ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ТА МЕРЕЖ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методи та алгоритми балансування навантаження в інфраструктурі хмарних обчислень»

на здобуття освітнього ступеня магістра
зі спеціальності 172 Телекомунікації та радіотехніка
освітньо-професійної програми Телекомунікаційні системи та мережі

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

_____ Вадим Вшивков

Виконав: здобувач вищої освіти гр.
ТСДМ-62 Вадим Вшивков
Керівник: Домрачева Катерина Олексіївна
Кандидат технічних наук, Доцент
Рецензент: Лариса Дакова Валеріївна
Кандидат технічних наук, Доцент

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Телекомунікацій

Кафедра Телекомунікаційних систем та мереж
Ступінь вищої освіти магістр
Спеціальність 172 Телекомунікації та радіотехніка
Освітньо-професійна програма Телекомунікаційні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою Віктор Заїка
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_Вшивкова Вадима Володимировича_____
1. Тема кваліфікаційної роботи: _Методи та алгоритми балансування_____ навантаження в інфраструктурі хмарних обчислень_____
керівник кваліфікаційної роботи _Домрачева Катерина Олексіївна_____ Кандидат технічних наук, Доцент_____,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 2025 р. № ___
2. Строк подання кваліфікаційної роботи «___» _____ 2025 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з хмарних обчислень і балансування навантаження, хмарна платформа Amazon Web Services, інструменти контейнеризації, моніторингу та навантажувального тестування.____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз хмарних обчислювальних систем та методів балансування навантаження.
2. Проєктування і розгортання експериментального середовища в AWS з використанням контейнеризації та систем моніторингу.
3. Проведення навантажувальних експериментів, порівняння алгоритмів балансування навантаження та формування рекомендацій.
5. Перелік ілюстративного матеріалу: презентація
1. Слайди з темою роботи, актуальністю, метою та завданнями дослідження.
2. Слайди з описом архітектури хмарної інфраструктури та експериментального стенду.
3. Слайди з характеристикою алгоритмів балансування навантаження та сценаріїв експериментів.
4. Слайди з основними результатами, практичними рекомендаціями та висновками.
6. Дата видачі завдання «___» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз літератури з хмарних обчислень і балансування	27.10-31.10	виконано
2	Аналіз методів і алгоритмів балансування навантаження	01.11-05.11	виконано
3	Формування вимог до експериментального середовища	06.11-10.11	виконано
4	Проєктування архітектури хмарної інфраструктури	11.11-15.11	виконано
5	Вибір інструментів контейнеризації й моніторингу	16.11-20.11	виконано
6	Реалізація середовища та навантажувальні тести	21.11-25.11	виконано
7	Оформлення тексту роботи та реферату	26.11-30.11	виконано
8	Підготовка демонстраційних матеріалів і захисту	01.12-05.12	виконано

Здобувач вищої освіти _____ Вадим Вшивков____

Керівник
кваліфікаційної роботи _____ Катерина Домрачева_

РЕФЕРАТ

Текст магістерської кваліфікаційної роботи містить 101 сторінку, 30 рисунків, 6 таблиць та 34 джерела.

Об’єкт дослідження – процеси та механізми балансування навантаження в середовищах хмарних обчислень.

Предмет дослідження – методи й алгоритми балансування навантаження в інфраструктурах хмарних обчислень.

Метою роботи є аналіз, моделювання та експериментальне оцінювання сучасних методів і алгоритмів балансування навантаження в хмарних обчисленнях з особливим акцентом на адаптивні підходи, які забезпечують високу продуктивність, стабільність і дотримання вимог SLA за різних типів навантаження, а також на перспективні прогностичні та ML-орієнтовані методи.

Для досягнення поставленої мети в роботі розв’язано такі завдання: узагальнено теоретичні засади та виконано класифікацію алгоритмів балансування навантаження; спроєктовано та реалізовано експериментальне хмарне середовище на базі AWS та Kubernetes; визначено й відтворено чотири типові сценарії навантаження; розроблено методику та практичні рекомендації щодо вибору алгоритмів балансування навантаження за різних умов функціонування.

Методи дослідження включають теоретичний аналіз, експериментальне моделювання в хмарному середовищі на базі AWS та Kubernetes, навантажувальне та продуктивнісне тестування за чотирма сценаріями навантаження (рівномірне навантаження, ступінчасте зростання навантаження (ramp-up), імпульсне навантаження (burst) та деградація вузла), а також статистичне оцінювання затримки, пропускної здатності, стабільності та SLA-орієнтованих метрик.

У роботі представлено комплексне дослідження стратегій балансування навантаження, придатних для сучасних хмарних інфраструктур. Теоретична частина містить детальну класифікацію статичних, динамічних, адаптивних та алгоритмів на основі машинного навчання, а також аналіз ключових показників ефективності, зокрема затримки, пропускної здатності, стабільності та дотримання SLA. Для моделювання реальних умов експлуатації спроєктовано та розгорнуто

експериментальне хмарне середовище на базі AWS та Kubernetes, у межах якого досліджено роботу алгоритмів Round Robin, Least Connections, Shortest Response та Envoy adaptive за визначеними сценаріями навантаження. Окрему увагу приділено огляду прогностичних ML-підходів до балансування навантаження та можливостей їх інтеграції в хмарні системи.

Отримані результати показують, що статичні алгоритми залишаються придатними лише для передбачуваного, рівномірного трафіку, тоді як динамічні алгоритми можуть демонструвати нестабільну поведінку за пікових навантажень і деградації ресурсів. Адаптивне балансування навантаження з використанням Envoy adaptive забезпечує найкращу оперативну чутливість і стабільність у реальному часі за змішаних і нерівномірних профілів навантаження. Прогностичні та ML-орієнтовані методи розглянуто як перспективний напрям, що потенційно здатен підвищити ефективність балансування завдяки проактивному масштабуванню й самонавчанню. Підсумкові рекомендації визначають найбільш доцільні алгоритми для кожного сценарію навантаження та окреслюють напрями подальших досліджень, зокрема розроблення гібридних і самонавчальних систем балансування навантаження.

Ключові слова: хмарні обчислення, балансування навантаження, Kubernetes, адаптивні алгоритми, Envoy adaptive, прогностичні методи, машинне навчання, оцінювання продуктивності, угоди про рівень сервісу (SLA).

ABSTRACT

The master’s thesis comprises 101 pages, 30 figures, 6 tables, and 34 references.

The object of research is the processes and mechanisms of load balancing in cloud computing environments.

The subject of research is the methods and algorithms for load balancing in cloud computing infrastructures.

The aim of the thesis is to analyse, model, and experimentally evaluate modern load balancing methods and algorithms in cloud computing, with a particular focus on adaptive approaches that ensure high performance, stability, and compliance with SLA requirements under different types of workload, as well as on promising predictive and ML-oriented methods.

To achieve this aim, the following tasks are addressed: to generalise the theoretical foundations and provide a classification of load balancing algorithms; to design and implement an experimental cloud environment based on AWS and Kubernetes; to define and reproduce four typical workload scenarios; and to develop a methodology and practical recommendations for choosing load balancing algorithms under different operating conditions.

The research methods include theoretical analysis, experimental modelling in a cloud environment based on AWS and Kubernetes, load and performance testing under four workload scenarios (uniform load, ramp-up load, burst load, and node degradation), as well as statistical evaluation of latency, throughput, stability, and SLA-oriented metrics.

The thesis presents a comprehensive study of load balancing strategies applicable to modern cloud infrastructures. The theoretical part provides a detailed classification of static, dynamic, adaptive, and machine-learning-based algorithms, together with an analysis of key performance indicators such as latency, throughput, stability, and SLA compliance. To model realistic operational conditions, an experimental cloud environment based on AWS and Kubernetes was designed and deployed, within which the operation of Round Robin, Least Connections, Shortest Response, and Envoy adaptive algorithms was investigated under the defined workload scenarios. Special

attention is given to a review of predictive ML-based approaches to load balancing and the possibilities of their integration into cloud systems.

The results show that static algorithms remain suitable only for predictable, uniform traffic, whereas dynamic algorithms may exhibit unstable behaviour under peak loads and resource degradation. Adaptive load balancing with Envoy adaptive provides the best real-time responsiveness and stability for mixed and non-uniform workload profiles. Predictive and ML-oriented methods are considered as a promising direction that can potentially improve load balancing efficiency through proactive scaling and self-learning. The final recommendations identify the most appropriate algorithms for each workload scenario and outline directions for further research, including the development of hybrid and self-learning load balancing systems.

KEYWORDS: CLOUD COMPUTING, LOAD BALANCING, KUBERNETES, ADAPTIVE ALGORITHMS, ENVOY ADAPTIVE, PREDICTIVE METHODS, MACHINE LEARNING, PERFORMANCE EVALUATION, SERVICE LEVEL AGREEMENTS (SLA).

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Телекомунікацій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Вшивков В. В. до захисту кваліфікаційної роботи
за спеціальністю 172 Телекомунікації та радіотехніка
освітньо-професійної програми Телекомунікаційні системи та мережі
на тему: «Методи та алгоритми балансування навантаження в інфраструктурі хмарних обчислень».
Кваліфікаційна робота і рецензія додаються.
Директор ННІ Кравченко Владислав Ігорович

Висновок керівника кваліфікаційної роботи

Здобувач Вшивков Вадим Володимирович під час виконання магістерської кваліфікаційної роботи показав достатній рівень теоретичної та практичної підготовки, уміння працювати з сучасними інформаційними технологіями та науково-технічною літературою. У процесі розв’язання поставлених завдань проявив відповідальність, наполегливість і здатність самостійно опрацьовувати отримані результати, своєчасно враховував зауваження та рекомендації керівника.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Вшивкова В. В. на оцінку «**добре**» та присвоїти йому(їй) кваліфікацію магістра.

Керівник кваліфікаційної роботи _____ Катерина Домрачева

«____» _____ 2025 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Вшивков В. В. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою ТСМ _____ Віктор Заїка

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу
на здобуття освітнього ступеня магістра

здобувача вищої освіти Вшивкова Вадима Володимировича
на тему «Методи та алгоритми балансування навантаження в інфраструктурі хмарних обчислень»

Актуальність.

Актуальність дослідження зумовлена стрімким зростанням обсягів трафіку в хмарних обчислювальних середовищах, широким упровадженням мікросервісних архітектур та жорсткими вимогами до доступності й якості обслуговування (SLA). Це потребує використання ефективних методів балансування навантаження, здатних забезпечувати стійку роботу сервісів за різних профілів трафіку та часткової деградації ресурсів. У роботі обґрунтовано доцільність детального аналізу та експериментальної оцінки алгоритмів балансування в інфраструктурі AWS та Kubernetes, що формує актуальну науково-прикладну задачу підвищення ефективності хмарних систем.

Позитивні сторони.

1. Згідно з поданим планом, зміст кваліфікаційної роботи повністю відповідає вимогам до магістерських досліджень, має чітку структуру, логічну послідовність викладу та належний рівень опрацювання теоретичного матеріалу.
2. Розгорнуто й налаштовано експериментальне середовище на базі AWS та Kubernetes із використанням сучасних інструментів моніторингу та навантажувального тестування (Prometheus, Grafana, k6), що дозволило отримати репрезентативні кількісні результати для чотирьох сценаріїв навантаження.
3. На основі аналізу результатів проведено обґрунтоване порівняння алгоритмів Round Robin, Least Connections, SR (random/approx) та Envoy adaptive та сформульовано практичні рекомендації щодо вибору методів балансування для різних режимів роботи хмарних сервісів, що свідчить про прикладну цінність роботи.

Недоліки.

1. Аналіз інтелектуальних та прогнозних (ML-орієнтованих) підходів до балансування навантаження має переважно оглядовий характер; більш глибока кількісна оцінка таких методів могла б посилити наукову складову дослідження.
2. Доцільним було б розширити розгляд механізмів автоматичного масштабування та енергоефективності, зокрема шляхом додаткових експериментів або моделювання взаємодії балансування з autoscaling-політиками.

Відзначені зауваження не впливають на загальну позитивну оцінку валіфікаційної роботи магістерської.

