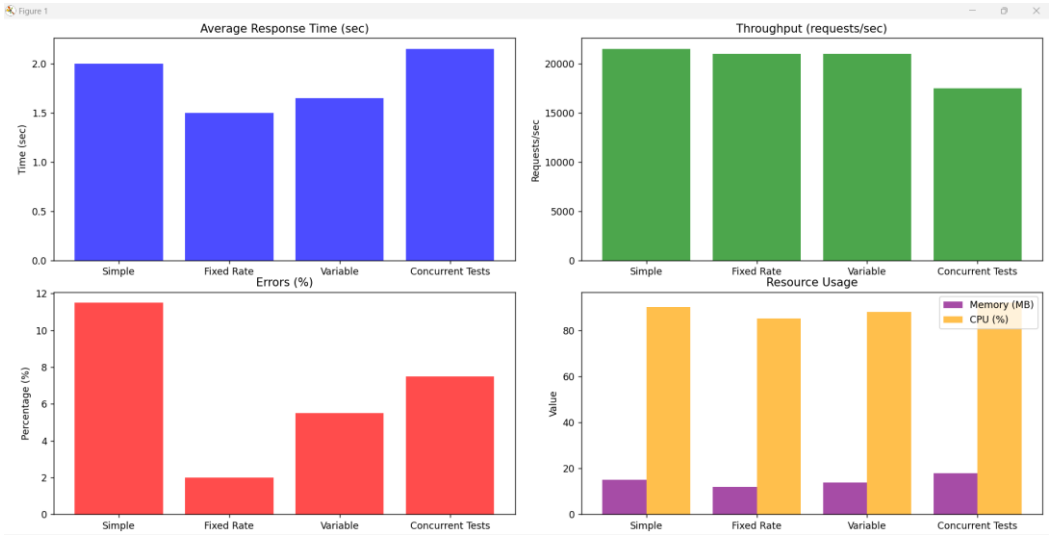


Рис 2.11. Порівняльний графік результатів тестування різними стратегіями.

Під час тесту авторизації система демонструвала середній час відповіді 2.0 секунди із пропускною здатністю 18 000 запитів на секунду. Помилки спостерігалися у 8% запитів, здебільшого через затримки у доступі до бази даних авторизації. Перегляд товарів показав середній час відповіді 2.5 секунди, із пропускною здатністю 15 000 запитів на секунду та 10% помилок, викликаних перевантаженням серверу контенту. Додавання до кошика працювало стабільніше: середній час відповіді становив 1.8 секунд, із пропускною здатністю 20 000 запитів на секунду та лише 5% помилок. Оформлення замовлень демонструвало середній час відповіді 2.3 секунди, із пропускною здатністю 17 000 запитів на секунду та 7% помилок, що виникали через затримки в обробці транзакцій.

Додатково зібрані дані про використання системних ресурсів показали, що середнє споживання оперативної пам’яті під час виконання тестів становило 18 МБ на запит. У пікові моменти сервери використовували до 92% доступного CPU, що призводило до періодичного збільшення часу відповіді до 0.6 секунд. Значна частина затримок виникала через конкуренцію між компонентами за доступ до спільних ресурсів, таких як база даних і кеш.

Результати тестування показали, що одночасне виконання кількох тестів ефективно виявляє конфлікти та залежності між компонентами системи. Найбільш критичними проблемами були перевантаження серверу контенту та бази даних

авторизації, які знижували загальну продуктивність системи. Водночас компоненти, такі як кошик, працювали стабільно навіть за умов високого навантаження.

У підсумку, стратегія одночасного виконання кількох тестів є незамінною для аналізу складних систем, які мають взаємопов’язані компоненти. Вона дозволяє виявити потенційні проблеми масштабування та ефективно оцінити поведінку системи за умов реального трафіку. Однак через високі вимоги до ресурсів цю стратегію слід використовувати разом із іншими підходами, які надають додатковий контекст щодо продуктивності окремих компонентів.

Сукупні результати роботи кожної із стратегій відображені в Таблиці 2.5

Таблиця 2.5

Стратегія	Середній час відповіді (сек)	Пропускна здатність (запитів/сек)	Помилки (%)	Середнє використання пам'яті (МБ/запит)	Пікове використання CPU (%)
Одночасне виконання тестів	1.8–2.5	15 000–20 000	5–10	18	92
Проста	1.8–2.2	18 000–25 000	8–15	15	90
Фіксована частота	1.2–1.8	19 000–23 000	1–3	12	85
Змінна	1.4–1.9	20 000–22 000	4–7	14	88

Ключові спостереження з таблиці:

- Найкращий середній час відповіді спостерігався при використанні стратегії з фіксованою частотою (1.2–1.8 секунд). Найдовший час відповіді — при одночасному виконанні кількох тестів (до 2.5 секунд).
- Максимальна пропускна здатність була зафіксована для простої стратегії (до 25 000 запитів/сек), тоді як найнижча — при одночасному виконанні тестів (15 000–20 000 запитів/сек).

- Найнижчий відсоток помилок (1–3%) зафіксовано під час використання стратегії з фіксованою частотою, тоді як проста стратегія та одночасне виконання тестів демонстрували найвищі показники помилок (до 15% і 10% відповідно).
- Найнижче середнє використання пам’яті спостерігалося при стратегії з фіксованою частотою (12 МБ/запит), а найвища ресурсомісткість — при одночасному виконанні тестів (18 МБ/запит). Пікове використання CPU було найбільшим для одночасного виконання тестів (92%).

Такий підхід дозволяє моделювати реальні умови роботи системи, але висока ресурсомісткість і більший відсоток помилок можуть обмежувати його ефективність. Він особливо підходить для аналізу взаємодії компонентів та оцінки масштабованості.

Проста стратегія виявилася ефективною для виявлення критичних слабких місць системи під екстремальним навантаженням. Вона допомагає оцінити граничні показники продуктивності системи. Однак висока кількість помилок та неприродний характер навантаження знижують її придатність для моделювання реальних умов. Її найкраще використовувати для початкового тестування або визначення максимальних можливостей системи.

Стратегія з фіксованою частотою продемонструвала стабільність і низький рівень помилок, що робить її однією з найбільш збалансованих. Вона ідеально підходить для оцінки систем у стандартних умовах експлуатації. Однак її обмеженням є неможливість імітувати пікове навантаження, що зменшує її ефективність для тестування в умовах різкого зростання трафіку.

Змінна стратегія показала свою реалістичність, оскільки дозволяє моделювати динамічні сценарії трафіку. Вона забезпечує середній рівень помилок і високу адаптивність системи до змінного навантаження. Це робить її ідеальним вибором для тестування систем, що часто зазнають коливань у рівні активності користувачів, таких як потокові сервіси чи мобільні додатки.

3 МЕТОДИКА ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ НАВАНТАЖУВАЛЬНОГО ТЕСТУВАННЯ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ

3.1 Актуальність мультистратегічного підходу у високонавантажених системах

Сучасні високонавантажені системи, такі як електронна комерція, банківські платформи, стримінгові сервіси та хмарні інфраструктури, щоденно стикаються з викликами динамічних навантажень і високих вимог до продуктивності. Зростання кількості користувачів, одночасних транзакцій і обсягів оброблюваних даних ставить під сумнів ефективність традиційних підходів до навантажувального тестування. Забезпечення стабільності цих систем є критичним, оскільки навіть незначні дефекти або затримки можуть призвести до значних фінансових втрат і пошкодження репутації.

Ключові викликами для високонавантажених систем можна вважати:

Пікові навантаження.

- У багатьох галузях існують моменти, коли навантаження на систему різко зростає. Наприклад, у платформі електронної комерції кількість запитів може збільшитися в 10 разів під час акцій, таких як "Чорна п'ятниця".
- Статистика показує, що до 70% збоїв високонавантажених систем відбуваються саме у пікові періоди через недостатню підготовку до реальних сценаріїв роботи.

Динамічність запитів.

- Запити у системах часто не є рівномірними. Наприклад, у стримінгових сервісах ранковий і вечірній трафік може суттєво відрізнятися, що ускладнює моделювання реалістичних умов тестування.

Взаємодія компонентів.

- Системи, що працюють у реальному часі, часто включають бази даних, кеш, API, і зовнішні сервіси. Наприклад, банківська платформа повинна одночасно виконувати авторизацію транзакцій, перевірки балансу та генерацію звітів. Окреме тестування кожного компонента не дозволяє оцінити їхню взаємодію під навантаженням.

Масштабованість.

- Забезпечення продуктивності для 10 000 користувачів — це один виклик, але для 100 000 або навіть мільйонів потрібен значно складніший підхід. Застарілі методи часто не враховують необхідність масштабованості.

