DOI: https://doi.org/10.15407/kvt217.03.024

КОВАЛЬ Р.Ю., аспірант
https://orcid.org/0009-0004-1798-3872, e-mail: kovalr2000@gmail.com
Міжнародний науково-навчальний центр інформаційних технологій
та систем НАН України та МОН України,
просп. Акад. Глушкова, 40, Київ, 03187, Україна

МЕТОД ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Вступ. *Це дослідження є актуальним, оскільки високонавантажені системи в наш час є левовою часткою всіх розробок у сфері інформаційних технологій, бо можуть одночасно підтримувати велику кількість запитів від користувачів, обробляти великі обсяги даних та здійснювати складні обчислення, є високоефективними, їх просто змінювати, додавати новий функціонал, вони забезпечують гарантії безпеки інформації користувачів та підтримують масштабування. Чим швидше вони збільшуються, тим складніше контролювати інфраструктурні ресурси. Коли система отримує приріст аудиторії — відповідно зростає частота і кількість запитів. З цього випливає, що чим більше запитів — тим більшого масштабування потребує система. Отже, високонавантажені системи — це системи, які потрібно весь час масштабувати, правильно підбирати інфраструктуру і в цілому загальні архітектурні концепції. У цьому і полягає складність реалізації таких рішень, але з перспективи бізнесу — це вартує докладених зусиль.*

Мета. *Розроблення способу підвищення ефективності високонавантажених систем на рівні архітектурних рішень.*

Методи. *Інформаційно-аналітичне дослідження, математичне моделювання та алгоритмічний аналіз підходів до підвищення ефективності високонавантажених систем.*

Результати. *Для розроблення методу підвищення ефективності розглянуто теоретичне підґрунтя видів архітектури високонавантажених систем. Проведено порівняльний аналіз наявних архітектурних підходів таких сучасних систем. На основі принципів контейнеризації та оркестрування даних застосунків було запропоновано використовувати додатковий модифікований проксі шар для обміну даними, що дає змогу зменшити час оброблення великої кількості запитів.*

Висновки. *Розроблений спосіб підвищення ефективності високонавантаженої системи на основі мікросервісної архітектури дає змогу розгортати та масштабувати в хмарі складні системи програмного забезпечення.*

Ключові слова: *запит, ефективність, високонавантажений, інфраструктура, архітектура, масштабування, мікросервіс, контейнер, проксі, сервер, хмарні технології.*

ВСТУП

У сучасному цифровому світі вимоги до ефективності та безпеки систем зросли. Завдяки розвитку електронної комерції, онлайн-ігор та інших веб-застосунків системи з великим навантаженням стають все важливішими в нашому щоденному житті. Ці системи мають бути здатними до оброблення великих обсягів трафіку та даних, а також гарантувати швидкий і безпечний досвід користувача.

Що таке високонавантажена система? Це та система, яка може витримати велике одночасне зростання запитів і при цьому забезпечити однаково високу якість обслуговування для всіх користувачів, незалежно від їх кількості — чи це буде десять або мільйони. Що ж, збільшення користувачів може призвести до більшого прибутку для системи, однак водночас більша кількість часу, протягом якого система недоступна, може призвести до втрати потенційного доходу для компанії. Прикладами таких великих систем можуть бути YouTube, Amazon, Google, Netflix, Facebook.

Одним із наслідків цього є зростання потреби в ефективних методах оптимізації систем з великим навантаженням. Розроблення таких методів стає ключовим для забезпечення надійності та доступності цих систем. Залежність від таких систем робить цю тему надзвичайно актуальною як для промисловості, так і для наукової спільноти.

У статті розглядаються різні аспекти збільшення ефективності високонавантажених систем. Аналізуються такі важливі теми, як масштабованість, кешування, балансування навантаження, інструменти моніторингу та архітектурні шаблони мікросервісів. Вивчаючи ці аспекти, ми отримуємо глибше розуміння проблем і можливостей, пов’язаних із системами високого навантаження, і навчаємося розробляти та впроваджувати ефективні рішення**.**

Крім того, стаття має метою вивчення еволюції технологій та підходів у цій сфері. З урахуванням постійного розвитку хмарних обчислень, периферійних обчислень і розподілених систем [1], важливо бути обізнаними щодо останніх розробок як для практиків, так і для дослідників. Досліджуючи зазначені теми, стаття спрямована на викладення усталених найкращих практик, а також новітніх тенденцій.

Отже, в статті розглянуто цю тему як з академічної точки зору, так і як дослідження, яке має практичне застосування. Це сприятиме обміну знаннями між академічними колами та промисловістю, що допоможе розробити інноваційні рішення, відповідні потребам сучасного цифрового світу.

