ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА ЕЛЕКТРОННА ІНЖЕНЕРІЯ
INFORMATION AND COMMUNICATION TECHNOLOGIES, ELECTRONIC ENGINEERING

№ 4 (2), 2024

РОЗРОБКА ПЛАТФОРМИ ДЛЯ ДОСЛІДЖЕННЯ АВТОМАТИЧНОГО МАСШТАБУВАННЯ КОНТЕЙНЕРІВ ТА БАЛАНСУВАННЯ НАВАНТАЖЕННЯ У РОЗПОДІЛЕНИХ СИСТЕМАХ

Г. Бешлей [ORCID: 0000-0001-5392-3499], **С. Боднар** [ORCID: 0009-0007-0235-5074], **М. Селюченко** [ORCID: 0009-0007-7801-8605], **М. Бешлей** [ORCID: 0000-0002-7122-2319], **М. Климаш** [ORCID: 0000-0003-2867-1482]

Національний університет "Львівська політехніка", вул. С. Бандери, 12, Львів, 79013, Україна

Відповідальний за рукопис: М. Бешлей (e-mail: mykola.i.beshlei@lpnu.ua)

(Подано 15 травня 2024)

Забезпечення якості обслуговування (Quality of Service, QoS) визначено як ключове завдання для розподілених систем, оскільки задоволення потреб користувачів є важливим аспектом їх успішного функціонування. Більшість рішень для автомасштабування контейнерів зосереджені на оптимізації ресурсів та управлінні витратами. Проте ці рішення часто не враховують динамічні вимоги користувачів до якості обслуговування (QoS), що призводить до затримок у виділенні ресурсів та зниження якості обслуговування. Існуючі алгоритми автомасштабування та балансування навантаження неадекватно враховують динаміку навантаження, що є суттєвою проблемою. Крім того, традиційні платформи для тестування нових алгоритмів, такі як Azure та AWS, є комерційними та закритими, що обмежує можливості для перевірки інноваційних підходів. У зв'язку з цим існує потреба у відкритих та доступних платформах, які дозволяють дослідникам та розробникам ефективно тестувати та впроваджувати нові алгоритми балансування навантаження та автоматичного масштабування. Для вирішення цих проблем необхідний новий підхід, заснований на глибокому розумінні контексту використання ресурсів та потреб користувачів, що забезпечить високу якість обслуговування та підвищить ефективність розподілених систем. Новизна даної роботи полягає у розробці нової платформи для дослідження методів автоматичного масштабування контейнерів та алгоритмів балансування навантаження. Створена віртуалізована сервісна платформа дозволила на практиці оцінити переваги та недоліки алгоритмів у реальних умовах. Наприклад, використання алгоритму "Round Robin" при затримці запитів у 50 мс призводило до завантаження серверів на 96.2% і середнього часу затримки у 679 мс. Впровадження алгоритму "Weighted Round Robin" та автоматичного масштабування контейнерів дозволило зменшити завантаження серверів до 56.1% та середню затримку до 11.8 мс. Отримані результати можуть стати основою для подальшого розроблення та впровадження алгоритмів у розподілених системах, що дозволить покращити якість обслуговування та загальну ефективність цих систем.

Ключові слова: *розподілені системи, балансування навантаження, віртуалізація, автоматичне масштабування, архітектура*.
УДК: 621.391

1. Вступ

Вплив великого навантаження на розподілені системи може мати значні наслідки. Під час високого навантаження збільшується час обробки запитів веб-сервісами, що може призвести до затримок у відповіді на запити користувачів, обмеженої пропускної здатності та навіть відмов у обслуговуванні [1]. Забезпечення якості обслуговування (Quality of Service, QoS) стає ключовим завданням для розподілених систем, оскільки задоволення потреб користувачів є важливим аспектом успішного функціонування таких систем.

Для досягнення високої якості обслуговування в умовах інтенсивного трафіку компанії часто змушені розгортати додаткові сервери та збільшувати обчислювальні ресурси, що призводить до зростання витрат і споживання енергії. Сучасні системи оркестрування контейнерів, такі як Kubernetes, використовувані хмарними операторами, як Amazon Web Services (AWS) та Microsoft Azure, застосовують правила автоматичного масштабування з фіксованими пороговими значеннями для задоволення бізнес-вимог, спираючись на моніторинг таких показників, як завантаження процесора та використання пам'яті [2-5].

Проте існує необхідність у системах, які можуть автоматично збільшувати або зменшувати ресурси під час змін активності сервісів. Це можна реалізувати за допомогою автомасштабування на основі доступності сервісів, що дозволяє динамічно регулювати кількість контейнерів залежно від навантаження. Це підхід допоможе не тільки підтримувати необхідний рівень QoS, але й оптимізувати використання обчислювальних ресурсів, знижуючи витрати та споживання енергії [6].

Наукові дослідження підкреслюють, що більшість рішень для автомасштабування контейнерів зосереджені на оптимізації ресурсів та ефективному управлінні витратами. Проте виникають проблеми із забезпеченням динамічних вимог користувачів до QoS та оптимізацією використання обчислювальних ресурсів. Існуючі алгоритми автомасштабування та балансування навантаження контейнерів неадекватно враховують динаміку робочого навантаження, що призводить до затримок у виділенні ресурсів та зниження QoS [7].

Традиційні платформи для тестування нових алгоритмів балансування навантаження, такі як Azure та AWS, є комерційними та з закритим вихідним кодом, що обмежує можливості для перевірки інноваційних підходів. Існує потреба у відкритих та доступних платформах, які дозволяють дослідникам та розробникам ефективно тестувати і впроваджувати нові алгоритми балансування навантаження та автоматичного масштабування [8-10]. Для вирішення цих проблем необхідний новий підхід, заснований на глибокому розумінні контексту використання ресурсів та потреб користувачів, що забезпечить високу якість обслуговування та підвищить ефективність розподілених систем.