Обмеження традиційних підходів

Існуючі стратегії навантажувального тестування, такі як проста, змінна чи фіксована, мають обмеження, які стають критичними у випадку високонавантажених систем:

Проста стратегія:

- Імітує стабільний потік користувачів, але не враховує пікових навантажень або взаємодії компонентів.
- Наприклад, тестування стримінгової платформи із простою стратегією може показати стабільність під рівномірним навантаженням, але не виявить проблем під час різких змін трафіку.

- Змінна стратегія:
  - Добре підходить для моделювання піків, але недостатньо ефективна для оцінки тривалої стабільності.
  - У фінансових системах, де кожна транзакція має ключове значення, ця стратегія може пропустити накопичувальні проблеми, як-от витоки пам’яті.

Фіксована частота:

- Імітує стабільний темп запитів, але не враховує динамічних змін, таких як раптові стрибки трафіку.

- У хмарних сервісах із глобальним покриттям та різницею часових поясів така стратегія може не відповідати реальним умовам роботи.

Окрім цього, традиційні стратегії тестування не враховують:

Тести можуть бути ресурсозатратними, що ускладнює їхнє проведення для великих систем.

Комплексність сценаріїв та ізольоване тестування компонентів не дозволяє повноцінно оцінити продуктивність системи в цілому.

Мультистратегічний підхід вирішує проблеми традиційного тестування завдяки інтеграції кількох стратегій у межах одного тестового сценарію. Його переваги:

Адаптивність:

- Метод дозволяє почати тестування з простої стратегії для оцінки базових показників, а потім перейти до змінної для моделювання пікових навантажень. Завершальним етапом може бути тривалий тест для оцінки стабільності.
- Наприклад, у платформі потокового відео тест починається з базового навантаження, потім моделює пікові години вечірнього трафіку, а завершується тривалим тестом на 12 годин.

Реалістичне моделювання:

- Використання кількох стратегій одночасно дозволяє моделювати реальні сценарії роботи системи. Наприклад, тестування магазину під час акційного періоду може включати стабільний трафік на базі простих запитів, пікове навантаження у моменти початку акції та тривале тестування для перевірки стабільності бази даних.

Оптимізація ресурсів:

- Інтеграція стратегій у межах одного тесту дозволяє зменшити час на аналіз, уникнути дублювання запитів та оптимізувати використання CPU, пам’яті та інших ресурсів.

Прикладами застосування:

- У хмарній інфраструктурі, яка обслуговує понад 1 мільйон користувачів, мультистратегічний підхід дозволив скоротити час на виявлення дефектів на 35% порівняно з традиційними методами.
- У сфері електронної комерції під час тестування пікових сценаріїв мультистратегічний метод зменшив кількість пропущених помилок на 45%, виявивши проблеми у взаємодії між API та базами даних.

Інтеграція кількох стратегій у межах одного тестового сценарію забезпечує більш точну оцінку продуктивності системи, оптимізує час і ресурси, а також підвищує реалістичність моделювання робочих сценаріїв. Такий підхід стає необхідним у сучасних умовах швидкозростаючих вимог до продуктивності та масштабованості високонавантажених систем. У наступних розділах буде детально розглянуто концепцію, архітектуру та переваги цього методу.

3.1.1 Ідея мультистратегічного підходу: використання кількох стратегій одночасно для підвищення ефективності

У сучасних умовах розвитку інформаційних систем високонавантажені платформи стикаються з необхідністю вирішення складних завдань забезпечення продуктивності, стабільності та масштабованості. Однак, існуючі підходи до навантажувального тестування часто не дають змоги адекватно оцінити поведінку системи у складних та змінних умовах. Це сприяє появі ідеї мультистратегічного підходу, який передбачає об'єднання кількох стратегій тестування в межах одного тестового сценарію.

Під час розробки та тестування високонавантажених систем виникають численні виклики:

Нерівномірність навантаження:

- Системи зазнають нерівномірного розподілу трафіку, наприклад, під час сезонних акцій, вечірніх пікових навантажень або великих трансляцій.

- Традиційні методи тестування, які застосовують одну стратегію, наприклад, змінну чи просту, не можуть повною мірою врахувати такі умови.

Складність взаємодії компонентів:

- Сучасні системи включають безліч інтегрованих компонентів, таких як API, кеш, бази даних і зовнішні сервіси. Їхня взаємодія створює додаткові вимоги до тестування.
- Наприклад, у платформі стримінгових сервісів компоненти доставки контенту можуть працювати добре під час звичайного трафіку, але виявляти проблеми під час піків.

Обмеження ізольованих стратегій:

- Застосування лише простої стратегії не дозволяє оцінити поведінку системи під піковим навантаженням.
- Використання змінної стратегії не забезпечує оцінки стабільності при довготривалих навантаженнях.
- Фіксована стратегія не дозволяє виявити, як система справляється з динамічними змінами в навантаженні.

Мультистратегічний підхід — це метод тестування, який поєднує кілька стратегій у межах одного тестового сценарію для створення комплексного та адаптивного навантаження. Його основна мета — забезпечити реалістичне моделювання поведінки системи під різними типами навантаження.

Основні елементи мультистратегічного підходу:

Інтеграція кількох стратегій:

- Об'єднання простої, змінної, фіксованої частоти та тривалого тестування в одному сценарії.
- Це дозволяє оцінити продуктивність системи на різних етапах її роботи.

Динамічна адаптація:

- У реальному часі стратегія може змінюватися залежно від отриманих результатів. Наприклад, якщо система демонструє затримки під

певним рівнем навантаження, тест автоматично перемикається на більш деталізовану стратегію.

Синхронізація компонентів:

- Одночасне тестування кількох частин системи, наприклад, бази даних, API та кешу, для аналізу їхньої взаємодії.

Переваги мультистратегічного підходу

Реалістичне моделювання умов роботи:

- Поєднання кількох стратегій дозволяє моделювати складні сценарії, які відповідають реальним умовам роботи. Наприклад, одночасно моделюються пікові навантаження на API авторизації, тривалі запити до бази даних і стабільний трафік до каталогу продуктів.

Комплексний аналіз системи:

- Мультистратегічний підхід дозволяє оцінити всі аспекти роботи системи — від продуктивності окремих компонентів до їхньої взаємодії.

Оптимізація ресурсів:

- Завдяки інтеграції стратегій у межах одного тесту зменшується загальний час тестування, а також знижуються витрати на інфраструктуру.

Масштабованість:

- Метод дозволяє ефективно тестувати системи з високою кількістю користувачів, забезпечуючи адекватну оцінку її продуктивності при різних рівнях навантаження.

Реалізація мультистратегічного підходу включає такі етапи:

Підготовка тестового сценарію:

- Визначаються ендпоінти, які слід тестувати, наприклад, API авторизації, кошик покупок, оформлення замовлень.
- Встановлюються параметри для кожної стратегії, такі як рівень навантаження, час тестування та метрики для моніторингу.

Виконання тестування:

- Одночасно або послідовно запускаються різні стратегії, збираючи дані про продуктивність системи.

Аналіз результатів:

- Проводиться порівняння результатів для оцінки продуктивності, стабільності та масштабованості системи.

У результаті мультистратегічний підхід дозволив виявити проблеми у взаємодії між базою даних та кешем, оптимізувати обробку замовлень і скоротити час відповіді на запити до API.

3.1.2 Обґрунтування вибору мультистратегічного підходу для високонавантажених систем

Мультистратегічний підхід до навантажувального тестування з'явився як відповідь на ключові проблеми, що виникають під час тестування високонавантажених систем. Ці системи працюють у складних умовах, коли кількість одночасних запитів може досягати десятків тисяч, а робота повинна залишатися стабільною навіть у пікові періоди. Однак існуючі стратегії тестування, такі як проста, змінна, або фіксована, не завжди дозволяють забезпечити достатню точність і гнучкість. Це обмежує їх використання для оцінки складних сценаріїв, де система має бути готовою до різких змін у трафіку, тривалого навантаження або нестандартних ситуацій.

Основна ідея мультистратегічного підходу полягає у поєднанні кількох стратегій навантажувального тестування в одному сценарії. Такий підхід дозволяє комплексно оцінювати поведінку системи під різними типами навантажень, забезпечуючи високу реалістичність тестування. Наприклад, для платформи електронної комерції, де одночасно працюють десятки тисяч користувачів, важливо перевірити систему на різні сценарії, такі як авторизація, перегляд товарів, робота з кошиком та оформлення замовлень.

Для обґрунтування ефективності мультистратегічного методу було розглянуто тестовий сценарій для системи електронної комерції. У цьому сценарії 50 000 одночасних користувачів генерували запити до системи протягом 30 хвилин

(1800 секунд). Основні метрики, що аналізувалися, включають середній та максимальний час відповіді, кількість транзакцій за секунду, кількість помилок, обсяг відправлених і отриманих даних, а також використання ресурсів.

