Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: **Методи та засоби балансування трафіку між сервісами в service mesh на платформі Istio**

Виконав: студент(ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Дерягін В.Л. (прізвище та ініціали)

Керівник (підпис) Луцик Н.С. (прізвище та ініціали)

Нормоконтроль (підпис) Тиш Є.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) Литвиненко Я.В. (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«17» листопада 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Дерягіну Віктору Леонідовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи та засоби балансування трафіку між сервісами в service mesh на платформі Istio*

Керівник роботи *Луцик Надія Степанівна, к.т.н., доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *14* » *листопада 2025* року № 4/7-987

2. Термін подання студентом завершеної роботи *16 грудня 2025 р.*

3. Вихідні дані до роботи *Принципи організації мікросервісної архітектури, архітектура та компоненти service mesh, алгоритми балансування навантаження, механізми маршрутизації трафіку в Istio, методи оцінки продуктивності розподілених систем*

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ
1 Аналіз архітектури мікросервісів та технологій service mesh
2 Методика дослідження ефективності алгоритмів балансування навантаження
3 Експериментальне дослідження алгоритмів балансування в Istio service mesh
4 Охорона праці та безпека в надзвичайних ситуаціях
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

1. Актуальність і мета дослідження.
2. Завдання, об'єкт і предмет, наукова новизна і практична цінність дослідження.
3. Діаграма архітектури роботи Istio.
4. Схема тестового середовища.
5. Етапи проведення експерименту.
6. Результати проведеного експерименту
7. Порівняльний аналіз алгоритмів
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., зав.каф. КС*	*04.12.2025*	
Безпека в надзвичайних ситуаціях	*Стручок В.С., ст. викладач каф. ОХ*		

7. Дата видачі завдання *17.11.2025р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Огляд літературних джерел. Постановка завдання на кваліфікаційну роботу.*	*17.11.2025р. – 23.11.2025р.*	*Викон.*
2.	*Робота над другим розділом, присвяченим формалізації критеріїв ефективності маршрутизації трафіку та розробці методики експериментального дослідження алгоритмів балансування навантаження в середовищі Istio service mesh.*	*24.11.2025р. – 02.12.2025р.*	*Викон.*
3.	*Робота над третім розділом, присвяченим проєктуванню експериментального середовища, проведенню навантажувальних тестів та аналізу результатів дослідження алгоритмів балансування навантаження в мікросервісній системі.*	*03.12.2025р. – 10.12.2025р.*	*Викон.*
4.	*Охорона праці та безпека в надзвичайних ситуаціях*	*04.12.2025р. – 10.12.2025р.*	*Викон.*
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*11.12.2025р. – 19.12.2025р.*	*Викон.*
6.	*Попередній захист кваліфікаційної роботи магістра*	*16.12.2025р.*	*Викон.*
7.	*Захист кваліфікаційної роботи магістра*	*22.12.2025р.*	*Викон.*

Студент _____ *Дерягін В.Л.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Луцик Н.С.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дерягін В.Л. Методи та засоби балансування трафіку між сервісами в service mesh на платформі Istio: робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп'ютерна інженерія / наук.кер. Луцик Н.С. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2025.

Ключові слова: мікросервісна архітектура, service mesh, Istio, балансування навантаження, маршрутизація трафіку, продуктивність, латентність.

Кваліфікаційна робота присвячена дослідженню ефективності взаємодії мікросервісів у розподілених програмних системах з використанням service mesh платформи Istio. У роботі виконано аналіз архітектури мікросервісних систем, розглянуто концепцію service mesh та проведено порівняльний огляд існуючих реалізацій. Основну увагу зосереджено на механізмах управління трафіком в Istio та алгоритмах балансування навантаження.

У межах роботи спроєктовано експериментальне середовище на базі Kubernetes кластера з Istio service mesh та розроблено методику навантажувального тестування з різними профілями навантаження. Проведено експериментальне дослідження алгоритмів round-robin, least connections та random з використанням метрик латентності, пропускної здатності, частоти помилок і рівномірності розподілу запитів. Отримані результати дозволили сформулювати практичні рекомендації щодо вибору алгоритму балансування навантаження залежно від характеристик мікросервісної системи та умов її експлуатації.

ANNOTATION

Deriahin V.L. Methods and tools for traffic balancing between services in a service mesh on the Istio platform. Master's Graduation Thesis: speciality 123 — Computer engineering / supervisor Lutsyk N.S. Ternopil: Ternopil Ivan Puluj National Technical University, 2025.

Keywords: microservice architecture, service mesh, Istio, load balancing, traffic routing, performance, latency.

This thesis investigates the efficiency of microservice interaction in distributed software systems using the Istio service mesh platform. The paper analyzes the architecture of microservice systems, examines the service mesh concept, and conducts a comparative review of existing implementations. Primary attention is focused on traffic management mechanisms in Istio and load balancing algorithms.

As part of the study, an experimental environment was designed based on a Kubernetes cluster with the Istio service mesh, and a load testing methodology using various load profiles was developed. Experimental research was conducted on the round-robin, least connections, and random algorithms using metrics such as latency, throughput, error rate, and request distribution uniformity. The obtained results allowed for the formulation of practical recommendations regarding the selection of a load balancing algorithm depending on the characteristics of the microservice system and its operating conditions.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

gRPC (англ. gRPC Remote Procedure Call) високопродуктивний rpc-фреймворк для взаємодії сервісів

TLS (англ. Transport Layer Security) криптографічний протокол захисту мережевих з’єднань

CRD (англ. Custom Resource Definition) механізм розширення Kubernetes API власними ресурсами

CNI (англ. Container Network Interface) стандарт підключення мереж для контейнерів

eBPF (англ. extended Berkeley Packet Filter) технологія виконання програм у ядрі linux

SLO (англ. Service Level Objective) цільовий показник якості сервісу

SLA (англ. Service Level Agreement) договірні зобов’язання щодо рівня сервісу

ВСТУП

Актуальність роботи. Сучасні розподілені програмні системи дедалі частіше будуються на основі мікросервісної архітектури, яка передбачає декомпозицію монолітного застосунку на набір автономних сервісів із чітко визначеними інтерфейсами взаємодії. Такий підхід забезпечує незалежність розгортання, технологічну гетерогенність та горизонтальне масштабування окремих компонентів системи. Водночас перехід до мікросервісів породжує нові виклики, пов'язані з міжсервісною комунікацією: кожен виклик між сервісами є мережевим запитом, що додає латентність та створює потенційну точку відмови.

Вибір оптимального алгоритму балансування навантаження суттєво впливає на продуктивність мікросервісної системи. Проте існуючі дослідження переважно фокусуються на загальній оцінці накладних витрат інфраструктури service mesh без детального порівняння впливу конкретних стратегій балансування на латентність, частоту помилок та рівномірність розподілу запитів за різних профілів навантаження. Це обумовлює необхідність проведення експериментального дослідження з чітко визначеною методологією та контрольованими умовами.

Метою кваліфікаційної роботи є підвищення ефективності взаємодії мікросервісів шляхом експериментального дослідження та порівняльного аналізу алгоритмів маршрутизації трафіку в service mesh платформі Istio.

Завдання кваліфікаційної роботи:

- проаналізувати архітектуру мікросервісних систем та проблеми міжсервісної взаємодії;
- дослідити концепцію service mesh та провести порівняльний аналіз існуючих платформ;
- вивчити механізми управління трафіком в Istio та алгоритми балансування навантаження;
- формалізувати критерії оцінки ефективності маршрутизації трафіку;
- спроєктувати експериментальне середовище на базі Kubernetes кластера з Istio;

– розробити методику проведення експериментів з різними профілями навантаження;

– провести експериментальне дослідження алгоритмів round-robin, least connections та random;

– сформулювати практичні рекомендації щодо вибору алгоритму балансування залежно від характеристик системи.

Відповідно до мети та завдань кваліфікаційної роботи визначено її об'єкт та предмет.

Об'єкт дослідження: процес маршрутизації трафіку між мікросервісами в середовищі Kubernetes з використанням service mesh.

Предмет дослідження: алгоритми балансування навантаження платформи Istio та їх вплив на продуктивність мікросервісної системи.

Методи дослідження. У роботі використано методи системного аналізу для дослідження архітектури service mesh та механізмів управління трафіком; методи теорії ймовірностей та математичної статистики для формалізації критеріїв ефективності та аналізу результатів експериментів; методи навантажувального тестування для оцінки продуктивності системи за різних профілів навантаження; методи експериментального дослідження для порівняльного аналізу алгоритмів балансування.

Наукова новизна одержаних результатів. Запропоновано методику порівняльного аналізу алгоритмів балансування навантаження в service mesh середовищі, яка, на відміну від існуючих підходів, враховує комплекс метрик продуктивності (перцентилі латентності p50, p95, p99, частоту помилок, коефіцієнт варіації розподілу навантаження) та передбачає дослідження за трьома профілями навантаження (стабільний, сплески, насичення), що дозволяє сформулювати обґрунтовані рекомендації щодо вибору алгоритму залежно від характеристик конкретної системи.

Практичне значення результатів кваліфікаційної роботи. Розроблено експериментальне середовище на базі Kubernetes кластера з Istio service mesh, що може бути використане для подальших досліджень продуктивності мікросервісних

систем. Сформульовано практичні рекомендації щодо вибору алгоритму балансування навантаження залежно від профілю навантаження та вимог до латентності, які можуть застосовуватися при проєктуванні та експлуатації production-систем.

Публікації. Результати дослідження апробовано на XIII науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології», XIV міжнародній науково-технічній конференція молодих учених та студентів «Актуальні задачі сучасних технологій», у вигляді тез конференцій.

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається із вступу, 4 розділів, висновків, списку використаних джерел та додатку.

РОЗДІЛ 1
АНАЛІЗ АРХІТЕКТУРИ МІКРОСЕРВІСІВ ТА ТЕХНОЛОГІЙ SERVICE MESH

1.1. Архітектура мікросервісів та проблеми міжсервісної взаємодії

Мікросервісна архітектура є підходом до розробки програмного забезпечення, за якого застосунок будується як набір невеликих автономних сервісів, кожен з яких виконує окрему бізнес-функцію та взаємодіє з іншими сервісами через чітко визначені програмні інтерфейси [4-6]. На відміну від монолітної архітектури, де всі компоненти системи тісно пов'язані та розгортаються як єдиний артефакт, мікросервіси забезпечують можливість незалежної розробки, тестування та розгортання окремих частин системи.

Основними характеристиками мікросервісної архітектури є декомпозиція за бізнес-функціями, автономність сервісів, децентралізоване управління даними та інфраструктурна автоматизація. Кожен мікросервіс володіє власним сховищем даних, що забезпечує слабку зв'язність між компонентами та дозволяє використовувати різні технології зберігання залежно від потреб конкретного сервісу.

Перехід до мікросервісної архітектури надає низку переваг для організацій, що розробляють складні програмні системи [7]. Незалежність розгортання дозволяє командам випускати оновлення окремих сервісів без необхідності перерозгортання всієї системи. Технологічна гетерогенність забезпечує можливість вибору оптимального стеку технологій для кожного сервісу. Горизонтальне масштабування дозволяє збільшувати потужність лише тих компонентів, які цього потребують, що підвищує ефективність використання ресурсів.

Проте мікросервісна архітектура створює нові виклики, більшість з яких пов'язана з міжсервісною комунікацією [8, 9]. У монолітному застосунку виклики між компонентами відбуваються в межах одного процесу та є надзвичайно швидкими. У мікросервісній системі кожен виклик між сервісами є мережевим запитом, що додає затримку та створює потенційну точку відмови.

Проблема мережевої латентності стає особливо гострою в системах з глибокими ланцюгами викликів. Якщо обробка одного користувацького запиту потребує послідовних викликів до декількох мікросервісів, загальна затримка накопичується. При цьому кожен мережевий виклик характеризується варіативністю часу відповіді, що ускладнює прогнозування продуктивності системи.

Надійність міжсервісної взаємодії є критичним аспектом, оскільки тимчасова недоступність одного сервісу може призвести до каскадних відмов у всій системі. Без належних механізмів захисту перевантажений або несправний сервіс може спричинити вичерпання ресурсів у сервісах, що його викликають, поширюючи проблему далі по ланцюгу залежностей.

Балансування навантаження між екземплярами сервісів є ще одним викликом, що потребує вирішення на рівні інфраструктури. У динамічних середовищах, де кількість екземплярів сервісу може змінюватися залежно від навантаження, необхідні механізми автоматичного виявлення доступних екземплярів та розподілу запитів між ними.

Забезпечення спостережуваності в мікросервісних системах є значно складнішим порівняно з монолітними застосунками. Розподілене трасування запитів, агрегація метрик з багатьох сервісів та централізоване логування потребують спеціалізованої інфраструктури та інструментарію.

Безпека міжсервісної комунікації вимагає впровадження механізмів автентифікації та авторизації на рівні кожного сервісу, шифрування трафіку між сервісами та управління сертифікатами в масштабах усієї системи.

Традиційно вирішення зазначених проблем покладалося на розробників кожного окремого сервісу, що призводило до дублювання коду, неузгодженості реалізацій та ускладнення підтримки. Бібліотеки для забезпечення надійності комунікації, такі як Hystrix для Java або Polly для .NET, частково вирішували ці проблеми, проте їх використання потребувало модифікації коду застосунків та обмежувало технологічний вибір.

1.2. Призначення та реалізації service mesh

Service mesh є виділеним інфраструктурним рівнем для забезпечення надійної, безпечної та спостережуваної комунікації між мікросервісами [10, 6]. Ключовою особливістю цього підходу є винесення функціональності управління трафіком з коду застосунків до інфраструктурного рівня, що забезпечує прозорість для розробників та уніфікацію політик комунікації в масштабах усієї системи.

Архітектура service mesh (рис. 1.1) базується на патерні sidecar proxy, за якого поряд з кожним екземпляром застосунку розгортається проксі-сервер, що перехоплює весь вхідний та вихідний мережевий трафік. Проксі-сервери утворюють так звану площину даних (data plane), через яку проходить увесь міжсервісний трафік. Управління конфігурацією проксі-серверів здійснюється централізовано через площину управління (control plane), що забезпечує узгодженість політик та спрощує адміністрування.

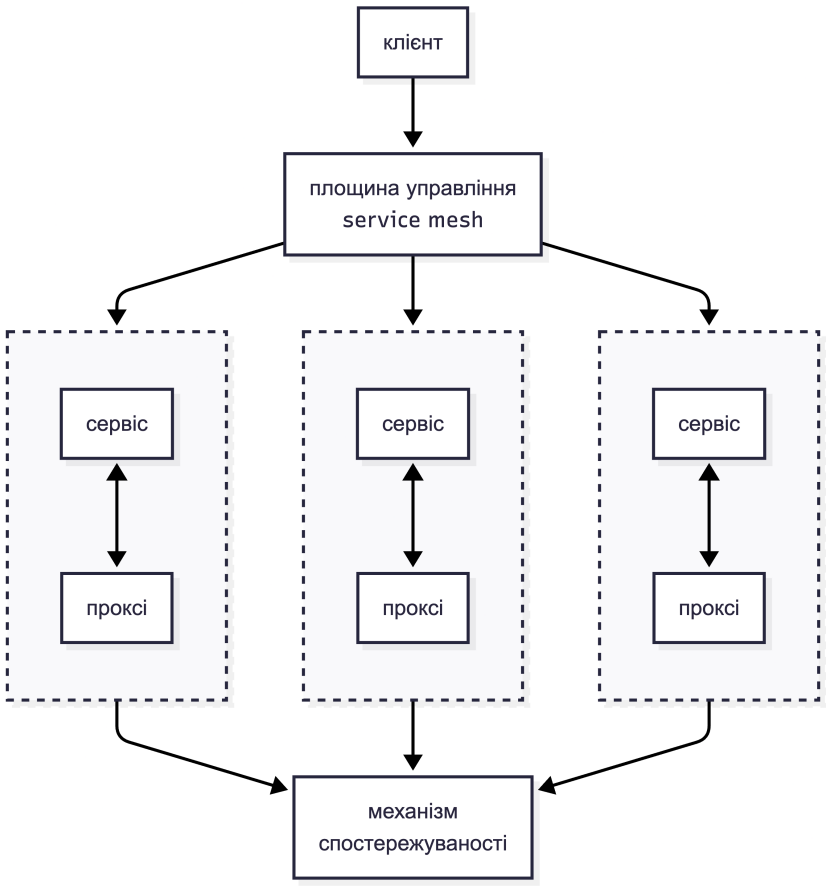

Рис. 1.1. Діаграма архітектури service mesh

Основними функціями service mesh є управління трафіком, забезпечення безпеки та спостережуваність. Управління трафіком включає балансування навантаження, маршрутизацію на основі правил, повторні спроби, тайм-аути та механізми circuit breaker. Безпека забезпечується через взаємну TLS-автентифікацію між сервісами, управління сертифікатами та політики авторизації [11. Спостережуваність реалізується через автоматичний збір метрик, розподілене трасування та логування запитів.

На сьогодні існує декілька зрілих реалізацій service mesh (табл. 1.1), кожна з яких має свої особливості та сфери застосування.

Istio є однією з найбільш функціональних та поширених платформ service mesh [12]. Проєкт було започатковано компаніями Google, IBM та Lyft у 2017 році. Istio використовує Envoy як проксі-сервер площини даних та надає багатий набір можливостей для управління трафіком, безпеки та спостережуваності [13]. Платформа підтримує розгортання в Kubernetes та на віртуальних машинах, що забезпечує гнучкість інтеграції з існуючою інфраструктурою.

Linkerd є легковагою альтернативою Istio, орієнтованою на простоту використання та мінімальне споживання ресурсів. Проєкт розвивається компанією Buoyant та є першою реалізацією концепції service mesh. Linkerd використовує власний проксі-сервер, написаний мовою Rust, що забезпечує низьку латентність та ефективне використання пам'яті. Платформа фокусується на основних функціях service mesh без надмірної складності конфігурації.

Consul Connect є компонентом платформи HashiCorp Consul, що додає можливості service mesh до існуючого інструменту виявлення сервісів та управління конфігурацією. Consul Connect підтримує як Envoy, так і власний вбудований проксі, та забезпечує інтеграцію з іншими продуктами HashiCorp, такими як Vault для управління секретами.

Cilium є service mesh, що використовує технологію eBPF для забезпечення мережевих функцій на рівні ядра Linux. Такий підхід дозволяє досягти високої продуктивності та низької латентності без використання традиційних sidecar-проксі.

Cilium є відносно новою платформою, що активно розвивається та набуває популярності в середовищах з високими вимогами до продуктивності.

Таблиця 1.1

Порівняння платформ service mesh

Характеристика	Istio	Linkerd	Consul Connect	Cilium
Проксі	Envoy	linkerd2-proxy	Envoy/вбудований	eBPF
Мова реалізації	C++	Rust	C++/Go	C/eBPF
Споживання ресурсів	Високе	Низьке	Середнє	Низьке
Функціональність	Висока	Середня	Середня	Середня
Складність	Висока	Низька	Середня	Середня
Зрілість	Висока	Висока	Висока	Середня

Вибір платформи service mesh залежить від конкретних вимог організації, існуючої інфраструктури та рівня експертизи команди. Istio залишається найбільш функціональним рішенням, що підходить для складних сценаріїв з розвиненими вимогами до управління трафіком та безпеки.