Висновки статті стануть корисними для розробників програмного забезпечення та архітекторів бізнес рішень. Отримана інформація буде також важливою для підприємств і організацій, які покладаються на високонавантажені системи для підтримки своїх операцій і задоволення потреб своїх клієнтів.

ПОСТАНОВКА ЗАВДАННЯ

Високонавантажені системи є масштабними IT-системами, які мають оптимально обробляти значну кількість вхідних запитів та трафіку, часто у реальному часі або з найменшою затримкою. Ці системи мають бути здатними до оброблення великої кількості одночасних запитів від користувачів, транзакцій та операцій оброблення даних, зберігаючи високий рівень ефективності, доступності та масштабованості.

Однією з ключових проблем, які виникають під час розроблення таких систем, є забезпечення стабільної роботи систем за великих навантажень. Постійне збільшення кількості користувачів та обсягів даних вимагає ефективних рішень для оптимізації використання ресурсів та швидкого реагування на запити.

Безпека є ще однією серйозною проблемою для високонавантажених систем. Зі зростанням кількості користувачів та обсягів даних зростають і ризики кібератак, витоку даних та інших загроз для безпеки. Ефективні методи захисту, такі як автентифікація, авторизація та шифрування, є важливими складовими частинами проєктування та розроблення високонавантажених систем.

Зрештою, проблема масштабованості відіграє ключову роль. Зростання обсягів даних та трафіку вимагає ефективних методів масштабування, щоб забезпечити стабільну та безперервну роботу системи навіть під час зростання її обсягів.

Усі ці проблеми становлять основу для проведення досліджень з метою розроблення та впровадження ефективних методів підвищення продуктивності та безпеки високонавантажених систем.

Зі зростанням будь-якої системи з'являються нові сервіси, утиліти та компоненти, між якими складно налаштовувати зв'язки та пам'ятати всі нюанси. Чим більше функціоналу, тим більше можливостей для помилок або критичних місць за високого навантаження. Дуже часто комунікація між компонентами через мережу або доступ до бази даних стають такими критичними точками (bottleneck) [2].

У зв'язку зі зростанням популярності інтернету та цифрових технологій підприємства та організації стали розробляти нові програми та послуги, які потребують оброблення запитів від великої кількості користувачів, транзакцій та операцій оброблення даних.

Спочатку високонавантажені системи використовувалися в основному великими корпораціями та державними установами для підтримки внутрішніх операцій, таких як нарахування заробітної плати та управління людськими ресурсами, фінансової звітності та аналізу даних.

З плином часу, з поширенням інтернету та появою онлайн-бізнесів, системи з високим навантаженням стали невід'ємною частиною електронної комерції, онлайн-ринків, соціальних мереж та інших програм, які потребують оброблення великих обсягів трафіку та даних.

Технології, які використовують для створення високонавантажених систем, постійно розвиваються та вдосконалюються. Разом із розробленням нових апаратних і програмних рішень, таких як хмарні обчислення, віртуалізація та контейнеризація [3], вони спрощують проєктування, створення та керування системами з високим навантаженням.

Сьогодні високонавантажені системи широко використовуються в різних галузях промисловості та застосуваннях, що є важливими для багатьох підприємств і організацій. Цим компаніям потрібно обробляти великі обсяги трафіку та даних ефективно та результативно. З розвитком інтернету та інших цифрових технологій високонавантажені системи стають ще більш важливими, оскільки компанії продовжують розробляти нові застосунки та послуги, які потребують оброблення великих обсягів даних і трафіку майже в реальному часі. Наприклад, на (Рис. 1) наведено архітектурну діаграму для розробки високонавантаженої системи банкінгу на основі AWS [4].

Деякі типові випадки використання високонавантажених систем охоплюють:

1. роздрібну торгівлю в інтернеті: вебсайти електронної комерції та онлайн-ринки, які обробляють велику кількість замовлень, платежів і взаємодії з клієнтами;
2. соціальні медіа: вебсайти та програми соціальних мереж, які обслуговують велику кількість профілів користувачів, публікацій і взаємодій;
3. ігри: багатокористувацькі онлайн-ігри, які обслуговують велику кількість гравців і ігрових сеансів;
4. фінансові послуги: вебсайти банківських і фінансових послуг та програми, які обробляють велику кількість транзакцій, платежів і взаємодій з клієнтами;
5. охорону здоров'я: системи електронних медичних записів і телемедичні платформи, які обробляють великі обсяги даних пацієнтів і взаємодій;
6. транспорт: додатки для спільного використання поїздок і системи керування транспортом, які обробляють велику кількість транспортних засобів і пасажирських запитів.

