УДК 004.896 **doi: 10.26906/SUNZ.2024.1.122**

С. Ю. Леонов, Д. А. Тиртишний

Національний технічний університет «Харківський політехнічний інститут», Харків, Україна

ДОСЛІДЖЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРНОЇ ЧАСТИНИ КОМП'ЮТЕРНОЇ СИСТЕМИ НА ОСНОВІ РОЗРОБЛЕНОГО ФРЕЙМВОРКУ

Анотація. У даній статті були вивчені методи та використання оригінально розробленого фреймворку для тестування продуктивності серверної частини комп'ютеної системи. Було проаналізовано завдання та виклики, які стоять перед розробниками при оптимізації продуктивності серверів. Проведено моделювання роботи сервера та створено фреймворк, що включає інструменти для оптимізації навантаження, моніторингу, аналізу даних, керування тестовим процесом та розподілу ресурсів. Експерименти проведено на реальних веб-додатках, що використовувалися як тестове середовище. Система прийняття рішень на основі запропонованого фреймворку призводить до оптимізації продуктивності серверної частини веб-додатків, що значно поліпшує їхню працездатність і надійність. В ході досліджень були визначені та підтверджені ознаки подальшого покращення продуктивності веб-додатків.

Ключові слова: тестування продуктивності; комп'ютерна система; веб-додатки; фреймворк; оптимізація; розподіл ресурсів; система прийняття рішень.

Вступ

Тестування продуктивності веб-сайту — це засіб забезпечення якості, який передбачає використання автоматизованого тестування, що імітує роботу певної кількості бізнес користувачів з їх транзакціями [1, 2]. Цей вид тестування є обов'язковим у процесі покращення якості кінцевого продукту [3].

Причина полягає в тому, що будь-якого роду перебої або проблеми, пов'язані з поганою продуктивністю-можуть стати причиною відмови клієнтів від використання конкретного програмного забезпечення [4].

Розглянемо ситуацію: чорна п'ятниця, день скидок, інтернет-магазин розраховує на велику кількість продажів у цей день. І, раптом, сервер відмовляє, перестає працювати, усі люди виходять з сайту та починають робити покупки в магазинах конкурентах. Скільки грошей втратила би компанія AliExpress, якби їх сервера відмовили у чорну п'ятницю. Тобто, стабільність роботи є чи не найбільш важливішим атрибутом Web-сайту, а тестування продуктивності створене саме для забезпечення цієї стабільності. Таке тестування дає можливість переконатися, що випробуване програмне забезпечення чи аплікація добре працюють при критичних умовах [5] та допомагає визначити, наскільки швидко деякі конкретні частини її системи реагують у найгірших умовах [6].

Мета статті. Виконати дослідження та розробити фреймворк для тестування продуктивності серверної частини веб-додатків.

Основна частина

В залежності від характеристик системи, які підлягають перевірці, виділяють декілька типів тестування продуктивності [7, 8]:

- Performance Testing (Тест продуктивності) — це будь-який тест, який перевіряє стабільність, продуктивність, масштабованість та / або пропускну спроможність веб-сайту.
- Capacity Test, Volume Testing (Тест на місткість) — допомагає визначити, скільки користувачів / об'єм їх даних може обробляти веб-сайт або додаток, перш ніж продуктивність або стабільність стають неприйнятними.
- Load Test (Тест на завантаження) — це тестування реакції системи на зміну навантаження (в межах допустимого). Різниця із Performance Testing (Тест продуктивності) в тому, що навантаження може вимірюватися навмисне не на піку.
- Stress Test (Стрес-тест) — як показує його назва, подібне тестування приведе до того, що Ваша програма виконуватиметься за ненормальних умов. Це дозволить дізнатись, її здатність до регенерації, після завершення дії стресу. Стресове тестування також показує, які компоненти зникнуть на крайньому рівні.
- Stability Test(Тест стабільності на витривалість) — це тривалий тест, який використовується для оцінки продуктивності та / або стабільності програми у часі. Корисний, тому що при проведенні цього виду тестування здійснюється спостереження за споживанням пам'яті для виявлення потенційних витоків. Крім того, таке тестування виявляє деградацію продуктивності, котра виражається у зниженні швидкості обробки інформації та / або збільшенні часу відповіді аплікації після тривалої роботи, порівняно з початком тесту [9].
- Smoke Test (Димовий тест) — це короткі цикли тестів, які проводяться під дуже низьким навантаженням. Цей вид продуктивного тестування підкреслює, що програма працює як очікується. Цей термін походить від апаратного тестування, де, якщо дим генерується (буквально), це означає, що тест не вдався, і більше тестування не потрібне.