1.3. Архітектура платформи Istio

Istio реалізує архітектуру service mesh через поєднання площини даних та площини управління (рис. 1.2). Площина даних складається з проксі-серверів Envoy, що розгортаються як sidecar-контейнери поряд з кожним робочим навантаженням. Площина управління забезпечує централізоване управління конфігурацією, видачу сертифікатів та збір телеметрії.

Envoy є високопродуктивним проксі-сервером, розробленим компанією Lyft та переданим до Cloud Native Computing Foundation. Envoy підтримує протоколи HTTP/1.1, HTTP/2 та gRPC, забезпечує розширені можливості балансування навантаження, автоматичні повторні спроби, circuit breaker та детальну телеметрію [14]. Архітектура Envoy базується на подієво-орієнтованій моделі з використанням

неблокуючого вводу-виводу, що забезпечує високу продуктивність при низькому споживанні ресурсів.

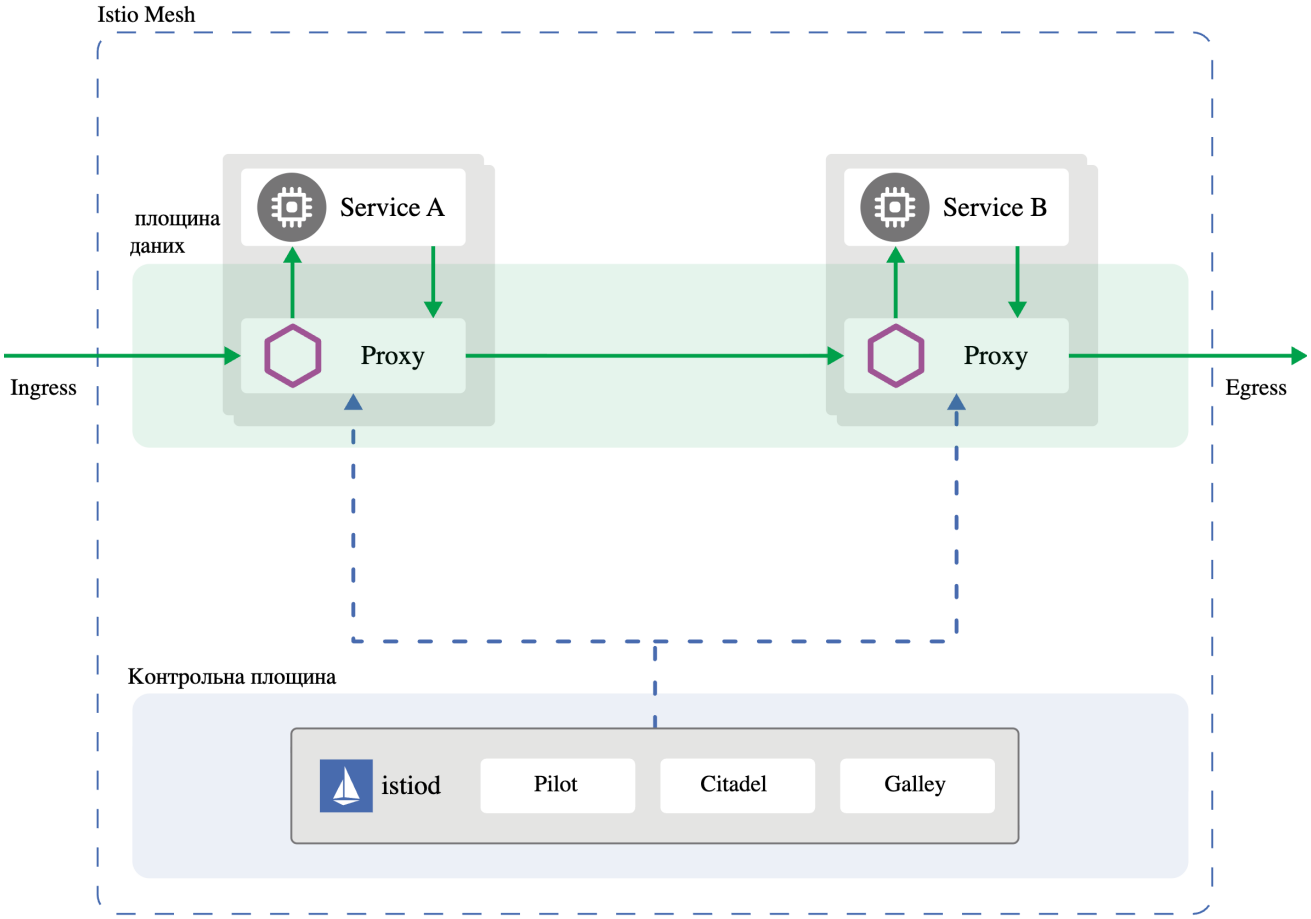

Рис. 1.2. Діаграма архітектури роботи Istio service mesh

У контексті Kubernetes кожен pod автоматично отримує sidecar-контейнер з Envoy при ввімкненні ін'єкції для відповідного namespace. Увесь вхідний та вихідний трафік pod перенаправляється через Envoy за допомогою правил iptables, що забезпечує прозорість для застосунку. Застосунок продовжує надсилати запити на localhost, не усвідомлюючи наявності проксі.

Istiod є монолітним компонентом площини управління, що об'єднує функціональність, яка раніше була розподілена між окремими компонентами Pilot, Citadel та Galley. Istiod відповідає за конфігурацію проксі-серверів Envoy, видачу та ротацію сертифікатів для взаємної TLS-автентифікації, а також валідацію конфігурації Istio.

Конфігурація управління трафіком в Istio здійснюється через набір власних ресурсів Kubernetes (Custom Resource Definitions). Основними ресурсами є VirtualService, DestinationRule, Gateway та ServiceEntry.

VirtualService визначає правила маршрутизації трафіку до сервісу. Цей ресурс дозволяє налаштувати розподіл трафіку між різними версіями сервісу, маршрутизацію на основі заголовків запиту, перенаправлення та переписування URL. VirtualService є потужним інструментом для реалізації стратегій розгортання, таких як canary releases та A/B тестування.

DestinationRule визначає політики, що застосовуються до трафіку після прийняття рішення про маршрутизацію. Цей ресурс дозволяє налаштувати алгоритм балансування навантаження, параметри connection pool, налаштування circuit breaker та TLS-режим для з'єднань з сервісом. DestinationRule також визначає підмножини (subsets) сервісу, що використовуються для маршрутизації трафіку до конкретних версій.

Gateway налаштовує точки входу на межі mesh для вхідного та вихідного трафіку. Ingress Gateway приймає зовнішній трафік та маршрутизує його до внутрішніх сервісів відповідно до правил VirtualService. Egress Gateway контролює вихідний трафік з mesh до зовнішніх сервісів.

ServiceEntry дозволяє додавати зовнішні сервіси до реєстру сервісів Istio, забезпечуючи можливість застосування політик управління трафіком до викликів зовнішніх API.

Механізм балансування навантаження в Istio реалізується на рівні проксі Envoy та конфігурується через DestinationRule. Envoy підтримує декілька алгоритмів балансування, включаючи round-robin, least connections, random та ring hash. Вибір алгоритму впливає на розподіл запитів між екземплярами сервісу та, відповідно, на загальну продуктивність системи.

Istio також надає механізми забезпечення надійності комунікації. Автоматичні повторні спроби дозволяють повторити невдалий запит до іншого екземпляра сервісу без втручання застосунку. Тайм-аути обмежують час очікування відповіді та

запобігають блокуванню ресурсів. Circuit breaker відстежує кількість помилок та тимчасово припиняє надсилання запитів до несправного екземпляра.

Спостережуваність в Istio забезпечується автоматичним збором метрик, розподіленим трасуванням та логуванням. Envoy генерує детальні метрики про кожен запит, включаючи латентність, коди відповіді та обсяг переданих даних. Ці метрики експортуються у форматі Prometheus та можуть візуалізуватися в Grafana. Розподілене трасування реалізується через інтеграцію з Jaeger або Zipkin та дозволяє відстежувати шлях запиту через всі сервіси системи.

1.4. Огляд алгоритмів балансування навантаження в Istio

Балансування навантаження є критичним аспектом забезпечення ефективної роботи мікросервісних систем [15, 16, 17]. Правильний вибір алгоритму балансування дозволяє рівномірно розподілити запити між екземплярами сервісу, мінімізувати латентність та максимізувати використання ресурсів. Istio, через проксі Envoy, підтримує декілька алгоритмів балансування навантаження, кожен з яких має свої характеристики та сфери застосування [18, 19, 20].

Round-robin є найпростішим алгоритмом балансування, за якого запити послідовно розподіляються між усіма доступними екземплярами сервісу по колу. Кожен наступний запит надсилається до наступного екземпляра в списку, після досягнення кінця списку цикл починається спочатку. Алгоритм round-robin є статистично справедливим у довгостроковій перспективі, оскільки кожен екземпляр отримує однакову кількість запитів. Проте він не враховує поточне навантаження на екземпляри та час обробки запитів, що може призвести до нерівномірного розподілу навантаження при неоднорідних запитах.

Перевагами round-robin є простота реалізації, передбачуваність поведінки та відсутність накладних витрат на відстеження стану екземплярів. Недоліком є неспроможність адаптуватися до відмінностей у продуктивності екземплярів або складності запитів.

Least connections (найменше з'єднань) є алгоритмом, що враховує поточне навантаження на екземпляри сервісу. Кожен новий запит надсилається до екземпляра з найменшою кількістю активних з'єднань. Такий підхід дозволяє автоматично адаптуватися до відмінностей у часі обробки запитів: екземпляри, що обробляють запити швидше, звільняють з'єднання раніше та отримують більше нових запитів.

Алгоритм least connections є особливо ефективним для сервісів з неоднорідними запитами, де час обробки суттєво варіюється. Він також добре працює в ситуаціях, коли екземпляри мають різну продуктивність, наприклад, через відмінності в апаратному забезпеченні або конкуренцію за ресурси.

Недоліком least connections є необхідність відстеження кількості активних з'єднань до кожного екземпляра, що додає накладні витрати. Крім того, алгоритм може демонструвати неоптимальну поведінку на початку роботи системи, коли всі екземпляри мають нуль з'єднань.

Random є алгоритмом, що випадковим чином обирає екземпляр для кожного запиту. При великій кількості запитів розподіл наближається до рівномірного, проте для окремих коротких періодів можливі значні відхилення. Алгоритм random є простим у реалізації та не потребує синхронізації стану між балансувальниками, що робить його придатним для розподілених систем.

Варто зазначити, що поведінка алгоритму random суттєво залежить від якості генератора псевдовипадкових чисел та розміру вибірки. При малій кількості запитів (до кількох сотень) розподіл може значно відхилятися від рівномірного, що призводить до тимчасового перевантаження окремих екземплярів. Цей ефект нівелюється при збільшенні інтенсивності трафіку, коли закон великих чисел забезпечує статистичну збіжність до рівномірного розподілу. На практиці random часто використовується як fallback-стратегія або в системах, де простота та відсутність залежностей між балансувальниками є пріоритетом над оптимальністю розподілу.

Порівнюючи три базові алгоритми, можна виділити ключову відмінність у їх підході до прийняття рішень. Round-robin є повністю детермінованим та не використовує жодної інформації про стан системи. Random додає стохастичність, але

також ігнорує поточний стан екземплярів. Least connections є єдиним з трьох, що приймає рішення на основі актуальної інформації про навантаження, що теоретично дозволяє досягти кращої адаптивності. Проте ця перевага має свою ціну у вигляді накладних витрат на підтримку та синхронізацію стану, що може бути критичним у високонавантажених системах з мільйонами запитів на секунду.

Weighted round-robin є розширенням базового round-robin, що дозволяє призначати вагові коефіцієнти екземплярам сервісу. Екземпляри з більшою вагою отримують пропорційно більше запитів. Такий підхід дозволяє враховувати відмінності в потужності серверів або поступово вводити нові екземпляри в експлуатацію.

Ring hash (consistent hashing) є алгоритмом, що забезпечує афінність сесій шляхом хешування певного атрибуту запиту (наприклад, IP-адреси клієнта або значення заголовка) та маршрутизації до відповідного екземпляра на хеш-кільці. Такий підхід гарантує, що запити з однаковим ключем завжди надсилаються до одного екземпляра, що є корисним для кешування на стороні сервера.

В Istio конфігурація алгоритму балансування здійснюється через ресурс DestinationRule. Приклад конфігурації для алгоритму least connections наведено на рис. 1.3.

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: my-service-destination
spec:
  host: my-service
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
```

Рис. 1.3. Лістинг конфігурації DestinationRule для алгоритму least connections

Istio також підтримує локально-зважене балансування (locality-aware load balancing), що надає пріоритет екземплярам у тій самій зоні доступності або регіоні. Такий підхід зменшує мережеву латентність та витрати на міжзональний трафік у хмарних середовищах.

Додатково Envoy підтримує механізм outlier detection, що автоматично виключає з балансування екземпляри, які демонструють підвищену частоту помилок або надмірну латентність. Це забезпечує self-healing поведінку системи та підвищує загальну надійність.

Вибір оптимального алгоритму балансування залежить від характеристик конкретної системи та профілю навантаження. Для сервісів з однорідними швидкими запитами round-robin може бути достатнім. Для сервісів з варіативним часом обробки least connections зазвичай демонструє кращі результати. Системи з вимогами до афінності сесій потребують ring hash або подібних алгоритмів.

1.5. Аналіз існуючих досліджень та підходів до оптимізації трафіку

Проблематика оптимізації трафіку в мікросервісних системах та service mesh середовищах активно досліджується науковою спільнотою. Існуючі роботи охоплюють широкий спектр питань, від теоретичного аналізу алгоритмів балансування до практичних досліджень продуктивності конкретних платформ [6, 21].

Дослідження продуктивності service mesh платформ проводяться як академічними установами, так і індустріальними компаніями [22]. Порівняльні бенчмарки демонструють, що впровадження service mesh додає певну латентність до кожного запиту через проходження трафіку через sidecar-проксі [23]. Величина цієї латентності варіюється залежно від платформи та конфігурації, типово складаючи від 1 до 10 мілісекунд [22].

Дослідження впливу алгоритмів балансування на продуктивність систем проводилися в контексті різних технологій [15, 18]. Класичні роботи з теорії черг надають математичний апарат для аналізу поведінки систем масового

обслуговування під різними політиками балансування. Модель M/M/k дозволяє оцінити середній час очікування в системі з k серверами та пуассонівським потоком запитів [24].

Практичні дослідження алгоритмів балансування в контексті веб-сервісів демонструють, що least connections зазвичай перевершує round-robin для робочих навантажень з варіативним часом обробки [15, 19]. Проте різниця стає менш значущою для однорідних навантажень з швидкими запитами [18].

Адаптивні алгоритми балансування, що динамічно коригують розподіл на основі спостережуваних метрик, є предметом активних досліджень [19, 25]. Підходи на основі машинного навчання дозволяють прогнозувати оптимальний розподіл навантаження на основі історичних даних та поточного стану системи [25, 26]. Проте складність впровадження та інтерпретації таких алгоритмів обмежує їх практичне застосування [26].

Дослідження впливу мережевої топології на ефективність балансування є особливо актуальними для хмарних середовищ [27]. Локально-зважене балансування, що враховує розташування екземплярів у різних зонах доступності, може суттєво зменшити латентність та витрати на трафік [27, 28]. Проте надмірна локалізація може призвести до нерівномірного навантаження при нерівномірному розподілі клієнтів.

Питання масштабованості площини управління service mesh досліджуються в контексті великих розгортань з тисячами сервісів [29]. Централізована архітектура площини управління може стати вузьким місцем при великій кількості проксі-серверів, що потребують конфігурації. Рішення включають горизонтальне масштабування компонентів управління та оптимізацію протоколів синхронізації конфігурації.

Безпека service mesh є окремим напрямком досліджень, що охоплює питання взаємної автентифікації, авторизації на основі атрибутів та захисту від атак [9, 30]. Автоматична ротація сертифікатів та zero-trust мережева модель забезпечують високий рівень безпеки без модифікації коду застосунків [30].

Незважаючи на значний обсяг існуючих досліджень, бракує систематизованих експериментальних робіт, що порівнюють алгоритми балансування в Istio під різними

профілями навантаження. Більшість існуючих бенчмарків фокусуються на загальній латентності service mesh без детального аналізу впливу конкретних алгоритмів балансування [22]. Це обумовлює необхідність проведення комплексного експериментального дослідження з чітко визначеною методологією та контрольованими умовами.

1.6. Висновки до розділу 1

У першому розділі було проведено аналітичний огляд предметної області дослідження. Розглянуто архітектуру мікросервісів та ключові проблеми міжсервісної взаємодії. Досліджено концепцію service mesh та проведено порівняння основних платформ, включаючи Istio, Linkerd, Consul Connect та Cilium. Детально розглянуто архітектуру Istio, механізми управління трафіком та алгоритми балансування навантаження.

На основі аналізу існуючих досліджень встановлено, що не існує достатньо систематизованих експериментальних робіт з порівняння алгоритмів балансування в Istio під різними профілями навантаження. Це обумовлює необхідність проведення експериментального дослідження з чітко визначеною методологією. Сформульовано задачі дослідження та визначено очікувані результати.

РОЗДІЛ 2
МЕТОДИКА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

2.1. Формалізація критеріїв ефективності маршрутизації трафіку

Для проведення коректного порівняльного аналізу алгоритмів балансування навантаження необхідно визначити формальні критерії оцінки ефективності. Ці критерії повинні відображати ключові аспекти продуктивності системи, бути вимірюваними в умовах експерименту та відповідати реальним вимогам до production-систем. У даному розділі формалізовано метрики, що використовуватимуться для оцінки досліджуваних алгоритмів.

Латентність запитів є основним показником продуктивності для більшості веб-сервісів та API. У загальному випадку латентність визначається як інтервал часу між відправленням запиту клієнтом та отриманням повної відповіді. Проте в контексті мікросервісної архітектури з service mesh це визначення потребує уточнення, оскільки запит проходить через декілька компонентів, кожен з яких вносить свій внесок у загальну затримку [31].

Декомпозиція латентності в Istio service mesh включає декілька компонентів. Мережева затримка між клієнтом та Ingress Gateway залежить від географічної відстані та якості мережевого з'єднання. Час обробки на Ingress Gateway включає парсинг HTTP-запиту, застосування політик маршрутизації та прийняття рішення про балансування. Мережева затримка між Gateway та цільовим подом зазвичай є мінімальною в межах одного кластера (менше 1 мс). Час обробки на sidecar-проксі Envoy цільового поду включає перехоплення трафіку через iptables, обробку на рівні Envoy та передачу до контейнера застосунку. Власне час обробки запиту застосунком є основним компонентом для більшості сервісів. Час обробки відповіді на зворотному шляху включає аналогічні етапи у зворотному напрямку.

У двохрівневій архітектурі (frontend → backend) ця декомпозиція ускладнюється, оскільки frontend-сервіс виконує вихідний виклик до backend, який

також проходить через sidecar-проксі обох сервісів. Таким чином, один користувацький запит генерує щонайменше два рішення про балансування: на рівні Gateway (вибір екземпляра frontend) та на рівні sidecar frontend (вибір екземпляра backend).

Для характеристики розподілу латентності використовуються статистичні показники на основі перцентилів. Перцентиль є значенням, нижче якого знаходиться певний відсоток спостережень у вибірці. Формально, якщо L є випадковою величиною, що описує латентність запиту, то p-й перцентиль Lp визначається як найменше значення x, для якого ймовірність $P(L \leq x)$ становить не менше p/100.

У практиці моніторингу мікросервісних систем використовуються три ключові перцентилі латентності, кожен з яких має своє призначення та інтерпретацію.

P50 (медіана) показує значення латентності, нижче якого знаходиться половина всіх запитів. Медіана характеризує типовий час відповіді для більшості користувачів та є основним показником центральної тенденції розподілу. На відміну від середнього арифметичного, медіана є стійкою до викидів — окремі дуже повільні запити не впливають на її значення.

P95 (95-й перцентиль) показує латентність, яку не перевищують 95% запитів. Цей показник є критично важливим для оцінки користувацького досвіду, оскільки характеризує поведінку системи для переважної більшості користувачів, виключаючи лише найгірші 5% випадків. Саме p95 часто використовується у Service Level Objectives (SLO) та Service Level Agreements (SLA). Наприклад, типовий SLO може визначати: "95% запитів повинні оброблятися менше ніж за 200 мс".

P99 (99-й перцентиль) відображає так звану хвостову латентність (tail latency), найгірші випадки, що трапляються з частотою 1%. Хвостова латентність є особливо важливою з декількох причин. По-перше, для високонавантажених систем з мільйонами запитів на добу навіть 1% означає тисячі користувачів, що отримують поганий досвід. По-друге, в мікросервісних системах хвостова латентність має властивість накопичуватися: якщо користувацький запит викликає 10 сервісів паралельно, загальна латентність визначається найповільнішим з них, що статистично зміщує розподіл у бік p99 окремих сервісів. По-третє, p99 часто виявляє

проблеми, що не видимі на рівні p50 та p95: garbage collection паузи, холодні старти, мережеві ретрансмісії.

Збір метрик латентності в Istio здійснюється автоматично sidecar-проксі Envoy.

Гістограма в Prometheus реалізується через набір лічильників (buckets), кожен з яких відповідає за певний діапазон значень. Стандартна конфігурація Istio використовує такі межі для лічильників: 1, 5, 10, 25, 50, 100, 250, 500, 1000, 2500, 5000, 10000 мілісекунд. Кожен лічильник є кумулятивним — містить кількість запитів з латентністю, меншою або рівною граничному значенню.

Обчислення перцентилів з гістограми виконується функцією histogram_quantile у PromQL. Для обчислення p99 латентності backend-сервісу використовується запит показаний на Рис 2.1.