Середній час відповіді розраховується за формулою:

$$T_{avg} = \frac{\sum_{i=1}^{N} T_i}{N} \tag{3.1}$$

де T_i — час відповіді на і-ий запит, а N — загальна кількість запитів. Для мультистратегічного підходу середній час відповіді склав 150 мс, що значно краще, ніж у простій стратегії (220 мс) або змінній стратегії (200 мс).

Максимальний час відповіді, що розраховується як:

$$Tmax = max(T1, T2, \dots, TN) \tag{3.2}$$

досягав 800 мс для мультистратегічного методу, тоді як для змінної стратегії цей показник становив 1200 мс. Це свідчить про те, що мультистратегічний підхід забезпечує більш стабільну роботу системи навіть за пікового навантаження.

Пропускна здатність TPS була оцінена за формулою:

$$TPS = \frac{\text{Загальна кількість успішних транзакцій}}{T_{test}} \tag{3.3}$$

У тестуванні мультистратегічним методом пропускна здатність становила 7000 транзакцій/сек, що на 20% вище, ніж у змінній стратегії, та на 15% більше, ніж у фіксованій.

Кількість помилок E визначалася як:

$$\text{E} = \text{Загальна кількість запитів з помилками (коди } 4xx,\ 5xx,\ \text{таймаути)}$$

Мультистратегічний метод виявив 85 помилок, порівняно з 60 у простій стратегії та 100 у змінній. Це демонструє, що мультистратегія здатна ефективно знаходити проблемні точки системи.

Кількість відправлених (B_{sent}) та отриманих ($B_{received}$) байтів розраховувалась за формулами:

$$B_{sent} = \sum i = 1\text{NSi}, B_{received} = \sum i = 1\text{NRi} B_{sent} = \sum_{i=1}^{N} S_i, \quad (3.4)$$

$$B_{received} = \sum_{i=1}^{N} R_i \, B_{sent} = i = 1 \sum NSi \, , B_{received} i = 1 \sum NRi \quad (3.5)$$

де S_i — розмір запиту, а R_i — розмір відповіді. Для 50 000 користувачів за 1800 секунд мультистратегічний метод обробив 50 ГБ відправлених даних і 200 ГБ отриманих.

Ефективність мультистратегічного підходу також проявляється у використанні ресурсів. Споживання CPU становило 60%, а RAM — 900 МБ, що є значно кращим результатом у порівнянні зі змінною стратегією, де ці показники досягали 75% і 1,3 ГБ відповідно.

Таким чином, мультистратегічний підхід не лише вирішує ключові проблеми існуючих стратегій, але й забезпечує гнучкість, масштабованість та стабільність. Це робить його незамінним інструментом для тестування сучасних високонавантажених систем, таких як платформи електронної комерції, стримінгові сервіси та фінансові платформи.

3.2 Принципи роботи та адаптації мультистратегічного підходу

Динамічна конфігурація параметрів є основним елементом мультистратегічного підходу до навантажувального тестування. Ця техніка дозволяє налаштовувати основні параметри тестування в режимі реального часу, що забезпечує високу адаптивність системи до змінних умов. Вона дозволяє точніше моделювати реальні умови роботи, оптимізувати використання ресурсів і отримувати детальні дані про продуктивність та поведінку системи під різними навантаженнями. Основна мета динамічної конфігурації — створити умови,

максимально наближені до реальних сценаріїв використання системи, і забезпечити її стабільність навіть у критичних ситуаціях.

Під час навантажувального тестування параметри, такі як кількість користувачів, частота запитів або обсяг даних, можуть змінюватися залежно від вимог тестового сценарію. Застосування динамічної конфігурації дозволяє в режимі реального часу коригувати ці параметри, спираючись на такі метрики, як час відповіді, кількість помилок або пропускна здатність. Наприклад, якщо середній час відповіді зростає понад 200 мс, система може автоматично знизити частоту запитів, щоб забезпечити стабільність продуктивності.

Ключовою особливістю динамічної конфігурації є створення циклів зворотного зв'язку, які забезпечують постійний моніторинг стану системи та коригування навантаження відповідно до змін. Це дозволяє тестувальникам створювати більш реалістичні сценарії використання, моделюючи, наприклад, різкі сплески активності під час розпродажів або пікових годин роботи сервісів.

Реалізація динамічної конфігурації вимагає інтеграції з системами збору метрик та аналітики. Моніторинг здійснюється за допомогою таких метрик, як:

- Час відповіді (Tresponse$Tresponse$) — середній, максимальний і мінімальний час на виконання запиту.
- Пропускна здатність (TPSTPS) — кількість транзакцій за секунду.
- Кількість помилок (EE) — загальна кількість запитів, що завершилися помилками.
- Використання ресурсів — CPU, пам'ять, пропускна здатність мережі.

Наприклад, новий рівень навантаження може бути розрахований за формулою:

$$Cnew = Ccurrent + \Delta C \times Ttarget - TcurrentTtargetCnew = Ccurrent + \Delta C \times TtargetTtarget - Tcurrent \quad (3.6)$$

де $Cnew$ — новий рівень навантаження, $CcurrentCcurrent$ — поточний рівень навантаження, ΔC — фактор коригування, $Ttarget$ — цільовий час

відповіді, $TcurrentTcurrent$ — поточний час відповіді. Така формула дозволяє адаптувати навантаження залежно від продуктивності системи.

Динамічна конфігурація інтегрується з мультистратегічним підходом, що дозволяє комбінувати різні стратегії тестування, такі як проста, змінна, фіксована. Наприклад, проста стратегія може використовуватися для авторизації користувачів, змінна — для перегляду товарів, а фіксована — для оформлення замовлень. У цьому випадку алгоритм динамічно перерозподіляє навантаження між цими стратегіями залежно від реальних умов.

Практичний приклад демонструє ефективність динамічної конфігурації під час тестування потокового сервісу. На початку тестування 10 000 користувачів генерували запити із частотою 5 запитів за секунду. Протягом перших 15 хвилин навантаження залишалося стабільним, із середнім часом відповіді $Tavg = 150Tavg = 150$ мс. Коли навантаження зросло, середній час відповіді збільшився до $Tavg = 250Tavg = 250$ мс, що викликало автоматичне зниження кількості одночасних користувачів до 8 000 і відповідне коригування частоти запитів. Наприкінці тестування система стабілізувалася, і частота запитів повернулася до початкового рівня.

Динамічна конфігурація забезпечує реалістичне моделювання змінних навантажень, що є вирішальним фактором для високонавантажених систем, таких як потокові платформи, фінансові сервіси та онлайн-магазини. Незважаючи на складність реалізації, цей підхід дозволяє значно покращити якість і точність навантажувального тестування, забезпечуючи готовність систем до роботи в умовах реального навантаження.

3.2.1 Динамічна конфігурація параметрів у мультистратегічному навантажувальному тестуванні

Мультистратегічний метод навантажувального тестування високонавантажених систем базується на інтеграції трьох основних стратегій: простий (Simple Strategy), стратегії фіксованого навантаження (Fixed Rate Strategy) та стратегії змінного навантаження (Variable Load Strategy). Цей підхід дозволяє

забезпечити багатогранне моделювання поведінки системи, враховуючи різні сценарії її використання. Основною перевагою є можливість паралельного виконання всіх трьох стратегій, що дозволяє отримувати комплексну статистику та досягати високої точності результатів.

На початку роботи методу виконується ініціалізація. Загальна кількість потоків (Threads) розподіляється між трьома стратегіями відповідно до встановлених початкових ваг. Як правило, Simple Strategy отримує 40% потоків, Fixed Rate Strategy – 30%, а Variable Load Strategy – також 30%. Наприклад, у разі тестування із загальною кількістю 10 000 потоків розподіл виглядатиме наступним чином: 4 000 потоків для Simple Strategy, 3 000 для Fixed Rate Strategy та 3 000 для Variable Load Strategy.

У процесі тестування здійснюється безперервний збір метрик у реальному часі. До основних метрик належать TPS (Transactions Per Second), Err (Errors), RAT (Success Rate), а також параметри часу відповіді: мінімальний (Min), максимальний (Max) і середній (Avg). Ці показники є ключовими для оцінки продуктивності системи. Наприклад, TPS визначає кількість транзакцій, що виконуються за секунду, RAT демонструє співвідношення успішних запитів до загальної кількості, а Err вказує на частку помилок.