2. Розгоробка віртуалізованої сервісної платформи.

Для тестування процесу надання веб-сервісів користувачам у віртуалізованому обчислювальному середовищі, побудованому на сучасних інформаційних технологіях та алгоритмах, таких як Java, Spring Boot, Docker, Hibernate, HSQL DB, JQuery, AJAX та HighCharts, була розроблена універсальна сервісна платформа (рис.1). Основна цінність цієї платформи полягає в її багатофункціональності та гнучкості. Вона може бути розгорнута на одній фізичній машині для локального тестування і відлагодження конкретних сценаріїв надання послуг або на розподілених фізичних машинах для моделювання та дослідження сценаріїв масштабованих розподілених систем. Платформа використовує Java та Spring Boot для створення надійних і масштабованих веб-додатків, Docker для ізоляції додатків і легкого управління залежностями, Hibernate і HSQL DB для ефективного управління даними, а також JQuery, AJAX та HighCharts для забезпечення динамічного і інтерактивного інтерфейсу користувача. Функціонуючи як сервісна система з вбудованими функціями балансування навантаження, платформа дозволяє ефективно розподіляти

запити користувачів між серверами та підтримує дослідження і тестування нових алгоритмів балансування навантаження, що сприяє покращенню якості обслуговування та оптимізації використання ресурсів. Крім того, платформа може бути використана для реалізації реальних веб-сервісів, забезпечуючи високий рівень якості обслуговування завдяки ефективному управлінню ресурсами та динамічному масштабуванню. Розроблена сервісна платформа підтримує як горизонтальне, так і вертикальне масштабування на програмному рівні, що дозволяє динамічно адаптувати ресурси відповідно до поточних потреб. У системі впроваджений власний балансувальник навантаження, який використовує алгоритми Round Robin та Weighted Round Robin для оптимального розподілу запитів між серверами. Окрім цього, є можливість розширення функціоналу для підвищення ефективності розподілу.

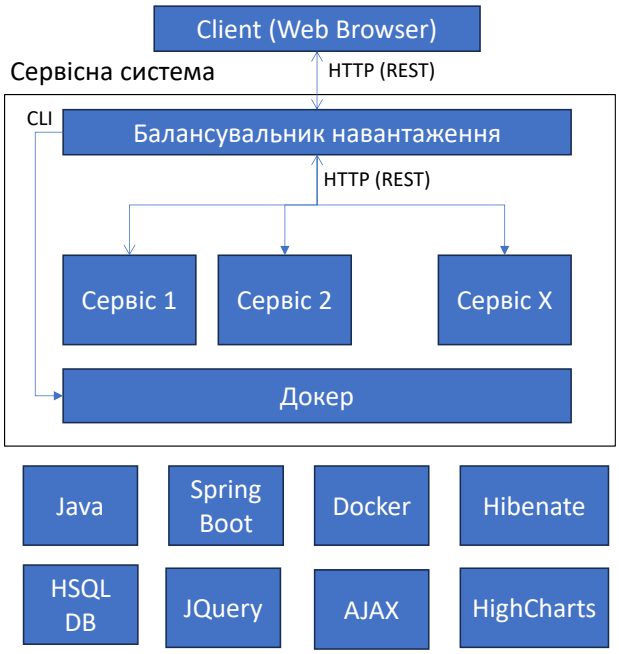


Рис. 1. Структура розробленої віртуалізованої сервісної платформи для дослідження алгоритмів балансування навантаження

Система збирає та аналізує статистичні дані про кількість оброблених запитів і завантаження центрального процесора кожного сервера. Це забезпечує можливість моніторингу та детального аналізу продуктивності системи. Також був створений користувацький інтерфейс для веб-браузера, який значно спрощує процес управління ресурсами, проведення експериментів та моніторинг стану системи.

Створене тестове середовище включає наступні компоненти:

Веб-клієнт: інтерфейс, який використовується користувачем для взаємодії з додатком. Веб-клієнт надсилає запити до балансувальника навантаження, який подальше розподіляє ці запити між серверами або контейнерами.

Балансувальник навантаження (LB): програмний компонент, який приймає запити від веб-клієнта і розподіляє їх між серверами або контейнерами. Основна мета балансувальника навантаження полягає в забезпеченні рівномірного розподілу трафіку для збільшення пропускної здатності та надійності системи..

Java: Це мова програмування та середовище виконання, використане для реалізації балансувальника навантаження та серверів. Java відзначається високою продуктивністю та широкими можливостями для розробки програмного забезпечення.

Сервери або Docker-контейнери: Це обчислювальні ресурси, на яких розгорнуті додатки або сервіси. Сервери або Docker-контейнери обробляють запити від веб-клієнта та надають відповідні відповіді.

Взаємодія між компонентами у тестовому середовищі відбувається наступним чином (рис.2). Веб-клієнт надсилає запит до балансувальника навантаження, який обробляє його і надсилає відповідь клієнту. На цьому етапі балансувальник визначає, який із серверів або контейнерів буде обробляти запит.

Після отримання запиту, балансувальник запитує у серверів інформацію про завантаження CPU контейнерів за допомогою команди Docker stats. Цей запит дозволяє отримати актуальну інформацію про використання ресурсів кожним контейнером, що є важливим для прийняття рішень про оптимізацію розподілу навантаження. Балансувальник аналізує отримані дані і на основі цієї інформації може вирішити, чи необхідно створити новий контейнер для обробки запитів або видалити наявний, який неефективно використовує ресурси.