```
histogram_quantile(
  0.99,
  sum(
    rate(
      istio_request_duration_milliseconds_bucket{
        destination_service_name = "backend-service"
      }[5m])) by (le))
```

Рис. 2.1. Лістинг для обчислення p99 латентності за допомогою PromQL

Цей запит обчислює 99-й перцентиль на основі швидкості зміни лічильників buckets за останні 5 хвилин. Функція histogram_quantile виконує лінійну інтерполяцію між boundaries для отримання точного значення перцентиля.

Слід враховувати обмеження гістограмного підходу. Точність обчислення перцентилів залежить від розташування boundaries відносно реальних значень. Якщо більшість запитів мають латентність між двома сусідніми boundaries (наприклад, між 50 та 100 мс), інтерполяція може давати неточні результати. Для критичних систем рекомендується налаштувати межі відповідно до очікуваного розподілу латентності.

End-to-end латентність (E2E latency) вимірюється на стороні клієнта та включає повний час від відправлення запиту до отримання відповіді. У контексті

експериментів E2E латентність вимірюється генератором навантаження Fortio [32]. Ця метрика є найбільш релевантною з точки зору користувача.

Різниця між E2E латентністю (виміряною Fortio) та латентністю на рівні Istio (виміряною Envoy) дозволяє оцінити накладні витрати service mesh. У типових конфігураціях ця різниця складає 2-5 мс на запит.

Пропускна здатність (throughput) визначає кількість запитів, яку система успішно обробляє за одиницю часу. Пропускна здатність вимірюється у запитах за секунду (RPS) та є важливим показником масштабованості системи.

Слід розрізняти номінальну та ефективну пропускну здатність. Номінальна пропускна здатність — це інтенсивність генерації запитів клієнтом, що задається параметрами генератора навантаження. Ефективна пропускна здатність — це кількість успішно оброблених запитів за секунду, що може бути меншою за номінальну при перевантаженні системи.

В Istio пропускна здатність розраховується на основі метрики istio_requests_total — лічильника усіх оброблених запитів. Метрика має labels для фільтрації за кодом HTTP-відповіді, що дозволяє окремо рахувати успішні (2xx) та неуспішні (4xx, 5xx) запити. PromQL-запит для обчислення ефективної пропускної здатності наведено на рис. 2.2.

```
sum(
  rate(
    istio_requests_total{
      destination_service_name = "backend-service",
      response_code =~ "2.."
    }[1m]))
```

Рис. 2.2. Лістинг для обчислення пропускної здатності за допомогою PromQL

Цей запит обчислює швидкість зростання лічильника успішних запитів за останню хвилину.

Частота помилок (error rate) визначає відсоток запитів, що завершуються з помилкою. В контексті HTTP-сервісів помилками вважаються відповіді з кодами

статусу 5xx (серверні помилки). Коди 4xx (клієнтські помилки) зазвичай не враховуються, оскільки вони відображають помилки у вхідних даних, а не проблеми системи.

Частота помилок обчислюється як відношення кількості запитів з помилками до загальної кількості запитів. В Istio частота помилок відстежується через метрику istio_requests_total з фільтрацією за полем response_code. Відповідний PromQL-запит наведено на рис. 2.3.

```
(
  sum(
    rate(
      istio_requests_total{
        destination_service_name = "backend-service",
        response_code =~ "5.."
      }[5m]))
  /
  sum(
    rate(
      istio_requests_total{
        destination_service_name = "backend-service"
      }[5m]))
) * 100
```

Рис. 2.3. Лістинг для обчислення частоти помилок за допомогою PromQL

Додатково корисною є метрика envoy_cluster_upstream_rq_xx на рівні Envoy, що надає детальнішу інформацію про помилки з розбивкою за типами (502 Bad Gateway, 503 Service Unavailable, 504 Gateway Timeout).

Для production-систем типовими вимогами є частота помилок менше 0.1% (три дев'ятки доступності — 99.9%) для критичних сервісів та менше 1% для некритичних. У контексті навантажувального тестування підвищення частоти помилок є індикатором досягнення межі пропускної здатності системи або спрацювання механізмів захисту (circuit breaking).

Рівномірність розподілу навантаження є однією з ключових метрик оцінки якості балансування у розподілених системах. Ця характеристика визначає наскільки ефективно алгоритм балансування розподіляє запити між екземплярами сервісу, забезпечуючи оптимальне використання доступних обчислювальних ресурсів. Нерівномірний розподіл може призводити до перевантаження окремих екземплярів (так званих hotspots), що збільшує латентність для частини запитів та підвищує ризик відмов. Крім того, нерівномірне навантаження знижує загальну пропускну здатність системи, оскільки частина серверів працює на межі своїх можливостей, тоді як інші залишаються недовантаженими. Для моніторингу цієї метрики зазвичай використовують статистичні показники, такі як стандартне відхилення або коефіцієнт варіації кількості запитів на кожен екземпляр.

Для кількісної оцінки рівномірності використовується коефіцієнт варіації (Cv), що визначається як відношення стандартного відхилення до середнього значення:

$$C_v = \frac{\sigma}{\mu}, \tag{2.1}$$

де σ — стандартне відхилення кількості запитів між екземплярами;

μ — середня кількість запитів на екземпляр.

Коефіцієнт варіації є безрозмірною величиною, що дозволяє порівнювати варіабельність розподілів з різними середніми значеннями. Для ідеально рівномірного розподілу Cv = 0. На практиці інтерпретація значень Cv така: менше 0.05 — відмінний розподіл, 0.05-0.10 — хороший розподіл, 0.10-0.20 — прийнятний розподіл, більше 0.20 — потребує уваги, більше 0.30 — проблематичний розподіл.

Для алгоритму least connections очікується, що кількість активних з'єднань буде більш збалансованою між екземплярами порівняно з round-robin та random, особливо при варіативному часі обробки. Якщо один екземпляр обробляє запити повільніше, least connections автоматично зменшить кількість нових запитів до нього, підтримуючи баланс активних з'єднань.

2.2. Проєктування експериментального середовища

Для проведення експериментальних досліджень необхідно спроєктувати тестове середовище, що забезпечить контрольовані умови, відтворюваність результатів та реалістичне моделювання production-сценаріїв. Середовище повинно дозволяти змінювати параметри балансування без модифікації коду застосунків та забезпечувати детальний збір метрик [33].

Архітектура тестового середовища (рис. 2.4) базується на двохрівневій топології мікросервісів, що є типовою для реальних систем [26, 28]. Така архітектура включає Ingress Gateway як точку входу, frontend-сервіс як проміжний рівень та backend-сервіс як кінцевий обробник. Двохрівнева топологія дозволяє дослідити балансування на двох рівнях: на рівні ingress-трафіку (від Gateway до frontend) та на рівні міжсервісної взаємодії (від frontend до backend).

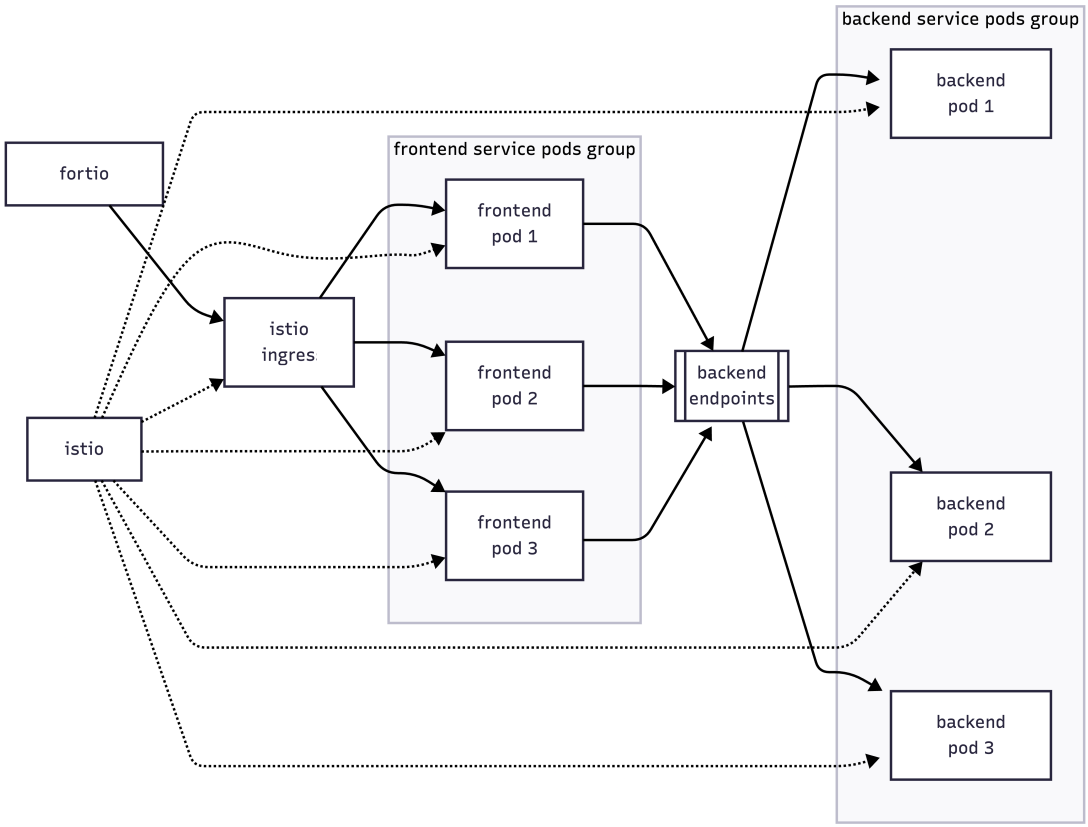

Рис. 2.4. Діаграма архітектури тестового середовища

Вибір двохрівневої архітектури обумовлений кількома факторами. По-перше, вона відображає типовий патерн "API Gateway → Backend Service" або "BFF → Domain Service", поширений у production-системах. По-друге, вона дозволяє дослідити накопичення ефектів балансування: рішення на першому рівні впливає на те, який екземпляр frontend прийматиме рішення на другому рівні. По-третє, вона створює реалістичний ланцюг латентності з двома точками варіативності.

Kubernetes кластер є платформою для розгортання тестового середовища. Для експериментів використовується кластер з такою конфігурацією:

Вузли кластера включають один Control Plane вузол та три Worker вузли. Control Plane вузол розміщує компоненти Kubernetes (kube-apiserver, etcd, kube-scheduler, kube-controller-manager) та Istiod — площину управління Istio. Worker вузли розміщують робочі навантаження та мають ідентичні характеристики для забезпечення однорідності.

Мережева конфігурація кластера використовує Calico як CNI (Container Network Interface) plugin з налаштуванням VXLAN для overlay-мережі. Calico забезпечує мережеву політику на рівні pod та ефективну маршрутизацію трафіку між вузлами.

Розподіл подів між Worker вузлами здійснюється Kubernetes scheduler з урахуванням anti-affinity правил [27]. Для забезпечення реалістичного моделювання мережевої взаємодії поди frontend та backend розподіляються між усіма трьома Worker вузлами.

Istio service mesh встановлюється з використанням istioctl та профілю default. Профіль default включає Istiod (площина управління), Istio Ingress Gateway та базові конфігурації телеметрії. Версія Istio: 1.20.

Автоматична ін'єкція sidecar вмикається для namespace test-mesh через label, команд для застосування наведена на рис. 2.5:

```
kubectl label namespace test-mesh istio-injection=enabled
```

Рис. 2.5. Команда для застосування мітки istio-injection

Після цього кожен pod, створений у namespace test-mesh, автоматично отримує sidecar-контейнер istio-proxy (Envoy).

Ресурси sidecar-проксі налаштовуються через annotations на рівні namespace або окремих deployment. Для експериментів використовується конфігурація наведена на рис. 2.6.