Висновок: кваліфікаційна робота на здобуття ступеня магістра заслуговує оцінку "**добре**", а здобувач Вшивков Вадим Володимирович заслуговує присвоєння кваліфікації: магістр з електронних комунікацій та радіотехніки.

Рецензент: _____ _Лариса Дакова_к. т. н., доцент__

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 12
ВСТУП 14
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНИХ ОБЧИСЛЕННЯХ 20
1.1. Поняття та архітектура хмарних обчислень 20
1.2. Типи сервісних моделей (IaaS, PaaS, SaaS) та їх вплив на балансування навантаження 22
1.3. Проблематика нерівномірного розподілу навантаження в хмарних середовищах 25
1.4. Роль балансування навантаження у забезпеченні масштабованості та відмовостійкості 27
1.5. Класифікація методів і алгоритмів балансування навантаження 30
1.6. Статичні алгоритми балансування навантаження 31
1.7. Динамічні алгоритми балансування навантаження 34
1.8. Інтелектуальні та прогнозні методи балансування 37
1.9. Архітектурні рішення та рівні реалізації балансування 40
1.10. Критерії оцінювання ефективності балансування 45
1.11. Огляд сучасних досліджень та технологічних рішень у сфері балансування в хмарних інфраструктурах 50
1.12. Висновки до розділу 1 55
РОЗДІЛ 2. АНАЛІТИКО-ДОСЛІДНИЦЬКИЙ АНАЛІЗ ФУНКЦІОНУВАННЯ СИСТЕМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ 58
2.1. Характеристика бази дослідження та інфраструктури хмарної платформи 58
2.2. Джерела фактичних даних, показники оцінювання та період дослідження 60
2.3. Аналітичний огляд результатів роботи алгоритмів балансування 64
2.4. Висновки до розділу 2 66
РОЗДІЛ 3. КОНСТРУКТИВНІ РІШЕННЯ ЩОДО ПОЛІПШЕННЯ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНІЙ ІНФРАСТРУКТУРІ 69

3.1. Узагальнення результатів аналітико-дослідницького розділу 69
3.2. Методика вибору алгоритмів балансування навантаження у хмарній платформі 76
3.3. Рекомендації щодо вибору алгоритмів для типових сценаріїв навантаження 80
3.4. Проєкт адаптивної системи балансування навантаження на базі AWS та Kubernetes 100
3.5. Оцінка очікуваних ефектів від впровадження запропонованих рішень 108
3.6. Висновки до розділу 3 109
ВИСНОВКИ 111
ПЕРЕЛІК ПОСИЛАНЬ 114
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 118

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ALB – Application Load Balancer, балансувальник рівня застосунків (AWS)

ALBE – Adaptive Load Balancing (Envoy), адаптивне балансування, що реагує на зміни метрик продуктивності (Envoy)

API – Application Programming Interface, інтерфейс прикладного програмування

Autoscaling – автоматичне масштабування ресурсів у хмарних середовищах

AWS – Amazon Web Services, хмарна інфраструктурна платформа

CloudWatch – сервіс моніторингу та логування AWS

CPU – Central Processing Unit, центральний процесор

CSV – Comma-Separated Values, табличний формат даних

EC2 – Elastic Compute Cloud, віртуальні машини AWS

Energy – метрика енергоспоживання в алгоритмах оптимізації

Grafana – система візуалізації часових рядів

HAProxy – високопродуктивний балансувальник навантаження

HPA – Horizontal Pod Autoscaler, автоматичне масштабування Pod-ів

HTTP – HyperText Transfer Protocol

IaaS – Infrastructure as a Service, інфраструктура як послуга

Ingress – компонент Kubernetes для маршрутизації HTTP-трафіку

IP – Internet Protocol

JMeter – інструмент навантажувального тестування

Kubernetes – система оркестрації контейнерів

L4 / L7 – рівні мережевої моделі OSI (транспортний і прикладний)

Latency – затримка (час відповіді)

LC – Least Connections, розподіл запитів на вузол із найменшою кількістю активних з’єднань

Load Balancer – балансувальник навантаження

Locust – інструмент генерації тестового трафіку

LSTM – Long Short-Term Memory, нейромережа для прогнозування

ML – Machine Learning, машинне навчання

ML-LB – ML-Predictive Load Balancing, балансування на основі машинного навчання з прогнозуванням навантаження

NAT – Network Address Translation

Nginx – веб-сервер та L7 Ingress-контролер

NLB – Network Load Balancer, L4-балансувальник AWS

PaaS – Platform as a Service, платформа як послуга

Pod – базова обчислювальна одиниця в Kubernetes

Prometheus – система моніторингу часових рядів

RPS – Requests per Second, кількість запитів за секунду

RR – Round Robin, алгоритм циклічного розподілу запитів

SaaS – Software as a Service, програмне забезпечення як послуга

SLA – Service Level Agreement, угода про рівень сервісу

Service Mesh – архітектурний шар керування мережею мікросервісів

SR – Shortest Response, вибір вузла з найменшим часом відповіді

SRT – Shortest Response Time, алгоритм мінімального часу відповіді

TCP/UDP – транспортні протоколи

Throughput (TPS) – пропускна здатність системи

WRR – Weighted Round Robin, зважене циклічне балансування

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів даних, кількості користувачів та складності програмних систем. Все ширше впроваджуються хмарні обчислення, які дають змогу гнучко масштабувати ресурси, забезпечувати високу доступність сервісів і скорочувати витрати на підтримку інфраструктури. Для бізнесу, державних установ та критично важливих сервісів особливої ваги набувають стабільність роботи, дотримання угод про рівень обслуговування (SLA), а також раціональне використання обчислювальних ресурсів.

Нерівномірний розподіл навантаження між вузлами хмарної інфраструктури призводить до локальних перевантажень, зростання часу відповіді, зниження пропускної здатності та можливих відмов сервісів. Ці проблеми загострюються за умов високодинамічних і непередбачуваних потоків запитів, характерних для веб-застосунків, мікросервісних архітектур та сервісів обробки даних у реальному часі. У таких умовах балансування навантаження стає ключовим механізмом забезпечення масштабованості, відмовостійкості та стабільної якості обслуговування.

У сучасних наукових і прикладних роботах запропоновано широкий спектр статичних, динамічних, адаптивних та інтелектуальних алгоритмів балансування навантаження, розроблено різноманітні архітектурні підходи до побудови хмарних платформ та сервіс-меш-рішень. Разом з тим, недостатньо розробленими залишаються питання комплексного порівняння алгоритмів у реалістичному хмарному середовищі, поєднання класичних і прогнозних підходів, а також методики вибору алгоритмів для типових сценаріїв навантаження. Особливо актуальним є дослідження роботи таких алгоритмів у зв’язці з інструментами моніторингу та autoscaling у середовищі хмарних провайдерів і контейнерних оркестраторів.

Виходячи з цього, дослідження методів та алгоритмів балансування навантаження в інфраструктурі хмарних обчислень, розроблення експериментального середовища на базі сучасних хмарних та контейнерних технологій і формування практичних рекомендацій щодо вибору алгоритмів для різних типів навантаження є актуальним науковим та прикладним завданням.

Мета і завдання дослідження. Мета роботи полягає в підвищенні ефективності балансування навантаження в інфраструктурі хмарних обчислень шляхом розроблення та обґрунтування методичних і практичних підходів до вибору та застосування методів і алгоритмів розподілу запитів за різних сценаріїв навантаження.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

1. Проаналізувати теоретичні засади хмарних обчислень та класифікувати основні методи й алгоритми балансування навантаження в хмарних середовищах.
2. Сформувати систему критеріїв оцінювання ефективності балансування (середній час відповіді, квантилі затримок, пропускна здатність, стабільність під навантаженням, виконання SLA, енергоефективність) та обґрунтувати доцільність їх комплексного застосування.
3. Розробити архітектуру експериментального середовища на базі хмарної платформи та кластера Kubernetes з використанням багаторівневих (L4/L7) балансувальників, системи моніторингу та інструментів навантажувального тестування.
4. Реалізувати експериментальний стенд у хмарному середовищі, визначити та налаштувати сценарії навантаження (рівномірне, зростаюче, пікове, асиметричне, з деградацією вузлів) і забезпечити збір необхідних метрик.
5. Провести серію навантажувальних експериментів для статичних, динамічних, адаптивних та прогнозних алгоритмів балансування навантаження й отримати кількісні показники їх ефективності за сформованою системою критеріїв.

6. Здійснити порівняльний аналіз ефективності алгоритмів за різних сценаріїв навантаження, виявити їх сильні та слабкі сторони.
7. Розробити методику вибору алгоритмів балансування для типових профілів навантаження та запропонувати проєкт адаптивної системи балансування навантаження в хмарній інфраструктурі.

Об’єкт дослідження – процес розподілу навантаження між обчислювальними ресурсами в інфраструктурі хмарних обчислень, який забезпечує надання користувачам доступу до сервісів із заданими показниками продуктивності, надійності та якості обслуговування.

Предмет дослідження – методи, алгоритми та архітектурні рішення балансування навантаження в хмарних платформах на базі віртуалізації та контейнеризації (зокрема із використанням Kubernetes та багаторівневих балансувальників), а також їх вплив на показники ефективності, стабільності роботи та відповідності вимогам SLA за різних типів навантаження.

Методи дослідження. Для досягнення поставленої мети використано сукупність взаємодоповнювальних методів дослідження. Теоретичну частину роботи виконано із застосуванням методів аналізу й узагальнення наукових джерел, системного підходу до класифікації методів і алгоритмів балансування навантаження, формалізації критеріїв оцінювання та побудови логічних моделей.

Практичну частину дослідження здійснено на основі експериментального моделювання в хмарній інфраструктурі: розгорнуто кластер Kubernetes, налаштовано мережеві та прикладні балансувальники, систему моніторингу та середовище навантажувального тестування. Для оцінювання ефективності алгоритмів використано статистичні методи обробки результатів експериментів (обчислення середніх значень, квантилів, аналіз варіації та розподілу показників).

Наукова новизна одержаних результатів полягає в наступному:

- узагальнено теоретичні засади хмарних обчислень у контексті задач балансування навантаження, запропоновано уточнену класифікацію статичних, динамічних, адаптивних та інтелектуальних алгоритмів балансування з урахуванням їх застосування у хмарних платформах;

- сформовано комплексну систему критеріїв оцінювання ефективності балансування навантаження, яка враховує час відповіді, пропускну здатність, стабільність під навантаженням, виконання SLA та енергоефективність і дозволяє порівнювати різні алгоритми в єдиному експериментальному середовищі;
- розроблено та реалізовано експериментальне середовище на базі хмарної платформи з використанням кластера Kubernetes, багаторівневих балансувальників, системи моніторингу та інструментів навантажувального тестування, що забезпечує можливість детального аналізу поведінки алгоритмів за різних сценаріїв навантаження;
- запропоновано методику комплексного порівняння статичних, динамічних, адаптивних та прогнозних алгоритмів балансування навантаження у типовій для хмарних сервісів інфраструктурі;
- розроблено проєкт адаптивної системи балансування навантаження з інтеграцією прогнозної моделі, що враховує очікувану інтенсивність запитів і дозволяє підвищити стабільність роботи системи та рівень виконання SLA у динамічних умовах.

Апробація результатів та публікації. Основні результати дослідження викладено у тезах доповідей, підготовлених і поданих до участі в профільних науково-практичних конференціях, присвячених проблемам хмарних обчислень, балансування навантаження та інтелектуальних методів керування ресурсами. Матеріали роботи обговорювалися на засіданнях випускової кафедри, що дало змогу уточнити постановку задачі, структуру експериментального середовища та інтерпретацію отриманих результатів.

Теоретична значущість роботи полягає в подальшому розвитку та систематизації уявлень про методи й алгоритми балансування навантаження в хмарних інфраструктурах, формуванні цілісної класифікації підходів та уточненні їх місця в архітектурі хмарних систем.