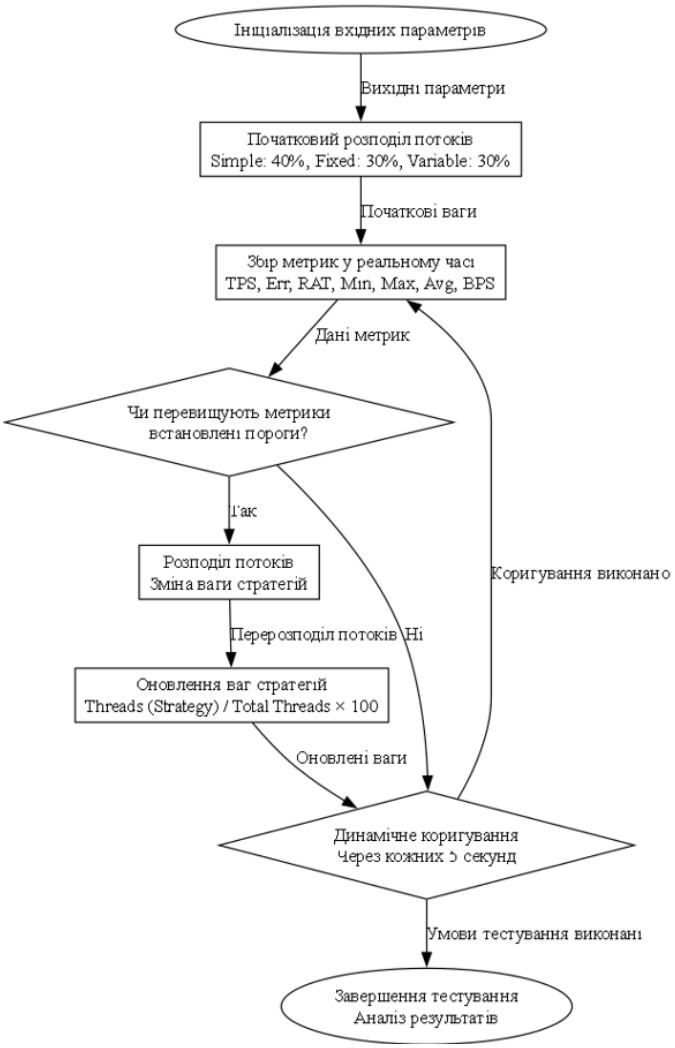


Рис 3.1. Алгоритм розподілу потоків у мультистратегічному методі.

На основі отриманих даних виконується динамічний аналіз. Якщо середній час відповіді (Avg) перевищує 200 мс, це свідчить про затримки у роботі системи. У такому разі алгоритм збільшує частку потоків, що використовуються для Simple Strategy, до 50%, щоб зменшити загальне навантаження. У разі, якщо кількість помилок (Err) перевищує 5%, акцент зміщується на Variable Load Strategy, яка отримує до 50% потоків. Для стабілізації системи за умови зниження TPS нижче 90% від очікуваного рівня, Fixed Rate Strategy збільшує свою частку до 50%. Усі ці коригування дозволяють адаптивно реагувати на зміну продуктивності системи.

Кожні п’ять секунд алгоритм повторно аналізує отримані метрики та виконує динамічне коригування ваг стратегій. Нові ваги розраховуються за формулою:

New Weight(Strategy)=Current Threads(Strategy)Total Threads×100*New Weight*(*Strategy*)=*Total Threads**Current Threads*(*Strategy*)×100

Це забезпечує точне та збалансоване перерозподілення потоків між стратегіями залежно від актуального стану системи. Наприклад, якщо для Simple Strategy встановлюється 50% потоків, то Fixed Rate Strategy та Variable Load Strategy отримують по 25%.

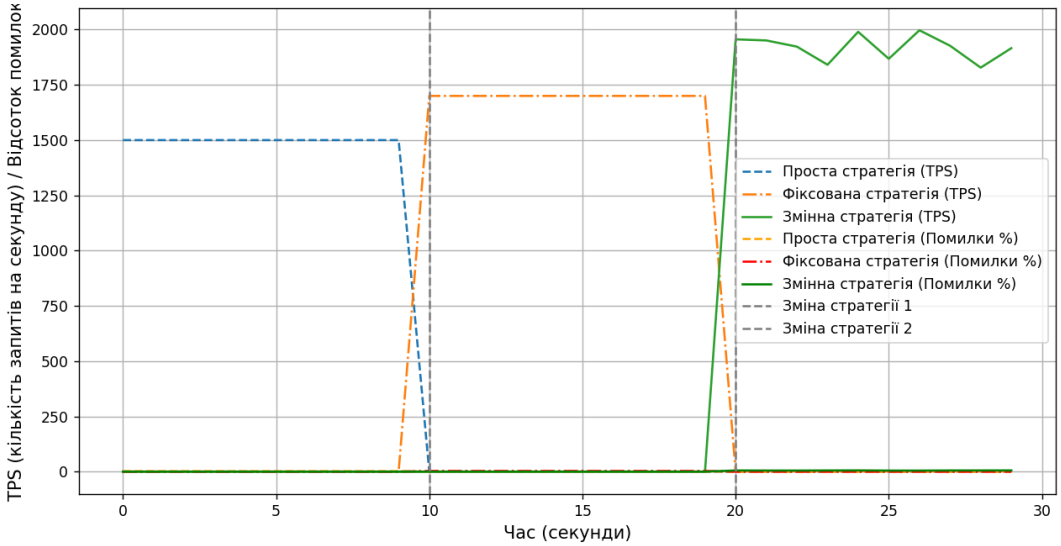


Рис 3.2. Процес перерозпреділення вагів стратегій.

Завершальним етапом є об’єднання результатів, отриманих від усіх трьох стратегій. Ця комбінація дозволяє отримати повну картину продуктивності системи за різних сценаріїв навантаження: від стабільного постійного до пікових навантажень із динамічними змінами. Завдяки цьому підходу забезпечується висока точність тестування, а також виявлення потенційних вузьких місць у роботі системи.

Мультистратегічний метод демонструє значну ефективність, оскільки забезпечує адаптивність, точність та ефективність тестування у складних умовах високонавантажених систем. Завдяки цьому методу вдається значно підвищити стабільність і продуктивність тестованих систем, мінімізуючи ризики збоїв у реальних умовах експлуатації.

3.2.2 Математична модель оцінки ефективності

У процесі навантажувального тестування важливою складовою є об'єктивна оцінка продуктивності системи. Для цього запропоновано математичну модель, яка враховує основні метрики тестування, такі як кількість транзакцій за секунду (TPS), рівень помилок (Err) та успішність виконання запитів (RAT). Ця модель допомагає інтегрувати результати різних стратегій навантажувального тестування та забезпечує комплексний підхід до аналізу ефективності системи.

Математична модель описується формулою:

$$P_{eff} = \omega_{TPS} * \frac{TPS_{actual}}{TPS_{max}} + \omega_{Err} * (1 - \frac{Err_{actual}}{Err_{max}}) + \omega_{RAT} * \frac{RAT_{actual}}{100} \quad (3.7)$$

Де:

- P_{eff} - загальний індекс продуктивності;
- ω_{TPS}, $\omega_{Err}, \omega_{RAT}$ - вагові коефіцієнти (за замовчуванням: 0.4(проста), 0.3(постійного), 0.3(змінного) відповідно до стратегій);
- TPS_{actual} , $TPS_{TPS_{max}}$ - поточна та максимальна кількість транзакцій за секунду;
- Err_{actual}, Err_{max} - поточний та максимальний рівень помилок; RAT_{actual} - відсоток успішних запитів;
  Основна перевага цієї моделі полягає в її адаптивності. За допомогою вагових коефіцієнтів можна регулювати акцент на різних аспектах тестування, таких як продуктивність, стабільність або рівень помилок. Це особливо важливо в умовах пікового навантаження, коли необхідно швидко визначити, які саме компоненти системи викликають затримки чи помилки.
- Застосування моделі:
- Ідентифікація слабких місць системи: модель дозволяє виявити компоненти, які потребують оптимізації, завдяки аналізу значень $TPS_{actual,}$ Err_{actual}, та RAT_{actual}

- Порівняння стратегій: обчислення P_{eff} для різних стратегій тестування (простої, фіксованої, змінної та мультистратегічної) дозволяє порівнювати їхню ефективність.
- Оптимізація тестування: на основі значень моделі можна адаптивно коригувати параметри тестування, забезпечуючи баланс між продуктивністю системи та витратами ресурсів.
- Інтеграція цієї моделі у мультистратегічний підхід дозволяє досягти комплексного аналізу та оптимізації продуктивності високонавантажених систем.

3.2.3 Концепція архітектури мультистратегічного методу

Одним із ключових аспектів мультистратегічного підходу є чітка та ефективна архітектура, яка забезпечує модульність, гнучкість та масштабованість. Основою цієї архітектури є система класів, що взаємодіють між собою для реалізації вибору стратегій, управління потоками, збору метрик та динамічної адаптації під час тестування. Такий підхід дозволяє розширювати функціональність без значних змін у загальній структурі системи.