Для реалізації фреймворку тестування продуктивності серверної частини потрібно спочатку дослідити різні інструменти для тестування продуктивності, вивчити їх взаємодію та розробити архітектурну діаграму (рис. 1). Для того щоб забезпечити процес неперервної підтримки тестів та зрозумілий інтерфейс для їх запуску, була використана зв'язка Jenkins + Git. У [10, 11] описані Jenkins та Git. У фреймворку тестування продуктивності Git використовується для того щоб зберігати тестові файли та файл з параметрами запуску тестів.

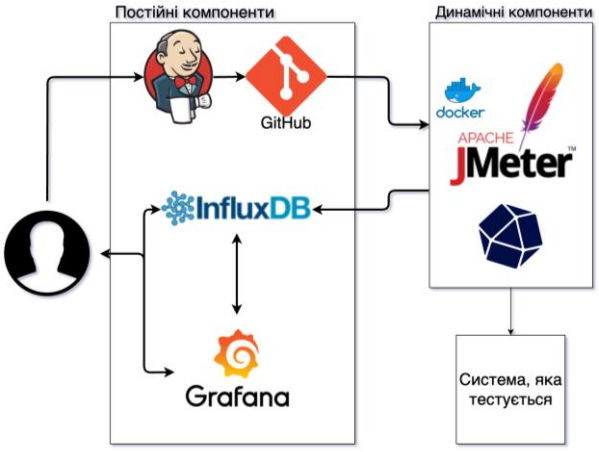


Рис. 1. Високорівнева архітектура фреймворку тестування продуктивності

Jenkins у свою чергу, надає можливість запуску тестів через зрозумілий графічний інтерфейс та з можливістю вибору параметрів запуску: кількість віртуальних користувачів, тип тесту, тестовий сценарій, тощо (рис. 2).

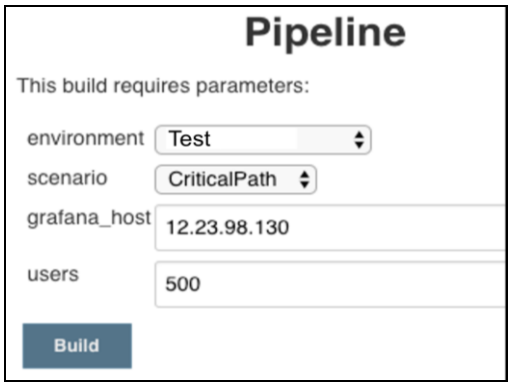


Рис. 2. Параметри запуска тесту в Jenkins

Детальний опис параметрів:

- environment – вибір системи, над якою буде проводитись тестування;
- scenario – вибір тестового сценарію, який буде протестований;
- grafana_host – адреса бази даних, яка зберігає результати та робить їх візуальне відображення (у вигляді графіків та таблиць);
- users – кількість віртуальних користувачів, які будуть робити запити до серверу одночасно.

Apache JMeter — інструмент для проведення навантажувального тестування, що розробляється Apache Software Foundation, підпроекту Jakarta. Хоча спочатку JMeter розроблявся як засіб тестування веб-застосунків, натепер він здатний проводити навантажувальні тести для JDBC-з'єднань, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP і TCP.

Цікава можливість – створення великої кількості запитів за допомогою декількох комп'ютерів при управлінні цим процесом з одного з них. Архітектура підтримує плагіни сторонніх розробників і дозволяє доповнювати інструмент новими функціями.

Docker — інструментарій для управління ізольованими Linux-контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.

Сирцевий код Docker написаний мовою Go і поширюється під ліцензією Apache 2.0. Інструментарій базується на застосуванні вбудованих в ядро Linux штатних механізмів ізоляції на основі просторів імен (namespaces) і груп управління (cgroups). Для створення контейнерів використовуються скрипти lxc. Для формування контейнера досить завантажити базовий образ оточення (команда docker pull base), після чого можна запускати в ізольованих оточеннях довільні програми (наприклад, для запуску bash можна виконати docker run -i -t base/bin/bash).