Коли створюється новий контейнер, інформація про нього зберігається в базі даних системи. Контейнер підтверджує свою готовність до роботи, після чого він може приймати запити. Цей процес дозволяє забезпечити динамічне масштабування ресурсів відповідно до поточних потреб системи.

Після цього балансувальник вибирає відповідний сервер для обробки запиту від веб-клієнта. Він зберігає статистику про запит, яка включає інформацію про час обробки та використання ресурсів. Запит перенаправляється до вибраного сервера, де він обробляється. Сервер виконує необхідні операції, обробляє дані і генерує відповідь, яку надсилає назад до балансувальника.

Балансувальник отримує відповідь від сервера і пересилає її веб-клієнту. На цьому етапі він також фіксує час, витрачений на обробку запиту, та обчислює затримки. Ці дані використовуються для подальшого аналізу продуктивності системи та виявлення можливих вузьких місць.

Такий механізм забезпечує ефективну взаємодію між компонентами, оптимізацію використання ресурсів та стабільну роботу системи. Завдяки збиранню та аналізу статистичних даних про кількість оброблених запитів і завантаження центрального процесора кожного сервера, система може адаптуватися до змін у навантаженні і забезпечувати високу продуктивність та надійність роботи.

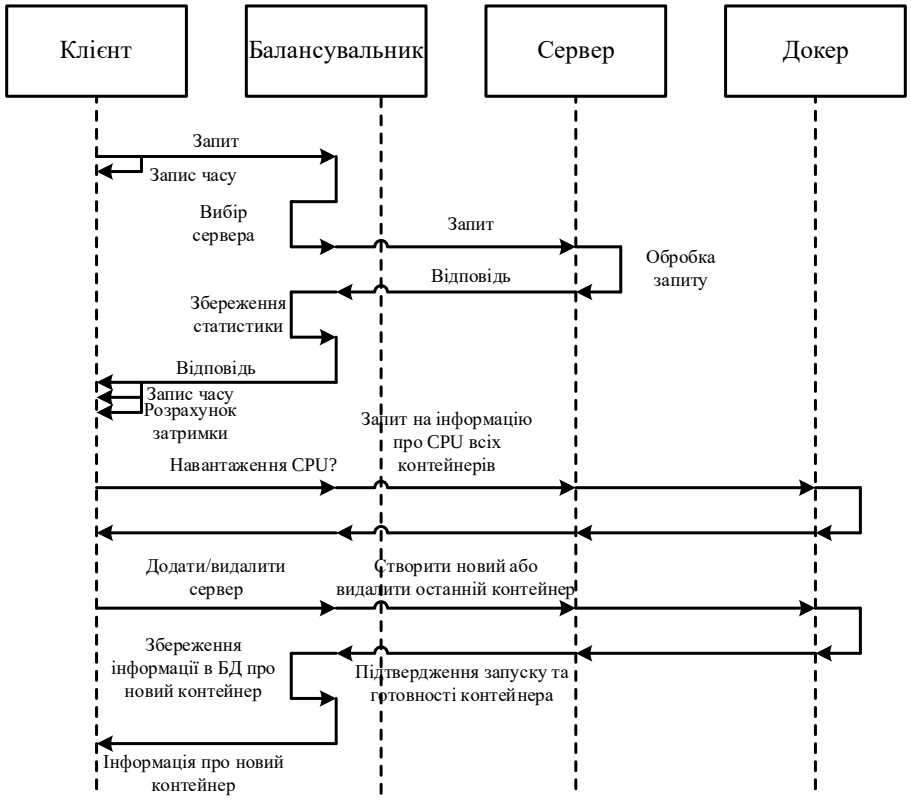


Рис. 2. Діаграма взаємодії компонентів розробленої платформи

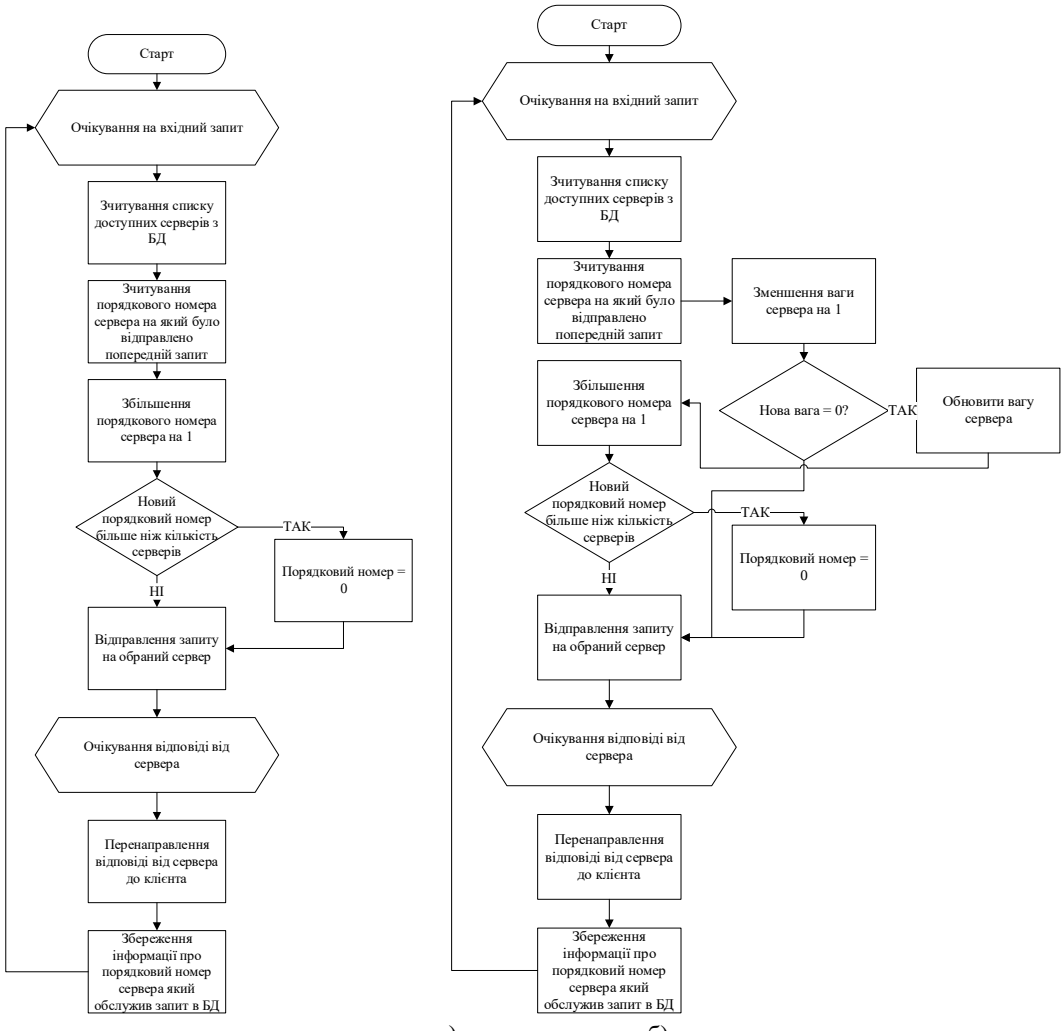
Основні програмні функції платформи, що використовуються для дослідження автоматичного масштабування контейнерів та балансування навантаження у розподілених системах, описані нижче.

1. **"/api/greetings"**: повертає дані у форматі JSON, що містять колекцію об'єктів типу Greeting з бази даних.
2. **"/api/create-addition-container/{quota}"**: обробляє GET-запит за шляхом "/api/create-addition-container/{quota}", де {quota} є змінною, що передається у запиті. Запит дозволяє управляти Docker через командний рядок, зокрема, динамічно додавати контейнери. Параметр quota встановлює обмеження для контролю та регулювання використання певних ресурсів або функцій. Під час тестування навантаження на Docker контейнер, керування квотою процесорної потужності дозволяє регулювати і обмежувати використання CPU контейнером під час інтенсивного навантаження. Це сприяє ефективному проведенню тестів та виявленню можливих проблем чи нестабільностей системи під навантаженням, забезпечуючи стабільність і передбачуваність продуктивності контейнера.
3. **"/api/weighted_round_robin"**: реалізує алгоритм зваженого циклу. Використання ваги важливе для нерівномірного розподілу навантаження між серверами, визначаючи пропорції навантаження, яке отримують різні сервери. У даній роботі вага сервера обчислюється за формулою serverQuota / maxQuota * 10, де serverQuota - це обмеження сервера, а maxQuota – це обмеження всіх серверів.