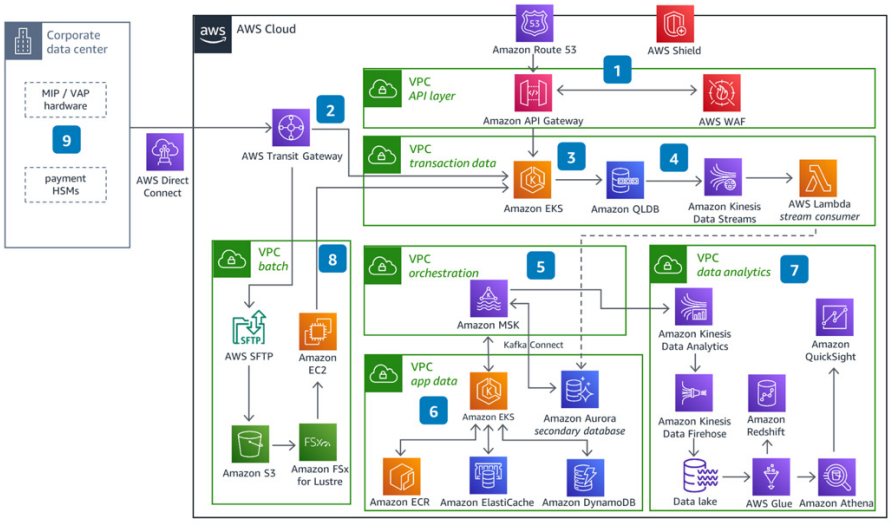

Рис. 1. Схема архітектури системи банкінгу в AWS

Також до прикладів високонавантажених систем можна віднести такі платформи як сервіс купівлі та доставки Amazon, відеоплатформа YouTube, соцмережі Facebook, Instagram, Twitter тощо.

Для відповіді на вимоги систем з високим навантаженням необхідно, щоб системи були розроблені та оптимізовані для забезпечення ефективності, масштабованості та доступності. Це зазвичай передбачає використання методів, таких як балансування навантаження, кешування та розподіл даних для управління вхідними запитами, а також розгортання кількох серверів, баз даних та інших ресурсів для забезпечення надійності та ефективності. Крім того, системи з високим навантаженням повинні бути безпечними, захищаючи конфіденційні дані та гарантуючи конфіденційність користувачів і клієнтів.

Загалом, системи з високим навантаженням відіграють вирішальну роль у багатьох сферах сучасного суспільства, даючи змогу компаніям, урядам та іншим організаціям ефективно обробляти великі обсяги даних, трафіку та транзакцій, а також забезпечувати високоякісну взаємодію з користувачами та клієнтами.

Метою роботи є розроблення способу підвищення ефективності високонавантажених систем на рівні архітектурних рішень.

ТЕОРЕТИЧНЕ ПІДГРУНТЯ СИСТЕМ НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Мікросервісна архітектура (MSA) [5] полягає у розбитті застосунку на невеликі незалежні сервіси, які взаємодіють з допомогою чітко визначених API-інтерфейсів. Цей підхід передбачає, що кожен сервіс відповідає за певну бізнес-можливість (Рис. 2), наприклад, автентифікацію користувачів або оброблення платежів. Кожен мікросервіс має свою власну базу коду, базу даних та інфраструктуру, що дає їм змогу працювати незалежно. MSA є розвитком сервіс-орієнтованої архітектури (SOA), де програма розділена на менші незалежні сервіси. Однак, в MSA кожен сервіс зазвичай працює у власному процесі і спілкується з іншими через мережу. Ця архітектура надає системі масштабованості, гнучкості та стійкості, що робить її добре придатною для систем із високим навантаженням. Основна перевага MSA полягає у здатності кожного сервісу працювати незалежно, що полегшує розгортання, масштабування [6] та розвиток системи.

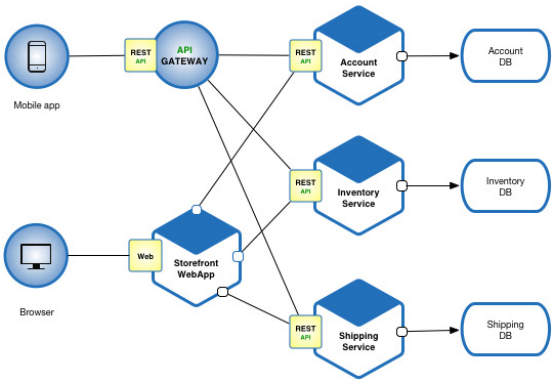

Рис. 2. Схема принципу мікросервісної архітектури

Особливостями архітектури мікросервісів є [7]:

Деякі недоліки та виклики, пов'язані з архітектурою мікросервісів, охоплюють:

1. підвищену складність керування: збільшена кількість невеликих сервісів може призвести до підвищення складності керування розгортанням, тестуванням та моніторингом системи;
2. накладні витрати на мережевий зв'язок: багато мікросервісів потребують інтенсивної комунікації через мережу, що може збільшити затримки та витрати на мережевий зв'язок;
3. складність розподіленої системи: високий рівень розподіленості може ускладнити забезпечення належної взаємодії між усіма компонентами системи;
4. операційні витрати: збільшена кількість сервісів може збільшити операційні витрати на розгортання, масштабування та моніторинг;
5. узгодженість даних: складність забезпечення узгодженості даних може зростати, оскільки вони розподілені між багатьма сервісами;
6. складність тестування: незалежність та велика кількість сервісів можуть зробити тестування більш складним та трудомістким;
7. гранулярність сервісів: визначення правильного рівня деталізації для мікросервісів може бути складним завданням, оскільки вони повинні бути достатньо маленькими для ефективного управління, але не надто маленькими, щоб не перевантажувати систему.

Дійсно, архітектура мікросервісів стала дуже популярною у сучасному комерційному світі через свої переваги у створенні та підтримці складних програмних застосунків. Даючи змогу розбивати застосунки на менші, незалежні компоненти, мікросервіси дають змогу організаціям ефективніше розробляти, розгортати та модифікувати застосунки.

У системах з високим навантаженням, де потрібно обробляти великі обсяги даних та транзакцій, архітектура мікросервісів особливо корисна, оскільки вона дає змогу гнучко масштабувати окремі компоненти системи залежно від потреб та навантаження. Це полегшує розподілення навантаження та забезпечує ефективніше використання ресурсів.

Основні переваги мікросервісної архітектури охоплюють підвищену модульність, децентралізацію, незалежність, масштабованість та стійкість. Ці характеристики роблять її привабливим вибором для розроблення систем з високим навантаженням у сучасному комерційному середовищі.

ЗАГАЛЬНИЙ АЛГОРИТМ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Цей підхід, який передбачає використання окремого рівня інфраструктури для керування міжсервісним зв'язком і потоком трафіку, є дієвим способом полегшення розроблення, розгортання та підтримки систем з великим навантаженням. Він спрощує комунікацію між сервісами, полегшує вирішення проблем, пов'язаних з навантаженням та маршрутизацією трафіку, і забезпечує більші надійність та ефективність системи. Крім того, використання проксі-серверів дає змогу керувати мережевим трафіком і кешувати ресурси, що може покращити швидкодію та продуктивність системи [8].

Рис. 3. Принцип роботи проксі

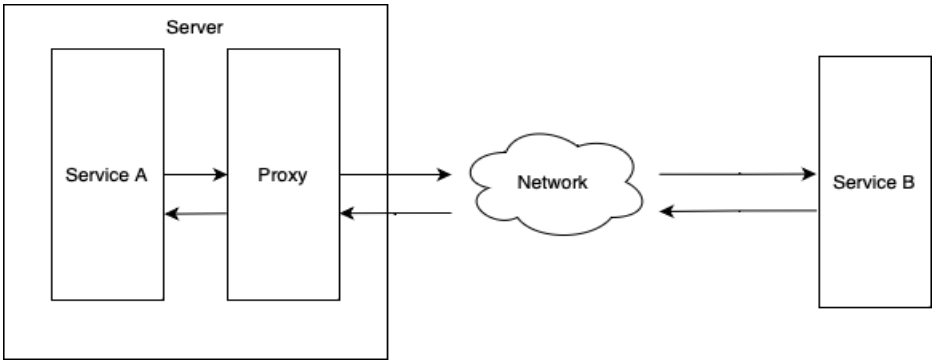

Рис. 4. Схема базової взаємодії сервісів через проксі

Виділення логіки взаємодії між компонентами з виведенням на окремий рівень інфраструктури дає змогу кожному сервісу фокусуватися на своїй основній функційності, замість того, щоб вирішувати проблеми, пов'язані з маршрутизацією та керуванням трафіком. Це полегшує розроблення та підтримку сервісів, а також дає змогу швидше реагувати на зміни та вдосконалення в системі.

У підсумку, використання окремого рівня інфраструктури для керування міжсервісним зв'язком і потоком трафіку, а також використання проксі-серверів, є ефективним способом полегшення розроблення та підтримки високонавантажених систем. Він допомагає забезпечити більшу масштабованість, надійність та ефективність системи в цілому.

Рисунок 3 відображає базову взаємодію сервісів через проксі шар у системі мікросервісів. На ньому рисунку видно, що кожен сервіс має свій власний проксі, через який відбувається вся маршрутизація, взаємодія, налаштування та моніторинг.