Методична значущість полягає в розробленні методики комплексного оцінювання ефективності алгоритмів балансування, яка охоплює відбір критеріїв,

побудову експериментального середовища, вибір сценаріїв навантаження, організацію збору та аналізу метрик. Запропоновані підходи можуть бути використані як основа для подальших досліджень і лабораторних практикумів з курсів, пов’язаних із хмарними обчисленнями, розподіленими системами та DevOps-практиками.

Практична значущість результатів полягає в можливості прямого застосування розробленої методики та запропонованих рекомендацій при проєктуванні, налаштуванні й експлуатації хмарних інфраструктур. Експериментальне середовище та проєкт адаптивної системи балансування можуть бути використані для побудови й оптимізації рішень балансування навантаження у реальних хмарних платформах, що сприяє підвищенню стабільності роботи сервісів, зниженню ризику порушення SLA та більш раціональному використанню обчислювальних ресурсів.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та демонстраційних матеріалів.

У першому розділі наведено теоретико-методичні основи балансування навантаження в хмарних обчисленнях: розглянуто поняття та архітектуру хмарних обчислень, типи сервісних моделей та їх вплив на балансування, проаналізовано проблематику нерівномірного розподілу навантаження, роль балансування у забезпеченні масштабованості та відмовостійкості, класифіковано методи й алгоритми балансування, описано архітектурні рівні реалізації балансування та сформовано критерії оцінювання ефективності.

У другому розділі подано аналітико-дослідницький аналіз функціонування системи балансування навантаження: охарактеризовано базу дослідження та інфраструктуру хмарної платформи, описано джерела фактичних даних і показники оцінювання, наведено результати роботи алгоритмів балансування за різних сценаріїв навантаження та здійснено їх аналітичний огляд.

У третьому розділі розроблено конструктивні рішення щодо поліпшення балансування навантаження в хмарній інфраструктурі: узагальнено результати аналітико-дослідницького розділу, запропоновано методику вибору алгоритмів для

типових сценаріїв навантаження, сформовано рекомендації щодо їх практичного застосування, представлено проєкт адаптивної системи балансування навантаження на базі хмарної платформи та Kubernetes і виконано оцінку очікуваних ефектів від впровадження запропонованих рішень.

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНИХ ОБЧИСЛЕННЯХ

У цьому розділі розкрито теоретичні засади балансування навантаження в хмарних обчисленнях, узагальнено існуючі підходи до організації хмарної інфраструктури та ролі балансувальників у забезпеченні масштабованості й відмовостійкості сервісів. Проведено критичний огляд основних методів і алгоритмів балансування, проаналізовано критерії їх ефективності та сучасні тенденції розвитку відповідних технологічних рішень. Окрему увагу приділено методичним підходам до подальшого дослідження, що ґрунтується на порівняльному аналізі алгоритмів за вибраними метриками продуктивності в умовах різних типів навантаження.

1.1. Поняття та архітектура хмарних обчислень

Сучасні хмарні обчислення ґрунтуються на ідеї динамічного розподілу ресурсів між користувачами, що забезпечує еластичність, масштабованість та відмовостійкість систем [1]. Хмарні обчислення - це модель надання обчислювальних ресурсів за потреби через інтернет; у таких системах критично важливим є механізм ефективного розподілу навантаження.

Архітектуру хмарних обчислень у вигляді багаторівневої моделі подано на рисунку 1.1. Віртуалізація створює основу для масштабування та балансування навантаження. Такий підхід значно знижує витрати на обслуговування IT-систем і підвищує гнучкість бізнес-процесів.

Архітектура хмарних обчислень традиційно представлена багаторівневою структурою, що включає такі основні компоненти:

- Інфраструктурний рівень (Infrastructure Layer) - забезпечує фізичні ресурси (сервери, сховища, мережі), на базі яких створюються віртуальні ресурси.

Віртуалізація здійснюється за допомогою гіпервізорів (наприклад, VMware ESXi, KVM, Xen, Hyper-V), що дозволяють розподіляти обчислювальні ресурси між ізольованими віртуальними машинами.

- Платформний рівень (Platform Layer) - надає середовище для розробки, розгортання та керування додатками. Сюди входять інструменти оркестрації контейнерів (Kubernetes, Docker Swarm), системи управління базами даних, черги повідомлень, фреймворки CI/CD тощо.
- Рівень програмних сервісів (Application Layer) - представлений прикладними програмами та сервісами, які безпосередньо використовуються кінцевими користувачами через веб-інтерфейси або API (наприклад, Google Workspace, Microsoft 365, Amazon S3).

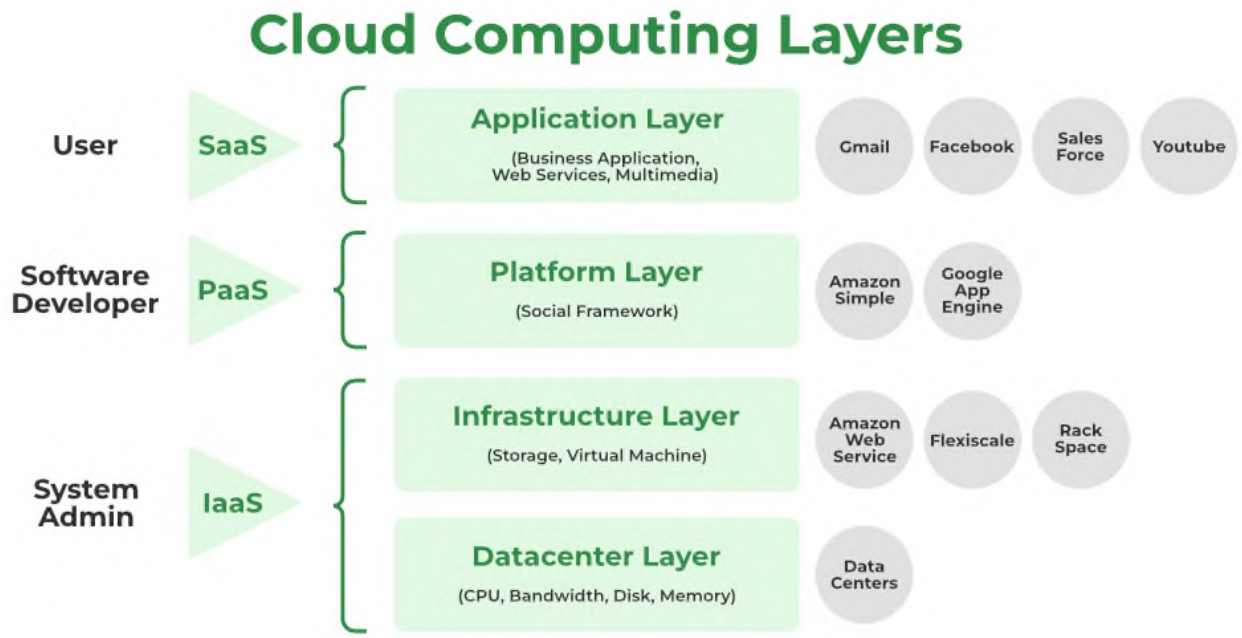


Рисунок 1.1 Архітектура хмарних обчислень: рівні та компоненти

Архітектура хмар визначає можливості масштабування та вибір механізмів балансування. Таким чином, архітектура хмарних обчислень формує основу для реалізації ефективних механізмів розподілу навантаження, оскільки саме структура рівнів і типи взаємодії компонентів визначають подальші можливості масштабування та відмовостійкості системи. Обрані моделі розгортання (public, private, hybrid cloud), спосіб організації сервісів (монолітний чи мікросервісний підхід), а також спосіб маршрутизації трафіку між вузлами безпосередньо

впливають на те, які алгоритми балансування можуть бути задіяні, наскільки ефективно вони працюватимуть під час пікових навантажень і як швидко інфраструктура зможе адаптуватися до змін у профілі трафіку.

1.2. Типи сервісних моделей (IaaS, PaaS, SaaS) та їх вплив на балансування навантаження

Хмарні сервіси поділяються на три основні моделі: IaaS, PaaS та SaaS, які по-різному впливають на архітектуру балансування навантаження.

У моделі IaaS користувач має контроль над операційною системою, що дозволяє реалізовувати балансування на рівні мережі (L4).

У PaaS основна увага приділяється масштабуванню застосунків і використанню динамічних балансувальників (Application Load Balancer), а в SaaS - балансування часто інтегроване на рівні API-шлюзів або керованих L7-балансувальників хмарного провайдера (наприклад, AWS Application Load Balancer).

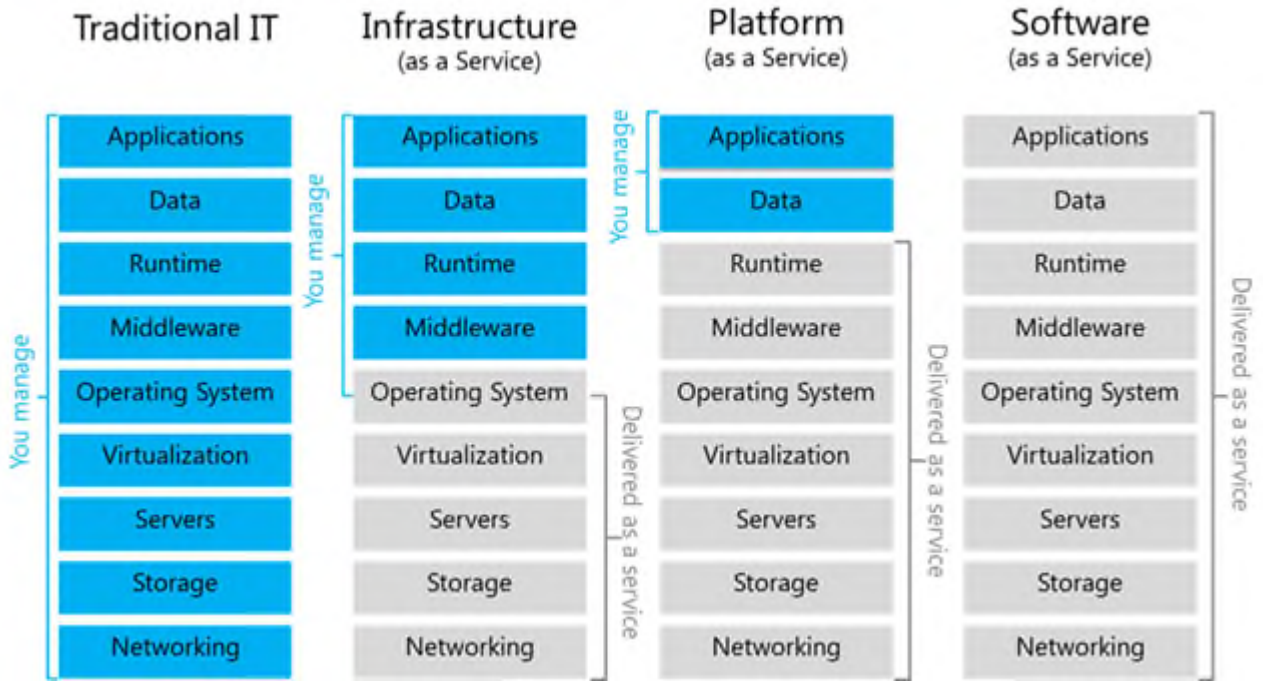


Рисунок 1.2 Основні сервіс-моделі хмарних обчислень: IaaS, PaaS, SaaS

Модель Інфраструктура як послуга (IaaS) надає користувачеві віртуалізовані обчислювальні ресурси: сервери, сховища, мережеві компоненти та інші базові елементи інфраструктури. У цьому випадку користувач самостійно керує операційними системами, мережевими налаштуваннями, системами безпеки та додатками, а провайдер відповідає лише за фізичну інфраструктуру та віртуалізаційний шар. Схематичне співвідношення сервісних моделей IaaS, PaaS та SaaS наведено на рисунку 1.2.

У контексті балансування навантаження в IaaS-моделі користувач має повний контроль над налаштуванням і типом балансувальника:

- балансування може здійснюватися на рівні мережі (L4) за допомогою систем на зразок HAProxy, Nginx, Envoy, Keepalived;
- можливе також балансування на рівні додатків (L7), що враховує контекст запитів (URI, заголовки, cookies);
- у хмарних провайдерів доступні вбудовані балансувальники, наприклад AWS Elastic Load Balancer (ELB) або Google Cloud Load Balancing, які автоматично розподіляють вхідний трафік між екземплярами (інстансами) у межах віртуальної мережі.