4. **"/api/containers-cpu-usage"**: виконує запит до Docker для отримання інформації про використання CPU контейнерами та повертає ці дані у відповідь на запит клієнта. Для цього використовуються вбудовані засоби Docker, зокрема команда docker stats, яка надає актуальні статистичні дані про кожен контейнер, включаючи використання процесора. У реалізації методу спершу створюється об'єкт Process для виконання команди docker stats за допомогою методу Runtime.getRuntime().exec(cmd). Ця команда дозволяє отримати живі дані про використання ресурсів контейнерами, які потім обробляються для виділення ідентифікатора контейнера та відповідних показників CPU використання. Після виконання команди, використовується BufferedReader для зчитування вихідного потоку процесу через process.getInputStream(). Кожен рядок вихідного потоку додається до об'єкта StringBuilder result, який містить усю отриману інформацію, розділену новими рядками. Наприкінці метод формує відповідь, що містить отримані дані (result.toString()), та HTTP-статус HttpStatus.OK, що підтверджує успішне завершення запиту..
5. **"/api/list-containers"**: відповідає на запит, повертаючи список серверів.
6. **"/api/delete-last-container"**: API-метод видаляє останній контейнер за його ідентифікатором. Щоб отримати ID останнього контейнера, використовується клас Process та команда Process process = Runtime.getRuntime().exec("docker ps -q --latest"). Це дозволяє отримати ідентифікатор останнього контейнера. З допомогою класу BufferedReader зчитується вивід цієї команди. Після отримання ідентифікатора виконується видалення контейнера за допомогою команди docker rm -f та заданого ідентифікатора.
7. **"/api/list-containers-rps"**: збирає статистику щодо кількості оброблених запитів на секунду кожним контейнером. Балансувальник утримує дані про кількість оброблених запитів кожним сервером у своїй базі даних. Після кожного запиту балансувальник оновлює дані лічильника для відповідного сервера. При виклику операції "list-containers-rps" балансувальник зчитує всю таблицю з інформацією про кількість оброблених запитів кожним сервером і передає її контролеру. Після цього лічильники запитів обнуляються для всіх серверів. Метод дозволяє контролеру отримувати дані про кількість запитів, які були оброблені протягом конкретного періоду часу, наприклад, за одну секунду. Ця інформація дозволяє моніторити навантаження на сервери та оцінювати ефективність роботи балансувальника..

На рис. 3 зображені блок-схеми основних досліджуваних алгоритмів балансування навантаження, а саме round robin (рис. 3а) та weighted round robin (рис. 3б).