Проксі служить посередником між клієнтами і самим сервісом, приймаючи запити від клієнтів і пересилаючи їх до відповідного сервісу. Це дає змогу вивести логіку маршрутизації та взаємодії на окремий рівень інфраструктури, що полегшує розроблення та підтримку системи.

Крім того, така архітектура дає змогу легко масштабувати систему, оскільки можна додавати або видаляти проксі-сервери залежно від навантаження та потреб системи. Це забезпечує більшу гнучкість та ефективність керування мережевим трафіком та взаємодією сервісів.

Ідея використання проксі-сервісу полягає у тому, що сервіс А спочатку направляє свій запит до проксі замість безпосереднього виклику сервісу В, як показано на рисунку 4. Це забезпечує більш гнучке керування мережевим трафіком та взаємодією сервісів.

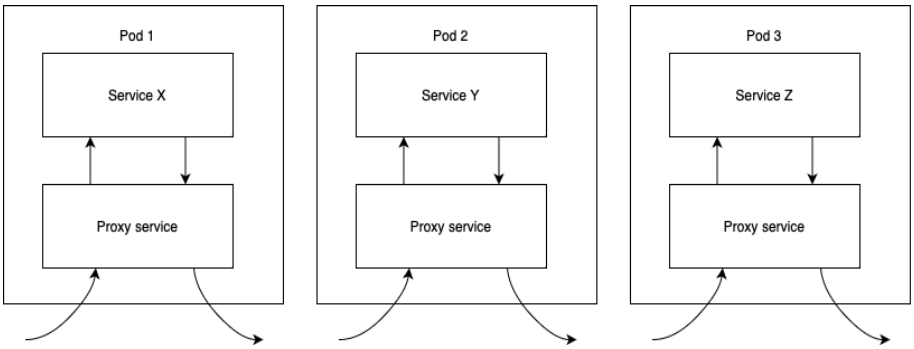

Рис. 5. Приклад можливої комунікації декількох сервісів

Оскільки і сервіс А і проксі-сервіс знаходяться в одному контейнері або процесі, можна здійснити виклик на локальний хост через зворотний мережевий інтерфейс (loopback network interface). Після отримання запиту проксі-сервіс виконує виклик сервісу В через зовнішній мережевий адаптер серверу. Подібно, коли сервіс В надсилає відповідь, вона повертається до сервісу А через проксі-сервіс.

У системі з багатьма сервісами взаємодія виглядатиме як показано на рисунку 5. Важливою рисою цього підходу є те, що проксі-сервіс обробляє і вхідний і вихідний трафік, що дає змогу керувати всією взаємодією сервісів у системі.

Введення проксі-менеджера до системи дасть змогу ефективно керувати шаром проксі-сервісів і забезпечити централізоване управління мікросервісами. Проксі-менеджер виступає як централізований центр управління для всіх мікросервісів у системі, де його основна мета полягає в керуванні трафіком між ними.

Основні функції проксі-менеджера охоплюють:

1. керування конфігурацією: проксі-менеджер зберігає конфігурації та налаштування проксі-сервісів, що дає змогу легко керувати їхньою поведінкою та параметрами;
2. моніторинг та керування: він відповідає за моніторинг роботи мікросервісів та проксі-сервісів і забезпечує їхню стабільну роботу. У разі виявлення проблем або збоїв проксі-менеджер може автоматично вживати відповідних заходів для відновлення роботи системи;
3. маршрутизацію трафіку: проксі-менеджер відповідає за маршрутизацію трафіку між мікросервісами, що дає змогу ефективно керувати потоками даних та оптимізувати їхню взаємодію;
4. балансування навантаження: він забезпечує рівномірний розподіл трафіку між різними екземплярами мікросервісів для підвищення надійності та ефективності системи.

Проксі-менеджер динамічно адаптується до змін у середовищі мікросервісів, спрощуючи процес додавання нових сервісів або виявлення та відновлення неполадок у наявних сервісах. Це дає змогу забезпечити надійну та ефективну роботу системи в умовах змін і навантаження.