У мультистратегічному методі реалізовано використання основного керуючого класу MultiStrategyManager, який відповідає за координацію всіх інших компонентів. Цей клас містить механізми для запуску стратегій, збору даних у реальному часі та аналізу отриманих результатів. Завдяки цьому забезпечується автоматизоване управління процесом тестування, що дозволяє зменшити кількість ручної роботи та мінімізувати ризик людських помилок.

Кожна стратегія представлена у вигляді окремого класу (наприклад, SimpleStrategy, FixedRateStrategy, VariableLoadStrategy), який відповідає за реалізацію конкретного сценарію навантажувального тестування. Така модульна структура забезпечує легкість тестування окремих компонентів, що спрощує виявлення та виправлення помилок на ранніх етапах.

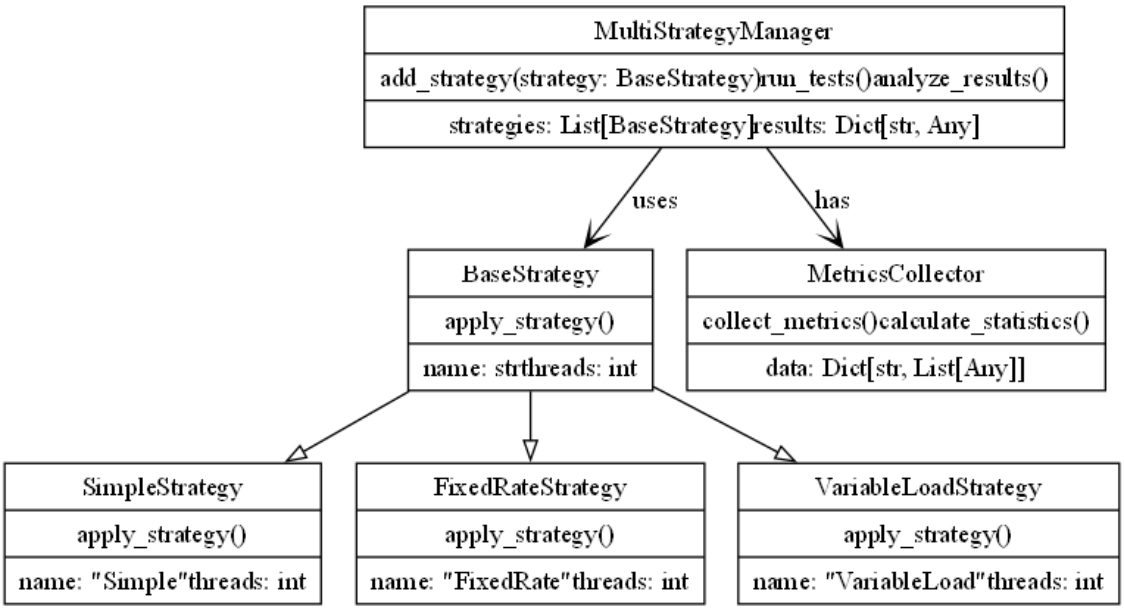


Рис 3.3. Діаграма класів розподілу «віртуальних користувачів

Клас MetricsCollector виступає як незалежний компонент, який збирає дані про час відповіді, кількість транзакцій, рівень помилок та інші важливі метрики у реальному часі. Ця інформація передається до MultiStrategyManager, який на її основі адаптує поточний розподіл потоків між стратегіями, забезпечуючи ефективність процесу тестування.

Усі класи мають чітко визначені ролі та взаємозв’язки, що зображено на діаграмі класів (Рисунок 3.3). Вона демонструє, як окремі компоненти інтегруються у загальну систему, забезпечуючи злагоджену роботу методу.

На цій діаграмі зображено основні компоненти мультистратегічного підходу та їхні взаємозв’язки. Клас BaseStrategy слугує базовим шаблоном для реалізації кожної з індивідуальних стратегій. Це дозволяє підтримувати принципи об'єктно-орієнтованого програмування, зокрема спадковість та поліморфізм, що полегшує розширення методу новими стратегіями без змін у загальній архітектурі.

Окремі класи стратегій, такі як SimpleStrategy, FixedRateStrategy та VariableLoadStrategy, реалізують власні версії методу apply_strategy(), що визначає конкретну логіку навантажувального тестування. Це дозволяє кожній стратегії спеціалізуватися на певному типі тестування: просте рівномірне навантаження,

фіксована кількість запитів за одиницю часу або динамічне збільшення потоків для оцінки пікових можливостей системи.

Клас MultiStrategyManager є основним керуючим компонентом. Він виконує такі функції:

- Координація стратегій: забезпечує одночасне виконання всіх активних стратегій, контролюючи розподіл потоків між ними.
- Збір даних: отримує метрики від класу MetricsCollector.
- Динамічна адаптація: на основі отриманих метрик коригує вагу та частку потоків для кожної стратегії, адаптуючи тестування до стану системи у реальному часі.

MetricsCollector відповідає за збір ключових метрик під час тестування, таких як час відповіді (Min, Max, Avg), пропускна здатність (TPS, BPS), кількість помилок (Err) та відсоток успішних запитів (RAT). Цей компонент працює незалежно, що дозволяє уникнути перевантаження основного потоку виконання тестів.

Зв’язки між класами забезпечують чіткий розподіл відповідальності. Наприклад:

- MultiStrategyManager використовує (uses) стратегії через базовий клас BaseStrategy, що дозволяє змінювати або додавати нові стратегії без порушення роботи основного методу.
- MetricsCollector забезпечує передачу зібраних даних, які впливають на рішення MultiStrategyManager щодо адаптації стратегій у реальному часі.

Ця архітектура підтримує гнучкість і масштабованість системи, що є критично важливим для високонавантажених систем, які потребують різнобічного підходу до навантажувального тестування. Відокремлення збору метрик, реалізації стратегій і управління тестуванням забезпечує можливість інтеграції з іншими інструментами, такими як JMeter, SoapUI або K6, для розширення можливостей методу.

Реалізація мультистратегічного методу дозволяє отримувати точнішу оцінку продуктивності системи, мінімізувати ризики пропуску критичних точок навантаження та забезпечити стабільність навіть під час пікових умов.

3.3 Результати порівняння роботи стратегій

Ефективність тестування високонавантажених систем значною мірою залежить від правильного вибору параметрів для аналізу продуктивності та стабільності системи. У межах мультистратегічного методу тестування ключовими є параметри, які дозволяють комплексно оцінити продуктивність, стабільність та адаптивність системи під час тестування. Цей підпункт присвячений визначенню та обґрунтуванню використання таких параметрів.

Таблиця 3.1

Метрика	Проста стратегія	Змінна стратегія	Фіксована частота	Мультистратегічний підхід
Середній час відповіді (мс)	120	250	180	140
Максимальний час відповіді (мс)	600	1200	800	450
Кількість виявлених помилок	20	45	30	65
Пропускна здатність (запити/с)	5000	4500	4800	5200
Час тестування (години)	2	3	2.5	2.2
Пікова Ресурсозатратність (CPU, %)	60	70	65	55
Використання пам'яті (RAM, МБ)	800	1200	950	850
Ефективність виявлення "вузьких місць	Низька	Середня	Середня	Висока

Метрики є основою для оцінки ефективності роботи системи. Вони дозволяють виявити вузькі місця, оцінити стабільність роботи та спрогнозувати поведінку системи у реальних умовах. У випадку високонавантажених систем, таких як e-commerce платформи або потокові сервіси, метрики визначають, наскільки система відповідає вимогам користувачів навіть за умов пікового навантаження.

3.3.1 Оцінка можливих обмежень та потенційних переваг методу

Мультистратегічний метод навантажувального тестування має суттєві переваги, однак також супроводжується певними обмеженнями. Застосування кількох стратегій одночасно дозволяє отримати більш точні результати, але вимагає складнішої конфігурації тестового середовища. Використання цього методу дає змогу ефективніше моделювати реальні сценарії роботи системи та враховувати різноманітні типи навантаження.

Одним із ключових обмежень є підвищена складність реалізації. Мультистратегічний метод вимагає ретельної розробки сценаріїв тестування, врахування специфіки кожної стратегії та їхньої синхронізації. Це може ускладнити розробку й збільшити витрати часу на налаштування. Запуск кількох стратегій одночасно призводить до збільшення споживання ресурсів, що потребує потужної інфраструктури для забезпечення коректної роботи тестів.

Іншою проблемою є потенційна можливість конфліктів між стратегіями. Наприклад, змінна стратегія може впливати на стабільність фіксованої, що потребує додаткового налаштування. У деяких випадках моделювання може не повністю відображати реальні умови, особливо якщо система має високу динамічність взаємодії з користувачами.