Алгоритм Round Robin починається з очікування вхідного запиту від клієнта. Після отримання запиту зчитується список доступних серверів з бази даних. Далі визначається порядковий номер сервера, на який було надіслано попередній запит, і цей номер збільшується на одиницю. Якщо новий порядковий номер перевищує кількість доступних серверів, його встановлюють на 0. Запит надсилається на сервер, визначений оновленим порядковим номером. Сервер обробляє запит і надсилає відповідь. Відповідь перенаправляється від сервера до клієнта, а інформація про порядковий номер сервера, який обслуговував запит, зберігається в базі даних.

Алгоритм Weighted Round Robin (рис. 3б) подібно до попереднього алгоритму, процес починається з очікування вхідного запиту. Після отримання запиту зчитується список доступних серверів з бази даних, а також порядковий номер сервера, на який було надіслано попередній запит. Порядковий номер сервера збільшується на одиницю, а вага сервера зменшується на одиницю. Якщо нова вага сервера дорівнює нулю, вона оновлюється до початкового значення, а порядковий номер встановлюється на 0. Якщо новий порядковий номер перевищує кількість доступних серверів, його також встановлюють на 0. Запит надсилається на сервер, визначений за оновленим порядковим номером і вагою. Сервер обробляє запит і надсилає відповідь, яка перенаправляється клієнту. Інформація про порядковий номер сервера, який обслуговував запит, зберігається в базі даних.



а) б)

Рис. 3. Блок схеми алгоритмів round robin (а) та weighted round robin

Обидва алгоритми забезпечують рівномірний розподіл навантаження між серверами, але алгоритм weighted round robin враховує вагу серверів, що дозволяє більш ефективно використовувати ресурси системи.

3. Дослідження алгоритмів балансування round robin та weighted round robin в умовах автомасштабування контейнерів

У ході експериментального дослідження початковий етап передбачав навантаження одного сервера. За умов затримки запитів у 50 мс, що еквівалентно 20 запитам на секунду, середнє завантаження сервера досягало 96.2% (рис.3 а). Такий високий рівень завантаження призвів до суттєвого збільшення затримки відповідей, що показано на рис. 3.6б, де значення затримки становило 679 мс. Як видно з результатів експерименту, порогове значення у 100 мс перевищене.

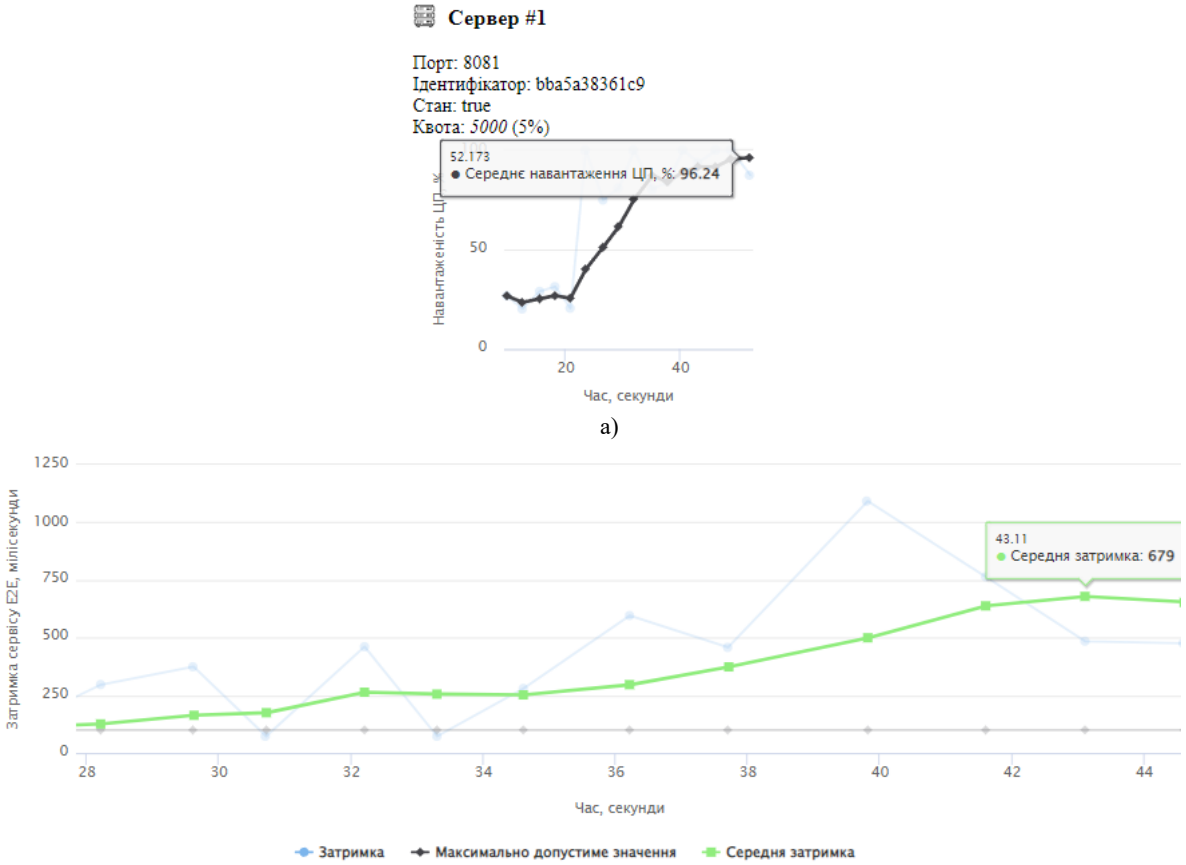


Рис. 4. Навантаження процесора (а) при високій нагрузці та затримка заитів обслуговування (б) в цей момент часу

Для зниження рівня навантаження було додано ще один ідентичний сервер із квотою 5000. Використовуючи алгоритм round robin, який було обрано у веб-клієнті, навантаження починають рівномірно розподіляти між двома серверами (рис. 5а). Завдяки цьому підходу кожен сервер обробляє приблизно половину загального навантаження, що сприяє зменшенню затримки відповідей і покращенню загальної продуктивності системи. Згідно з рис. 5б, після додавання додаткового сервера середній час затримки стабілізувався на значенні 73.8 мс.