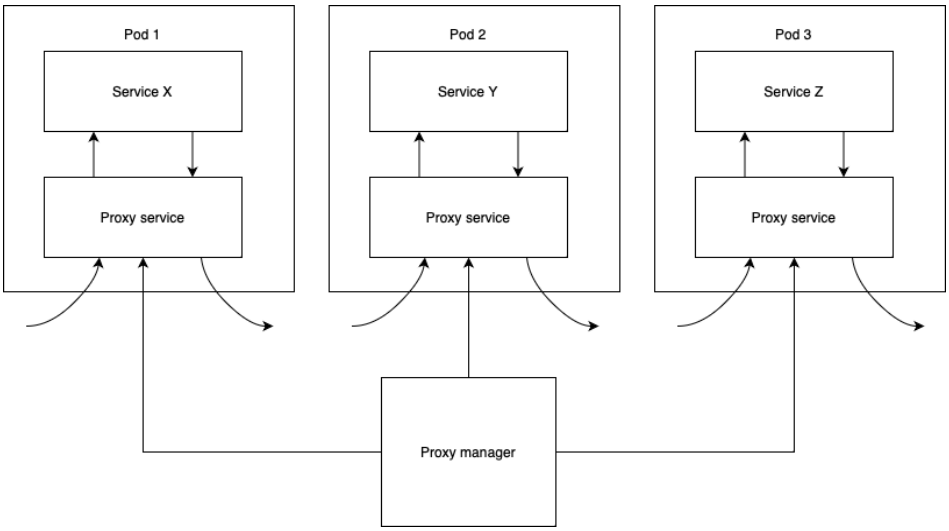

Рис. 6. Схема системи з проксі-менеджером

Описаний підхід з використанням проксі-менеджера і проксі-сервісів (Рис. 6) може ефективно інтегрувати в себе різноманітні властивості для збільшення ефективності роботи системи. Розглянемо, які саме властивості можна інтегрувати:

1. масштабування: підхід з використанням проксі-сервісів дає змогу легко масштабувати окремі компоненти системи, оскільки кожен сервіс може мати свій власний екземпляр проксі-сервісу, який обробляє трафік. Це дає змогу гнучко розподіляти ресурси та реагувати на зміни в навантаженні;
2. балансування навантаження [9]: проксі-менеджер може відповідати за балансування навантаження між різними екземплярами проксі-сервісів або мікросервісів, що дає змогу рівномірно розподілити трафік і забезпечити оптимальну продуктивність системи;
3. кешування запитів: проксі-сервіси можуть використовувати кешування для збереження результатів попередньо оброблених запитів і швидкого доступу до них у майбутньому. Це допомагає зменшити час відповіді на запити і покращує продуктивність системи;
4. шаблон «Вимикач» [10]: проксі-менеджер може використовувати шаблон «Вимикач» для автоматичного виявлення та відключення непрацюючих або несправних сервісів. Це дає змогу уникнути витрат ресурсів на взаємодію з недоступними компонентами та забезпечує неперервну роботу системи;
5. моніторинг та збір метрик: проксі-менеджер може вести моніторинг роботи мікросервісів та проксі-сервісів і збирати метрики про їхню продуктивність та стан. Це дає змогу виявляти проблеми та оптимізувати роботу системи для досягнення більшої ефективності.

Інтеграція цих властивостей дасть змогу створити потужну та ефективну систему, яка забезпечить високу продуктивність та надійність у роботі навіть в умовах інтенсивного навантаження.

Ці переваги, що надаються застосуванням проксі-шару в архітектурі мікросервісів, є дійсно важливими з точки зору розроблення та ефективності системи. Ще раз окреслимо їх:

1. спрощена комунікація: застосування проксі шару дає змогу розробникам абстрагуватися від складності мережевої комунікації між сервісами. Це дає змогу їм зосередитися на написанні бізнес-логіки та функціональності, замість вирішення проблем мережевої взаємодії;

2. покращена простежуваність: забезпечення видимості мережевого трафіку між сервісами дає змогу розробникам легше відстежувати та аналізувати проблеми, які виникають в системі. Це допомагає забезпечити швидке виявлення та усунення неполадок;

3. гнучкість: застосування проксі-шару дає змогу розробникам легко переключатися між різними комунікаційними протоколами та технологіями, не потребуючи змін у коді програми. Це забезпечує гнучкість та можливість швидкого реагування на зміни у вимогах бізнесу;

4. узгодженість: проксі шар забезпечує послідовне застосування налаштувань конфігурації до всіх сервісів у системі, незалежно від їх мови або технологічного стеку. Це дає змогу забезпечити єдність та стандартизацію керуванння та конфігурації.

Загалом, проксі-шар в архітектурі мікросервісів сприяє полегшенню розробки, гнучкості та узгодженості системи, що є критичними аспектами у розробленні та експлуатації складних застосунків.

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

Проведене тестування [11] зосереджувалося на порівнянні ефективності двох систем. Обидві системи спроектовані на основі мікросервісної архітектури. Перша система використовувала запропонований метод підвищення ефективності високонавантажених систем, тоді як друга система була більш стандартизованою і не використовувала цей метод.

Тестування проводилося на різних рівнях навантаження, що починалися з 1-го запиту на секунду і досягали 14000 запитів на секунду. Результати цього порівняння задокументовано і подано у відповідних табл. 1–2.