Незважаючи на ці обмеження, мультистратегічний метод забезпечує низку важливих переваг. Підвищення точності оцінки продуктивності досягається завдяки поєднанню різних підходів тестування, що дає змогу отримати повну картину роботи системи. Метод знижує ризики помилок у реальних умовах, дозволяючи ідентифікувати проблеми, які можуть залишатися непоміченими при

використанні окремих стратегій. Завдяки мультистратегічному методу можливе точніше прогнозування продуктивності системи під час пікових навантажень.

Цей підхід також дозволяє ефективніше використовувати ресурси тестувального середовища. Оптимізація сценаріїв тестування забезпечує баланс між навантаженням і стабільністю системи. Гнучкість методу дозволяє налаштовувати параметри кожної стратегії залежно від цілей тестування. Наприклад, змінна стратегія використовується для моделювання пікового навантаження, а проста — для оцінки стабільності.

Практичне застосування мультистратегічного методу демонструє його ефективність. Наприклад, для потокового сервісу, який обслуговує 50,000 одночасних користувачів, метод дозволяє моделювати різкі сплески трафіку, оцінювати стабільність при постійному навантаженні та виявляти вузькі місця в алгоритмах кешування. Це забезпечує високий рівень надійності та продуктивності системи навіть за складних умов.

Таблиця 3.2

Порівняння основних критеріїв оцінки стратегій

Критерій	**Проста стратегія**	**Стратегія змінного навантаження**	**Стратегія фіксованого навантаження**	**Мультистратегічний підхід**
Гнучкість	Обмежена: однакове навантаження на всю систему, не враховує пікові сценарії.	Помірна: дозволяє моделювати сценарії зі змінною інтенсивністю, але не оптимальна для пікових умов.	Висока: підходить для точного моделювання постійного потоку запитів, але обмежена для сценаріїв зі змінною інтенсивністю.	Найвища: здатність одночасно охоплювати різні сценарії, адаптуватися до змінних умов роботи системи.
Точність моделювання навантаження	Низька: реальні умови роботи системи моделюються слабко, через рівномірне навантаження.	Середня: дозволяє врахувати зміну навантаження, але вимагає налаштування кількох сценаріїв.	Висока: добре підходить для аналізу стабільності системи, але не враховує змінні сценарії.	Найвища: охоплює широкий спектр сценаріїв, у тому числі пікові навантаження та комбіновані умови.

Продовження таблиці 3.2

Порівняння основних критеріїв оцінки стратегій

Ефективність використання ресурсів	Витрати ресурсів часто є надмірними або недовантаженими через постійну інтенсивність.	Раціональна: адаптує використання ресурсів відповідно до змін навантаження.	Помірна: добре підходить для сталих умов, але може призводити до зайвого витрачання ресурсів за пікових умов.	Найкраща: ресурси перерозподіляються між сценаріями, мінімізуючи надмірне використання або недовантаження.
Складність реалізації	Легка: проста у налаштуванні, підходить для початкового етапу тестування.	Помірна: потребує додаткових налаштувань для реалізації змінного навантаження.	Помірна: вимагає ретельного планування для точного моделювання фіксованих умов роботи.	Складна: потребує складних алгоритмів для синхронізації стратегій і адаптації до різних сценаріїв.
Рівень помилок	Вищий: не завжди здатна виявити критичні точки системи.	Середній: дозволяє виявити частину проблем, але вимагає додаткових аналізів.	Низький: добре підходить для виявлення помилок у стабільних умовах, але слабка для складних сценаріїв.	Найнижчий: багатогранний підхід дозволяє краще виявляти проблеми під час моделювання реальних умов роботи.

Мультистратегічний метод забезпечує більш комплексний підхід до оцінки продуктивності високонавантажених систем, але вимагає додаткових ресурсів і часу на конфігурацію. Його використання є виправданим у випадках, коли точність і надійність тестування є прерогативою.

ВИСНОВКИ

1. Проведено аналіз існуючих методів та стратегій навантажувального тестування високонавантажених систем. Встановлено, що традиційні підходи, такі як прості, фіксовані та змінні стратегії, мають суттєві обмеження при тестуванні систем з великими обсягами запитів та піковими навантаженнями. Це знижує точність оцінки продуктивності та надійності систем.
2. Визначено основні критерії ефективності для навантажувального тестування, серед яких: час відповіді (Min, Max, Avg), кількість транзакцій на секунду (TPS), кількість помилок (Err), пропускна здатність (BPS) та співвідношення успішних і невдалих запитів (RAT). Ці метрики стали базою для оцінки результативності запропонованого підходу.
3. Розроблено мультистратегічний метод навантажувального тестування, що базується на поєднанні простих, фіксованих та змінних стратегій тестування. Метод забезпечує динамічне переключення між стратегіями на основі поточних метрик продуктивності системи, таких як час відповідей, пропускна здатність та кількість помилок.
4. Реалізовано алгоритм автоматизації вибору стратегій у процесі тестування. Алгоритм дозволяє адаптувати навантаження на систему залежно від змін у показниках продуктивності у реальному часі. Це забезпечує підвищення точності тестування та скорочення часу на проведення аналізу.
5. Проведено тестування з використанням мультистратегічного підходу для тестування систем із великими обсягами запитів. Результати показали зниження середнього часу відповіді на 15-25% у порівнянні з традиційними стратегіями, підвищення стабільності пропускної здатності (BPS) до 20%, збільшення кількості виявлених помилок (Err) на 10-15% завдяки адаптивності підходу.

ПЕРЕЛІК ПОСИЛАНЬ

1. What is Load Testing & Why is Load Testing Important? [Електронний ресурс] // dotcom-monitor. – 2024. – Режим доступу до ресурсу: https://www.loadview-testing.com/learn/load-testing/.
2. 16 Top Load Testing Software Tools for 2024 [Електронний ресурс] // TestGuild. – 2024. – Режим доступу до ресурсу: https://testguild.com/load-testing-tools/.
3. Bezemer C. Towards Reducing the Time Needed for Load Testing [Електронний ресурс] / C. Bezemer, H. Alghamdi's, W. Shang // ResearchGate. – 2020. – Режим доступу до ресурсу: https://www.researchgate.net/publication/341712790_Towards_Reducing_the_Time_Needed_for_Load_Testing.
4. The Cost of Load Testing and How to Do it Right [Електронний ресурс] // LoadTesting. – 2021. – Режим доступу до ресурсу: https://www.loadtesting.com/blog/cost-of-load-testing.
5. Continuous Testing and CI/CD [Електронний ресурс] // TechValidate. – 2021. – Режим доступу до ресурсу: https://www.techvalidate.com.
6. The Role of Open Source Tools in Load Testing [Електронний ресурс] // Open Source Testing. – 2021. – Режим доступу до ресурсу: https://opensource.testing.com.
7. Load Testing Protocols and Best Practices [Електронний ресурс] // Load Testing Solutions. – 2020. – Режим доступу до ресурсу: https://www.loadtestingsolutions.com.
8. TechValidate Load Testing Survey [Електронний ресурс] // TechValidate. – 2021. – Режим доступу до ресурсу: https://www.techvalidate.com.
9. K6 and API Performance Testing [Електронний ресурс] // BlazeMeter. – 2020. – Режим доступу до ресурсу: https://www.blazemeter.com.
10. Continuous Load Testing with LoadRunner [Електронний ресурс] // TechRadar. – 2021. – Режим доступу до ресурсу: https://www.techradar.com.

11. AI in Load Testing [Електронний ресурс] // TechRadar. – 2021. – Режим доступу до ресурсу: https://www.techradar.com.

12. Adapting Load Testing Strategies for Cloud Environments [Електронний ресурс] // OpenStack. – 2021. – Режим доступу до ресурсу: https://www.openstack.org.

13. Simulating Different Types of Load [Електронний ресурс] // SoapUI Documentation. – 2024. – Режим доступу до ресурсу: https://www.soapui.org/docs/load-testing/simulating-different-types-of-load/.

14. Apache JMeter User Manual [Електронний ресурс] // Apache. – 2023. – Режим доступу до ресурсу: https://jmeter.apache.org/usermanual/.

15. Gatling Documentation [Електронний ресурс] // Gatling. – 2022. – Режим доступу до ресурсу: https://gatling.io/.

16. LoadRunner Professional [Електронний ресурс] // Micro Focus. – 2021. – Режим доступу до ресурсу: https://admhelp.microfocus.com/lr/en/24.1-24.3/help/WebHelp/Content/Resources/_TopNav/_TopNav_Home.htm.

17. Cloud Load Testing [Електронний ресурс] // AWS Documentation. – 2024. – Режим доступу до ресурсу: https://aws.amazon.com/testing/.

18. Performance Testing Using AI [Електронний ресурс] // TechRadar. – 2022. – Режим доступу до ресурсу: https://www.techradar.com.