```
sidecar.istio.io/proxyCPU: "100m"
sidecar.istio.io/proxyCPULimit: "500m"
sidecar.istio.io/proxyMemory: "128Mi"
sidecar.istio.io/proxyMemoryLimit: "256Mi"
```

Рис. 2.6. Лістинг конфігурації Istio

Ці значення забезпечують достатні ресурси для обробки тестового навантаження без надмірного споживання.

Prometheus встановлюється як частина Istio addons для збору метрик.

Prometheus автоматично виявляє Envoy sidecar через Kubernetes service discovery та збирає метрики з endpoint :15020/stats/prometheus кожного поду.

Kiali встановлюється для візуалізації топології mesh та моніторингу в реальному часі. Kiali інтегрується з Prometheus та надає графічний інтерфейс для перегляду потоків трафіку, статусу сервісів та конфігурації Istio [34].

Конфігурація тестових сервісів визначає характеристики frontend та backend компонентів. Обидва сервіси реалізовані як прості HTTP-сервери на Node.js, що дозволяє точно контролювати час відповіді.

Frontend Service моделює проміжний рівень обробки. Сервіс приймає HTTP-запит, виконує мінімальну обробку (симуляція валідації), здійснює HTTP-виклик до backend-service та повертає результат клієнту.

Час власної обробки frontend генерується за нормальним розподілом з параметрами μ=20ms та σ=5ms. Нормальний розподіл обрано для моделювання стабільного сервісу з невеликою варіативністю — більшість запитів обробляються близько до середнього значення.

Backend Service моделює основну бізнес-логіку з варіативним часом обробки. Час відповіді генерується за експоненційним розподілом з параметром λ=1/50 (mean=50ms).

Вибір експоненційного розподілу для моделювання часу обробки на backend обумовлений його математичними властивостями, які добре відповідають реальним умовам роботи серверних систем. Експоненційний розподіл є "memoryless" — час, що залишився до завершення обробки запиту, не залежить від часу, що вже минув з моменту його початку. Ця властивість добре моделює сценарії, де час обробки визначається зовнішніми факторами, які важко передбачити: запити до бази даних з різним обсягом даних, що повертаються, виклики зовнішніх API з непередбачуваною латентністю, операції з різним рівнем кешування та попадання в кеш. Важливою характеристикою експоненційного розподілу є наявність "важкого хвоста" — значна частина запитів (приблизно 5%) матиме латентність, що значно перевищує середнє значення. Саме ця особливість створює умови підвищеного навантаження, за яких різниця в ефективності між різними алгоритмами балансування стає помітною та статистично значущою.

Реалізація затримки у коді сервісів використовує асинхронний sleep з генерацією випадкового значення відповідно до заданого розподілу. Для нормального розподілу використовується алгоритм Box-Muller [35]. Для експоненційного — інверсія функції розподілу, згідно формули:

$$delay = -mean \times \ln(random(C)), \quad (2.2)$$

Де delay — величина затримки у мілісекундах;

mean — початкова величина затримки у мілісекундах;

C — константа якою регулюється амплітуда змін у розподілі.

Istio Gateway та VirtualService налаштовують точку входу для зовнішнього трафіку.

Генератор навантаження Fortio розгортається на окремій машині поза кластером або в окремому namespace без sidecar-ін'єкції. Це забезпечує точне

вимірювання E2E латентності без додаткових накладних витрат від service mesh на стороні клієнта.

Fortio обрано з таких причин: розроблений командою Istio з розумінням специфіки service mesh, підтримує точний контроль частоти запитів (QPS) на відміну від інструментів, що контролюють лише concurrency, надає детальну гістограму латентності з обчисленням перцентилів, має низькі власні накладні витрати та стабільну генерацію навантаження, підтримує експорт результатів у JSON для автоматизованого аналізу.

2.3. Методика проведення експериментів

Методика проведення експериментів визначає послідовність дій, параметри тестування та процедури збору даних. Коректна методика забезпечує відтворюваність результатів та можливість порівняння між різними конфігураціями.

Матриця експериментів формується як комбінація алгоритмів балансування та профілів навантаження. Дослідження охоплює три алгоритми (round-robin, least connections, random), та три профілі навантаження (стабільний, сплески, насичення). Це дає 9 унікальних конфігурацій.

Для забезпечення статистичної значущості кожна конфігурація тестується 5 разів. П'ять повторень є мінімально достатньою кількістю для виявлення аномальних результатів та оцінки варіабельності. При меншій кількості повторень випадкові фактори (мережеві затримки, garbage collection, конкуренція за ресурси) можуть суттєво спотворити результати. При більшій кількості повторень виграш у точності не виправдовує додаткові витрати часу.

Загальна кількість експериментальних прогонів: 3 алгоритми × 3 профілі × 5 повторень = 45 прогонів.

Структура матриці експериментів наведена в таблиці 2.1.

Таблиця 2.1

Матриця експериментів

Алгоритм	Стабільний	Сплески	Насичення	Всього
round-robin	5	5	5	15
least connections	5	5	5	15
random	5	5	5	15
всього	15	15	15	45

Профілі навантаження моделюють різні сценарії реального використання системи. Кожен профіль характеризується патерном генерації запитів та має специфічну мету дослідження.

Профіль стабільного навантаження (Stable Load) моделює штатний режим роботи системи з постійною інтенсивністю трафіку.

Мета профілю — встановити базову лінію (baseline) продуктивності для кожного алгоритму. При стабільному навантаженні система працює в штатному режимі без перевантажень, що дозволяє оцінити типову латентність та рівномірність розподілу запитів. Очікується, що всі три алгоритми покажуть подібні результати при стабільному навантаженні нижче точки насичення.

Профіль сплесків навантаження (Bursty Load) моделює нерівномірний трафік з періодичними піками, що є типовим для багатьох веб-сервісів (маркетингові кампанії, публікації в соціальних мережах, початок робочого дня).

Мета даного профілю навантаження полягає в оцінці реактивності алгоритмів балансування та їх здатності швидко адаптуватися до динамічних змін у характері навантаження на систему.

Профіль насичення (Saturation/Stress) моделює перевантаження системи, коли інтенсивність запитів перевищує пропускну здатність.

Мета профілю — визначити точку деградації сервісу та перевірити роботу механізмів захисту. При перевищенні пропускної здатності очікується: зростання латентності (особливо p99), збільшення частоти помилок через спрацювання circuit

breaker (параметри connectionPool в DestinationRule), можлива нерівномірність розподілу.

Параметри профілів навантаження зведено в таблиці 2.2.

Таблиця 2.2

Параметри профілів навантаження

Профіль	RPS	Workers	Тривалість	Патерн	Мета
Стабільний	50	10	5 хв	Константний	Baseline
Сплески	10↔200	5↔50	5 хв	Циклічний	Реактивність
Насичення	500	100	5 хв	Константний	Точка деградації

Процедура проведення експерименту складається з п'яти етапів (рис 2.7), що забезпечують консистентність умов між прогонами та мінімізують вплив зовнішніх факторів.

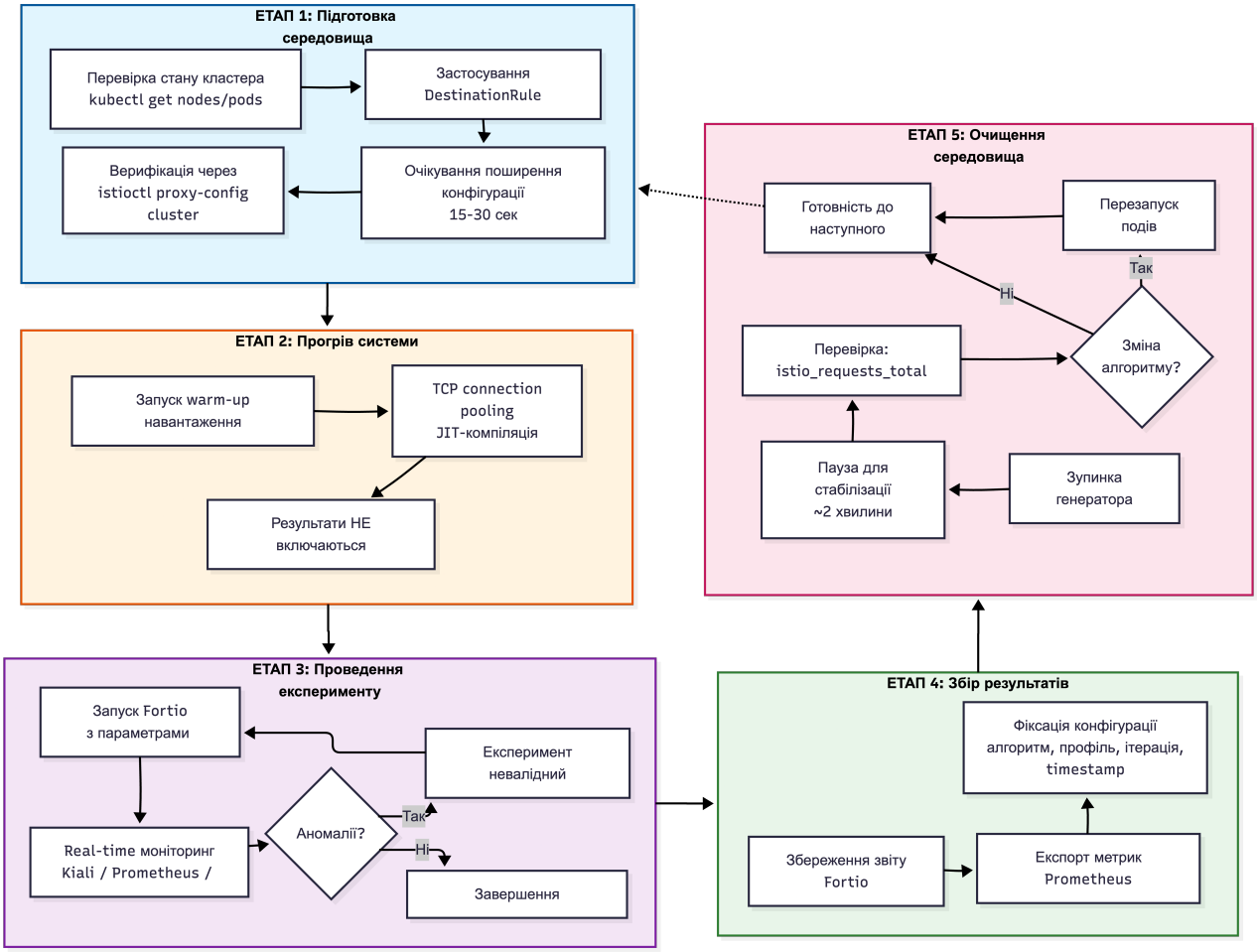

Рис. 2.7. Діаграма експериментального процесу

Першим етапом експерименту є підготовка середовища. На цьому кроці перевіряється загальний стан кластера за допомогою команд kubectl get nodes та kubectl get pods -n test-mesh. Усі вузли та поди мають перебувати у стані Ready або Running. Після цього застосовується DestinationRule з обраним алгоритмом балансування навантаження. Далі необхідно дочекатися поширення конфігурації, що зазвичай займає 15–30 секунд, протягом яких Istiod розповсюджує оновлення до всіх Envoy sidecar. Коректність застосування конфігурації перевіряється за допомогою istioctl proxy-config cluster.

Другим етапом є прогрів системи (warm-up), який часто недооцінюється, але є критично важливим для отримання коректних результатів. Мета прогріву полягає у виведенні системи в стабільний робочий стан перед початком вимірювань. Необхідність цього етапу зумовлена кількома факторами. По-перше, механізми TCP connection pooling означають, що перші запити створюють нові TCP-з’єднання, тоді як наступні використовують уже встановлені, що працює значно швидше. По-друге, для сервісів на базі JVM перші запити виконуються з використанням інтерпретатора, а лише згодом активується JIT-компіляція з генерацією оптимізованого машинного коду. Крім того, під час початкових запитів заповнюються різноманітні кеші, зокрема DNS-кеші, кеші з’єднань Envoy та внутрішні кеші застосунків. Окремо варто враховувати й autoscaling: якщо використовується HPA, системі може знадобитися додатковий час для масштабування до необхідного рівня.

Для прогріву використовується навантаження тривалістю 30 секунд з інтенсивністю близько 20% від цільового профілю. Результати, отримані на цьому етапі, не включаються до подальшого аналізу.

Третій етап — це безпосереднє проведення експерименту. На цьому кроці запускається генератор навантаження Fortio з параметрами відповідного профілю. Протягом усього експерименту здійснюється моніторинг стану системи в реальному часі через Kiali, а також спостереження за ключовими метриками в Prometheus та Grafana. Усі аномалії, зокрема перезапуски подів, мережеві помилки або випадки OOM, фіксуються окремо. Якщо під час експерименту відбувається будь-яка подібна аномалія, результат вважається невалідним, а експеримент повторюється.

На четвертому етапі виконується збір результатів. Одразу після завершення навантаження, протягом приблизно однієї хвилини, зберігається звіт Fortio у форматі JSON, експортуються метрики з Prometheus за період експерименту, а також фіксується повна конфігурація запуску, включно з використаним алгоритмом балансування, профілем навантаження, номером ітерації та часовою міткою.

Prometheus-запити для експорту метрик латентності p50/p95/p99 наведено на рис. 2.8.

```
histogram_quantile(
  0.50,
  sum(
    rate(
      istio_request_duration_milliseconds_bucket{
        destination_service_name = "backend-service"
      }[5m])) by (le))
histogram_quantile(
  0.95,
  sum(
    rate(
      istio_request_duration_milliseconds_bucket{
        destination_service_name = "backend-service"
      }[5m])) by (le))
histogram_quantile(
  0.99,
  sum(
    rate(
      istio_request_duration_milliseconds_bucket{
        destination_service_name = "backend-service"
      }[5m])) by (le))
```

Рис. 2.8. Лістинг запитів Prometheus для експорту потрібних метрик латентності

Запит для отримання розподілу запитів по подах наведено на рис. 2.9.

```
sum(
  increase(
    istio_requests_total{
      destination_service_name = "backend-service"
    }[5m])) by (pod_name)
```

Рис. 2.9. Лістинг запиту Prometheus для отримання розподілу запитів по подах

Запит для обрахунку частоти помилок наведено на рис. 2.10