Таблиця 1. Оброблення запитів високонавантаженої системи із застосуванням запропонованого підходу

Кількість запитів за секунду	**Час оброблення (мс)**	**Час оброблення (с)**
1	890.926	0.890
10	4502.541	4.502
100	25647.125	25.647
500	105167.841	105.167
1000	190520.877	190.521
2000	270123.146	270.123
5000	635125.591	635.125
10000	830564.629	830.564
12000	959340.284	959.340
14000	1111591.491	1111.591

Таблиця 2. Оброблення запитів високонавантаженої системи без застосування запропонованого підходу

Кількість запитів за секунду	Час оброблення (мс)	Час оброблення (с)
1	90.176	0.091
10	881.968	0.881
100	9164.125	9.164
500	46917.578	46.917
1000	92467.987	92.467
2000	240641.095	240.641
5000	670063.153	670.063
10000	950842.592	950.842
12000	1153391.012	1153.391
14000	1434535.012	1434.535

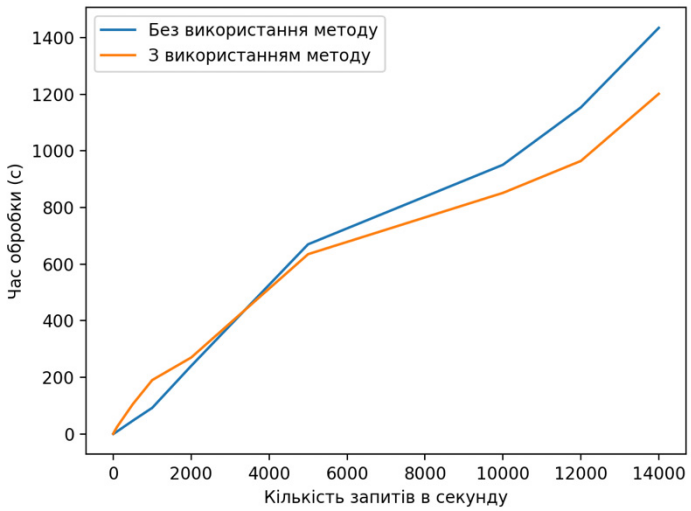

Рис. 7. Графік порівняння часу оброблення запитів систем

На графіку (Рис. 7), видно, що запропонований метод підвищення ефективності починає діяти за великої кількості запитів. Зокрема, починаючи з межі від 4000 запитів за секунду, час оброблення запитів починає зменшуватися порівняно з традиційною системою. Це свідчить про те, що запропонований метод діє ефективно та може покращити продуктивність системи за великого обсягу запитів.

Отримане підвищення ефективності, яка полягає у швидкості оброблення запитів системи, зростає в межах від 5,9 % (для 5000 запитів) до 22,5 % (для 14000 запитів). Однак спостерігається погіршення результатів для кількості запитів менше 5000 за секунду, особливо це помітно при 1-му запиті за секунду, коли традиційний підхід справляється краще майже у вісім разів. З цього можна зробити висновок, що запропонований метод є доцільним тільки для дійсно навантаженої системи, а не для ситуацій, коли користувачі роблять кілька запитів за одну секунду.

Таблиця 3. Зміни ефективності

Кількість запитів за секунду	Зміна в ефективності (%)
1	-778.02197802
10	-311.01021566
100	-79.86687036
500	-24.15542341
1000	-6.04215558
2000	-12.2514451
5000	+5.96033507
10000	+12.64963054
12000	+17.6046978
14000	+22.51210323

Для зручності аналізу наведемо табл. 3, в якій відображено, як змінюється ефективність запропонованого методу за різної кількості запитів. В таблиці час оброблення запитів з застосуванням методу порівнюється з часом оброблення без його застосування, а також додається знак "+" або "-", щоб вказати на збільшення чи зменшення ефективності відповідно.

Підвищення ефективності відбувається завдяки введенню додаткового шару мережевої інфраструктури, який керує трафіком між мікросервісами та контролює їх комунікацію. Це дає змогу зменшити навантаження на самі мікросервіси, оскільки вони більше не відповідають за управління трафіком. У результаті система може обробляти більше запитів за секунду, що забезпечує швидшу обробку всього потоку запитів порівняно з використанням тієї ж системи без цього шару.

Однак очевидним є те, що цей підхід програє стандартному при невеликому навантаженні, наприклад, коли кількість запитів менше 4000 за секунду. Наприклад, оброблення одного запиту може бути в 8 разів повільнішою. Це свідчить про те, що цей спосіб підвищення ефективності може застосовуватися переважно у високонавантажених системах.