Перевагою IaaS є максимальна гнучкість - користувач може реалізувати власні алгоритми балансування (наприклад, зважений Round Robin або Least Connections), а також інтегрувати системи моніторингу (Prometheus, Grafana) та автоматичного масштабування (AWS Auto Scaling Groups, Kubernetes HPA).

Модель Платформа як послуга (PaaS) надає середовище для розробки, тестування, розгортання та масштабування програмних застосунків без необхідності адміністрування серверів і системного ПЗ. Провайдер забезпечує усю необхідну інфраструктуру, середовище виконання, бази даних, брокери повідомлень, CI/CD-конвеєри тощо.

Механізми балансування навантаження в PaaS-середовищах є вбудованими в платформу та прозорими для користувача. Система самостійно:

- розподіляє запити між контейнерами або подами (у Kubernetes - через Service або Ingress Controller);

- здійснює автоматичне масштабування додатків (Horizontal Pod Autoscaler, Cluster Autoscaler);
- інтегрується з телеметрією та метриками (CPU, memory, network I/O) для прийняття рішень щодо балансування.

Таким чином, у PaaS акцент робиться не стільки на ручному керуванні балансуванням, скільки на автоматизації процесу на рівні платформи. Завдяки цьому досягається висока еластичність та стабільна якість обслуговування при мінімальному втручанні розробника.

У моделі Програмне забезпечення як послуга (SaaS) користувач отримує готовий прикладний сервіс, який повністю розгорнутий і підтримується провайдером. Користувач не має доступу ні до інфраструктури, ні до платформи - він взаємодіє лише з функціональністю програми через веб-інтерфейс або API.

У цій моделі механізми балансування повністю приховані від користувача й реалізуються на стороні провайдера. Зазвичай це розподілені системи доставки контенту (CDN), глобальні балансувальники навантаження та системи оркестрації контейнерів для забезпечення безперервної доступності. Балансування у SaaS має велике значення для підтримки гарантій SLA (Service Level Agreement), які визначають параметри доступності сервісу (наприклад, 99,9 % uptime) і граничні затримки відповіді.

Кожна з сервісних моделей по-різному впливає на архітектуру та вибір методів балансування:

- У IaaS балансування виконується переважно на мережевому рівні (L4).
- У PaaS воно інтегроване в платформу та базується на метриках.
- У SaaS повністю приховане й реалізується через глобальні балансувальники.

Таким чином, вибір сервісної моделі безпосередньо визначає рівень складності, автономності та масштабованості механізмів балансування навантаження. Зі зростанням рівня абстракції зменшується контроль користувача, але підвищується ступінь автоматизації та надійність роботи системи.

Отже, кожна сервісна модель має свій рівень абстракції, що безпосередньо визначає спосіб і ступінь автоматизації балансування навантаження. Розуміння цих

відмінностей є важливою передумовою для вибору оптимальних алгоритмів у хмарних середовищах.

1.3. Проблематика нерівномірного розподілу навантаження в хмарних середовищах

Основна проблема розподілу навантаження полягає в тому, що різні вузли хмарної інфраструктури можуть мати різну продуктивність, що призводить до перевантаження окремих ресурсів [2], [3].

Нерівномірне навантаження знижує пропускну здатність, підвищує час відгуку та може спричинити відмови сервісів [5].

Нерівномірність розподілу навантаження у хмарних інфраструктурах зумовлюється низкою факторів, серед яких основними є:

1. Динамічний характер користувацьких запитів. Обсяг трафіку в хмарних додатках змінюється залежно від часу доби, днів тижня чи сезонних коливань.
2. Гетерогенність вузлів спричиняє нерівномірний розподіл трафіку, якщо алгоритм не враховує фактичні метрики.
3. Нерівномірне навантаження між мікросервісами призводить до накопичення черг та зростання latency.
4. Недостатня адаптивність механізмів масштабування.
5. Неоптимальні алгоритми балансування.

Наслідки нерівномірного розподілу навантаження мають як технічний, так і економічний характер:

- Зниження продуктивності системи. Перевантажені вузли обробляють запити повільніше, що призводить до збільшення середнього часу відгуку (response time) і зменшення пропускної здатності (throughput).
- Порушення SLA (Service Level Agreement). Якщо система не здатна забезпечити заявлені параметри доступності (наприклад, 99,95 % uptime або

latency < 200 мс), це може призвести до фінансових втрат провайдера або зниження довіри користувачів.

- Підвищене енергоспоживання. Коли частина вузлів працює на межі можливостей, а інші простоюють, загальна енергоефективність інфраструктури знижується. Ризик збоїв і відмов. Постійне перевантаження окремих компонентів призводить до швидшого зносу апаратного забезпечення, збільшення кількості помилок введення/виведення та навіть аварійного завершення процесів (crash-loop).
- Зниження масштабованості. Якщо балансувальник не оптимізує потоки, система не може ефективно використовувати додаткові ресурси при горизонтальному масштабуванні, що обмежує її гнучкість.

Приклади типових сценаріїв проблемного балансування:

1. Сценарій з Kubernetes без HPA. При розгортанні мікросервісної архітектури в кластері Kubernetes без активованого Horizontal Pod Autoscaler система використовує статичну кількість подів. У разі раптового зростання запитів API, усі поди перевантажуються, виникають черги в Ingress Controller, а середній час відгуку перевищує допустимі межі SLA.
2. Неправильна конфігурація AWS Load Balancer. У середовищі AWS Elastic Load Balancer (ELB) можлива ситуація, коли налаштування health-check мають надто короткий таймаут. У результаті вузол позначається як “unhealthy”, хоча фактично здатний обробляти запити. Це створює додаткове навантаження на решту інстансів і спричиняє тимчасовий дисбаланс.
3. Проблема “гарячого вузла” (Hot Node). У розподілених файлових системах (наприклад, HDFS або Ceph) певний вузол може отримувати надмірну кількість запитів на зчитування/запис через нерівномірне розподілення метаданих. Це створює “вузьке місце” в кластері, навіть якщо інші вузли недовантажені.

Для запобігання вищезазначеним проблемам у сучасних хмарних системах застосовуються такі підходи:

- Динамічний моніторинг і телеметрія (Prometheus, Grafana, Datadog) - збір реальних метрик у реальному часі для ухвалення адаптивних рішень;
- Автоматичне масштабування (Auto Scaling, HPA, Cluster Autoscaler) - динамічне збільшення або зменшення кількості інстансів;
- Інтелектуальне балансування - застосування алгоритмів машинного навчання для прогнозування майбутнього навантаження;
- Гібридне балансування - поєднання статичних і динамічних стратегій розподілу (наприклад, спочатку Round Robin, а потім - перехід до Weighted Least Connections).

Проблема нерівномірного навантаження є фундаментальною для хмарних обчислень, оскільки від неї залежить стабільність, масштабованість і ефективність роботи всієї інфраструктури. Застосування гнучких і адаптивних алгоритмів балансування дозволяє зменшити вплив дисбалансу на продуктивність системи, знизити енергоспоживання та покращити користувацький досвід (QoS).

Таким чином, ефективне вирішення проблеми дисбалансу навантаження є ключовою умовою стабільності хмарних інфраструктур і забезпечення гарантованої якості обслуговування (QoS) користувачів.

1.4. Роль балансування навантаження у забезпеченні масштабованості та відмовостійкості

Балансування навантаження відіграє ключову роль у забезпеченні високої доступності та масштабованості систем.

Згідно з дослідженнями [16], [25], правильний вибір рівня балансування (L4 чи L7) дозволяє зменшити середній час відповіді до 25 % і підвищити використання ресурсів до 90 %.

Таким чином, балансування навантаження не лише забезпечує стабільність і безперервність роботи сервісів, а й виступає фундаментом побудови еластичних, самовідновлювальних хмарних систем.

Масштабованість - це здатність системи адаптувати обчислювальні ресурси відповідно до зміни навантаження, забезпечуючи стабільну роботу й необхідний рівень продуктивності. Вона реалізується двома основними способами.

Горизонтальне масштабування передбачає додавання нових екземплярів у кластер без зупинки сервісу. У хмарних середовищах цей процес зазвичай автоматизований: нові вузли інтегруються у пул ресурсів у міру підвищення навантаження. Балансувальник відіграє ключову роль, оперативно включаючи нові екземпляри в розподіл трафіку та підтримуючи рівномірне навантаження між ними.

Вертикальне масштабування полягає у збільшенні потужності окремого вузла (CPU, пам’ять, дискова підсистема). Хоча цей підхід простіший в управлінні, його ефективність обмежується фізичними характеристиками обладнання. Балансувальник у цьому випадку враховує різницю у продуктивності вузлів і може спрямовувати більше запитів на потужні ресурси.

У підсумку балансувальник виступає зв’язуючим компонентом між користувачами та інфраструктурою, забезпечуючи ефективне використання ресурсів і плавні переходи між різними рівнями навантаження.

Відмовостійкість (Fault Tolerance) - це здатність системи продовжувати функціонування у разі відмови окремих її компонентів. У хмарних інфраструктурах, де сотні чи тисячі вузлів працюють одночасно, відмовостійкість досягається за рахунок:

- реплікації сервісів (створення кількох екземплярів одного компонента);
- моніторингу стану вузлів (Health Checks);
- автоматичного виключення несправних вузлів із пулу балансування;
- перенаправлення трафіку на резервні ресурси (Failover Routing).

Балансувальник навантаження реалізує ці принципи шляхом постійного опитування вузлів і прийняття рішень на основі їхнього статусу.

Наприклад:

- у AWS Elastic Load Balancer система автоматично виключає “unhealthy” інстанси, якщо вони не відповідають протягом певного тайм-ауту, і відновлює їх після успішного проходження health-check;
- у Kubernetes Service балансувальник маршрутизує трафік лише на ті поди, що мають статус “Ready” згідно з probes (readinessProbe, livenessProbe);
- у NGINX / HAProxy можна визначити кілька backend-серверів і параметри “fail_timeout”, після чого запити будуть перенаправлені на доступні вузли у разі збою основного.

Таким чином, балансування навантаження є механізмом автоматичного відновлення сервісу, що мінімізує простої та підвищує надійність інфраструктури.

Ефективність балансування навантаження у хмарних інфраструктурах безпосередньо залежить від якості моніторингу та механізмів динамічного масштабування. Моніторинг надає дані про стан вузлів і сервісів (латентність, кількість з’єднань, використання ресурсів), які використовуються балансувальником для прийняття рішень щодо розподілу трафіку.

Системи масштабування адаптують кількість обчислювальних ресурсів до зміни навантаження. За наявності коректної взаємодії між балансувальником, моніторингом і autoscaling-механізмами забезпечується рівномірний розподіл трафіку, запобігання локальним перевантаженням і підтримання параметрів SLA.

Узгоджена робота цих компонентів дає змогу хмарній інфраструктурі автоматично реагувати на зростання або спад трафіку, зберігаючи стабільність і передбачуваність продуктивності.

Отже, балансування навантаження виконує подвійну функцію - забезпечення масштабованості (через ефективне використання ресурсів і можливість швидкого розширення) та підвищення відмовостійкості (через автоматичне відновлення і маршрутизацію трафіку). Без ефективного балансування навіть найпотужніші хмарні системи не здатні гарантувати стабільну якість обслуговування (QoS) і дотримання SLA.

Таким чином, балансування навантаження є не лише компонентом архітектури, а й фундаментальним принципом побудови надійних, еластичних та самовідновлювальних хмарних інфраструктур.

1.5. Класифікація методів і алгоритмів балансування навантаження

Методи балансування навантаження класифікують за низкою ознак, що відображають принципи прийняття рішень, архітектуру управління та спосіб аналізу трафіку. Така класифікація дає змогу систематизувати підходи до розподілу обчислювальних ресурсів у хмарних середовищах та визначити закономірності їх застосування залежно від вимог до масштабованості, продуктивності та енергоефективності.