```
(
  sum(
    rate(
      istio_requests_total{
        destination_service_name = "backend-service",
        response_code =~ "5.."
      }[5m]))
  /
  sum(
    rate(
      istio_requests_total{
        destination_service_name = "backend-service"
      }[5m]))
) * 100
```

Рис. 2.10. Лістинг запиту Prometheus для отримання частоти помилок

П’ятий етап присвячений очищенню середовища та підготовці до наступного прогону. Після завершення збору результатів зупиняється генератор навантаження, після чого системі надається час для стабілізації — приблизно дві хвилини. Ця пауза необхідна для завершення всіх активних запитів і коректного закриття мережевих з’єднань. Далі перевіряється відсутність залишкового навантаження: значення швидкості зростання метрики istio_requests_total має бути близьким до нуля. За потреби, зокрема під час переходу між тестуванням різних алгоритмів балансування,

виконується перезапуск подів з метою повного скидання внутрішнього стану системи та забезпечення однакових початкових умов для наступного експерименту.

Порядок виконання експериментів має значення для мінімізації систематичних похибок. Обрана стратегія — randomized block design: всі 5 повторень одного алгоритму з одним профілем виконуються послідовно (блок), порядок блоків рандомізується.

Приклад порядку виконання: least connections-stable (×5), round-robin-saturation (×5), random-bursty (×5), і т.д.

Такий підхід мінімізує вплив часових факторів (наприклад, зміни мережевої латентності протягом дня) на порівняння алгоритмів.

2.4. Методи аналізу та обробки результатів

Аналіз результатів експериментів передбачає систематичну обробку зібраних метрик та формування обґрунтованих висновків щодо ефективності алгоритмів балансування. Методи аналізу включають первинну обробку даних, агрегацію результатів повторних прогонів, порівняльний аналіз та візуалізацію.

Первинна обробка даних починається з валідації зібраних результатів. Для кожного прогону перевіряється: повнота даних (наявність усіх метрик), відсутність аномалій (перезапуски подів, мережеві помилки), консистентність (загальна кількість запитів відповідає очікуваній для профілю).

Прогони з аномаліями виключаються з аналізу та замінюються додатковими прогонами. Критерії виключення: перезапуск будь-якого поду під час експерименту, частота помилок більше 50% (свідчить про серйозну проблему, не пов'язану з алгоритмом), відхилення загальної кількості запитів більше ніж на 20% від очікуваної.

Агрегація результатів повторних прогонів здійснюється для кожної комбінації (алгоритм, профіль). Для метрик латентності (p50, p95, p99) обчислюються: середнє значення по 5 прогонах, стандартне відхилення, мінімум та максимум.

Стандартне відхилення є індикатором стабільності результатів. Низьке значення (< 5% від середнього) свідчить про високу відтворюваність. Високе значення може вказувати на: нестабільність системи, вплив зовнішніх факторів, недостатній прогрів, або реальну варіативність поведінки алгоритму.

Виявлення викидів здійснюється за правилом 2σ: якщо значення відхиляється від середнього більше ніж на 2 стандартних відхилення, воно вважається викидом. Викиди аналізуються окремо для виявлення причин та виключаються з агрегації (або весь прогон повторюється).

Порівняльний аналіз алгоритмів здійснюється для кожного профілю навантаження окремо. Для кожної метрики порівнюються агреговані значення трьох алгоритмів.

Формат представлення результатів використовуємо структуру як у таблиці 2.3.

Таблиця 2.3

Шаблон таблиці результатів (приклад для профілю "стабільний")

Метрика	round-robin	least connections	random	Найкращий
p50 (ms)	X ± σ	X ± σ	X ± σ	[алгоритм]
p95 (ms)	X ± σ	X ± σ	X ± σ	[алгоритм]
p99 (ms)	X ± σ	X ± σ	X ± σ	[алгоритм]
Error rate (%)	X ± σ	X ± σ	X ± σ	[алгоритм]
Cv розподілу	X ± σ	X ± σ	X ± σ	[алгоритм]

Аналіз статистичної значущості різниць враховує варіабельність результатів. Якщо довірчі інтервали двох алгоритмів перекриваються, різниця вважається статистично незначущою, навіть якщо середні значення відрізняються.

Формування висновків базується на комплексному аналізі всіх метрик та профілів.

Для кожного алгоритму визначаються: сильні сторони (за якими метриками та профілями алгоритм найкращий), слабкі сторони (за якими метриками та профілями алгоритм найгірший), оптимальні сценарії застосування.

Порівняльний висновок визначає: який алгоритм є найкращим overall (якщо такий є), за яких умов різниця між алгоритмами є значущою, за яких умов вибір алгоритму не має суттєвого значення.

Практичні рекомендації формулюються у вигляді decision matrix: "якщо характеристика системи X — рекомендується алгоритм Y".

Обмеження аналізу слід чітко зафіксувати. Результати отримані для конкретної конфігурації: 3 репліки, експоненційний розподіл затримки backend (mean=50ms), нормальний розподіл затримки frontend (μ=20ms). Результати можуть відрізнятися для інших конфігурацій: більшої кількості реплік, іншого розподілу затримки, реальних сервісів з більш складною поведінкою. Синтетичне навантаження Fortio може не відображати всіх особливостей реального трафіку (bursty patterns, long-tail distributions, correlated requests).

2.5. Висновки до розділу 2

У другому розділі визначено метрики для порівняльного аналізу алгоритмів балансування навантаження. Для забезпечення об'єктивності дослідження обрано критерії, що піддаються експериментальному вимірюванню та відповідають вимогам до реальних систем, причому основним індикатором продуктивності виступає латентність запитів.

Специфіка архітектури Istio service mesh вимагає детальної декомпозиції цього показника, оскільки сумарна затримка формується не лише часом виконання бізнес-логіки, а й інфраструктурними накладними витратами. До складових загального часу відгуку віднесено мережеву затримку, час обробки та парсингу на Ingress Gateway, а також етап перехоплення трафіку sidecar-проксі Envoy перед передачею його у цільовий контейнер застосунку.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ АЛГОРИТМІВ БАЛАНСУВАННЯ В ISTIO SERVICE MESH

3.1. Розгортання тестової інфраструктури

Для проведення експериментального дослідження алгоритмів балансування навантаження розгорнуто відповідне тестове середовище. Середовище реалізує двохрівневу архітектуру мікросервісів, що дозволяє дослідити роботу алгоритмів балансування як на рівні ingress-трафіку (від Gateway до frontend), так і на рівні міжсервісної взаємодії (від frontend до backend).

Kubernetes кластер складається з одного Control Plane вузла та трьох Worker вузлів з ідентичними характеристиками: 4 vCPU, 8 GB RAM, Ubuntu 22.04 LTS. Версія Kubernetes — 1.28, container runtime — containerd 1.7. Мережева підсистема використовує Calico CNI з VXLAN overlay для забезпечення зв'язності між подами на різних вузлах.

Istio service mesh версії 1.20 встановлено з профілем default, що включає компонент площини управління Istiod та Istio Ingress Gateway. Для забезпечення детального збору телеметрії параметр tracing.sampling встановлено на значення 100, що гарантує трасування кожного запиту. Автоматична ін'єкція sidecar-проксі Envoy активована для namespace test-mesh через відповідну мітку.

Тестові сервіси розгорнуто як зазначено в таблиці 3.1.

Таблиця 3.1

Конфігурація вузлів Kubernetes кластера

Параметр	Control Plane	Worker Node (×3)
CPU	4 vCPU	4 vCPU
RAM	8 GB	8 GB
Диск	100 GB SSD	50 GB SSD
Операційна система	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS
Kubernetes версія	1.28	1.28
Container runtime	containerd 1.7	containerd 1.7

Frontend Service реалізовано як HTTP-сервер на Node.js, що приймає запити, симулює затримку обробки за нормальним розподілом (μ=20ms, σ=5ms) та здійснює виклик до Backend Service.

Конфігурація включає podAntiAffinity для розподілу реплік між різними Worker вузлами, що забезпечує реалістичне моделювання мережевої взаємодії з урахуванням міжвузлового трафіку. Параметри затримки передаються через змінні середовища, що дозволяє модифікувати поведінку сервісу без перезбирання образу.

Backend Service симулює обробку бізнес-логіки з варіативним часом відповіді. Затримка генерується за експоненційним розподілом з параметром mean=50ms, що моделює реальні сценарії, де час обробки залежить від складності запиту, стану кешу або затримок зовнішніх залежностей.

Для кожного сервісу створено відповідний Kubernetes Service типу ClusterIP, що забезпечує стабільну DNS-адресу для міжсервісної комунікації. Istio автоматично перехоплює трафік до цих сервісів через sidecar-проксі та застосовує політики балансування.

Конфігурація Istio Gateway та VirtualService визначає точку входу для зовнішнього трафіку та правила маршрутизації. Gateway налаштовано на прийом HTTP-трафіку на порту 80 для хоста test.local. VirtualService маршрутизує всі вхідні запити до frontend-service.

Ключовим елементом експериментів є ресурс DestinationRule, що визначає алгоритм балансування навантаження для кожного сервісу. Для забезпечення консистентності експериментів однаковий алгоритм застосовується як до frontend-service, так і до backend-service.

Параметри connectionPool налаштовані таким чином: maxConnections=100 обмежує кількість TCP-з'єднань до кожного екземпляра, http2MaxRequests=1000 обмежує кількість паралельних HTTP/2 запитів. Ці значення забезпечують нормальну роботу при профілях стабільного навантаження та сплесків, але викликають спрацювання circuit breaker при профілі насичення.

Стек моніторингу розгорнуто як частину Istio addons. Prometheus збирає метрики з усіх Envoy sidecar через endpoint /stats/prometheus на порту 15020. Kiali

забезпечує візуалізацію топології mesh та моніторинг в реальному часі. Grafana використовується для побудови дашбордів з ключовими метриками експериментів.

Генератор навантаження Fortio версії 1.60 розгорнуто на окремій машині поза кластером для забезпечення точного вимірювання end-to-end латентності без впливу sidecar-проксі на стороні клієнта. Машина з'єднана з кластером через локальну мережу з латентністю менше 1 мс.

3.2. Проведення експериментів

Матриця експериментів включає 45 прогонів: 3 алгоритми балансування (round-robin, least connections, random) × 3 профілі навантаження (стабільний, сплески, насичення) × 5 повторень для кожної комбінації (табл. 3.2).

Таблиця 3.2

Параметри проведених експериментів

Параметр	Значення
Алгоритми балансування	round-robin least connections random
Профілі навантаження	Стабільний (50 RPS) Сплески (10↔200 RPS) Насичення (500 RPS)
Кількість повторень	5 для кожної комбінації
Тривалість прогону	300 секунд (5 хвилин)
Warm-up період	30 секунд (не включається в аналіз)
Пауза між прогонами	120 секунд
Загальна кількість прогонів	45

Порядок виконання експериментів рандомізовано на рівні блоків для мінімізації впливу часових факторів. Кожен блок складається з 5 послідовних повторень однієї комбінації (алгоритм, профіль), порядок блоків визначено генератором випадкових чисел.

Процедура проведення кожного прогону включала п'ять етапів. На етапі підготовки застосовувалася відповідна конфігурація DestinationRule та перевірялося її поширення до всіх Envoy sidecar за допомогою команди istioctl proxy-config cluster. Етап прогріву тривав 30 секунд з навантаженням 10 RPS для ініціалізації TCP connection pool та внутрішніх кешів. Основний етап експерименту тривав 300 секунд з параметрами відповідного профілю навантаження. На етапі збору результатів зберігалися звіти Fortio у форматі JSON та експортувалися метрики з Prometheus. Етап очищення включав паузу 120 секунд для завершення активних з'єднань та стабілізації системи.

Використано три різні профілі навантаження (рис. 3.1).

Профіль стабільного навантаження генерувався командою Fortio з параметрами: частота 50 RPS, 10 паралельних workers, тривалість 300 секунд. Цей профіль моделює штатний режим роботи системи без перевантажень.

Профіль сплесків реалізовано через скрипт, що чергує періоди низького навантаження (10 RPS, 5 workers, 30 секунд) та високого навантаження (200 RPS, 50 workers, 30 секунд). За 300 секунд виконується 5 повних циклів. Цей профіль моделює типову поведінку веб-сервісів з періодичними піками активності.

Профіль насичення генерувався з параметрами: частота 500 RPS, 100 паралельних workers, тривалість 300 секунд. Цей профіль навмисно перевищує пропускну здатність системи для дослідження поведінки алгоритмів в умовах перевантаження та спрацювання механізмів захисту.

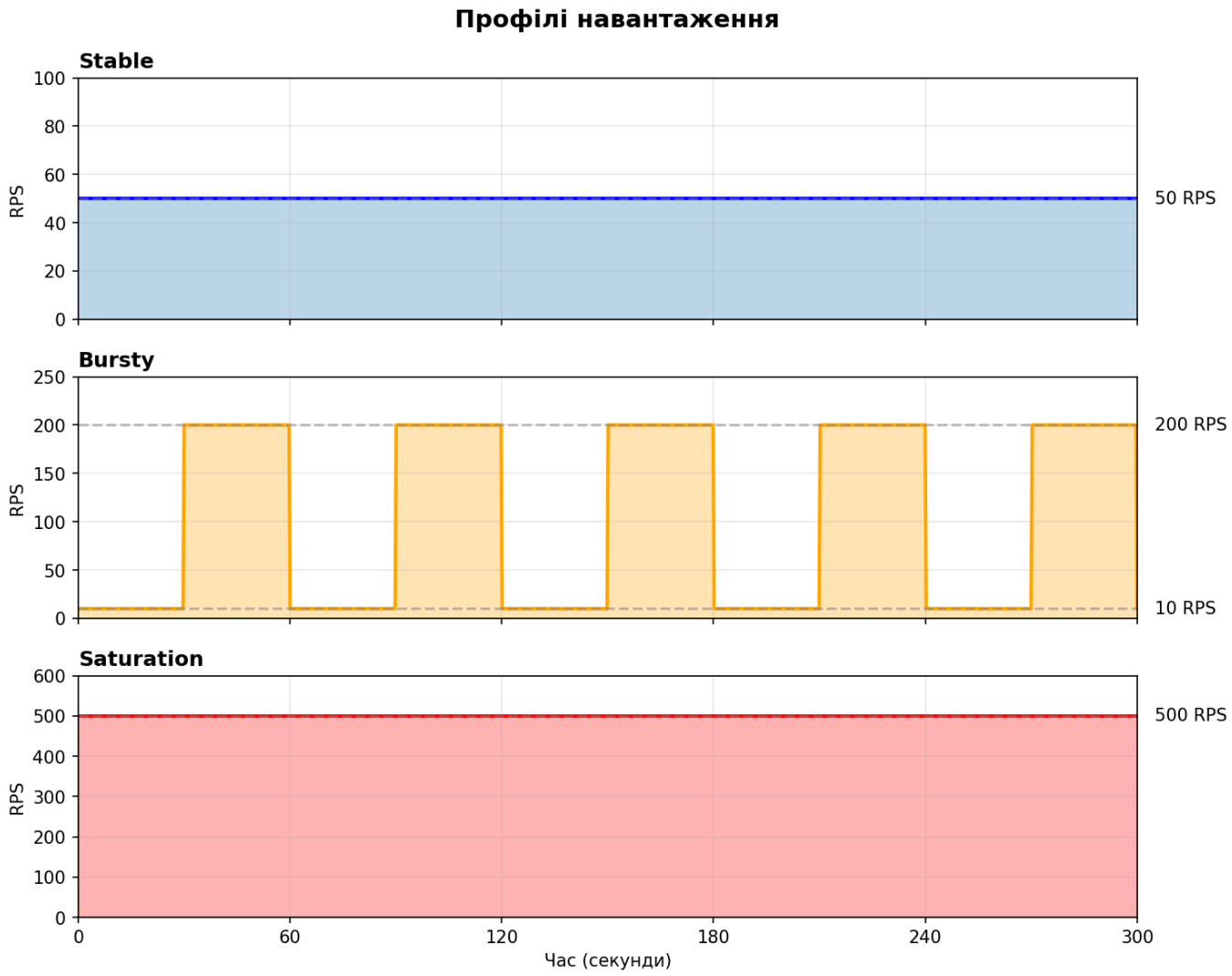

Рис. 3.1. Графік профілів навантаження

Під час проведення експериментів фіксувалися всі аномалії. З 45 запланованих прогонів 3 було визнано невалідними через перезапуск подів (2 випадки) та мережеву помилку на рівні інфраструктури (1 випадок). Ці прогони було повторено, загальна кількість фактично виконаних прогонів склала 48.

3.3. Результати вимірювання латентності

Латентність запитів є основним критерієм оцінки ефективності алгоритмів балансування. Вимірювання проводилися на двох рівнях: end-to-end латентність (від Fortio до отримання відповіді) та латентність окремих сервісів (за даними Envoy

sidecar). Результати представлено як середнє значення ± стандартне відхилення по 5 повторенням. Таблиця 3.3 містить результати при стабільному навантаженні.

Таблиця 3.3

Результати вимірювання латентності при стабільному навантаженні (50 RPS)

Алгоритм	p50 (мс)	p95 (мс)	p99 (мс)	E2E p99 (мс)
round-robin	71.3 ± 1.2	142.8 ± 3.4	198.4 ± 5.7	203.1 ± 5.9
least connections	70.8 ± 1.1	138.2 ± 2.9	187.6 ± 4.8	192.3 ± 5.1
random	72.1 ± 1.8	145.3 ± 4.1	203.7 ± 7.2	208.5 ± 7.4

При стабільному навантаженні всі три алгоритми демонструють подібну продуктивність на рівні медіани (p50): різниця між найкращим (least connections, 70.8 мс) та найгіршим (random, 72.1 мс) результатом складає лише 1.8%. Теоретично очікуване значення p50 становить приблизно 70 мс (20 мс frontend + 50 мс backend), що підтверджує коректність експериментальної установки.

Різниця між алгоритмами стає помітнішою на хвостових перцентилях. На рівні p99 least connections демонструє перевагу в 5.5% над round-robin та 7.9% над random. Це пояснюється здатністю least connections адаптуватися до варіативності часу обробки backend-сервісу: запити направляються до менш завантажених екземплярів, що зменшує накопичення черги на окремих подах.

Стандартне відхилення результатів для random є вищим (7.2 мс для p99 порівняно з 5.7 мс для round-robin), що свідчить про меншу передбачуваність поведінки цього алгоритму. Таблиця 3.4 містить результати при сплесках навантаження.

Таблиця 3.4

Результати вимірювання латентності при сплесках навантаження (10↔200 RPS)

Алгоритм	p50 (мс)	p95 (мс)	p99 (мс)	E2E p99 (мс)
round-robin	73.4 ± 2.1	168.5 ± 8.7	247.3 ± 14.2	253.8 ± 14.9
least connections	72.1 ± 1.6	152.4 ± 5.3	218.6 ± 9.1	224.2 ± 9.6
random	74.8 ± 3.2	178.9 ± 11.4	268.4 ± 18.6	275.1 ± 19.2

Профіль сплесків суттєво збільшує хвостову латентність для всіх алгоритмів через формування черг під час пікових періодів (рис. 3.2). Порівняно зі стабільним навантаженням, p99 зростає на 24.6% для round-robin, 16.5% для least connections та 31.8% для random.

Least connections демонструє найкращу адаптивність до змін навантаження. Під час переходу від низького до високого навантаження алгоритм швидко перерозподіляє запити до менш завантажених екземплярів, що обмежує зростання черги. Перевага least connections над round-robin на рівні p99 складає 11.6%, над random — 18.6%.

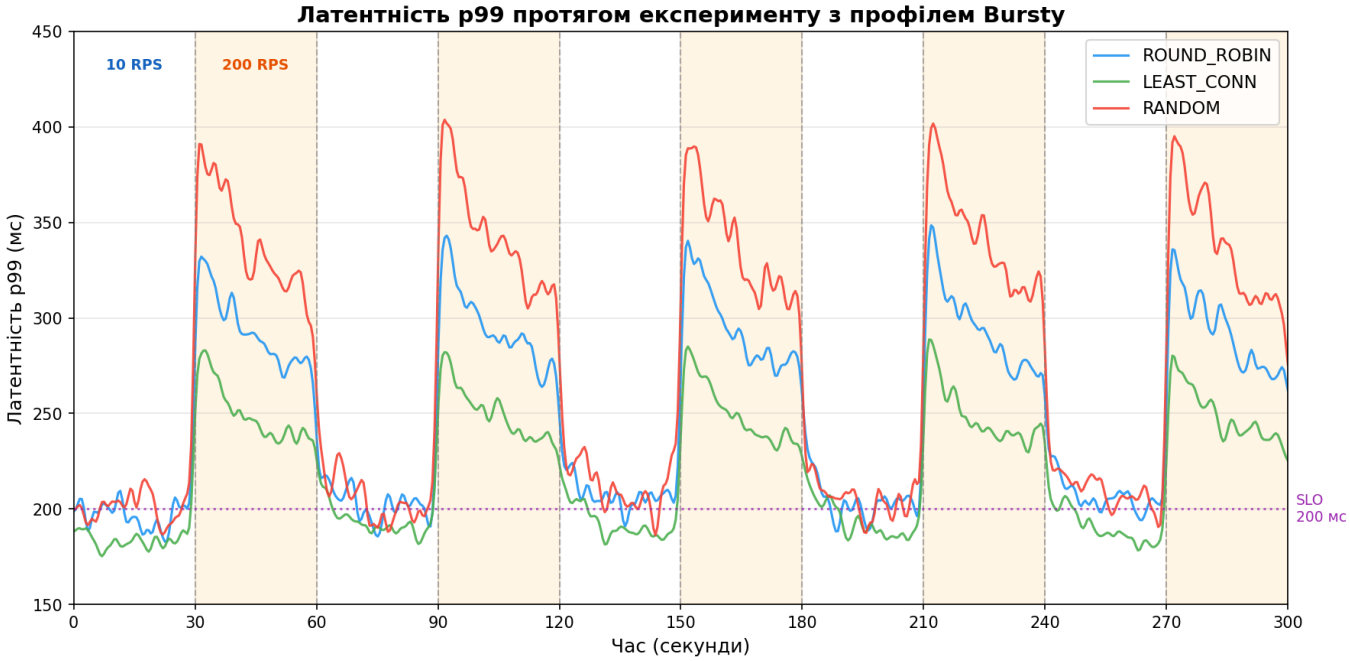

Рис. 3.2. Графік латентності p99 протягом експерименту зі сплесками

Random демонструє найгіршу поведінку при сплесках через відсутність механізму врахування поточного стану екземплярів. Випадковий вибір може направити декілька послідовних запитів до одного екземпляра, що вже обробляє довгий запит, створюючи локальне перевантаження.

Стандартне відхилення результатів при сплесках значно вище, ніж при стабільному навантаженні, що відображає варіативність поведінки системи залежно від фази навантаження в момент вимірювання.

Таблиця 3.5

Результати вимірювання латентності при насиченні (500 RPS)

Алгоритм	p50 (мс)	p95 (мс)	p99 (мс)	E2E p99 (мс)
round-robin	156.7 ± 12.4	487.3 ± 34.8	892.4 ± 67.3	923.7 ± 71.2