ВИСНОВКИ

На основі наявних технологій, методів та підходів до побудови високонавантажених систем на основі мікросервісної архітектури запропоновано спосіб покращення ефективності таких систем. Він полягає в додаванні нового інфраструктурного рівня у вигляді проксі-шару в кожен контейнер мікросервісу, який поєднуватиме в собі загальноприйняті шаблони програмування мікросервісних систем, такі як балансування навантаження, маршрутизацію трафіку, алгоритми повторних оброблень запитів, кешування запитів тощо. Окрім цього пропонується додати окремий сервіс з централізованим керуванням та налаштуванням. Очікується, що використання такого способу в сучасних IT-системах надасть змогу підвищити ефективність їх роботи.

REFERENCES

1. Shabani I. et al. Design of modern distributed systems based on microservices architecture *International Journal of Advanced Computer Science and Applications*. 2021, V .12, №. 2. https://doi.org/10.14569/IJACSA.2021.0120220
2. R. M. Munaf, J. Ahmed, F. Khakwani, T. Rana, "Microservices Architecture: Challenges and Proposed Conceptual Design. 2019 *International Conference on Communication Technologies (ComTech)*, 2019, pp. 82–87. https://doi.org/10.1109/COMTECH.2019.8737831
3. M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, M. Steinder, Performance Evaluation of Microservices Architectures Using Containers. 2015, *IEEE 14th International Symposium on Network Computing and Applications, Cambridge*, 2015, pp. 27–34. https://doi.org/10.1109/NCA.2015.49
4. Waseem M., Liang P. Microservices architecture in DevOps. *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*. IEEE, 2017, pp. 13–14. https://doi.org/10.1109/APSECW.2017.18
5. Richardson C. Microservices patterns: with examples in Java. Simon and Schuster, 2018.
6. Ghofrani J., Lübke D. Challenges of Microservices Architecture: A Survey on the State of the Practice. *ZEUS*, 2018, V. 2018, pp. 1–8.
7. Gan Y., Delimitrou C. The architectural implications of cloud microservices. *IEEE Computer Architecture Letters,* 2018, V. 17, №. 2, pp. 155–158. https://doi.org/10.1109/LCA.2018.2839189
8. Khazaei H. Efficiency analysis of provisioning microservices. *2016 IEEE International conference on cloud computing technology and science (CloudCom)*. IEEE, 2016, pp. 261–268. https://doi.org/10.1109/CloudCom.2016.0051
9. Niu Y., Liu F., Li Z. Load balancing across microservices. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 198–206. https://doi.org/10.1109/INFOCOM.2018.8486300
10. Molchanov H., Zhmaiev A. Circuit breaker in systems based on microservices architecture. 2018. https://doi.org/10.20998/2522-9052.2018.4.13
11. Aderaldo C. M. et al. Benchmark requirements for microservices architecture research. 2017 IEEE. *ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE).* IEEE, 2017, pp. 8–13. https://doi.org/10.1109/ECASE.2017.4

Received 20.05.2024

Koval R.Yu., PhD Student,
https://orcid.org/0009-0004-1798-3872, e-mail: kovalr2000@gmail.com
International Research and Training Center for Information

Technologies and Systems of the National Academy of Sciences
of Ukraine and the Ministry of Education and Science of Ukraine,
40, Akad. Glushkov av., Kyiv, 03187, Ukraine

METHOD OF INCREASING THE EFFICIENCY OF HIGH-LOAD SYSTEMS BASED ON MICROSERVICES ARCHITECTURE

Introduction. *This study is relevant for the following reasons: high-load systems nowadays occupy the lion's share of all developments in the field of information technology, because they can simultaneously support a large number of requests from end users, process large amounts of data and perform complex calculations, are highly efficient, easy to change, add new functionality, provide security guarantees for user information and support scaling. The faster they grow, the harder it is to control infrastructure resources. When the system receives an increase in the audience, the frequency and number of requests increases accordingly. It follows that the more requests, the more scaling the system needs. Thus,*

highly loaded systems are systems that need to be scaled all the time, with the right infrastructure and overall architectural concepts. This is the complexity of implementing such solutions, but from a business perspective, it is worth the effort.

***The purpose of the paper** is to develop a method of increasing the efficiency of high-load systems at the level of architectural solutions.*

***Methods.** Information-analytical research, mathematical modeling and algorithmic analysis of approaches to improving the efficiency of high-load systems.*

***Results.** In order to develop a method for improving efficiency, the theoretical basis of the types of architecture of high-load systems is considered. A comparative analysis of the existing architectural approaches of such modern systems is carried out. Based on the principles of containerization and orchestration of application data, it was proposed to use an additional modified proxy layer for data exchange to reduce the processing time of a large number of requests.*

***Conclusions.** A method for improving the efficiency of a highly loaded system based on a microservice architecture has been developed. Using this method will allow better deployment and scaling of complex software systems in the cloud.*

***Keywords:** request, efficiency, high-load, infrastructure, architecture, scaling, microservice, container, proxy, server, cloud technologies.*