Docker дозволяє нам динамічно створювати генератор навантаження, у вигляді jmeter. Це має ряд переваг: у разі якщо виникли проблеми с генератором навантаження ми можемо легко перезапустити його за допомогою створення нового docker контейнеру. Також це дозволяє створювати генератори навантаження із потрібною нам конфігурацією відразу, тобто нам не потрібно буде кожен раз налаштовувати усю систему, вона буде відразу налаштована.

Telegraf – це додаток, який записує усі параметри моніторингу системи до бази даних, які потім можна буде використовувати для створення графіків у Grafana. Що дозволить нам бачити, яка частина комп'ютера дала збій під час тесту: процесор, оперативна пам'ять, диск чи з'єднання з інтернетом. Тобто Telegraf дозволяє робити моніторинг системи-сервера, яка тестується, у реальному часі.

InfluxDB – це база даних на основі часових рядів у якій зберігаються результати наших тестів. Усі дані зберігаються у вигляді пари ключ-значення, де ключем є поточний час. Також у цю базу можливо записувати спеціальні теги, які допоможуть фільтрувати дані та групувати їх на Grafana.

Grafana – це додаток для візуалізації даних, які зберігаються в InfluxDB. Найголовніший візуальний об'єкт у Grafana – це dashboard. Dashboard-це набір рядків, в кожному з яких є одна чи декілька панелей. Панелі бувають різні: таблиці, цифрові панелі, повідомлення, кругові діаграми, теплові карти, тощо. Також є можливість дивитись результати за вибраний період часу, що також є дуже важливою можливістю.

Перед тим як почати розробку, потрібно зрозуміти як компоненти будуть взаємодіяти між собою. Почати потрібно з установки Docker, далі будуємо "Docker image"(образ докер) для Grafana та Influx. Образ Docker містить операційну систему, застосунок і всі його залежності. Образи в Docker складаються з шарів. Якщо треба отримати образ з веб-сервером, то беремо за основу образ з дистрибутивом операційної системи, додаємо залежність - веб-сервер, і записуємо це як новий образ, який матиме два шари – один з ОС, наступний з веб-сервером. Образами можна обмінюватись через DockerHub.

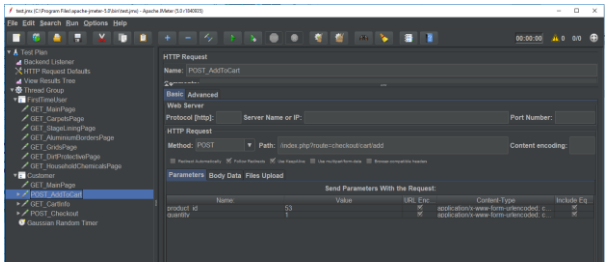
Саме з DockerHub можна завантажити офіційні образи InfluxDB, Grafana та Jmeter.

Після того, як образи були завантажені, їх потрібно встановити, або запустити. Запущені образи докерів називаються Docker Container. Запустимо контейнери бази даних InfluxDB та візуального додатку Grafana. Як тільки вони будуть запущені – ви зможете перейти по адресі машини на якій була запущена Grafana та продовжити настройку там. Приклад URL: https://localhost/grafana . Ці контейнери ми запускаємо відразу, бо вони є постійними компонентами. Доступ до них ми повинні мати постійно, аби завжди мати змогу подивитися результати попередніх тестів. Образ Jmeter буде запускатися динамічно, одразу після кліку на кнопку “build” у Jenkins.

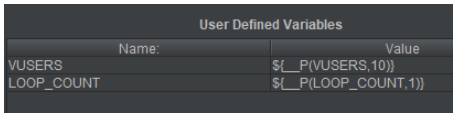
Наступним етапом буде установка Jmeter на свій комп’ютер для скриптування запитів до серверу на основі тестового сценарію. Jmeter складається з декількох основних компонентів: Thread Group, HTTP Requests, Listeners та таймери [20].

- Thread Group: налаштування усього сценарію, в яких вибираються кількість віртуальних користувачів, кількість запусків тестового сценарію;
- HTTP Requests: це саме наші запити до сервера, в них ми вказуємо URL, за яким ми відправляємо запит, тип запиту, та параметри;
- Listeners: слухачі, які запам’ятовують та агрегують результати тесту, в нашому випадку використовуємо спеціальний Backend Listener який буде писати результати в базу даних InfluxDB;
- Таймери дозволяють встановлювати паузи між запитами, щоб емулювати більш реальний сценарій використання сайту, бо навряд чи ви змогли би передивитися 5 сторінок сайту за 3 секунди.