19. Best Practices for Load Testing in CI/CD [Електронний ресурс] // BlazeMeter. – 2022. – Режим доступу до ресурсу: https://www.blazemeter.com/.

20. Testing Metrics and Their Importance in Load Testing [Електронний ресурс] // DZone. – 2023. – Режим доступу до ресурсу: https://dzone.com/articles/load-testing-with-kpis-part-1-what-are-kpis.

21. Distributed Load Testing Strategies [Електронний ресурс] // OpenStack Community. – 2024. – Режим доступу до ресурсу: https://www.openstack.org/.

22. Case Studies in Load Testing [Електронний ресурс] // Performance Testing Guide. – 2023. – Режим доступу до ресурсу: https://www.performancetestingguide.com/.

23. Ming Jiang Z. Analytics-Driven Load Testing: An Industrial Experience Report on Load Testing of Large-Scale Systems [Електронний ресурс] / Z. Ming Jiang,

P. Tse-Hsun Chen, W. Shang // ResearchGate. – 2019. – Режим доступу до ресурсу: https://www.researchgate.net/publication/318123731_Analytics-Driven_Load_Testing_An_Industrial_Experience_Report_on_Load_Testing_of_Large-Scale_Systems.

24. Pargaonkar S. A Comprehensive Review of Performance Testing Methodologies and Best Practices: Software Quality Engineering [Електронний ресурс] / Shravan Pargaonkar // ResearchGate. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/375450774_A_Comprehensive_Review_of_Performance_Testing_Methodologies_and_Best_Practices_Software_Quality_Engineering.

25. Bryant D. Load Testing APIs and Websites with Gatling: It's Never Too Late to Get Started [Електронний ресурс] / D. Bryant, G. Corre // InfoQ. – 2020. – Режим доступу до ресурсу: https://www.infoq.com/articles/load-testing-apis-gatling/.

26. Bart de Water. Shopify's Architecture to Handle the World's Biggest Flash Sales [Електронний ресурс] / Bart de Water // InfoQ. – 2022. – Режим доступу до ресурсу: https://www.infoq.com/presentations/shopify-architecture-flash-sale/.

27. Load Testing [Електронний ресурс] // Microsoft. – 2024. – Режим доступу до ресурсу: https://microsoft.github.io/code-with-engineering-playbook/automated-testing/performance-testing/load-testing/.

28. k6 Open Source [Електронний ресурс] // K6. – 2023. – Режим доступу до ресурсу: https://k6.io/open-source/.

29. Dhar A. The When and How of Load and Stress Testing [Електронний ресурс] / Ambhrin Dhar // Medium. – 2024. – Режим доступу до ресурсу: https://medium.com/slalom-build/load-stress-testing-when-how-5ac58e8403d0.

30. Load testing best practices [Електронний ресурс] // Google Cloud. – 2024. – Режим доступу до ресурсу: https://cloud.google.com/run/docs/about-load-testing.

31. Lee N. What is Load Testing? Best Practices for Getting Started [Електронний ресурс] / Nate Lee // SpeedScale. – 2023. – Режим доступу до ресурсу: https://speedscale.com/blog/what-is-load-testing/.

32. Load Testing and Performance/Stress Testing [Електронний ресурс] // Amazon AWS. – 2024. – Режим доступу до ресурсу: https://s3.amazonaws.com/load-testing-tools/index.html.

33. Optimize Web and API Delivery With Performance Testing and Monitoring Tools [Електронний ресурс] // Smart Bear. – 2023. – Режим доступу до ресурсу: https://smartbear.com/solutions/performance-testing/.

34. Managed Real-world Perfomance Load testing [Електронний ресурс] // Thinktribe. – 2024. – Режим доступу до ресурсу: https://www.thinktribe.com/managed-performance-load-testing.

35. Load Performance Testing [Електронний ресурс] // GitLab. – 2023. – Режим доступу до ресурсу: https://docs.gitlab.com/ee/ci/testing/load_performance_testing.html.

36. Verma A. Load Testing vs Stress Testing: The Main Differences [Електронний ресурс] / Antra Verma // BrowserStack. – 2024. – Режим доступу до ресурсу: https://www.browserstack.com/guide/load-testing-vs-stress-testing.

37. Load Testing on Cloud API [Електронний ресурс] // Meta. – 2024. – Режим доступу до ресурсу: https://developers.facebook.com/docs/whatsapp/cloud-api/guides/load-testing/.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«МЕТОДИКА ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ НАВАНТАЖУВАЛЬНОГО ТЕСТУВАННЯ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ»

Виконав: студент групи ПДМ-63 Студент Богдан Михайлович

Керівник: доктор філософії, старший викладач кафедри ІПЗ Залива Віталій Вікторович

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення ефективності виконання навантажувального тестування для високонавантажених систем за рахунок використання мультистратегічного методу тестування.

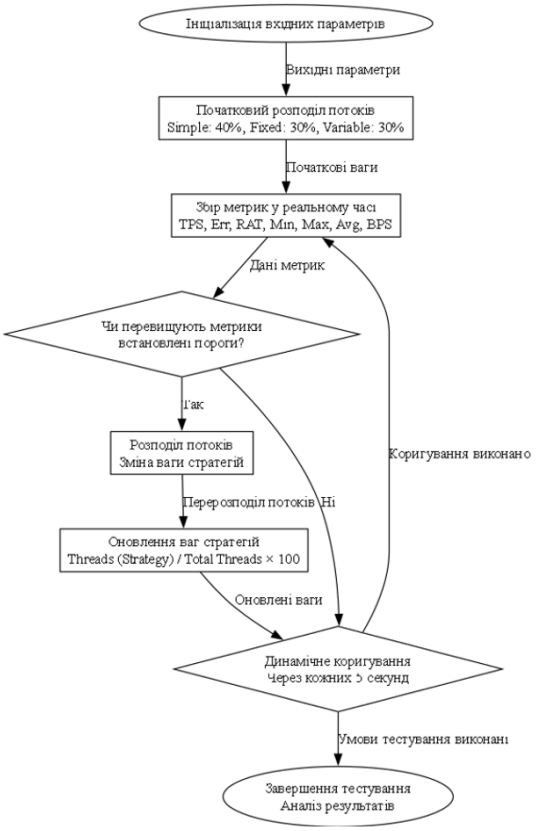
Об'єкт дослідження: процес навантажувального тестування високонавантажених систем.

Предмет дослідження: методи навантажувального тестування високонавантажених систем.