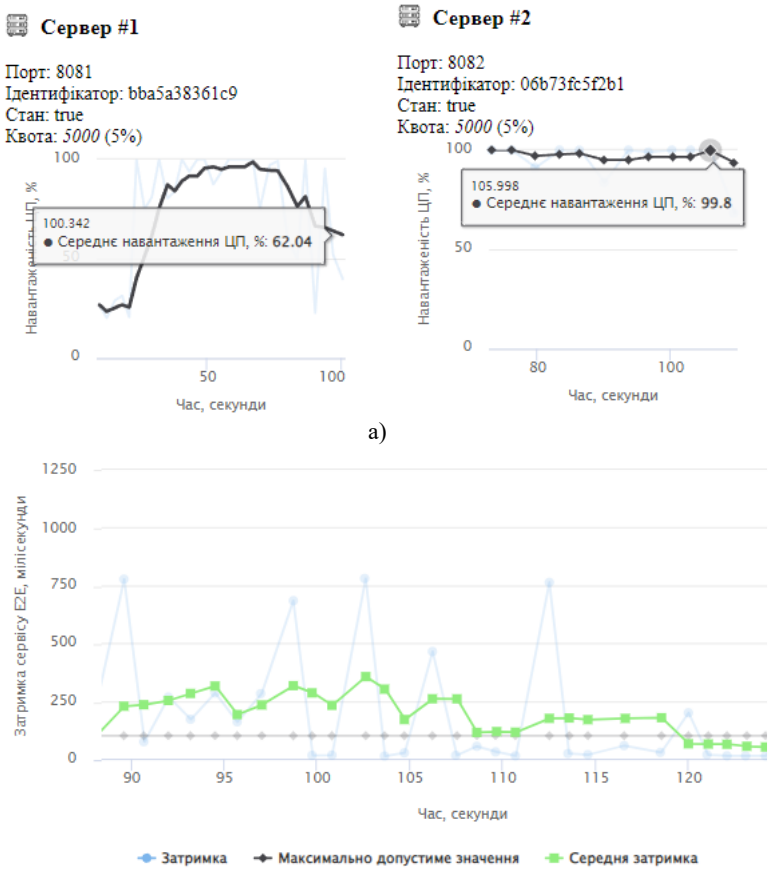


Рис. 5. Навантаження процесорів двох серверів при рівній нагрузці та затримка в цей момент часу

Алгоритм "round robin" є одним з найпоширеніших методів розподілу навантаження між серверами. Він працює шляхом циклічного розподілу запитів між серверами, незалежно від їхньої потужності. Однак дослідження показали, що цей алгоритм має суттєві недоліки, особливо при роботі з серверами різної потужності.

У ході досліджень було проведено тестування на трьох серверах з різною потужністю (рис. 6). Перший та другий сервери мали середню потужність, тоді як третій сервер був значно потужнішим. Результати показали, що алгоритм "round robin" не забезпечує рівномірного та ефективного розподілу навантаження. Перші два сервери були завантажені на 50.6% та 62.2% відповідно, тоді як третій сервер був завантажений лише на 41.6%. Це свідчить про значну нерівномірність у розподілі навантаження, що призводить до неефективного використання ресурсів потужнішого сервера.

Для вирішення цієї проблеми було впроваджено алгоритм "Weighted round robin", який враховує "вагу" кожного сервера. Вага визначається на основі їхньої потужності, що дозволяє більш ефективно розподіляти запити. Додаткове тестування (рис. 7а) показало, що після впровадження алгоритму "Weighted round robin" перші два сервери були завантажені на 20.7% та 30.2% відповідно, а третій сервер на 56.1%. Це означає, що навантаження на перші два сервери зменшилося на 29.9% та 32% відповідно, в той час як третій сервер став отримувати більше навантаження на 14.5%. Крім того, застосування алгоритму "Weighted round robin" покращило загальну продуктивність системи. Середня затримка зменшилась з 51 мс до 11.8 мс, що є значним покращенням (рис.7б). Це свідчить про те, що новий алгоритм не лише забезпечує більш рівномірне навантаження, але й підвищує швидкість обробки запитів.