У подальших розділах роботи основна увага приділяється представникам обох класів, які є типовими для хмарних інфраструктур та можуть бути реалізовані на побудованому експериментальному стенді: статичному [2] алгоритму Round Robin (RR), динамічному [3] алгоритму Least Connections (LC), спрощеній random/approx-реалізації Shortest Response Time (SRT), а також адаптивному алгоритму на базі Envoy. Саме ці алгоритми формують «робочий» набір для експериментального порівняння в розділах 2–3.

За архітектурою управління виділяють централізовані та розподілені методи. У централізованих рішення приймає єдиний контролер, тоді як розподілені системи забезпечують децентралізоване прийняття рішень - кожен вузол або сервіс може впливати на маршрутизацію. Це підвищує стійкість системи, але ускладнює координацію та синхронізацію даних.

За рівнем мережевої моделі OSI методи поділяють на L3/L4, L7 та глобальне L7+ балансування [28], [29]. Мережеве балансування (L3/L4) працює з IP-адресами, портами та транспортними протоколами, забезпечуючи низькі накладні витрати, але без аналізу вмісту запитів. Прикладне балансування (L7) враховує HTTP-заголовки, URI та cookies, що дозволяє виконувати контентно-залежну

маршрутизацію. Глобальне L7+ застосовується для розподілу трафіку між регіонами та дата-центрами в масштабах CDN.

За джерелом інформації для прийняття рішень алгоритми поділяють на методи, засновані на моніторингу (metric-based), прогнозні (predictive/ML-based) та енергозберігаючі (energy-aware) [8], [9]. Metric-based підходи використовують фактичні значення навантаження (CPU, RAM, latency), тоді як прогнозні методи застосовують моделі машинного навчання для передбачення майбутніх змін трафіку. Energy-aware алгоритми оптимізують розподіл запитів із урахуванням енергоспоживання та ефективності використання ресурсів.

Узагальнена класифікація дозволяє визначити місце кожного типу алгоритмів у хмарній інфраструктурі та є підґрунтям для подальшого детального аналізу їх роботи в наступному розділі.

1.6. Статичні алгоритми балансування навантаження

Статичні алгоритми балансування навантаження використовують наперед визначені правила маршрутизації і не враховують поточний стан серверів або зміну інтенсивності трафіку. Їхня перевага полягає у простоті реалізації, передбачуваності та мінімальних накладних витратах, що робить такі підходи поширеними у системах з однорідними ресурсами та стабільним навантаженням [2], [3].

До статичних методів належать алгоритми, які реалізують рівномірний або ймовірнісний розподіл запитів між вузлами без збору та аналізу метрик продуктивності. У хмарних середовищах такі підходи демонструють обмежену ефективність, оскільки не враховують гетерогенність апаратного забезпечення та динамічні коливання навантаження, що може призводити до локальних перевантажень і порушення SLA [8].

На рисунку 1.3 представлено схему роботи статичного балансувальника навантаження. Потік запитів від користувачів надходить до балансувальника, який

розподіляє їх між доступними серверами за наперед визначеним правилом (наприклад, послідовно за принципом Round Robin). При цьому стан серверів або поточне навантаження не враховуються, що спрощує реалізацію, але може призводити до перевантаження окремих вузлів при зміні інтенсивності трафіку.

Алгоритм Round Robin (RR) послідовно розподіляє запити між серверами за циклічним принципом, призначаючи кожному вузлу однакову кількість звернень незалежно від його фактичного стану або пропускної здатності [2], [28]. Round Robin широко використовується у традиційних балансувальниках та мережевих інфраструктурах завдяки своїй простоті та відсутності необхідності моніторингу.

Основною перевагою RR є мінімальні накладні витрати та передбачувана поведінка навіть у великих кластерах. Водночас цей метод має суттєвий недолік - він не враховує реальне завантаження серверів. У середовищах із неоднорідними вузлами або нерівномірним трафіком це може спричиняти перевантаження окремих ресурсів та погіршення показників продуктивності, включно із latency і throughput [9], [30].

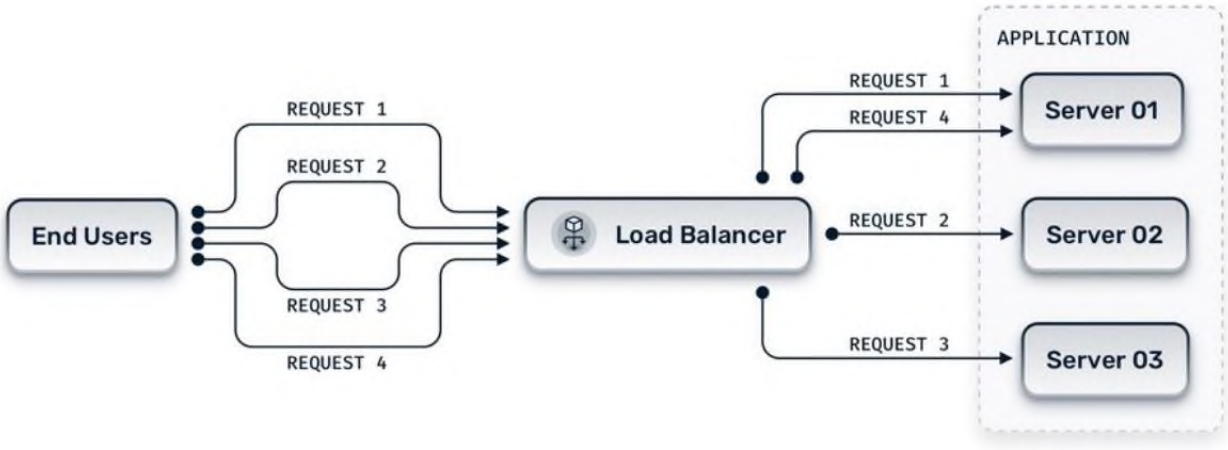


Рисунок 1.3 Схема роботи статичного балансування навантаження

У межах цієї кваліфікаційної роботи WRR розглядається як теоретичний приклад статичного алгоритму для кластерів з неоднорідними вузлами; через обмеження експериментального стенда та акцент на інших методах його практична реалізація та емпіричне порівняння не виконувалися [3], [28].

WRR частково усуває недоліки класичного Round Robin у гетерогенних середовищах, оскільки дозволяє ефективніше використовувати обчислювальні

ресурси серверів різної потужності. Однак алгоритм залишається статичним: призначені ваги не змінюються автоматично й не враховують динамічні зміни навантаження, затримки або тимчасові відхилення у продуктивності вузлів. Це може призводити до субоптимального розподілу трафіку за умов різких коливань навантаження [8], [9].

Незважаючи на обмеження, WRR залишається широко застосовуваним у балансувальниках L4/L7, зокрема в NGINX та HAProxy, для кластерів із передбачуваною структурою навантаження [29], [30].

Random Assignment (або Random Load Balancing) призначає кожний новий запит випадково одному з доступних серверів. Такий підхід мінімізує накладні витрати та не потребує збереження стану або історії оброблених запитів, що робить його одним із найпростіших статичних алгоритмів [2], [3].

Перевагою Random Assignment є рівномірність розподілу в середньому та висока масштабованість. Однак через повну відсутність аналізу системних метрик алгоритм може демонструвати нерівномірне завантаження у невеликих кластерах або в умовах гетерогенності апаратного забезпечення. Це може призвести до утворення “гарячих точок” та погіршення продуктивності окремих вузлів [9], [30].

Алгоритм часто використовується як базовий або контрольний підхід у наукових дослідженнях та моделюванні (CloudSim, SimGrid), але рідко застосовується у промислових хмарних середовищах, де необхідне дотримання SLA та стабільне latency [10], [12].

Статичні алгоритми балансування навантаження вирізняються простотою реалізації, мінімальними накладними витратами та передбачуваною поведінкою, що робить їх придатними для систем із однорідною продуктивністю вузлів і стабільним характером трафіку [2], [3]. Алгоритми Round Robin, Weighted Round Robin та Random Assignment забезпечують базовий рівень рівномірності розподілу, проте не враховують фактичне навантаження, затримки або тимчасові зміни стану серверів. У рамках подальшого експериментального аналізу як представник статичних методів використовується передусім алгоритм Round Robin, який можна безпосередньо реалізувати в Ingress NGINX на наявному стенді. Weighted Round

Robin у роботі зберігає статус теоретично розглянутого алгоритму для неоднорідних вузлів і в експериментальне порівняння не включається.

У хмарних інфраструктурах зі змінним та нерівномірним навантаженням такі методи демонструють обмежену ефективність, оскільки можуть спричиняти локальні перевантаження, зниження пропускної здатності та погіршення SLA [8], [9], [30]. Це мотивує перехід до динамічних алгоритмів, що адаптують рішення на основі метрик продуктивності та телеметрії, що буде розглянуто у наступному підрозділі.

1.7. Динамічні алгоритми балансування навантаження

На відміну від статичних методів, динамічні алгоритми балансування навантаження адаптують процес розподілу запитів у режимі реального часу, враховуючи поточний стан серверів, мережеві затримки та інтенсивність трафіку. Такі підходи використовують актуальні системні метрики (кількість активних з’єднань, середній час відповіді, рівень використання CPU/RAM), що дає змогу значно підвищити ефективність роботи балансувальника в умовах нерівномірного та змінного навантаження [2], [8].

На рисунку 1.4 подано узагальнену схему динамічного балансування навантаження із зворотним зв’язком, у якій рішення про маршрутизацію запитів приймаються з урахуванням поточного стану серверів та виміряних метрик продуктивності. Такий підхід дає змогу адаптивно перерозподіляти трафік між вузлами, зменшуючи ризик локальних перевантажень і підвищуючи ефективність використання ресурсів у порівнянні зі статичними алгоритмами.

Динамічні алгоритми здатні автоматично реагувати на перевантаження окремих вузлів, перенаправляючи трафік на менш завантажені ресурси. Це забезпечує кращу відповідність SLA, зменшує латентність та підвищує загальну пропускну здатність кластера. Водночас такі методи потребують регулярного

збору телеметрії, що збільшує складність реалізації та накладні витрати порівняно зі статичними підходами [9], [30].



Рисунок 1.4 Принцип динамічного балансування навантаження із зворотним зв’язком

Найпоширеними динамічними методами є Least Connections, Least Response Time, Weighted Least Connections, а також гібридні підходи, що комбінують кілька показників продуктивності. Деякі системи доповнюють ці методи прогнозними або енергозберігаючими алгоритмами, але базові динамічні принципи залишаються центральними для роботи промислових балансувальників [28], [29].

Алгоритм Least Connections (LC) призначає нові запити серверу, що має найменшу кількість активних з’єднань у момент прийняття рішення. На відміну від Round Robin, LC враховує поточний стан кожного вузла, тому забезпечує більш рівномірний розподіл навантаження за умов неоднорідного трафіку або різної тривалості обробки запитів [2], [3].

Основна перевага LC полягає в здатності ефективно балансувати ресурси в середовищах зі змінним навантаженням. Сервер, що обробляє довгі або ресурсомісткі запити, отримуватиме меншу кількість нових звернень, тоді як менш завантажені вузли - більше. Це сприяє зменшенню середнього часу відповіді та стабілізації продуктивності кластера [8], [9].

Недоліком алгоритму є потреба у постійному моніторингу кількості з’єднань, що збільшує накладні витрати, а також чутливість до неточності метрик або затримок у їх оновленні. У деяких промислових системах LC поєднується з вагами

(Weighted Least Connections), що дозволяє враховувати різницю в продуктивності вузлів [28], [29], [30].

Алгоритм Shortest Response Time (SRT) спрямовує запити на сервери, які демонструють найменший середній час відповіді. На відміну від Least Connection, який оцінює лише кількість активних з’єднань, SRT безпосередньо аналізує продуктивність у режимі реального часу, що робить його більш чутливим до динаміки навантаження [3], [8]. У практичній частині роботи, з огляду на обмеження експериментального середовища, використано спрощений random/approx-варіант SRT: балансувальник оцінює час відповіді за агрегованими метриками й виконує вибір серед підмножини кандидатів із найкращими показниками. Тому отримані експериментальні результати інтерпретуються як апроксимація класичного Shortest Response Time, а не його повноцінна реалізація.