least connections	134.2 ± 8.7	412.6 ± 24.1	756.8 ± 48.5	784.3 ± 51.8
random	168.4 ± 15.6	534.2 ± 42.7	978.6 ± 82.4	1012.4 ± 86.9

При насиченні система працює за межами нормальної пропускної здатності (табл. 3.5), що призводить до суттєвого зростання латентності для всіх алгоритмів. Медіана збільшується в 1.9-2.3 рази порівняно зі стабільним навантаженням, p99 — в 3.8-4.8 рази.

Least connections зберігає відносну перевагу і в умовах перевантаження. На рівні p99 алгоритм показує результат на 15.2% кращий за round-robin та на 22.7% кращий за random. Механізм врахування активних з'єднань дозволяє більш ефективно утилізувати доступну пропускну здатність, направляючи нові запити до екземплярів, що звільняються.

Random демонструє найгірші результати через нерівномірність розподілу, що посилюється при високому навантаженні. Окремі екземпляри отримують непропорційно багато запитів, формуючи довгі черги, тоді як інші залишаються недовантаженими.

Високі значення стандартного відхилення при насиченні (до 82.4 мс для p99 random) свідчать про нестабільність поведінки системи в цьому режимі, що є очікуваним для роботи за межами нормальної пропускної здатності (рис. 3.3).

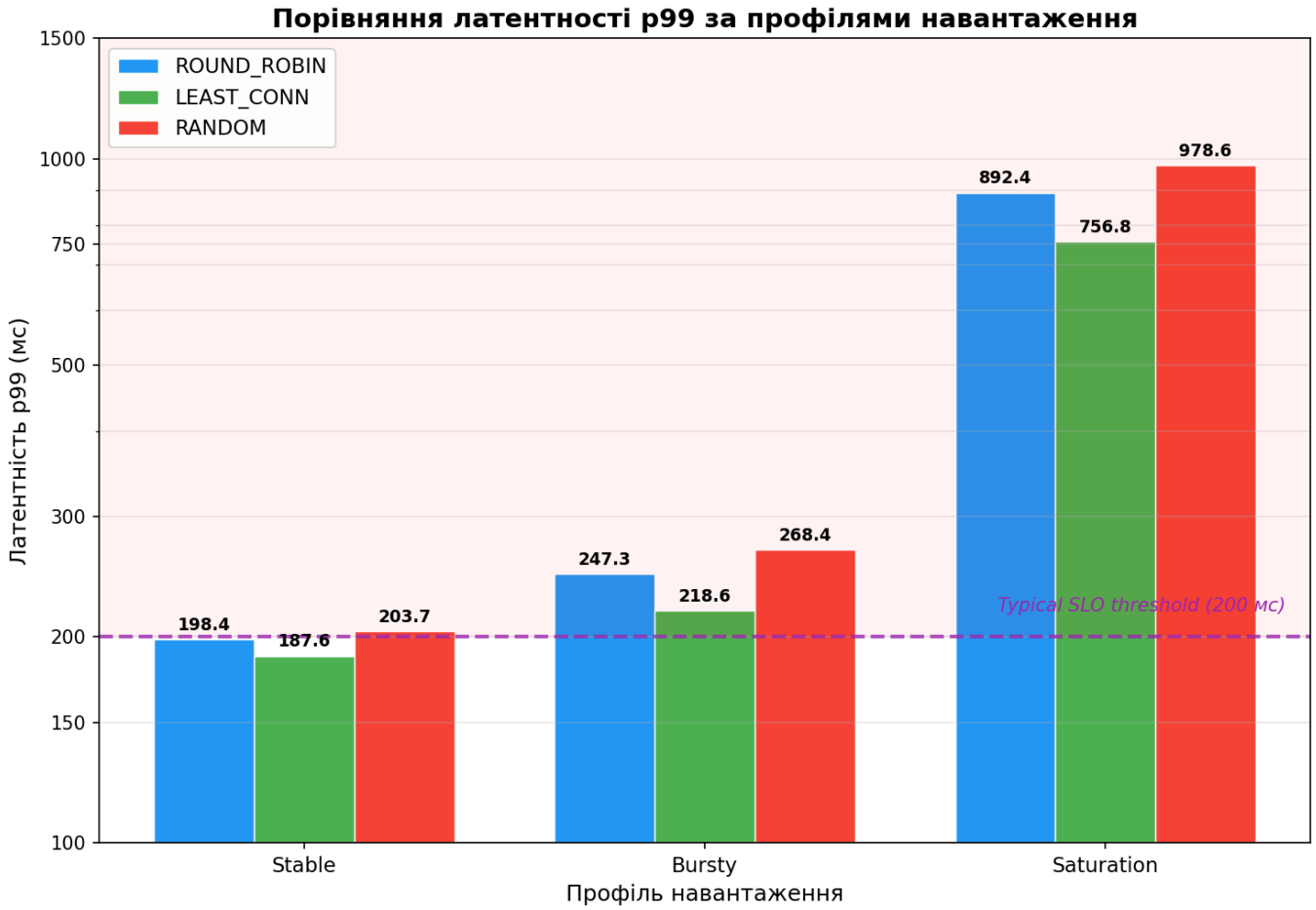

Рис. 3.3. Порівняльна діаграма латентності p99 за профілями навантаження

3.4. Результати вимірювання частоти помилок

Частота помилок характеризує надійність системи та здатність алгоритмів балансування запобігати перевантаженню окремих екземплярів. Помилками вважаються відповіді з HTTP-кодами 5xx, переважно 503 Service Unavailable, що генеруються при спрацюванні circuit breaker. Результати наведено у таблиці 3.6.

Таблиця 3.6

Частота помилок за профілями навантаження (%)

Алгоритм	Стабільний	Сплески	Насичення
round-robin	0.00 ± 0.00	0.12 ± 0.08	4.73 ± 0.89
least connections	0.00 ± 0.00	0.04 ± 0.03	3.21 ± 0.64
random	0.00 ± 0.00	0.23 ± 0.14	6.18 ± 1.12

При стабільному навантаженні жоден з алгоритмів не генерує помилок, що підтверджує коректність налаштування системи та достатність ресурсів для обробки 50 RPS.

При сплесках з'являються поодинокі помилки, спричинені короткочасним перевищенням лімітів connectionPool під час пікових фаз. Least connections демонструє найменшу частоту помилок (0.04%), оскільки ефективніше розподіляє пікове навантаження. random показує найвищу частоту (0.23%) через потенційне накопичення запитів на окремих екземплярах.

При насиченні частота помилок суттєво зростає для всіх алгоритмів через систематичне спрацювання circuit breaker. Least connections забезпечує найнижчу частоту помилок (3.21%), що на 32% менше за round-robin та на 48% менше за random. Ефективніший розподіл навантаження дозволяє максимізувати утилізацію доступної пропускної здатності перед відмовою.

Різниця між алгоритмами в частоті помилок корелює з різницею в хвостовій латентності: алгоритми з вищою p99 латентністю також демонструють вищу частоту помилок, оскільки обидва показники відображають ефективність розподілу навантаження.

3.5. Результати аналізу розподілу навантаження

Рівномірність розподілу запитів між екземплярами сервісу є важливим показником якості балансування, що безпосередньо впливає на латентність відповідей та загальну надійність системи. Нерівномірний розподіл призводить до ситуацій, коли окремі сервери працюють на межі своїх можливостей, тоді як інші залишаються недовантаженими. Для кількісної оцінки рівномірності розподілу в даному дослідженні використано коефіцієнт варіації (Cv) — безрозмірний статистичний показник, що визначається як відношення стандартного відхилення до середнього арифметичного значення кількості запитів, оброблених кожним екземпляром. Перевагою цієї метрики є можливість порівнювати розподіли з різними

абсолютними значеннями навантаження. Таблиця 3.7 містить результати підрахунку розподілу запитів між подами.

Таблиця 3.7

Розподіл запитів між подами backend-service при стабільному навантаженні

Алгоритм	Загальна к-сть запитів	Pod-1 (%)	Pod-2 (%)	Pod-3 (%)	Cv
round-robin	15000 ± 42	33.4 ± 0.2	33.3 ± 0.2	33.3 ± 0.2	0.002
least connections	15000 ± 38	33.8 ± 0.4	33.1 ± 0.3	33.1 ± 0.4	0.012
random	15000 ± 51	34.2 ± 1.1	32.4 ± 0.9	33.4 ± 1.0	0.027

При стабільному навантаженні всі три алгоритми забезпечують прийнятний розподіл з Cv < 0.05. round-robin демонструє практично ідеальну рівномірність (Cv = 0.002), що є очікуваним результатом для детермінованого циклічного алгоритму.

Least connections показує дещо вищу варіабельність (Cv = 0.012), оскільки розподіл залежить від часу обробки запитів. Екземпляри, що обробляють запити швидше, отримують більше нових запитів, що є бажаною адаптивною поведінкою.

Random має найвищу варіабельність (Cv = 0.027), що є статистично очікуваним результатом для випадкового вибору. При 15000 запитів та 3 екземплярах стандартне відхилення частки кожного екземпляра складає приблизно 0.9-1.1%, що відповідає теоретичному значенню.

Таблиця 3.8

Розподіл запитів між подами backend-service при сплесках навантаження

Алгоритм	Загальна к-сть запитів	Pod-1 (%)	Pod-2 (%)	Pod-3 (%)	Cv
round-robin	31500 ± 187	33.5 ± 0.3	33.2 ± 0.3	33.3 ± 0.3	0.005
least connections	31500 ± 156	35.2 ± 0.8	32.1 ± 0.6	32.7 ± 0.7	0.049
random	31500 ± 224	35.8 ± 1.8	31.2 ± 1.5	33.0 ± 1.6	0.069

При сплесках навантаження варіабельність розподілу зростає для всіх алгоритмів. round-robin зберігає низький Cv (0.005), оскільки циклічний перебір не залежить від інтенсивності навантаження (табл. 3.8).

Least connections демонструє помітно вищу варіабельність (Cv = 0.049), що відображає адаптивну поведінку алгоритму. Під час пікових фаз екземпляри з меншим часом обробки отримують більшу частку запитів, що є бажаним ефектом для мінімізації загальної латентності.

Random показує найвищу варіабельність (Cv = 0.069), яка посилюється при сплесках через короткі періоди високої інтенсивності, протягом яких статистичне вирівнювання не встигає відбутися.

Таблиця 3.9

Розподіл запитів між подами backend-service при насиченні

Алгоритм	Загальна к-сть запитів	Pod-1 (%)	Pod-2 (%)	Pod-3 (%)	Cv
round-robin	143265 ± 2841	33.4 ± 0.4	33.3 ± 0.4	33.3 ± 0.4	0.002
least connections	145347 ± 2156	31.2 ± 1.2	35.8 ± 0.9	33.0 ± 1.0	0.072
random	140892 ± 3567	28.7 ± 2.3	38.2 ± 1.9	33.1 ± 2.1	0.086

При насиченні загальна кількість успішно оброблених запитів відрізняється між алгоритмами через різну частоту помилок (табл. 3.9) (рис. 3.4). Least connections обробляє на 1.5% більше запитів ніж round-robin та на 3.2% більше ніж random, що відповідає нижчій частоті помилок.

Round-robin зберігає рівномірний розподіл навіть при перевантаженні, оскільки алгоритм не враховує стан екземплярів. Це призводить до того, що перевантажені екземпляри продовжують отримувати запити, збільшуючи загальну латентність.

Least connections при насиченні демонструє виражену нерівномірність (Cv = 0.072), направляючи більше запитів до екземплярів з швидшою обробкою. Хоча формально це збільшує дисбаланс, реально це є оптимальною стратегією максимізації пропускної здатності.

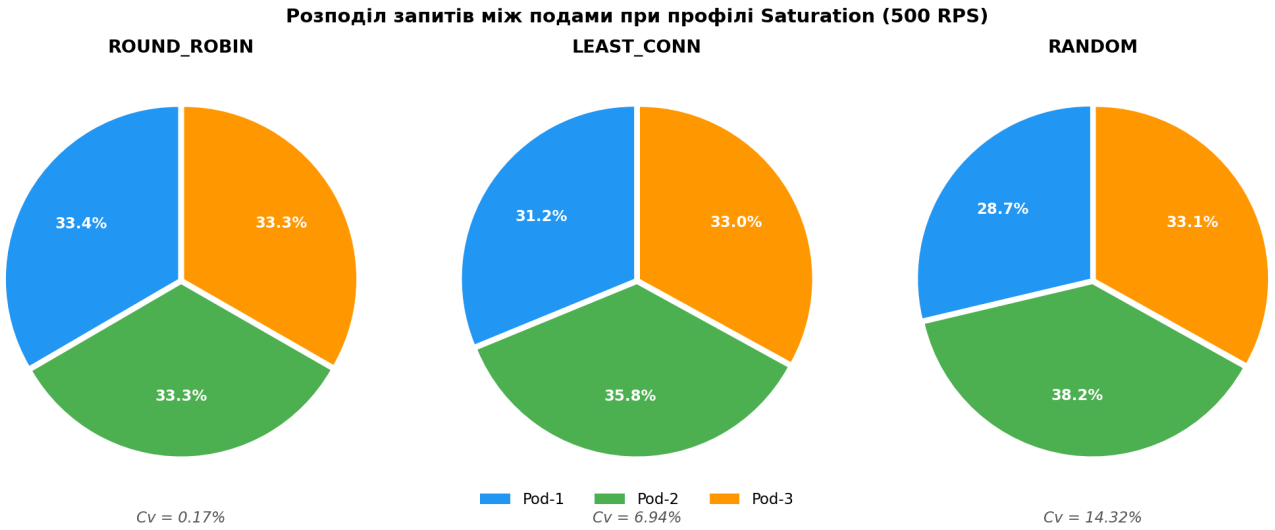

Рис. 3.4. Візуалізація розподілу запитів між подами

3.6. Аналіз дворівневого балансування

Специфікою тестового середовища є наявність двох точок прийняття рішень про балансування: на рівні Ingress Gateway (вибір екземпляра frontend) та на рівні sidecar frontend (вибір екземпляра backend). Аналіз впливу цієї архітектури дозволяє оцінити накопичення ефектів балансування.

Таблиця 3.10

Порівняння латентності frontend та backend при стабільному навантаженні

Алгоритм	Frontend p99 (мс)	Backend p99 (мс)	Overhead (мс)
round-robin	42.3 ± 2.1	147.8 ± 4.9	8.3 ± 1.2
least connections	41.8 ± 1.8	138.4 ± 4.2	7.4 ± 0.9
random	43.1 ± 2.6	151.2 ± 6.1	9.1 ± 1.4

Overhead визначено як різницю між E2E p99 та сумою Frontend p99 і Backend p99, що відображає накладні витрати service mesh (обробка на Gateway, sidecar latency, мережева затримка між компонентами) (табл. 3.10) (рис. 3.5).

Frontend демонструє значно меншу варіативність латентності порівняно з backend через використання нормального розподілу затримки. Різниця між

алгоритмами на рівні frontend є мінімальною (< 3%), тоді як на рівні backend досягає 9.2% між least connections та random.

Цей результат підтверджує припущення про те що вибір алгоритму балансування має більший вплив для сервісів з варіативним часом обробки. Для frontend з нормальним розподілом (σ=5ms) перевага least connections є незначною. Для backend з експоненційним розподілом (коефіцієнт варіації = 1.0) перевага least connections є суттєвою.

Overhead service mesh складає 7.4-9.1 мс, що відповідає типовим значенням для Istio та включає: обробку на Ingress Gateway (2-3 мс), обробку на sidecar frontend (1-2 мс), обробку на sidecar backend (1-2 мс), мережеву затримку (2-3 мс).

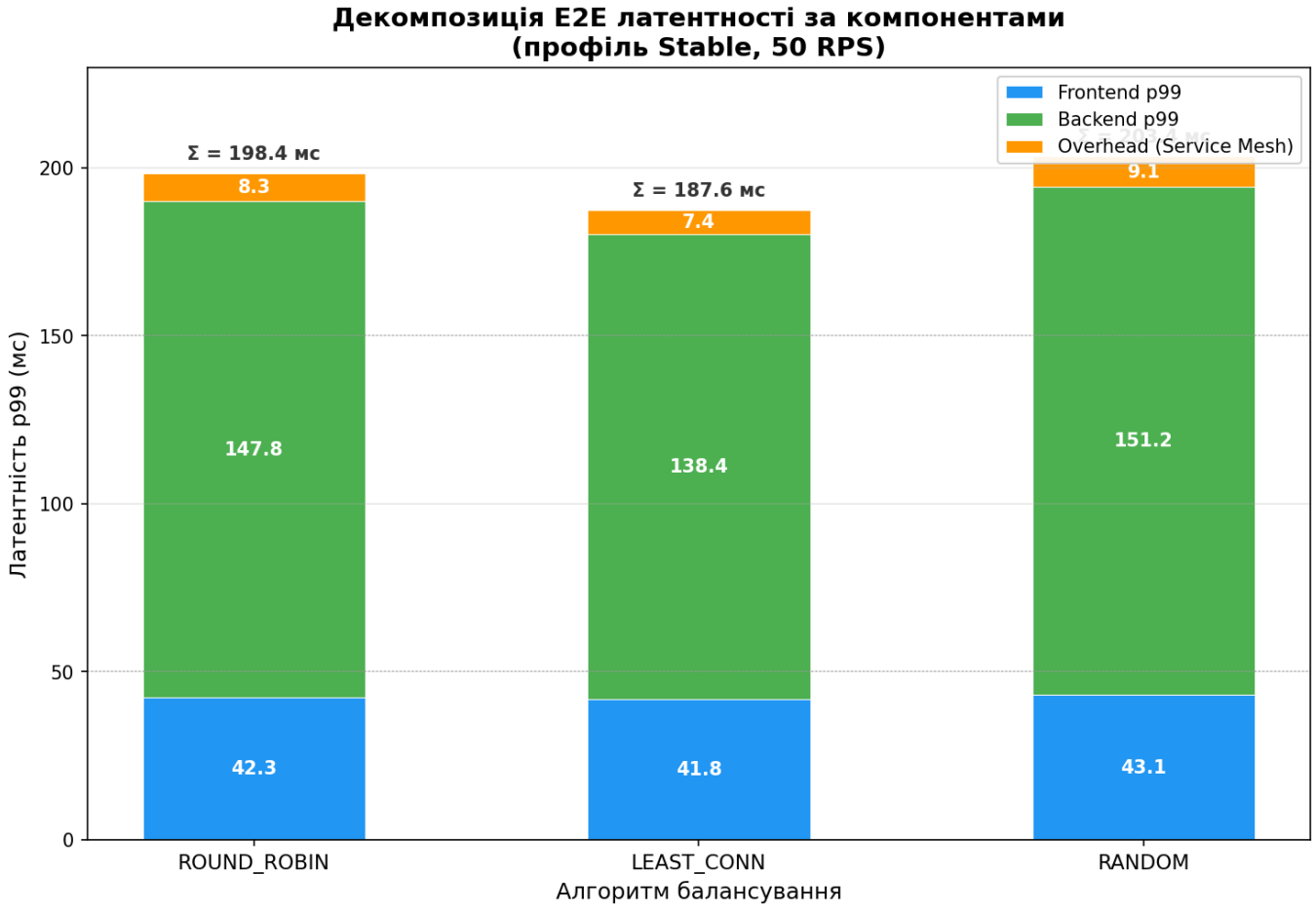

Рис. 3.5. Стовпчикова діаграма декомпозиції E2E латентності

3.7. Порівняльний аналіз алгоритмів

На основі отриманих результатів проведено комплексний порівняльний аналіз алгоритмів балансування за всіма метриками та профілями навантаження.

Таблиця 3.11

Зведена таблиця результатів (нормалізовано відносно round-robin = 100%)

Метрика	Профіль	round-robin	least connections	random
p99 латентність	Стабільний	100%	94.5%	102.7%
p99 латентність	Сплески	100%	88.4%	108.5%
p99 латентність	Насичення	100%	84.8%	109.7%
Частота помилок	Стабільний	100%	0%	0%
Частота помилок	Сплески	100%	33.3%	191.7%
Частота помилок	Насичення	100%	67.9%	130.7%
Рівномірність (1/Cv)	Сплески	—	—	—
Рівномірність (1/Cv)	Стабільний	100%	16.7%	7.4%
Рівномірність (1/Cv)	Насичення	100%	2.8%	2.3%

Нормалізація дозволяє порівняти відносну ефективність алгоритмів (табл. 3.11) (рис. 3.6). Значення менше 100% для латентності та частоти помилок свідчить про перевагу над round-robin. Для рівномірності вище значення відповідає кращому результату (менший Cv).

Least connections демонструє найкращі показники латентності та частоти помилок за всіма профілями навантаження. Перевага збільшується зі зростанням інтенсивності та варіативності навантаження: від 5.5% при стабільному до 15.2% при насиченні. Компромісом є нижча рівномірність розподілу, яка, однак, є бажаною адаптивною поведінкою.

Round-robin забезпечує найкращу рівномірність розподілу та передбачувану поведінку. Алгоритм є оптимальним вибором для сценаріїв, де рівномірність навантаження є пріоритетом (наприклад, для забезпечення консистентного використання ресурсів або коректної роботи автомасштабування).

Random демонструє найгірші показники за всіма метриками крім простоти реалізації. Алгоритм не рекомендується для production-використання, окрім специфічних сценаріїв, де потрібна відсутність будь-якої детермінованості у виборі екземплярів.

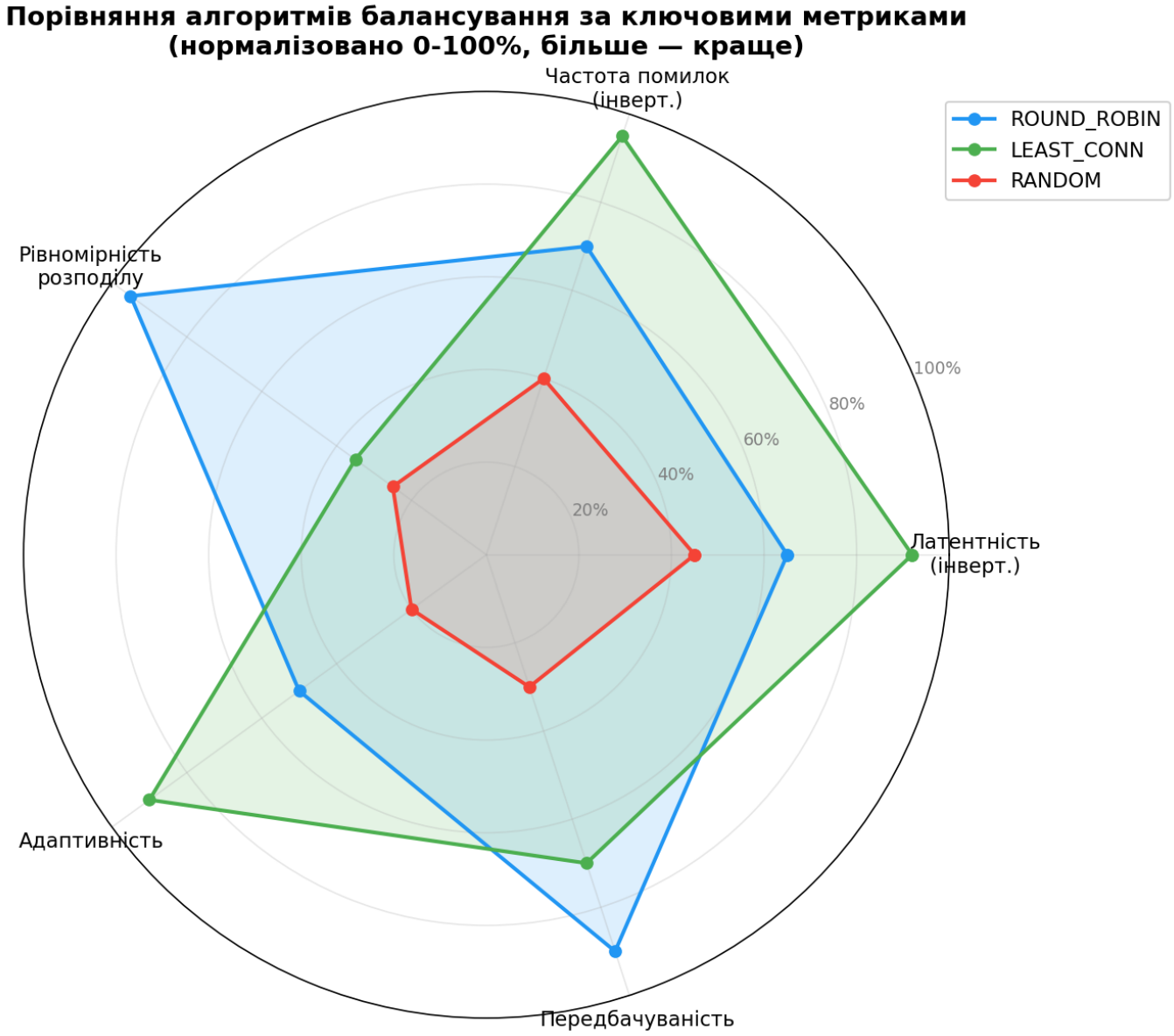

Рис. 3.6. Радарна діаграма порівняння алгоритмів

На основі проведеного дослідження сформульовано практичні рекомендації (табл. 3.12) щодо вибору алгоритму балансування навантаження в Istio service mesh.

Таблиця 3.12

Матриця вибору алгоритму балансування

Характеристика системи	Рекомендований алгоритм	Обґрунтування
Варіативний час обробки (Cv > 0.5)	least connections	Адаптація до різниці в продуктивності екземплярів
Стабільний час обробки (Cv < 0.2)	round-robin	Простота та передбачуваність достатні
Високі вимоги до p99 латентності	least connections	Мінімізація хвостової латентності
Вимоги до рівномірності навантаження	round-robin	Детермінований рівномірний розподіл
Часті сплески навантаження	least connections	Швидка адаптація до змін інтенсивності
Автомасштабування за CPU	round-robin	Рівномірне навантаження для коректних метрик

Для типової мікросервісної системи з backend-сервісами, що мають варіативний час обробки (звернення до бази даних, зовнішні API), рекомендується least connections. Перевага в 10-15% за латентністю p99 та 30-50% за частотою помилок виправдовує незначне збільшення складності.

Для frontend-сервісів та API Gateway з мінімальною власною логікою обробки round-robin є достатнім вибором. Простота та передбачуваність алгоритму спрощують діагностику проблем та забезпечують рівномірне навантаження для коректної роботи горизонтального автомасштабування.

Додаткові налаштування Istio, що рекомендується використовувати разом з обраним алгоритмом балансування, включають outlier detection та retry policy.

Параметри outlier detection забезпечують автоматичне виключення несправних екземплярів з балансування. consecutive5xxErrors=5 визначає поріг послідовних помилок для виключення. interval=30s встановлює період перевірки. baseEjectionTime=60s визначає мінімальний час виключення. maxEjectionPercent=50

обмежує максимальну частку виключених екземплярів для запобігання повній недоступності сервісу.

Retry policy забезпечує автоматичні повторні спроби при тимчасових помилках. attempts=3 визначає максимальну кількість спроб. perTryTimeout=200ms обмежує час очікування кожної спроби. retryOn визначає умови для повторної спроби: помилки 5xx, скидання з'єднання, помилки встановлення з'єднання.

Комбінація least connections з outlier detection та retry policy забезпечує максимальну надійність міжсервісної взаємодії, мінімізуючи вплив тимчасових відмов окремих екземплярів на загальну доступність системи.

3.8. Висновки до розділу 3

У третьому розділі розгорнуто тестову інфраструктуру на базі Kubernetes з Istio 1.20 та проведено 45 експериментів з трьома алгоритмами балансування за трьома профілями навантаження. Зібрано метрики латентності, частоти помилок та рівномірності розподілу.

Експериментально підтверджено перевагу least connections для сервісів з варіативним часом обробки: зниження p99 латентності на 5.5-15.2% та частоти помилок на 32% порівняно з round-robin залежно від профілю навантаження. Сформульовано практичні рекомендації щодо вибору алгоритму.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

У кваліфікаційній роботі магістра при розробці методів та засобів балансування трафіку між сервісами в service mesh на платформі Istio враховано вимоги з охорони праці та безпеки життєдіяльності. Виконання даної роботи передбачає тривалу роботу з комп'ютерною технікою, використання серверного та мережевого обладнання, а також аналіз і обробку значних обсягів інформації. У зв'язку з цим важливим є забезпечення безпечних і комфортних умов праці відповідно до чинного законодавства України.

Відповідно до Закону України «Про охорону праці» роботодавець зобов'язаний створити на кожному робочому місці умови праці, що відповідають нормативно-правовим актам, забезпечити функціонування системи управління охороною праці та здійснювати контроль за станом виробничого середовища. Закон визначає, що умови праці повинні бути безпечними для життя і здоров'я працівників, а також відповідати фізіологічним можливостям людини. Роботодавець несе відповідальність за порушення вимог охорони праці та зобов'язаний вживати профілактичних заходів для запобігання професійним захворюванням і нещасним випадкам [36].

Роботи, пов'язані з дослідженням і налаштуванням service mesh платформи Istio, належать до розумової праці з незначним фізичним навантаженням. Вони виконуються переважно у сидячому положенні з використанням персональних комп'ютерів і супроводжуються підвищеним зоровим та нервово-емоційним навантаженням. За таких умов вирішальне значення для збереження працездатності має відповідність параметрів мікроклімату приміщення встановленим санітарним нормам.

Мікроклімат виробничих приміщень визначається сукупністю параметрів, до яких належать температура повітря, відносна вологість, швидкість руху повітря та інтенсивність теплового випромінювання. Відхилення цих параметрів від

оптимальних значень може призводити до погіршення самопочуття, швидкої втоми, зниження концентрації уваги та продуктивності праці, що особливо критично при виконанні аналітичних і дослідницьких завдань.

Вимоги до мікроклімату виробничих приміщень в Україні регламентуються Державними санітарними нормами ДСН 3.3.6.042-99 «Державні санітарні норми мікроклімату виробничих приміщень» [40]. Згідно з цим нормативним документом, роботи з використанням комп'ютерної техніки належать до категорії легких робіт. Для даної категорії встановлено оптимальні та допустимі параметри мікроклімату залежно від пори року.