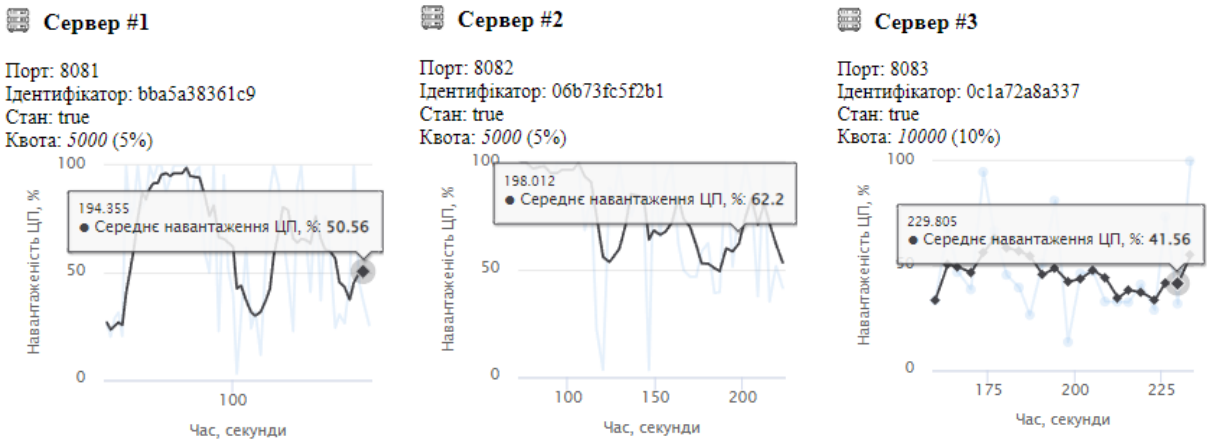


Рис. 6. Навантаження процесорів трьох серверів при однаковому навантаженні

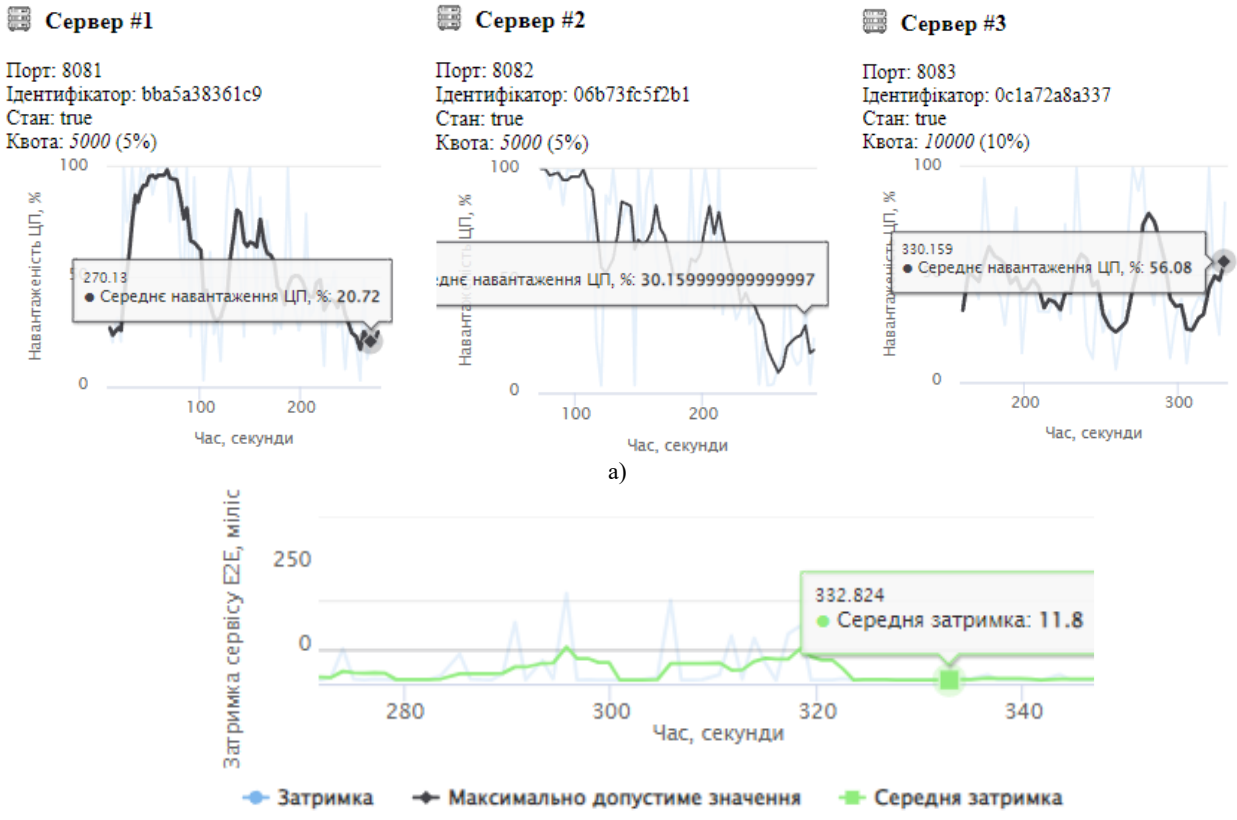


Рис. 7. Навантаження процесорів трьох серверів при більші нагрузці на третій (потужніший) сервер (а) та затримка в цей момент часу (б)

У сучасних умовах все більшої популярності набувають контейнери, які дозволяють швидко розгортати та масштабувати застосунки. Автоматичне масштабування контейнерів у поєднанні з алгоритмом "weighted round robin" може значно покращити продуктивність та надійність серверної інфраструктури. Завдяки автоматичному масштабуванню, система може динамічно реагувати на зміни навантаження, додаючи або видаляючи контейнери в залежності від поточних потреб. Це дозволяє оптимально використовувати доступні ресурси та забезпечувати високу доступність сервісів.

Автоматичне масштабування контейнерів працює шляхом моніторингу ключових метрик, таких як використання процесора, пам'яті або кількість запитів. Коли ці метрики перевищують задані пороги, система автоматично розгортає додаткові контейнери, щоб задовольнити підвищене

навантаження. Коли навантаження знижується, непотрібні контейнери видаляються, що дозволяє зекономити ресурси.

Висновок

У роботі розроблена сервісна платформа, яка дає можливість досліджувати процес обслуговування користувачів у розподілених сервісних системах шляхом моделювання інфраструктури довільної конфігурації та фактичних чи гіпотетичних сценаріїв функціонування розподілених систем. Використовучи розроблену сервісну платформу, у роботі проведено дослідження агоритмів балансування навантаження round robin та weighted round robin. Розроблена сервісна платформа має можливість розширення для інших нових алгоритмів балансування навантаження з метою поліпшення якості обслуговування користувачів у розподілених системах.