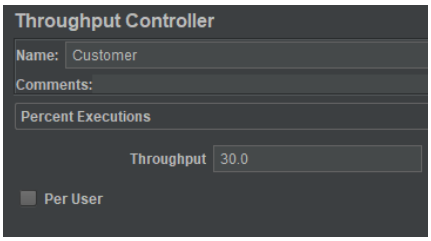
Інтерфейс Jmeter, та усі позначені вище його головні компоненти для тестового сценарію наведені на рис. 3, а.



а



б



в

Рис. 3. Тестовий сценарій у Jmeter

На рис. 3, б можна побачити 2 змінні VUSERS та LOOP_COUNT. Вони позначають кількість одночасних віртуальних користувачів та кількість запусків сценарію. Ці 2 змінні мають значення у виді змінної та значення за замовчанням. Це зроблено для того, щоб можна було передавати параметри при запуску тесту через файл з параметрами, або через Jenkins. Але якщо ніяких параметрів не передавати, то буде взято значення за замовчанням.

На рисунку 3, в можна побачити яким чином було реалізоване розділення кількості користувачів на конкретні запити до сервера. Був використаний пропускний контролер, котрий пропускав лише 30% від загальної кількості віртуальних користувачів.

Після того як тести були успішно розроблені у Jmeter, час додати їх на GIT та з'єднати Git репозиторій із Jenkins проектом. Після того як усе це зроблено залишається лише додати Pipeline Script до Jenkins проекту, який буде доставати тести з репозиторія, після цього створювати файл з параметрами, записувати туди параметри для тестів, вказані користувачем при запуску, далі він створює контейнер з Jmeter, в який додається тест та файл параметрів, після того тести проходять усередині контейнеру, контейнер видаляється, та результати відображаються у Grafana (рис. 4). Скрипт має бути написаний на мові Groovy.

```
stage('Execute tests') {

    sh "docker run --rm --volume
      `pwd`/Tests:/mnt/jmeter
      jmeter:latest -n -t
      /mnt/jmeter/${scenario}/test.jmx
      -q /mnt/jmeter/${scenario}/parameters.txt"
```

Рис. 4. Команда запуску контейнера Jmeter

Після цього залишається лише запустити тести та аналізувати результати.

Почнемо з тесту під навантаженням в 100 одночасних віртуальних користувачів (рис. 5)

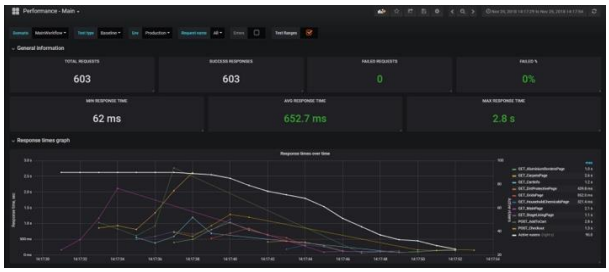


Рис. 5. Тестовий сценарій
100 віртуальними користувачами

На рисунку показано, що зі 100 одночасними користувачами система справляється дуже добре, максимальний час відповіді сервера клієнту склав 2.8 секунди, а середній лише 652 мс. На графіку, спочатку різко почала заходити велика кількість користувачів на сайт, тому час відповіді був доволі високий, але поступово користувачі закінчували свою роботу й показники покращувались. Верхня

біла лінія відображає кількість віртуальних користувачів у секунду часу. На піку, 90 користувачів в одну секунду запитували сервер. Для більш детального розбору кожного запиту потрібна детальна таблиця (рис. 6).

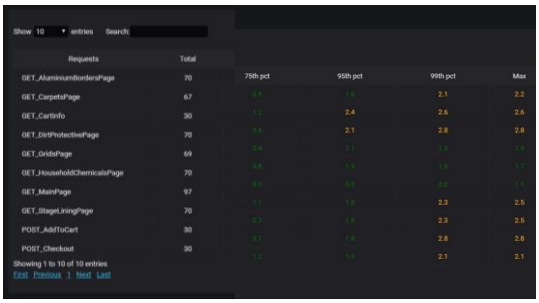


Рис. 6. Таблиця для сценарію зі 100 користувачами

Далі було прийняте рішення знайти кількість одночасних користувачів ближчу до максимуму, тест на місткість. Проведемо тестування з 200 одночасними користувачами (рис.7).