Для забезпечення комфортних умов праці при виконанні кваліфікаційної роботи у теплу пору року температура повітря у приміщенні повинна становити 22–24 °C, відносна вологість — 40–60 %, швидкість руху повітря — не більше 0,1 м/с. У холодну пору року рекомендована температура становить 20–22 °C при тих самих значеннях відносної вологості та швидкості руху повітря. Дотримання цих параметрів забезпечує оптимальний тепловий стан організму та знижує ризик розвитку перевтоми.

Особливого значення дотримання норм мікроклімату набуває у приміщеннях, де розміщується серверне або мережеве обладнання, що може бути додатковим джерелом тепловиділення. У таких умовах необхідно забезпечити ефективну вентиляцію та кондиціювання повітря з метою недопущення перегріву як обладнання, так і робочої зони персоналу. Недостатній повітрообмін або підвищена температура можуть негативно впливати на функціональний стан працівника та якість виконання робіт.

Раціональна організація робочого місця та дотримання оптимальних параметрів мікроклімату сприяють зменшенню нервово-емоційного напруження, підвищенню концентрації уваги та стабільності результатів досліджень. У поєднанні з регламентованим режимом праці та відпочинку це дозволяє забезпечити безпечні та комфортні умови виконання робіт, пов'язаних з аналізом трафіку, експериментальним дослідженням алгоритмів балансування навантаження та обробкою результатів.

Таким чином, у кваліфікаційній роботі магістра з теми методів та засобів балансування трафіку між сервісами в service mesh на платформі Istio враховано основні вимоги охорони праці. Дотримання положень Закону України «Про охорону праці» та вимог ДСН 3.3.6.042-99 щодо мікроклімату виробничих приміщень забезпечує створення безпечних і комфортних умов праці та сприяє збереженню працездатності і здоров’я працівників.

4.2. Безпека в надзвичайних ситуаціях

4.2.1. Джерела виникнення шуму і вібрацій. Заходи і засоби від шуму і вібрацій, гігієнічні та допустимі норми

Шум і вібрація є одними з найбільш поширених фізичних факторів виробничого середовища, які супроводжують діяльність людини в умовах техногенно насиченого простору. Їх дія проявляється як у промисловій сфері, так і в офісних та службових приміщеннях, де використовується комп’ютерна та серверна техніка. Вплив цих факторів може мати як безпосередній, так і накопичувальний характер, що зумовлює необхідність їх обов’язкового врахування при організації умов праці. Закон України «Про охорону праці» встановлює обов’язок роботодавця забезпечити такі умови праці, за яких рівні шкідливих і небезпечних факторів не перевищують гігієнічних нормативів і не створюють загрози життю та здоров’ю працівників [36].

Виробничий шум визначається як сукупність безладних звукових коливань, що відрізняються за інтенсивністю, спектральним складом та тривалістю дії. Основними джерелами шуму є технологічне обладнання, електричні машини, компресори, вентиляційні та кондиціонерні установки, а також допоміжні інженерні системи. У сучасних інформаційно-технічних комплексах, зокрема у серверних приміщеннях та центрах обробки даних, значний внесок у загальний шумовий фон вносять системи активного охолодження, мережеві комутатори та блоки живлення. У методичних посібниках з безпеки життєдіяльності підкреслюється, що навіть відносно помірні рівні шуму за умови тривалої дії можуть негативно впливати на функціональний стан організму людини [41].

Вібрація є механічними коливаннями твердих тіл, які виникають у процесі роботи машин і механізмів та передаються на організм людини через опорні поверхні або безпосередній контакт з обладнанням. Джерелами вібрації можуть бути обертові елементи машин, вентилятори, компресори, трансформатори та інші технічні пристрої. У виробничих і серверних приміщеннях вібраційні навантаження, як правило, мають локальний характер, однак за тривалого впливу вони можуть спричиняти функціональні порушення нервової системи, опорно-рухового апарату та периферичного кровообігу [41].

Негативний вплив шуму і вібрацій проявляється у вигляді підвищеної втомлюваності, зниження концентрації уваги, дратівливості та порушення сну. За тривалої дії можливий розвиток професійних захворювань, зокрема шумової приглухуватості та вібраційної хвороби. Особливу небезпеку становить поєднана дія шуму і вібрацій, яка підсилює їх шкідливий вплив і прискорює розвиток негативних змін в організмі людини. Саме тому у навчально-методичних матеріалах наголошується на необхідності комплексного підходу до оцінки та зниження рівнів цих факторів [42].

Гігієнічні та допустимі норми шуму встановлюються ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку», які регламентують граничні рівні шумового навантаження залежно від призначення приміщень і характеру виконуваних робіт [37]. Допустимі параметри загальної та локальної вібрації визначаються ДСН 3.3.6.039-99 «Державні санітарні норми виробничої загальної та локальної вібрації», що враховують частотні характеристики коливань, напрямок їх дії та тривалість впливу на організм людини [38].

Заходи і засоби захисту від шуму і вібрацій поділяються на організаційні, технічні та санітарно-гігієнічні. Організаційні заходи передбачають раціональне планування виробничих і службових приміщень, розміщення джерел підвищеного шуму в окремих зонах, обмеження часу перебування персоналу в умовах підвищеного шумового навантаження, а також впровадження раціональних режимів праці та відпочинку. Технічні заходи включають застосування малошумного обладнання, використання шумо- та вібропоглинаючих матеріалів, встановлення антивібраційних

опор, фундаментів і кожухів, а також регулярне технічне обслуговування обладнання. Санітарно-гігієнічні заходи передбачають систематичний контроль рівнів шуму і вібрацій та забезпечення їх відповідності нормативним значенням, що є необхідною умовою створення безпечного та комфортного виробничого середовища [42].

4.2.2. Основи фізіології праці й комфортних умов життєдіяльності. Класифікація основних форм діяльності людини. Особливості фізичної та розумової праці

Фізіологія праці є науковою дисципліною, що вивчає взаємодію організму людини з умовами виробничого середовища та трудового процесу. Вона досліджує закономірності змін функціонального стану органів і систем людини під впливом різних видів навантажень і розробляє рекомендації щодо раціональної організації праці. Закон України «Про охорону праці» визначає, що організація трудової діяльності повинна забезпечувати збереження життя і здоров’я працівників та відповідати їх фізіологічним можливостям [36].

Комфортні умови життєдіяльності формуються сукупністю факторів виробничого середовища, до яких належать параметри мікроклімату, рівні шуму і вібрацій, освітлення, ергономічні характеристики робочого місця, а також режим праці та відпочинку. Недотримання оптимальних умов призводить до розвитку перевтоми, нервово-емоційного напруження, зниження працездатності та підвищення ризику помилок у професійній діяльності. У нормативних документах з гігієнічної класифікації праці наголошується на необхідності комплексної оцінки умов праці з урахуванням важкості та напруженості трудового процесу [39].

Залежно від характеру навантаження розрізняють основні форми діяльності людини, серед яких виділяють фізичну та розумову працю. Фізична праця характеризується значним навантаженням на м’язову систему та супроводжується підвищеними енергетичними витратами. Вона пов’язана з виконанням механічних дій, переміщенням вантажів, роботою у вимушених або незручних позах, а також з повторюваними рухами. За надмірної інтенсивності фізичної праці відбувається

швидке виснаження енергетичних ресурсів організму, що проявляється зниженням точності рухів, уповільненням реакцій та підвищенням ризику травматизму.

Розумова праця характеризується переважним навантаженням на центральну нервову систему, органи зору та психоемоційну сферу. Вона пов’язана з обробкою великих обсягів інформації, логічним мисленням, аналізом даних, прийняттям відповідальних рішень та контролем складних процесів. Для фахівців у сфері інформаційних технологій, зокрема при дослідженні алгоритмів балансування трафіку в service mesh середовищі, характерна висока інтенсивність розумової праці, що вимагає тривалої концентрації уваги та високого рівня відповідальності. За відсутності раціонального режиму праці та відпочинку тривале нервово-емоційне напруження може призводити до функціональних розладів нервової системи, зниження працездатності та погіршення загального самопочуття [39].

Особливістю сучасної професійної діяльності є поєднання високої інтенсивності розумової праці з обмеженою руховою активністю, що створює передумови для розвитку гіподинамії. За таких умов особливого значення набуває організація комфортних умов життєдіяльності, які включають раціональне планування робочого місця, чергування періодів інтенсивної роботи з регламентованими перервами, а також дотримання гігієнічних вимог. Комплексний підхід до організації умов праці дозволяє знизити негативний вплив нервово-емоційного навантаження, забезпечити стабільний функціональний стан організму та підвищити ефективність професійної діяльності.

Таким чином, урахування основ фізіології праці, класифікації форм діяльності людини та особливостей фізичної і розумової праці є необхідною умовою створення безпечних і комфортних умов життєдіяльності. Це сприяє збереженню здоров’я і працездатності працівників, підвищенню якості виконання професійних завдань та відповідності організації праці вимогам чинного законодавства України.

ВИСНОВКИ

Основні наукові та практичні результати кваліфікаційної роботи можна узагальнити таким чином:

1) Проведено аналіз архітектури мікросервісних систем та визначено ключові проблеми міжсервісної взаємодії, зокрема вплив мережевої латентності, каскадних відмов і нерівномірного розподілу навантаження на продуктивність систем.

2) Досліджено концепцію service mesh та виконано порівняльний аналіз основних платформ, у результаті чого обґрунтовано доцільність використання Istio для експериментального дослідження механізмів управління трафіком.

3) Розглянуто архітектуру платформи Istio, її основні компоненти та механізми управління трафіком, а також проаналізовано алгоритми балансування навантаження round-robin, least connections та random.

4) Формалізовано критерії оцінки ефективності маршрутизації трафіку, що включають перцентилі латентності p50, p95, p99, пропускну здатність, частоту помилок та рівномірність розподілу навантаження між екземплярами сервісів.

5) Спроєктовано та реалізовано експериментальне середовище на базі Kubernetes кластера з Istio service mesh, яке забезпечує контрольовані умови та відтворюваність результатів навантажувального тестування.

6) Розроблено методику проведення експериментів із використанням генератора навантаження Fortio та різних профілів навантаження, що дозволяє комплексно оцінити поведінку алгоритмів балансування в умовах, наближених до production.

7) Проведено експериментальне дослідження впливу алгоритмів балансування на продуктивність мікросервісної системи та отримано кількісні результати щодо латентності, частоти помилок і рівномірності розподілу трафіку.

8) В результаті експерименту встановлено, що алгоритм least connections демонструє кращу адаптивність та нижчі значення хвостової латентності за умов варіативного часу обробки запитів, тоді як round-robin є ефективним для однорідних навантажень, а random характеризується підвищеною варіабельністю результатів.

9) Сформульовано практичні рекомендації щодо вибору алгоритму балансування в Istio залежно від профілю навантаження та вимог до продуктивності і стабільності мікросервісної системи.

10) Визначено обмеження проведеного дослідження та окреслено напрями подальших робіт, зокрема розширення експериментів на складніші топології та використання адаптивних і locality-aware стратегій балансування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. ТНТУ. 2024. 45 с.

2. Луцик Н. С., Луцків А. М., Осухівська Г. М., Тиш Є. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. ТНТУ. 2024. 44 с.

3. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп'ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль. ТНТУ, 2024. 32 с.

4. Микитишин А. Г., Митник М. М., Стухляк П. Д. Телекомунікаційні системи та мережі. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2017. 384 с.

5. Fowler M., Lewis J. Microservices: a definition of this new architectural term. 2014. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 18.11.2025).

6. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 616 p. ISBN: 978-1492034025.

7. Velepucha V., Flores P. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access. 2023. Vol. 11. P. 88339–88358. DOI: 10.1109/ACCESS.2023.3305687.

8. Свергун С., Жаровський Р. Тестування програмного забезпечення побудованого на мікросервісній архітектурі. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (7-8 грудня 2022 року).

Тернопіль: ТНТУ. 2022. С. 92.

9. Söylemez M., Tekinerdogan B. Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. Applied Sciences. 2022. Vol. 12, No. 11. Article 5507. DOI: 10.3390/app12115507.

10. Дячук, О.А.; Жаровський, Р.О. Використання SDN для оптимізації передачі даних в комп'ютерних мережах. Матеріали XI науково-технічна конференція Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (13-14 грудня 2023 року). Тернопіль: ТНТУ. 2023. С. 149-150.

11. Ковтун, Н.; Жаровський, Р. Аналіз засобів протидії вторгненням і атакам на комп'ютерні системи. Матеріали XII Міжнародна науково-технічна конференція молодих учених та студентів «Актуальні задачі сучасних технологій» (6-7 грудня 2023 року). Тернопіль: ТНТУ. 2023. С. 453-454.

12. Istio Documentation. URL: https://istio.io/latest/docs/ (дата звернення: 18.11.2025).

13. Envoy Proxy Documentation. Load Balancing. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/load_balancing (дата звернення: 18.11.2025).

14. LiveWyer. Service Meshes Decoded: A Performance Comparison of Istio vs Linkerd vs Cilium. 2024. URL: https://livewyer.io/blog/2024/05/08/comparison-of-service-meshes/ (дата звернення: 18.11.2025).

15. Bhattacharya R., Gao Y., Wood T. Dynamically Balancing Load with Overload Control for Microservices. ACM Transactions on Autonomous and Adaptive Systems. 2024. Vol. 19, No. 4. Article 22. P. 1–23. DOI: 10.1145/3676167.

16. Дерягін В.Л., Дрогобицький М.В., Луцик Н.С. Засоби автоматичної оптимізації трафіку між мікросервісами в Istio service mesh: Праці XIV наук.-техн. конф. (Тернопіль, ТНТУ ім. І. Пулюя, 11-12 грудня 2025 р.) с. 255-257.

17. Дерягін В.Л., Дрогобицький М.В., Луцик Н.С. Методи моніторингу та оптимізації взаємодії мікросервісів в Istio service mesh: Праці XIII наук.-техн. конф. (Тернопіль, ТНТУ ім. І. Пулюя, 17-18 грудня 2025 р.) с. 111.

18. Zhou J., Li X., Wang Q., Qin X., Miao W., Tian J. Balancing Load: An Adaptive Traffic Management Scheme for Microservices. 2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS). Hainan, China, 2023. P. 641–648. DOI: 10.1109/ICPADS56603.2022.00089.

19. Saxena D., Bhowmik B. Ways of Balancing Load in Microservice Architecture. Recent Advances in Signals and Systems. Springer, 2024. P. 379–396. DOI: 10.1007/978-981-97-4657-6_28.

20. Wan F., Wu X., Zhang Q. Chain-oriented Load Balancing in Microservice System. 2020 World Conference on Computing and Communication Technologies (WCCCT). Ho Chi Minh City, Vietnam, 2020. P. 10–14. DOI: 10.1109/WCCCT49815.2020.9036926.

21. Waseem M., Liang P., Shahin M. A Systematic Mapping Study on Microservices Architecture in DevOps. Journal of Systems and Software. 2020. Vol. 170. Article 110798. DOI: 10.1016/j.jss.2020.110798.

22. Bremler-Barr A., Harchol Y., Hay D., Lavi O., Mishali O. Technical Report: Performance Comparison of Service Mesh Frameworks: the MTLS Test Case. arXiv preprint. 2024. arXiv:2411.02267. 12 p.

23. Zhou X., Peng X., Xie T., Sun J., Ji C., Li W., Ding D. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. IEEE Transactions on Software Engineering. 2021. Vol. 47, No. 2. P. 243–260. DOI: 10.1109/TSE.2018.2887384.