На основі отриманих результатів дослідження встановлено, що для розподілених систем рекомендується використовувати алгоритми балансування навантаження, які забезпечують рівномірне розподілення навантаження між вузлами системи. Алгоритм Weighted round robin виявився ефективним у випадках, коли різні вузли мають різну потужність обчислювальних ресурсів. Алгоритм Round robin показав гарні результати у сценарії, де всі вузли мають однакову обчислювальну потужність. Таким чином, застосування алгоритму "weighted round robin" є оптимальним підходом для розподілу навантаження в умовах різної потужності серверів. Він забезпечує більш ефективне використання ресурсів та покращує загальну продуктивність серверної інфраструктури. Поєднання цього алгоритму з автоматичним масштабуванням контейнерів дозволяє досягти ще вищого рівня ефективності, гнучкості та надійності, що є критично важливим для сучасних IT-систем.

Список використаних літературних джерел

[1] *S. N. Srirama, M. Adhikari, and S. Paul, "Application deployment using containers with auto-scaling for microservices in cloud environment," J. Netw. Comput. Appl., vol. 160, no. 102629, p. 102629, 2020. https://doi.org/10.1016/j.jnca.2020.102629* .

[2] *E. Casalicchio, "A study on performance measures for auto-scaling CPU-intensive containerized applications," Cluster Comput., vol. 22, no. 3, pp. 995–1006, 2019, doi: 10.1007/s10586-018-02890-1.*

[3] *R. Sampaio, I. Beschastnikh, D. Maier, D. Bourne and V. Sundaresen, "Auto-tuning elastic applications in production," 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Melbourne, Australia, 2023, pp. 355-367, doi: 10.1109/ICSE-SEIP58684.2023.00038.*

[4] *N. Cruz Coulson, S. Sotiriadis and N. Bessis, "Adaptive Microservice Scaling for Elastic Applications," in IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4195-4202, May 2020, doi: 10.1109/JIOT.2020.2964405.*

[5] *M. Klymash, V. Romanchuk, M. Beshley and P. Arthur, "Investigation and simulation of system for data flow processing in multiservice nodes using virtualization mechanisms," 2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON), Kyiv, UKraine, 2017, pp. 989-992, doi: 10.1109/UKRCON.2017.8100397.*

[6] *M. Beshley, V. Romanchuk, M. Seliuchenko and A. Masiuk, "Investigation the modified priority queuing method based on virtualized network test bed," The Experience of Designing and Application of CAD Systems in Microelectronics, Lviv, Ukraine, 2015, pp. 1-4, doi: 10.1109/CADSM.2015.7230779.*

[7] *T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," J. Grid Comput., vol. 12, no. 4, pp. 559–592, 2014, doi: 10.1007/s10723-014-9314-7.*

[8] *"Autoscaling," Microsoft.com. https://learn.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling (accessed Nov. 18, 2023).*

[9] *S. Taherizadeh and M. Grobelnik, "Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications," Adv. Eng. Softw., vol. 140, no. 102734, p. 102734, 2020, doi: 10.1016/j.advengsoft.2019.102734.*

[10]S. Taherizadeh, V. Stankovski and J. Cho, "Dynamic Multi-level Auto-scaling Rules for Containerized Applications," in The Computer Journal, vol. 62, no. 2, pp. 174-197, Feb. 2019, doi: 10.1093/comjnl/bxy043.

[11]J. Dogani, R. Namvar, and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," Comput. Commun., vol. 209, pp. 120–150, 2023, doi: 10.1016/j.comcom.2023.06.010.

DEVELOPMENT OF A PLATFORM FOR RESEARCHING AUTOMATIC CONTAINER SCALING AND LOAD BALANCING IN DISTRIBUTED SYSTEMS

H. Beshley, S. Bodnar, M. Seliuchenko, M. Beshley, M. Klymash

Lviv Polytechnic National University, S. Bandery Str., 12, 79013, Lviv, Ukraine

Quality of Service (QoS) is identified as a key task for distributed systems because meeting user needs is an important aspect of their successful functioning. Most container autoscaling solutions focus on resource optimization and cost management. However, these solutions often do not consider the dynamic user requirements for Quality of Service (QoS), resulting in delays in resource allocation and a decrease in service quality. Existing autoscaling and load balancing algorithms inadequately account for load dynamics, which is a significant issue. Moreover, traditional platforms for testing new algorithms, such as Azure and AWS, are commercial and closed, limiting opportunities for validating innovative approaches. Consequently, there is a need for open and accessible platforms that allow researchers and developers to effectively test and implement new load balancing and autoscaling algorithms. To address these issues, a new approach based on a deep understanding of resource usage context and user needs is required to ensure high service quality and improve the efficiency of distributed systems. The novelty of this work lies in the development of a new platform for researching container autoscaling methods and load balancing algorithms. The created virtualized service platform enabled practical assessment of the advantages and disadvantages of algorithms under real conditions. For example, using the "Round Robin" algorithm with a 50 ms request delay resulted in server loads of 96.2% and an average delay time of 679 ms. Implementing the "Weighted Round Robin" algorithm and container autoscaling reduced server loads to 56.1% and the average delay to 11.8 ms. The results obtained can form the basis for further development and implementation of algorithms in distributed systems, which will improve service quality and overall efficiency of these systems.

Keywords: *distributed systems, load balancing, virtualization, automatic scaling, architecture.*