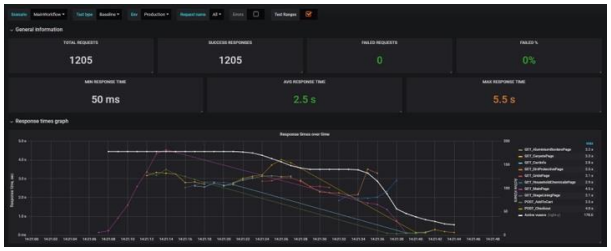


Рис. 7. Тест на місткість,
200 одночасних віртуальних користувачів

Середня швидкість відповіді сервера клієнту складає 2.5 секунди, що є нормою, але максимальний час вже виходить за норму, тобто можна вважати, що 200 користувачів є останньою межею перед початком нестабільної та повільної роботи сервера.

Наступним видом тестування було прийняте рішення зробити стресове тестування. Такий вид тестування допоможе нам побачити помилки, які викидає сервер, та навантаження на нього.

Проведемо тестування з 500 одночасними користувачами. (рис. 8)

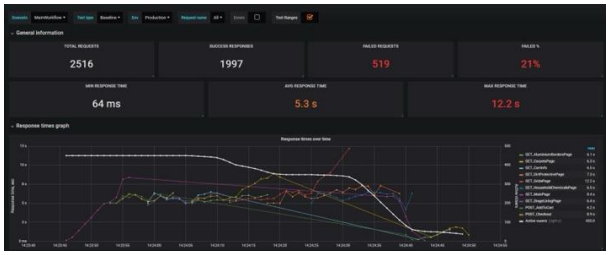


Рис. 8. Стрес тест, 500 віртуальних користувачів

При такому стресовому навантажені система вже стає нестабільною, та 21% запитів відбиваються з помилками. При цьому середній час відповіді сервера клієнту складає 5.3 секунди, а максимальний аж 12.2 секунди. Також можемо бачити, що найбільша кількість віртуальних користувачів у секунду, склала 450. Для того щоб розібратись з помилками, була розроблена відповідна панель з детальним описом помилок (рис. 9).

Рис. 9. Помилки при стрес тесті

Помилки були «Gateway time-out», що означає те, що клієнт не дочекався відповіді від сервера та вийшов з сесії. Це все стається у наслідок неймовірного навантаження на сервер. Після цього потрібно розібратись, а в чому все ж таки була проблема, чому сервер почав працювати повільно та нестабільно. Саме для цього нам і знадобиться інструмент моніторингу системи в реальному часі – Telegraf (рис. 10).

На рисунку вище ми бачимо, що як тільки на нашому сервері з'являється 500 одночасних користувачів, наш процесор завантажується на всі 100%, та тримає такий показник аж до кінця тесту і лише після цього, зменшує свої показники.

Рис. 10. Використання CPU сервером

Обговорення результатів

В ході дослідження були виконані дослідження з різними видами тестування продуктивності серверної частини веб-додатків. Після того були досліджені різні інструменти для розробки фреймворку для тестування продуктивності.

Після розробки фреймворку, був розроблений тест для прикладу, та завдяки цьому тестовому сценарію був протестований інтернет-магазин.

Тести продуктивності перевірили максимальну місткість сайту, його стабільність, та швидкість. Сайт може видавати відповідь на будь-який запит клієнта швидше ніж за 2.5 секунди при 100 одночасних користувачів на сайті. Також виявилось, що сайт може функціонувати навіть при 500 одночасних користувачів, але вже з деякими помилками, та затримками. Стабільна безпомилкова робота продовжується десь до 200 користувачів при нетривалому навантаженні на сервер.

При тривалому навантаженні від 100 одночасних юзерів, інтернет-магазин зберігає добрі показники швидкості, якості та стабільності. При стресових навантаженнях уся продуктивність впирається в процесор, тому порадою для власника інтернет-магазину буде зміна процесора на більш потужний, якщо він звичайно розраховує на такі великі навантаження на сервер.

Висновки

У результаті проведеного дослідження тестування продуктивності серверної частини веб-додатків та розробки фреймворку потрібно зазначити, що впровадження цього фреймворку дозволяє підвищити ефективність роботи та стабільність серверних веб-додатків.