24. Harchol-Balter M. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge: Cambridge University Press, 2013. 576 p. ISBN: 978-1107027503. DOI: 10.1017/CBO9781139226424.

25. Cui J., Chen P., Yu G. A Learning-based Dynamic Load Balancing Approach for Microservice Systems in Multi-cloud Environment. 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). Hong Kong, 2020. P. 334–341. DOI: 10.1109/ICPADS51040.2020.00053.

26. Zhong Z., Xu M., Rodriguez M.A., Xu C., Buyya R. Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions. ACM Computing

Surveys. 2022. Vol. 54, No. 10s. Article 217. P. 1–35. DOI: 10.1145/3510415.

27. Carrión C. Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. ACM Computing Surveys. 2022. Vol. 55, No. 7. Article 138. P. 1–37. DOI: 10.1145/3539606.

28. Toka L., Dobreff G., Fodor B., Sonkoly B. Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. IEEE Transactions on Network and Service Management. 2021. Vol. 18, No. 1. P. 958–972. DOI: 10.1109/TNSM.2021.3052837.

29. Liu J., Wang Q., Zhang S., Hu L., Da Silva D. Sora: A Latency Sensitive Approach for Microservice Soft Resource Adaptation. 24th International Middleware Conference (Middleware '23). Bologna, Italy, 2023. P. 1–14. DOI: 10.1145/3590140.3629109.

30. Yarygina T., Bagge A.H. Overcoming Security Challenges in Microservice Architectures. 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). Bamberg, Germany, 2018. P. 11–20. DOI: 10.1109/SOSE.2018.00011.

31. Bremler-Barr A., Harchol Y., Hay D., Lavi O., Mishali O. Technical Report: Performance Comparison of Service Mesh Frameworks: the MTLS Test Case. arXiv preprint. 2024. arXiv:2411.02267. 12 p.

32. Fortio Documentation. Load Testing Tool. URL: https://fortio.org (дата звернення: 28.11.2025).

33. Gan Y., Zhang Y., Cheng D., et al. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). Providence, RI, 2019. P. 3–18. DOI: 10.1145/3297858.3304013.

34. Kiali Documentation. URL: https://kiali.io/docs/ (дата звернення: 28.11.2025).

35. Box G.E.P., Muller M.E. A Note on the Generation of Random Normal Deviates. The Annals of Mathematical Statistics. 1958. Vol. 29, No. 2. P. 610–611. DOI: 10.1214/aoms/1177706645.

36. Закон України «Про охорону праці» №2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 05.12.2025).

37. ДСН 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку. URL: https://zakon.rada.gov.ua/rada/show/va037282-99 (дата звернення: 05.12.2025).

38. ДСН 3.3.6.039-99. Державні санітарні норми виробничої загальної та локальної вібрації. URL: https://zakon.rada.gov.ua/rada/show/va039282-99 (дата звернення: 05.12.2025).

39. Наказ Міністерства охорони здоров'я України від 08.04.2014 № 248 «Про затвердження Державних санітарних норм та правил "Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу"». URL: https://zakon.rada.gov.ua/laws/show/z0472-14 (дата звернення: 05.12.2025).

40. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. URL: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення: 05.12.2025).

41. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «Безпека в надзвичайних ситуаціях». Тернопіль: ФОП Паляниця В. А. 2022. 155 с.

42. Стручок В.С. Навчальний посібник « Техноекологія та цивільна безпека. Частина «Цивільна безпека» . Тернопіль: ФОП Паляниця В. А. 2022. 150 с.

Додаток А

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.75:004.7
В. Дерягін; М. Дрогобицький; Н. Луцик доктор філософії, доц.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

МЕТОДИ МОНІТОРИНГУ ТА ОПТИМІЗАЦІЇ ВЗАЄМОДІЇ МІКРОСЕРВІСІВ В ISTIO SERVICE MESH

UDC 004.75:004.7
V. Deriahin; M. Drohobytskyi; N. Lutsyk PhD., Assoc. Prof.

METHODS OF MONITORING AND OPTIMIZATION OF MICROSERVICES INTERACTION IN ISTIO SERVICE MESH

Розподілені мікросервісні системи потребують ефективних механізмів моніторингу та оптимізації міжсервісної комунікації для забезпечення надійності та продуктивності. Istio service mesh надає комплексний інструментарій для вирішення цих завдань без модифікації коду застосунків [1]. Зростання популярності мікросервісної архітектури у хмарних середовищах обумовлює необхідність впровадження спеціалізованих рішень для управління складною мережевою взаємодією між сотнями або тисячами окремих сервісів.

Моніторинг в Istio базується на автоматичному збиранні телеметрії через sidecar-проксі Envoy, які перехоплюють весь трафік між сервісами [2]. Система генерує три типи даних спостережуваності: метрики (latency, traffic volume, error rates), розподілене трасування запитів та логи доступу. Інтеграція з Prometheus забезпечує збір метрик, Grafana – їх візуалізацію, а Jaeger або Zipkin – аналіз трасування запитів через ланцюжки сервісів [1]. Ці інструменти дозволяють операторам ідентифікувати вузькі місця продуктивності та швидко локалізувати причини збоїв у розподілених системах.

Оптимізація взаємодії реалізується через механізми балансування навантаження та забезпечення відмовостійкості. Istio підтримує алгоритми Least Request, Round Robin, Random та Consistent Hash, що конфігуруються через DestinationRule [3]. Механізм Circuit Breaker автоматично обмежує запити до перевантажених сервісів при досягненні лімітів з'єднань (maxConnections) або очікуваних запитів (http1MaxPendingRequests), запобігаючи каскадним відмовам [1].

Outlier Detection доповнює оптимізацію автоматичним виключенням несправних екземплярів з пулу балансування на основі аналізу частоти помилок [3]. VirtualService дозволяє налаштовувати timeouts, retries та розподіл трафіку між версіями сервісів для A/B тестування та canary deployments [2].

Комбінація методів моніторингу та оптимізації в Istio забезпечує повну видимість стану системи та автоматичне реагування на проблеми продуктивності, підвищуючи надійність мікросервісних архітектур [4].

Література

1. Mendonça N.C., Box C., Manolache C., Ryan L., 2021. The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture. IEEE Software. Vol. 38, No. 5. P. 17–22.
2. Duque A.O., Klein C., Feng J., Cai X., Skubic B., Elmroth E., 2022. A Qualitative Evaluation of Service Mesh-based Traffic Management for Mobile Edge Cloud. IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). P. 210–219.
3. Alshuqayran N., Ali N., Evans R., 2016. A Systematic Mapping Study in Microservice Architecture. 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). P. 44–51.
4. Elkhatib Y., Povedano Poyato J., 2023. An Evaluation of Service Mesh Frameworks for Edge Systems. 6th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '23). ACM, New York, NY, USA. DOI: 10.1145/3578354.3592867.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

Для визначення реальної потреби конкретної культури ET_0 використовується коефіцієнт K_c, що зберігається у базі даних у вигляді "профілів росту". Система автоматично обирає відповідний коефіцієнт K_c залежно від фази вегетації, яка визначається за кількістю днів від моменту посадки. Апаратна частина базується на мікроконтролері ESP32, датчиках температури та вологості повітря і водяній помпі.

Розроблено архітектуру системи, що дозволяє перейти від реактивного до прогностичного управління поливом. Наукова новизна полягає в економічному обґрунтуванні гібридної моделі збору даних. Запропонований підхід дозволяє використовувати точний метод Пенмана-Монтейта без необхідності встановлення дорогих локальних піранометрів та анемометрів, що робить рішення економічно доцільним для впровадження у малих господарствах та теплицях.

Запропонована прогностична система дозволяє перейти від "поливу за розкладом" до "поливу за реальною потребою". Це створює передумови для значного скорочення витрат води та енергії, уникнення стресу рослин, спричиненого як нестачею, так і надлишком вологи, та, як наслідок, підвищення врожайності.

Література

1. Allen, R. G., Pereira, L. S., Raes, D., & Smith, M. (1998). Crop evapotranspiration - Guidelines for computing crop water requirements - FAO Irrigation and drainage paper 56. Rome: Food and Agriculture Organization of the United Nations.

2. Smith, J. Optimization of Water Usage in Automated Irrigation Systems. Journal of Agricultural Engineering. 2018. P. 201-214.

3. Brown, T., White, K. IoT and Data Analytics for Smart Irrigation. IEEE Transactions on Smart Agriculture. 2020. P. 145-160.

4. Chen, H., Wang, Y. Mathematical Models for Efficient Water Management. Journal of Environmental Systems. 2019. P. 98-112.

5. The Penman-Monteith Method C. 2–3. URL: https://www.researchgate.net/publication/241492864_The_Penman-Monteith_Method.

УДК 004.75

В.І. Дерягін, М.В. Дрогобицький, Н.С. Луцик доктор філософії, доц.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЗАСОБИ АВТОМАТИЧНОЇ ОПТИМІЗАЦІЇ ТРАФІКУ МІЖ МІКРОСЕРВІСАМИ В ISTIO SERVICE MESH

V.L. Deriahin, M.V. Drohobytskyi, N.S. Lutsyk Ph.D., Assoc. Prof.

TOOLS FOR AUTOMATIC OPTIMIZATION OF TRAFFIC BETWEEN MICROSERVICES IN ISTIO SERVICE MESH

Сучасна мікросервісна архітектура значно підвищує швидкість та гнучкість розробки програмного забезпечення, проте водночас збільшує операційну складність розподілених систем [1]. Перехід від монолітних застосунків до мікросервісів створює нові виклики у сфері управління міжсервісною комунікацією, моніторингу та забезпечення безпеки. Service mesh є перспективним підходом до вирішення цієї проблеми шляхом впровадження виділеного інфраструктурного рівня над мікросервісами без необхідності модифікації їх реалізації [1]. Istio, побудований на основі високопродуктивного Envoy proxy, є однією з найпоширеніших реалізацій

service mesh, що забезпечує автоматичне управління трафіком, безпеку та спостережуваність у хмарних середовищах [2].

Архітектура Istio базується на патерні sidecar proxy, де кожен мікросервіс супроводжується окремим контейнером Envoy, що перехоплює весь вхідний та вихідний трафік [2]. Така архітектура забезпечує повну прозорість мережевих взаємодій для застосунків — розробникам не потрібно впроваджувати логіку управління трафіком безпосередньо у код сервісів. Це дозволяє централізовано застосовувати політики маршрутизації, балансування навантаження та забезпечення відмовостійкості без змін у коді застосунків. Компонент control plane (Istiod) динамічно конфігурує всі проксі відповідно до поточної топології mesh через xDS API, забезпечуючи автоматичне оновлення конфігурації при зміні стану кластера [3].

Ключовим механізмом оптимізації трафіку є система балансування навантаження, що підтримує алгоритми Least Request (маршрутизація до екземпляра з найменшою кількістю активних запитів), Round Robin (послідовний розподіл), Random (випадковий вибір) та Consistent Hash (забезпечення session affinity на основі HTTP-заголовків або cookies) [1]. Конфігурація здійснюється через Custom Resource Definition DestinationRule, що дозволяє визначати політики для окремих сервісів або їх підмножин (subsets). За замовчуванням Istio використовує алгоритм Least Request, який динамічно адаптується до поточного навантаження кожного екземпляра сервісу. Дослідження показують, що service mesh забезпечує ефективне управління трафіком навіть у складних розподілених середовищах з тисячами сервісів [4].

Механізм Circuit Breaker є критичним патерном для створення стійких мікросервісних застосунків, що обмежує вплив відмов та затримок окремих сервісів на загальну продуктивність системи [2]. Цей патерн запозичений з електротехніки та реалізує принцип швидкої відмови для захисту системи від перевантаження. Istio реалізує circuit breaker через параметри connection pool: maxConnections (максимальна кількість з'єднань), http1MaxPendingRequests (максимальна кількість очікуваних запитів) та maxRequestsPerConnection. При досягненні встановлених лімітів circuit breaker автоматично блокує нові запити, забезпечуючи швидку відмову замість накопичення черги запитів до перевантаженого сервісу. Після певного періоду circuit breaker переходить у напіввідкритий стан, пропускаючи обмежену кількість тестових запитів для перевірки відновлення сервісу [3].

Outlier Detection доповнює circuit breaker механізмом автоматичного виключення несправних екземплярів із пулу балансування [3]. Система аналізує частоту помилок (consecutive5xxErrors) та виключає проблемні екземпляри на визначений період (baseEjectionTime). Параметр maxEjectionPercent контролює максимальний відсоток екземплярів, що можуть бути виключені одночасно, забезпечуючи мінімальну доступність сервісу. Такий підхід забезпечує самовідновлення системи та мінімізує вплив несправних компонентів на загальну продуктивність.

Додаткові можливості оптимізації надає механізм rate limiting, який дозволяє обмежувати кількість запитів до сервісу за одиницю часу. Це особливо важливо для захисту критичних сервісів від перевантаження під час пікових навантажень або DDoS-атак. Istio підтримує як локальний rate limiting на рівні окремого проксі, так і глобальний через зовнішній сервіс, що забезпечує узгоджене обмеження швидкості запитів у масштабах всього mesh.

Важливим аспектом оптимізації є також механізм retry з експоненційним backoff, який автоматично повторює невдалі запити з наростаючими інтервалами між спробами. Це дозволяє компенсувати тимчасові збої мережі або короткочасну недоступність сервісів без втручання на рівні застосунку. Конфігурація retry включає параметри

кількості спроб, таймаутів та умов, за яких запит вважається невдалим і потребує повторення.

Спостережуваність є невід'ємною складовою оптимізації трафіку в Istio. Система автоматично збирає метрики латентності, пропускної здатності та частоти помилок для кожного з'єднання між сервісами. Інтеграція з Prometheus та Grafana забезпечує візуалізацію цих метрик у реальному часі, а розподілене трасування через Jaeger або Zipkin дозволяє аналізувати шлях запиту через ланцюжок мікросервісів та ідентифікувати вузькі місця продуктивності.

VirtualService забезпечує гнучке управління маршрутизацією трафіку, включаючи розподіл між версіями сервісів для canary deployments та A/B тестування, налаштування timeouts та автоматичних retries з конфігурованою кількістю спроб [2]. Комбінація механізмів балансування навантаження, circuit breaker та outlier detection дозволяє будувати самовідновлювальні системи, де несправні компоненти автоматично ізолюються, а трафік перенаправляється до здорових екземплярів без втручання оператора.

Література

1. Li W., Lemieux Y., Gao J., Zhao Z., Han Y., 2019. Service Mesh: Challenges, State of the Art, and Future Research Opportunities. 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). P. 122–127. DOI: 10.1109/SOSE.2019.00026.

2. Saleh Sedghpour M.R., Klein C., Tordsson J., 2022. An Empirical Study of Service Mesh Traffic Management Policies for Microservices. Proceedings of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22). ACM, New York, NY, USA. P. 17–27. DOI: 10.1145/3489525.3511686.

3. Karn R., Das R., Pant D., Heikkonen J., Kanth R., 2022. Automated Testing and Resilience of Microservice's Network-link using Istio Service Mesh. Proceedings of the 31st Conference of Open Innovations Association FRUCT. DOI: 10.23919/FRUCT54823.2022.9770890.

4. Elkhatib Y., Povedano Poyato J., 2023. An Evaluation of Service Mesh Frameworks for Edge Systems. 6th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '23). ACM, New York, NY, USA. DOI: 10.1145/3578354.3592867.

УДК 004.6:004.855

Л. П. Дмитроца, к.т.н., доцент; О. І. Шубалий, аспірант
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ДОСЛІДЖЕННЯ СУЧАСНИХ ТРЕНДІВ АНАЛІТИКИ BIG DATA

L. P. Dmytrotsa Ph.D, Assoc. Prof.; O. I. Shubalyi, postgraduate student
(Ternopil Ivan Puluj National Technical University, Ukraine)

RESEARCH ON CURRENT TRENDS IN BIG DATA ANALYTICS

Від часу появи на початку 2000-х терміну "Big Data", в процесі активного розвитку цього напрямку Data Science, сучасні акценти досліджень у цій сфері змістилися від задач накопичення даних до пошуку шляхів покращення аналітичної обробки та якнайширшої інтеграції із штучним інтелектом. В той час як на старті “ери великих даних” для їх аналізу у більшій мірі застосовувалися класичні статистичні методи (регресія, баєсівський аналіз і т.п.) та традиційні реляційні бази даних, то