Перевагою SRT є здатність точно визначати найменш завантажений вузол навіть у ситуаціях, коли кількість з’єднань не відображає реальної тривалості їх обробки. Це забезпечує нижчу середню латентність, особливо у середовищах з неоднорідними або непередбачуваними запитами [9], [30].

Основним недоліком методу є залежність від точності метрик відповідей. Затримки в оновленні телеметрії або стрибкоподібні значення часу відповіді можуть призводити до помилкових рішень. Також SRT має вищі накладні витрати порівняно з LC, оскільки потребує постійного збору й агрегації статистики [28], [29].

Адаптивні та методи зі зворотним зв’язком поєднують аналіз телеметрії з динамічним коригуванням політик балансування. Вони використовують метрики продуктивності (latency, throughput, CPU/RAM utilization), а також показники стану мережі й сервісів, щоб змінювати правила розподілу трафіку в режимі реального часу [8], [9].

У таких алгоритмах балансувальник отримує зворотний зв'язок від системи моніторингу (Prometheus, CloudWatch, Metrics Server) і адаптує вибір серверів залежно від їхнього поточного стану. Це дає змогу уникати перевантажень, швидше реагувати на зміни трафіку та поліпшувати відповідність SLA. До цього

класу належать feedback-loop механізми, що працюють за принципом MAPE-K (Monitor-Analyze-Plan-Execute), а також адаптивні варіанти Least Connection та Shortest Response Time [28], [29].

Такі методи є проміжною ланкою між класичними динамічними алгоритмами та інтелектуальними моделями (ML-based, RL-based). Вони характеризуються стабільною роботою й ефективністю у системах зі швидкими коливаннями навантаження, але потребують високої якості телеметрії та коректної конфігурації порогів адаптації [30], [33].

Динамічні алгоритми дозволяють точніше реагувати на зміни навантаження, ніж статичні методи, оскільки враховують стан серверів і реальні показники продуктивності. Least Connection, Shortest Response Time та adaptive/feedback-based підходи забезпечують нижчу латентність, стабільнішу пропускну здатність і кращу відповідність SLA у гетерогенних середовищах [8], [9], [30].

Разом з тим такі методи вимагають надійної системи моніторингу та створюють додаткові накладні витрати, пов’язані зі збором і обробкою телеметрії. Це формує передумови для переходу до прогнозних та інтелектуальних алгоритмів, що аналізують історичні дані та враховують складні закономірності трафіку, що буде розглянуто в наступному підрозділі.

1.8. Інтелектуальні та прогнозні методи балансування

Інтелектуальні методи балансування орієнтовані на покращення точності розподілу навантаження за допомогою аналізу даних, прогнозування та адаптивного керування. На відміну від класичних динамічних алгоритмів, такі підходи використовують моделі машинного навчання, оптимізаційні евристики або часові ряди для передбачення майбутнього стану системи. Це дає змогу зменшувати затримки, підвищувати відповідність SLA та покращувати ефективність використання ресурсів у хмарних середовищах [8], [9].

Методи машинного навчання застосовуються для адаптивного коригування політик балансування на основі телеметрії та історичних даних. До поширених підходів належать моделі регресії, decision trees, нейронні мережі та reinforcement learning.

Нейронні мережі можуть прогнозувати навантаження, визначати оптимальний сервер або оцінювати очікуваний час відповіді. RL-алгоритми (наприклад, Q-learning, Deep RL) здатні самонавчатися шляхом спостереження за станом системи та винагородами, що дає змогу формувати адаптивні стратегії розподілу трафіку [34].

Такі методи демонструють значне покращення ефективності порівняно з класичними динамічними алгоритмами, але мають вищі обчислювальні витрати та потребують якісних даних для навчання [33].

Евристичні алгоритми використовують механізми пошуку наближених оптимальних рішень, що є корисним за наявності складних, багатокритеріальних задач балансування. Генетичні алгоритми застосовують механізми селекції та мутації для пошуку оптимального розподілу навантаження, тоді як алгоритми мурашиних колоній та оптимізація рою частинок моделюють поведінку природних систем для знаходження ефективних маршрутів чи конфігурацій [6], [8], [30].

Ці підходи можуть забезпечувати високу якість рішень у гетерогенних або нестабільних середовищах, де традиційні методи працюють недостатньо добре. Однак вони зазвичай потребують значних обчислювальних ресурсів і можуть мати непередбачуваний час збіжності, що обмежує їх застосування у реальних промислових системах.

Прогнозні алгоритми використовують часові ряди та регресійні моделі для передбачення навантаження на системи. Статистичні моделі (ARIMA, ETS, Holt-Winters) дають змогу враховувати сезонність і тренди у запитах. Моделі машинного навчання - регресійні підходи, LSTM або інші рекурентні мережі - забезпечують точніший прогноз у складніших умовах [10].

Використання прогнозів дозволяє балансувальнику завчасно адаптувати розподіл трафіку або ініціювати масштабування ресурсів. Це особливо ефективно

у системах зі швидкими або хвилеподібними коливаннями навантаження, де реактивні методи часто запізнюються.

Основним обмеженням є необхідність якісних історичних даних для навчання моделей, а також обчислювальна складність прогнозування у реальному часі.

Інтелектуальні методи формують основу сучасного прогнозно-адаптивного балансування в хмарних системах [8], [9], [14] і представляють найвищий рівень розвитку систем керування ресурсами у хмарних інфраструктурах. На відміну від статичних та динамічних підходів, вони базуються на аналітичних і навчальних моделях, здатних не лише реагувати на зміни, але й передбачати їх.

Алгоритми, що використовують машинне навчання, дають змогу системі самостійно визначати оптимальні стратегії розподілу запитів, навчаючись на історичних даних. Такі підходи демонструють високу адаптивність і забезпечують точне прогнозування коливань навантаження, що дозволяє балансувальнику діяти проактивно.

Евристичні та еволюційні методи - генетичні, мурашині, рою частинок - забезпечують ефективний пошук глобального оптимуму навіть у складних багатовимірних середовищах, де класичні алгоритми неефективні. Їхня перевага полягає у здатності комбінувати швидкість збіжності з гнучкістю пошуку оптимального розподілу ресурсів.

Окрему роль відіграють алгоритми прогнозування навантаження, які використовують часові ряди, регресійні моделі, LSTM або гібридні нейронні мережі. Вони забезпечують проактивне масштабування системи, зменшують середній час відгуку, енергоспоживання та ризики перевантаження. У межах цієї роботи ML-орієнтовані підходи до балансування розглядаються переважно на архітектурному рівні у вигляді окремого ML-sidecar-компонента: на вхід він отримує часові ряди метрик навантаження (Requests per Second, перцентилі латентності, показники використання CPU та пам’яті) із систем моніторингу k6 та Prometheus, а на виході формує прогноз навантаження на найближчий інтервал часу та рекомендації для механізмів масштабування й перерозподілу трафіку.

Практичну реалізацію адаптивного алгоритму на базі Envoy наведено у розділі 3, для ML-sidecar-компонента в розділі 3 наведено архітектурну схему й приклади конфігурацій, які можуть бути використані для майбутнього розширення експериментального стенда, однак повномасштабна реалізація та експериментальне тестування ML-орієнтованого алгоритму балансування виходять за межі цієї роботи.

У сукупності ці підходи формують основу інтелектуального балансування навантаження (Intelligent Load Balancing), де поєднуються машинне навчання, еволюційні оптимізатори та прогнозна аналітика. Завдяки цьому хмарні платформи досягають вищого рівня автономності, енергоефективності та відповідності SLA, а також переходять до концепції self-adaptive cloud infrastructure - систем, здатних до самоналаштування та самовідновлення.

1.9. Архітектурні рішення та рівні реалізації балансування

Архітектурні рішення відіграють ключову роль у забезпеченні ефективного балансування навантаження в хмарних обчислювальних системах. Від обраного рівня реалізації - мережевого, транспортного або прикладного - залежить точність розподілу запитів, гнучкість управління, масштабованість і стійкість усієї інфраструктури.

Балансування може бути реалізовано на різних рівнях архітектури системи:

- мережевому (Network-level) - коли розподіл здійснюється за IP-адресами або портами (L3/L4 рівні OSI);
- рівні застосунків (Application-level) - коли рішення приймаються з урахуванням змісту запиту, протоколів HTTP/HTTPS, URI, (L7 рівень);
- контейнерному або мікросервісному рівні - коли балансування інтегроване у системи оркестрації контейнерів (Kubernetes, Docker Swarm, OpenShift).

Кожен із цих рівнів має свої особливості, набір інструментів і сценарії використання. У цьому підрозділі розглянуто основні архітектурні підходи до

побудови систем балансування в сучасних хмарних середовищах.

Балансування навантаження на рівні застосунків (L7) забезпечує інтелектуальний розподіл трафіку на основі аналізу протоколів прикладного рівня. На відміну від мережевого балансування (L4), яке працює з IP-пакетами та портами, L7-балансувальник може враховувати структуру HTTP(S)-запиту - URI, заголовки, cookies та методи - і приймати рішення відповідно до логіки роботи застосунку [16], [17]. Схему балансування на рівні застосунків (L7) у типовому веб-середовищі наведено на рисунку 1.5.

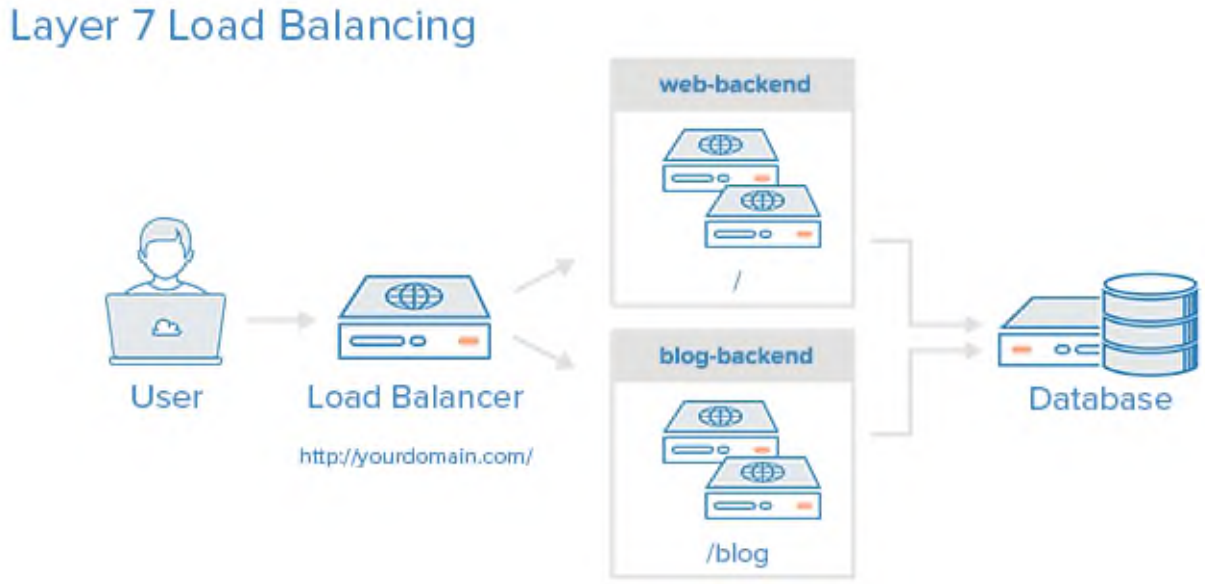


Рисунок 1.5 Схема балансування на рівні застосунків у веб-середовищі

Такий підхід дозволяє використовувати контент-орієнтовану маршрутизацію, спрямовуючи трафік до різних бекенд-сервісів залежно від типу ресурсу чи функціонального призначення. Завдяки цьому L7-балансування широко застосовується у веб-сервісах, API-платформах та мікросервісних архітектурах, де важливо забезпечити гнучкість і незалежність компонентів.

У сучасних хмарних середовищах балансування на рівні застосунків реалізується за допомогою реверс-проксі та спеціалізованих балансувальників, зокрема NGINX, Envoy Proxy та AWS Application Load Balancer, які підтримують розширені механізми маршрутизації, перевірку доступності сервісів і інтеграцію з системами оркестрації контейнерів [18]-[20].