Були детально досліджені особливості тестування продуктивності серверних веб-додатків, обговорені можливі проблеми, з якими можуть зіткнутися розробники, створені рекомендації щодо їхнього усунення.

Розроблений фреймворк включає інструменти для створення навантаження, моніторингу, аналізу даних, керування тестовим процесом, а також модулі для вдосконалення продуктивності. Він є зручним та гнучким інструментом для ефективного аналізу та вдосконалення продуктивності серверних веб-додатків.

Експерименти, проведені в рамках дослідження, підтвердили, що використання розробленого фреймворку призводить до значного покращення продуктивності серверних веб-додатків.

Отже, результати цього дослідження можуть бути використані розробниками веб-додатків для підвищення продуктивності та надійності своїх продуктів.

Список літератури

1. Леонов С. Ю., Тиртишний Д. А. Ключові аспекти тестування продуктивності. *Проблеми інформатики та моделювання* (ПІМ-2023): матеріали міжнародної науково-технічної конференції, Харків: НТУ «ХПІ», 2023. С. 69. URL: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/ee691cd3-44dd-4848-8631-5df0882a6c40/content
2. Still A. Web performance warrior. The Business of Speed. Published by O'Reilly Media, Inc., CA 95472, 2015. 50 p. URL: https://theswissbay.ch/pdf/Books/Computer%20science/O%27Reilly/web-performance-warrior.pdf
3. Buckler C. Jump Start Web Performance. SitePoint, 2020. 159 p. URL: https://www.oreilly.com/library/view/jump-start-web/9781098122799/
4. Донич Д. Види тестування програмного забезпечення. 2021. URL: https://lemon.school/blog/vydy-testuvannya-programnogo-zabezpechennya
5. Devaraj K. Software Testing Models. What it is, Types & How They Work? 2023. URL: https://testsigma.com/blog/software-testing-models/
6. Dustin E., Rashka J., Paul J. Automated Software Testing: Introduction, Management, and Performance. Addison-Wesley Prof., 1999. 608 p. URL: https://books.google.com.ua/books/about/Automated_Software_Testing.html?id=kl2H0G6EFf0C&redir_esc=y
7. Hoda R., Salleh N., Grundy J. The Rise and Evolution of Agile Software Devleopment. *IEEE Software*. 2018. Vol. 35, no. 5. P. 58-63. URL: https://ieeexplore.ieee.org/document/8409911
8. Dybå T., Dingsøyr T. Agile Project Management: From Self-Managing Teams to Large-ScaleDevelopment. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Florence, 2015. https://ieeexplore.ieee.org/document/7203121
9. Whiting E., Datta S. Performance Testing and Agile Software Development: A Systematic Review. 2021. 36 p. URL: https://www.researchgate.net/publication/351410867_Performance_Testing_and_Agile_Software_Development_A_Systematic_Review
10. Loadster. URL: https://loadster.app/guides/front-end-vs-back-end-performance/#the-8020-rule-of-web-performance
11. Носков В. І., Тиртишний Д. А. Тестування продуктивності web-сайту для зростання якості продукту. *Інформатика, управління та штучний інтелект* : матеріали 5-ї міжнар. наук.-техн. конф. студентів, магістрів та аспірантів, 20-22 листопада 2018 р. / наук. ред. В. Д. Дмитрієнко ; Нац. техн. ун-т "Харків. політехн. ін-т". Харків : НТУ "ХПІ", 2018. С. 65. URL: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/6476d887-10a4-48af-9b25-75a9e2e68856/content

Received (Надійшла) 12.12.2023
Accepted for publication (Прийнята до друку) 07.02.2024

Research of server-side performance testing and framework development

Sergey Leonov, Dmytro Tyrtyshnyi

Abstract. This article examines the methodologies and application of a unique framework developed for performance testing of the server-side of web applications. The academic study delves into the tasks and challenges developers commonly face while seeking to optimize server performance. Server operation was simulated and a comprehensive framework was developed. This framework incorporates comprehensive tools to create a load, perform monitoring functions, analyze data, and manage the test process. All experiments for this study were implemented on actual web applications providing a real-world test environment. The study shows that implementing the proposed framework optimizes the performance of the server-side of web applications. This significant improvement enhances operability and reliability. Our research has also narrowed down and confirmed specific indications that could lead to further improvements in web application performance.

Keywords: performance testing, server-side, web applications, framework, optimization, experiment, development.