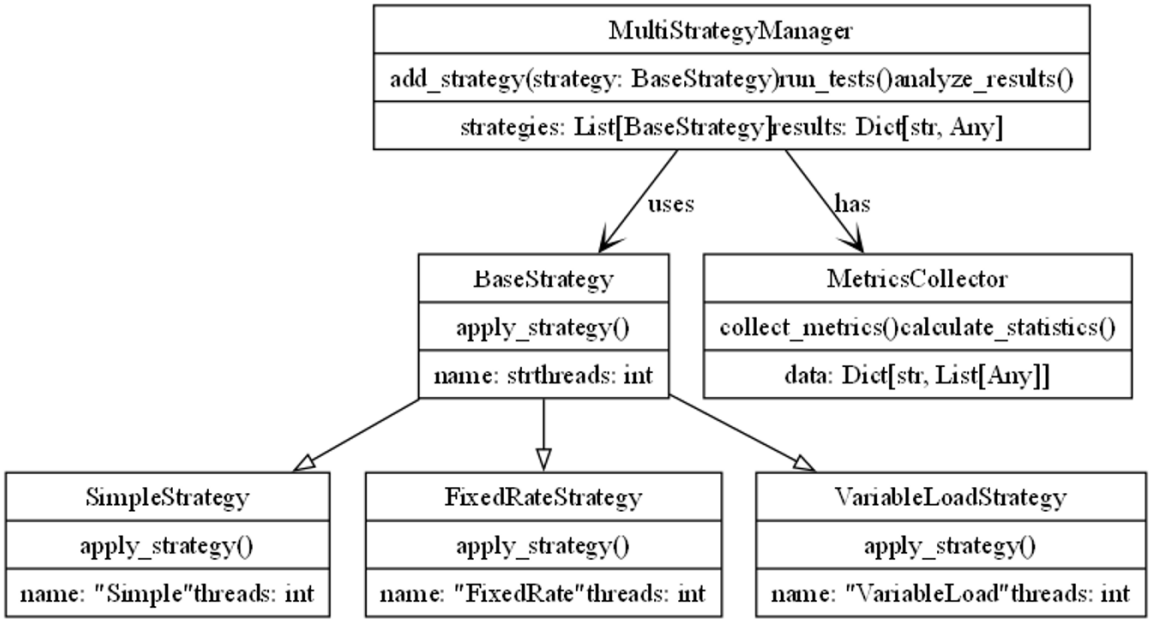
АНАЛІЗ МЕТОДІВ НАВАНТАЖУВАЛЬНОГО ТЕСТУВАННЯ

Критерій	Проста стратегія	Стратегія змінного навантаження	Стратегія фіксованого навантаження	Мультистратегічний підхід
Гнучкість	Обмежена: однакове навантаження на всю систему, не враховує пікові сценарії.	Помірна: дозволяє моделювати сценарії зі змінною інтенсивністю, але не оптимальна для пікових умов.	Висока: підходить для точного моделювання постійного потоку запитів, але обмежена для сценаріїв зі змінною інтенсивністю.	Найвища: здатність одночасно охоплювати різні сценарії, адаптуватися до змінних умов роботи системи.
Точність моделювання навантаження	Низька: реальні умови роботи системи моделюються слабко, через рівномірне навантаження.	Середня: дозволяє врахувати зміну навантаження, але вимагає налаштування кількох сценаріїв.	Висока: добре підходить для аналізу стабільності системи, але не враховує змінні сценарії.	Найвища: охоплює широкий спектр сценаріїв, у тому числі пікові навантаження та комбіновані умови.
Ефективність використання ресурсів	Витрати ресурсів часто є надмірними або недовантаженими через постійну інтенсивність.	Раціональна: адаптує використання ресурсів відповідно до змін навантаження.	Помірна: добре підходить для сталих умов, але може призводити до зайвого витрачання ресурсів за пікових умов.	Найкраща: ресурси перерозподіляються між сценаріями, мінімізуючи надмірне використання або недовантаження.
Складність реалізації	Легка: проста у налаштуванні, підходить для початкового етапу тестування.	Помірна: потребує додаткових налаштувань для реалізації змінного навантаження.	Помірна: вимагає ретельного планування для точного моделювання фіксованих умов роботи.	Складна: потребує складних алгоритмів для синхронізації стратегій і адаптації до різних сценаріїв.
Рівень помилок	Вищий: не завжди здатна виявити критичні точки системи.	Середній: дозволяє виявити частину проблем, але вимагає додаткових аналізів.	Низький: добре підходить для виявлення помилок у стабільних умовах, але слабка для складних сценаріїв.	Найнижчий: багатогранний підхід дозволяє краще виявляти проблеми під час моделювання реальних умов роботи.

АЛГОРИТМ РОЗПОДІЛУ «ВІРТУАЛЬНИХ КОРИСТУВАЧІВ» У МУЛЬТИСТРАТЕГІЧНОМУ МЕТОДІ



ДІАГРАМА КЛАСІВ РОЗПОДІЛУ «ВІРТУАЛЬНИХ КОРИСТУВАЧІВ»



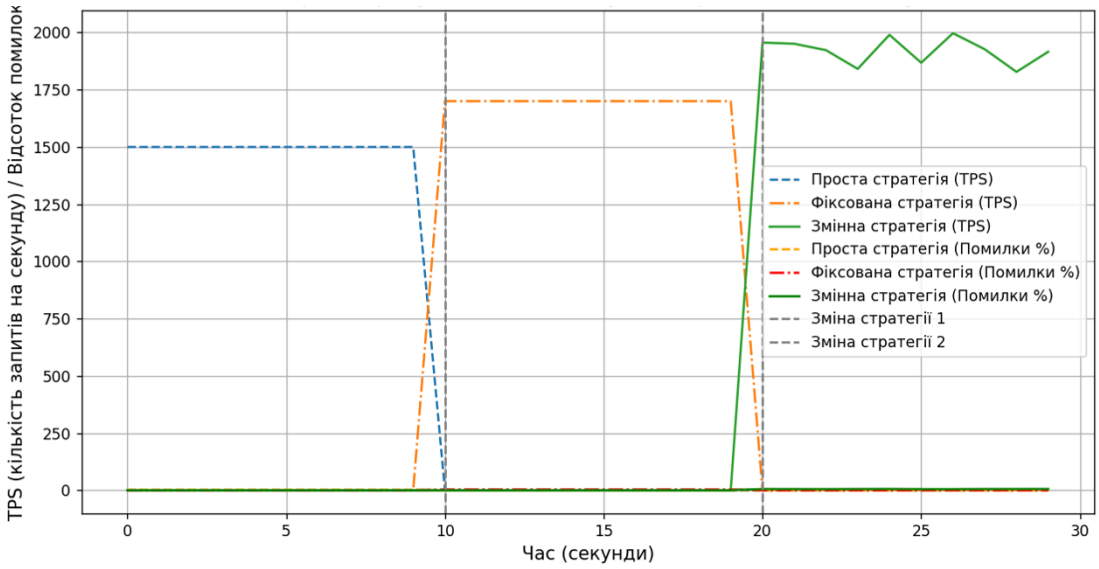
МАТЕМАТИЧНА МОДЕЛЬ МУЛЬТИСТРАТЕГІЧНОГО МЕТОДУ

$$P_{eff} = \omega_{TPS} * \frac{TPS_{actual}}{TPS_{max}} + \omega_{Err} * (1 - \frac{Err_{actual}}{Err_{max}}) + \omega_{RAT} * \frac{RAT_{actual}}{100}$$

Де:

- P_{eff} - загальний індекс продуктивності;
- ω_{TPS}, ω_{Err}, ω_{RAT} - вагові коефіцієнти (за замовчуванням: 0.4(проста), 0.3(постійного), 0.3(змінного) відповідно до стратегій);
- TPS_{actual} , $TPS_{TPS_{max}}$ - поточна та максимальна кількість транзакцій за секунду;
- Err_{actual}, Err_{max} - поточний та максимальний рівень помилок;
  RAT_{actual} - відсоток успішних запитів;

ПРОЦЕС РОЗПОДІЛУ НАВАНТАЖЕННЯ



ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ

Критерій	Проста стратегія	Фіксована стратегія	Змінна стратегія	Мультистратегічний метод
Транзакції за секунду	42	39	53	60
Відсоток знайдених помилок	77%	81%	85%	93%
Час відповіді мін./макс./середній	100 / 1200 / 600	80 / 1100 / 550	70 / 1050 / 520	60 / 1000 / 500
Стабільність часу відповідь (Max Response Time−Min Response Time)	1100	1020	980	940
Пропускна здатність у байтах/с	24,770	22,000	31,430	36.138
Потоки(«Кількість віртуальних юзерів»)	70000	70000	70000	70000
Використання ресурсів (%) CPU	70%	77%	82%	80%
Час тестування	20 хвилин	20 хвилин	20 хвилин	20 хвилин

ВИСНОВКИ

1. Проведено аналіз існуючих методів та стратегій навантажувального тестування високонавантажених систем. Встановлено, що традиційні підходи, такі як прості, фіксовані та змінні стратегії, мають суттєві обмеження при тестуванні систем з великими обсягами запитів та піковими навантаженнями. Це знижує точність оцінки продуктивності та надійності систем.

2. Визначено основні критерії ефективності для навантажувального тестування, серед яких: час відповіді (Min, Max, Avg), кількість транзакцій на секунду (TPS), кількість помилок (Err), пропускна здатність (BPS) та співвідношення успішних і невдалих запитів (RAT). Ці метрики стали базою для оцінки результативності запропонованого підходу.

3. Розроблено мультистратегічний метод навантажувального тестування, що базується на поєднанні простих, фіксованих та змінних стратегій тестування. Метод забезпечує динамічне переключення між стратегіями на основі поточних метрик продуктивності системи, таких як час відповідей, пропускна здатність та кількість помилок.

4. Реалізовано алгоритм автоматизації вибору стратегій у процесі тестування. Алгоритм дозволяє адаптувати навантаження на систему залежно від змін у показниках продуктивності у реальному часі. Це забезпечує підвищення точності тестування та скорочення часу на проведення аналізу.

5. Проведено тестування з використанням мультистратегічного підходу для тестування систем із великими обсягами запитів. Результати показали зниження середнього часу відповіді на 15-25% у порівнянні з традиційними стратегіями, підвищення стабільності пропускної здатності (BPS) до 20%, збільшення кількості виявлених помилок (Err) на 10-15% завдяки адаптивності підходу.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Студент Б. М., Залива В.В. Дослідження сучасних методів використання навантажувального тестування у створенні метасвіту// Зв’язок. Подано до друку

Тези доповідей:

1. Студент Б.М., Негоденко О.В. Дослідження можливостей технологій штучного інтелекту в автоматизації тестування. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». – Київ: ДУІКТ, 2024. – С.269-271.
2. Студент Б.М., Негоденко О.В. Інтеграція технологій та інструментів хмарних обчислень в автоматизованому тестуванні. // IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». – Київ: ДУІКТ, 2024. – С.177-178.