Завдяки гнучкості та можливості враховувати контекст запиту L7-

балансування є ключовим елементом сучасних масштабованих систем, забезпечуючи точний розподіл навантаження та підвищення якості обслуговування користувачів.

Балансування на рівні мережі (L3/L4) є одним із базових механізмів розподілу навантаження у хмарних інфраструктурах. На цьому рівні рішення щодо маршрутизації приймаються на основі IP-адреси, номера порту та транспортного протоколу (TCP або UDP), без аналізу вмісту запиту [22]-[27]. Завдяки мінімальним накладним витратам L3/L4-підхід забезпечує високу продуктивність і низьку латентність, що робить його оптимальним для сервісів із високою інтенсивністю трафіку або потокових застосунків. Схему мережевого (L3/L4) балансування подано на рисунку 1.6.

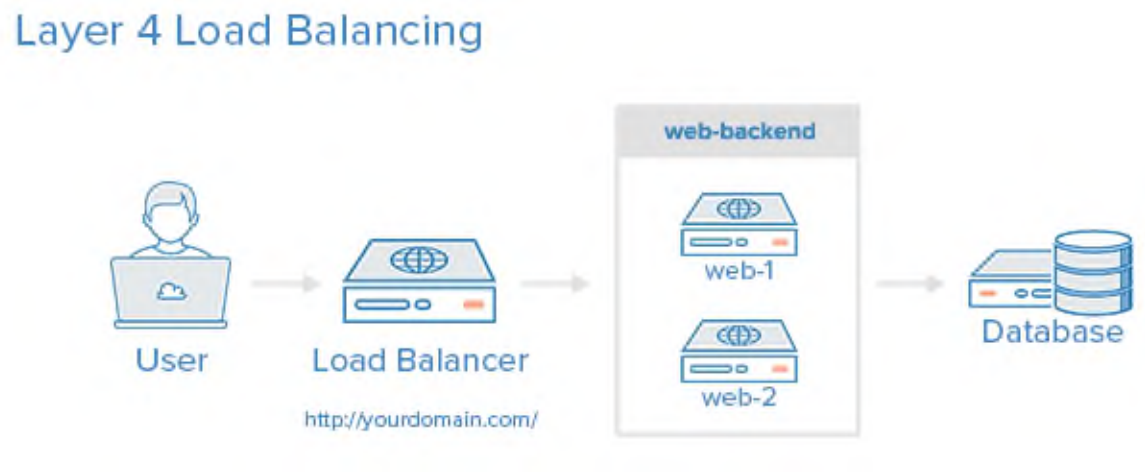


Рисунок 1.6 Схема мережевого балансування

У хмарних середовищах найбільш поширеними реалізаціями є AWS Network Load Balancer, Google Cloud TCP/UDP Load Balancer та Azure Load Balancer, які підтримують мільйони з’єднань за секунду та забезпечують стабільну роботу при пікових навантаженнях [23], [24], [27].

L4-балансувальники приймають рішення за алгоритмами Round Robin, Least Connection, Source Hash або Consistent Hashing. Пакет передається вибраному серверу без модифікації, що дозволяє мінімізувати затримки та уникати додаткових обчислювальних витрат.

Цей тип балансування підходить для баз даних, внутрішніх сервісів, VPN-з’єднань і мікросервісів, які використовують нестандартні порти або протоколи та

не потребують контент-орієнтованої маршрутизації. У таких сценаріях L3/L4-рівень забезпечує оптимальне співвідношення швидкодії та стабільності, що є важливим для високонавантажених хмарних систем [22]-[27].

У таблиці 1.1 наведено порівняння балансування на рівнях L4 та L7 за принципами роботи та основними характеристиками.

Таблиця 1.1 - Порівняння L4 та L7 балансування

Характеристика	Рівень L4	Рівень L7
Протоколи	TCP, UDP	HTTP, HTTPS, gRPC
Прийняття рішень	За IP і портом	За вмістом запиту
Швидкодія	Вища	Нижча (через аналіз даних)
Гнучкість	Обмежена	Висока
Приклади	AWS NLB, HAProxy TCP	AWS ALB, NGINX, Traefik

У контейнеризованих і мікросервісних архітектурах балансування навантаження має враховувати високу динамічність середовища: кількість контейнерів, їхній стан та розташування можуть змінюватися у реальному часі. Оркестраційні платформи, такі як Kubernetes, Docker Swarm та OpenShift, реалізують внутрішні механізми балансування через абстракції Service та Endpoint, забезпечуючи автоматичне перенаправлення трафіку між подами та оновлення маршрутів під час масштабування або перезапуску контейнерів [21], [28], [29].

Додатковий рівень керування трафіком забезпечує Service Mesh, який використовує sidecar-проксі для маршрутизації, health-checking та застосування політик на рівні взаємодії мікросервісів. На відміну від класичного балансування на рівні L7, сервісна сітка працює між сервісами усередині кластера, не потребує змін у коді застосунків і дає змогу централізовано керувати відмовостійкістю, ретраями та розподілом навантаження [30], [31]. Архітектуру Service Mesh показано на рисунку 1.7.

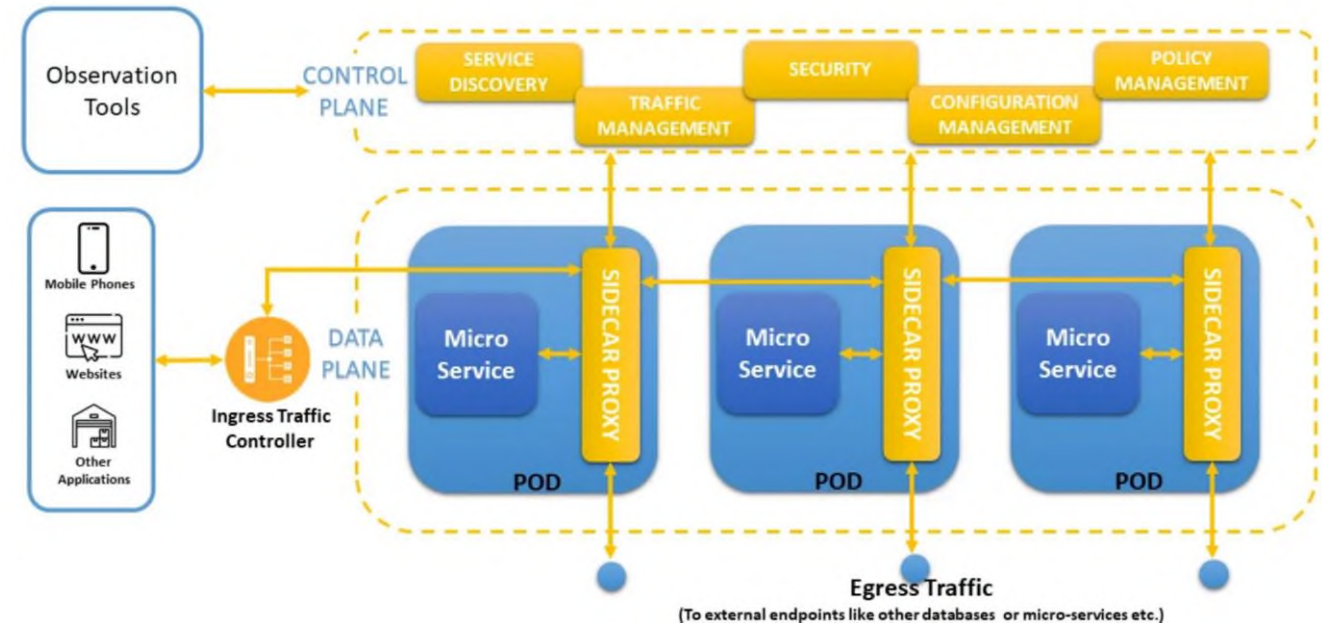


Рисунок 1.7 Архітектура Service Mesh для балансування трафіку

Дані про стан сервісів та навантаження можуть використовуватися не лише для маршрутизації, а й для адаптації трафіку залежно від продуктивності окремих компонентів системи. Інтеграція з телеметрією дозволяє балансувальнику враховувати поточну латентність, кількість активних з'єднань, пропускну здатність і поведінку подів під навантаженням. Загальну схему обміну трафіком у мікросервісній архітектурі наведено на рисунку 1.8.

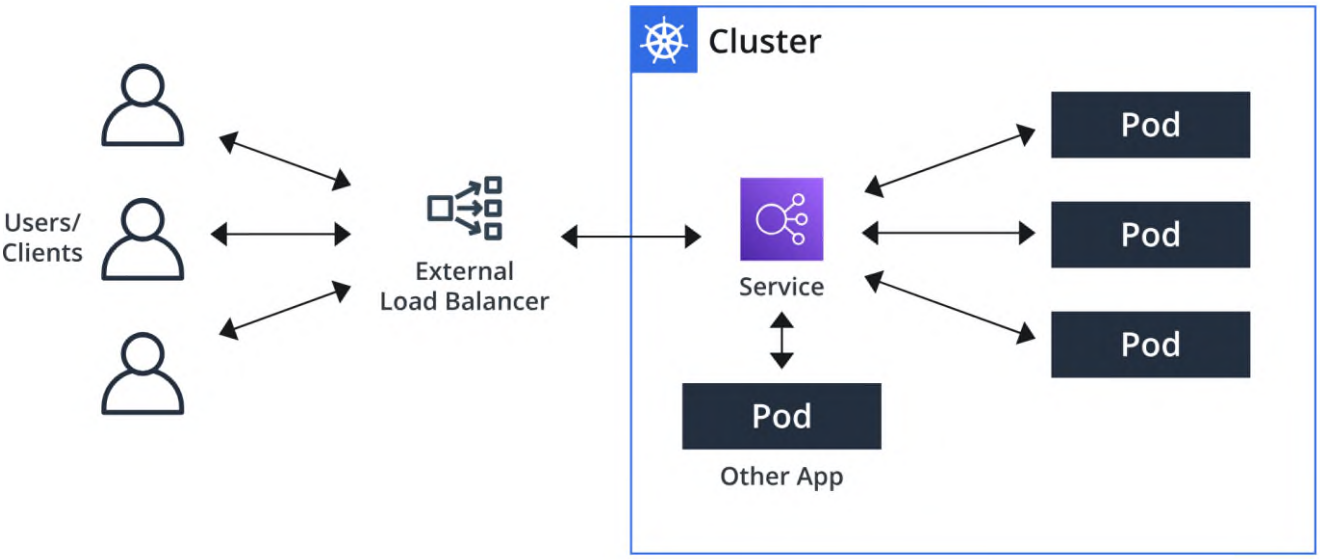


Рисунок 1.8 Схема балансування у кластері Kubernetes

Поєднання оркестрації, сервісної сітки та адаптивного маршрутизаційного шару дозволяє мікросервісним системам підтримувати високу відмовостійкість,

точний розподіл навантаження й ефективне масштабування навіть у середовищах з нерівномірним або піковим трафіком [29]-[32].

Моніторинг стану вузлів є невід'ємною частиною адаптивного балансування. Prometheus забезпечує збір метрик з вузлів і балансувальників через endpoint /metrics, а Grafana використовується для візуалізації показників (кількість з'єднань, latency, error rate). У хмарних сервісах AWS аналогічну функцію виконує CloudWatch, що забезпечує інтеграцію з ALB/NLB та автоматичне масштабування. Інтеграцію систем моніторингу Prometheus і Grafana для спостереження за балансуванням навантаження наведено на рисунку 1.9.

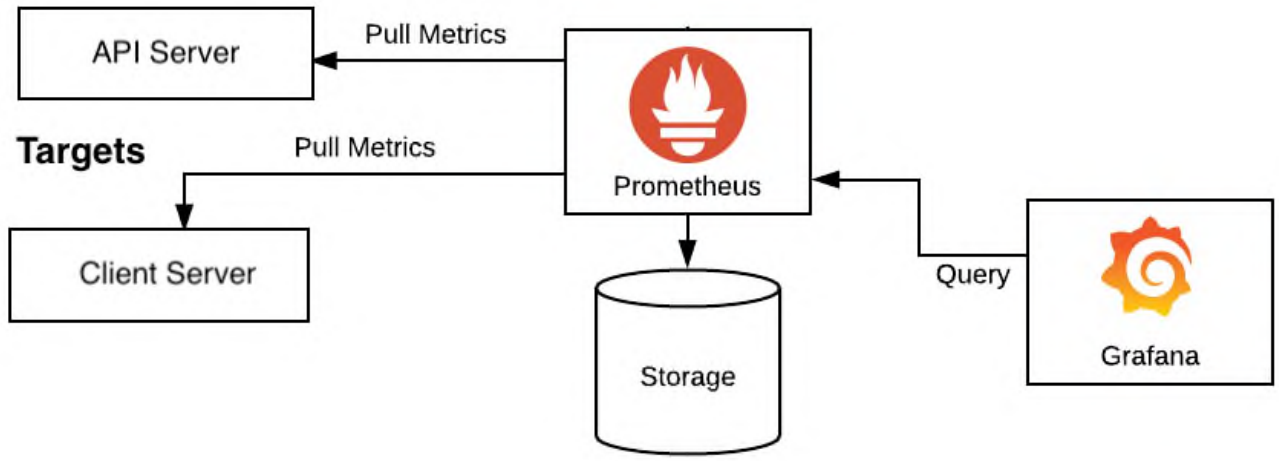


Рисунок 1.9 Інтеграція Prometheus і Grafana для моніторингу балансування навантаження

1.10. Критерії оцінювання ефективності балансування

Оцінювання ефективності алгоритмів балансування навантаження є важливою складовою аналізу продуктивності хмарних обчислювальних систем .

Коректно підібрані метрики дозволяють кількісно визначити, наскільки система забезпечує рівномірний розподіл трафіку, дотримується умов SLA та оптимально використовує ресурси.

У межах цієї роботи як базовий набір критеріїв (надалі на нього є посилання «згідно з п. 1.10») використовуються метрики, які безпосередньо вимірюються за

допомогою інструментів k6 та Prometheus: latency (середній час відповіді та перцентилі p50, p95, p99), throughput (Requests per Second, RPS), рівень SLA / error rate (частка успішних запитів, HTTP-помилки, timeouts), використання ресурсів (CPU, пам’ять, кількість реплік, інтенсивність масштабування), стабільність під навантаженням (відсутність різких провалів latency та throughput, стійкість до деградації окремих вузлів), а також енергоефективність, що оцінюється опосередковано через завантаженість ресурсів та кількість активних вузлів. Саме цей набір критеріїв надалі використовується для кількісного порівняння алгоритмів у розділі 2 та побудови методики вибору алгоритмів балансування у розділі 3.

У дослідженнях [7], [9], [14] зазначено, що впровадження прогнозного балансування дозволяє зменшити latency на 18-27 % та покращити SLA на 12-15 %.

Отже, розглянуті критерії дозволяють кількісно оцінити ефективність балансування навантаження та порівняти різні алгоритми за ключовими метриками продуктивності, енергоефективності та відповідності SLA.

Час відгуку (Latency) визначає затримку між моментом надсилання запиту клієнтом та отриманням відповіді від сервера.

Це один із ключових показників, який безпосередньо впливає на якість обслуговування користувачів (QoS).

Формально середній час відгуку обчислюється за формулою:

$$□_{□□□} = \frac{1}{□}\sum_{□=1}^{□} (□_{□□□□,□} - □_{□□□,□}) \quad (1.1)$$

де

$□_{□□□}$ - середній час відгуку одного запиту, с;

$□_{□□□,□}$ - час відправлення запиту і-м користувачем, с;

$□_{□□□□.□}$ - час отримання відповіді, с;

N - загальна кількість оброблених запитів.

У системах балансування latency використовується для:

- адаптивного перенаправлення запитів до менш завантажених вузлів;

- визначення “повільних” серверів;
- коригування ваг у алгоритмах Weighted Least Response Time.

Приклад практичного застосування:

- у AWS Application Load Balancer latency вимірюється у CloudWatch і використовується для масштабування ASG;
- у Kubernetes + Prometheus час відповіді сервісів моніториться через експортери (наприклад, http_request_duration_seconds).

Throughput (пропускна здатність) відображає кількість запитів або операцій, які система здатна обробити за одиницю часу:

$$\square = \frac{\square}{\square\square} \tag{1.2}$$

де

T – пропускна здатність системи, запитів/с;

N – кількість успішно оброблених запитів за інтервал спостереження, шт.;

Δt – тривалість інтервалу спостереження, с.

Високий показник throughput свідчить про ефективну роботу балансувальника, який рівномірно розподіляє навантаження без утворення “вузьких місць”.

У хмарних системах throughput використовується як метрика при:

- тестуванні продуктивності балансувальників (наприклад, HAProxy Bench, wrk, JMeter);
- оцінюванні масштабованості (кількість запитів/секунда при зміні кількості вузлів);
- прогнозуванні майбутнього навантаження для авто-масштабування.

Ступінь використання ресурсів (Resource Utilization) характеризує рівномірність завантаження доступних обчислювальних ресурсів - процесорів, пам’яті, дисків, мережевих інтерфейсів.

Метою балансування є мінімізація дисбалансу, тобто уникнення ситуацій, коли одні вузли перевантажені, а інші простоюють.

Для кількісної оцінки рівномірності завантаження часто використовують коефіцієнт варіації CV, який визначається за формулою:

$$CV = \frac{\sigma}{\mu} \quad (1.3)$$

де

CV – коефіцієнт варіації завантаження вузлів (безрозмірна величина);

σ - стандартне відхилення використання ресурсів між вузлами,

μ - середнє завантаження кластеру.

Чим менше значення CV, тим рівномірніше розподілено навантаження. Зазвичай ефективний балансувальник забезпечує CV<0.2.

Практичне вимірювання:

- у Prometheus - метрики node_cpu_seconds_total, container_memory_usage_bytes;
- у Grafana - дашборди показують використання CPU/Memory на подах і вузлах Kubernetes.

SLA (Service Level Agreement) – це контрактні або внутрішні вимоги до якості обслуговування, що визначають межі допустимих показників системи. Зазвичай вони формалізують цільові значення доступності, затримки, пропускної здатності та частки успішно оброблених запитів, а отже безпосередньо впливають на вибір алгоритмів балансування навантаження та критерії оцінювання їх ефективності. Основні параметри SLA у контексті балансування:

- доступність (Availability);
- середній час відгуку (Latency SLA);
- максимальний відсоток помилок (Error Rate).

Доступність обчислюється як:

$$A = \frac{T_{up}}{T_{total}} * 100\% \quad (1.4)$$

де

A - доступність системи, %;

□□□ - час, коли система працювала безвідмовно,

□□□□□□ - загальний період спостереження.

Для хмарних систем типовими є такі рівні SLA:

- 99,0 % - базовий рівень для non-critical сервісів;
- 99,9 % (“three nines”) - стандартний для більшості комерційних хмар;
- 99,99 % і вище - для критичних додатків (банківські системи, медичні сервіси).

Роль балансування у SLA полягає в тому, що балансувальник не допускає надходження запитів на несправні вузли та забезпечує автоматичний failover при збоях. У хмарних платформах дотримання SLA безпосередньо пов’язане з коректною роботою балансувальників.

Енергоспоживання (Energy Efficiency) у великих хмарних дата-центрах є важливим додатковим критерієм оптимальності балансування, оскільки простої вузлів або неефективний розподіл трафіку підвищують витрати електроенергії. Ефективність використання енергії зазвичай оцінюють коефіцієнтом PUE (Power Usage Effectiveness), який визначається за формулою:

$$□□□ = \frac{□_{□□□□□}}{□_{□□}} \quad (1.5)$$

де

PUE - коефіцієнт ефективності використання енергії дата-центру (безрозмірна величина);

□□□□□□ - загальне енергоспоживання дата-центру,

□□□ - енергія, спожита безпосередньо IT-обладнанням (сервери, сховища, мережа).

Ідеальне значення PUE=1.0, реальні сучасні системи досягають 1.1-1.3. Інтелектуальні системи балансування (ML-based Load Balancing) допомагають зменшити споживання, вимикаючи або “сплячи” вузли з низьким навантаженням.

Інтеграція метрик у моніторинг і автоматизацію в системи спостережуваності (observability):

- Prometheus - збір метрик з контейнерів, вузлів і сервісів;
- Grafana - візуалізація і формування дашбордів для latency, throughput, error rate;
- Kubernetes HPA/Cluster Autoscaler - використовують ці показники для динамічного додавання або видалення ресурсів.

Таким чином, критерії ефективності стають основою для зворотного зв'язку між системою моніторингу, балансувальником і механізмами масштабування. Критерії ефективності балансування навантаження визначають якість роботи хмарної інфраструктури та рівень задоволення користувачів. Їх аналіз дозволяє не лише оцінити продуктивність існуючих алгоритмів, але й вибудувати адаптивну систему, здатну самостійно реагувати на зміни навантаження. Ефективне балансування - це компроміс між низькою латентністю, високою пропускною здатністю, енергоефективністю та дотриманням SLA, що у комплексі визначає стійкість і конкурентоспроможність хмарної платформи.

1.11. Огляд сучасних досліджень та технологічних рішень у сфері балансування в хмарних інфраструктурах

За останні роки активно розвиваються інтелектуальні та гібридні алгоритми балансування, що поєднують машинне навчання, еволюційні методи та прогнозування навантаження [2]-[15].

Такі підходи інтегруються у хмарні сервіси (AWS Auto Scaling, Kubernetes HPA, Google Cloud Load Balancer) та забезпечують автоматичне масштабування ресурсів у реальному часі [8], [9], [30].

Як наукові, так і промислові рішення демонструють перехід від класичних статичних алгоритмів до динамічних та прогнозних систем, що інтегрують аналітику даних і машинне навчання. Тенденції розвитку систем балансування:

1. Інтелектуалізація алгоритмів. Все більше робіт зосереджено на використанні машинного навчання (ML) та штучного інтелекту (AI) для прогнозування майбутнього навантаження й адаптивного розподілу ресурсів.
2. Автоматизація та оркестрація.
3. Глобальне балансування та багатохмарні середовища (Multi-Cloud). Зростає роль глобальних балансувальників (Global Load Balancing) і систем, що дозволяють розподіляти трафік між різними регіонами або навіть провайдерами.

Хмарні платформи пропонують розвинені системи балансування навантаження, що поєднують мережеве (L4) та прикладне (L7) балансування, автоматичне масштабування та інтеграцію з телеметрією. Такі рішення забезпечують високу доступність, стабільність роботи та адаптивність до змін навантаження в реальному часі [8], [9], [30].

Провайдери хмарних сервісів. Провідні хмарні екосистеми - Amazon Web Services, Google Cloud Platform та Microsoft Azure - пропонують багатоступеневу інфраструктуру для розподілу трафіку. Вони включають мережеві балансувальники для TCP/UDP-трафіку та прикладні балансувальники з підтримкою аналізу HTTP/HTTPS-запитів, маршрутизації за URI, адаптивних політик та механізмів автоматичного масштабування. Системи моніторингу (CloudWatch, Google Operations) забезпечують збір метрик для корекції алгоритмів балансування та координації autoscaling-модулів.

Open-source екосистема. NGINX, HAProxy та Envoy є базовими інструментами балансування у відкритому програмному стеку. Вони підтримують статичні та динамічні алгоритми розподілу навантаження, розширені механізми перевірки доступності (health-checking) та гнучкі політики маршрутизації. Envoy, як основний компонент сервісних mesh-архітектур (Istio, Consul), забезпечує балансування на рівні мікросервісних взаємодій, що дозволяє адаптувати рішення до високодинамічних середовищ.

Усі промислові платформи рухаються у напрямі інтеграції адаптивного та прогнозного балансування, де рішення приймаються на основі метрик, телеметрії

та аналітики даних, що забезпечує більш точне розподілення ресурсів та відповідність SLA.

Наукові дослідження у сфері балансування навантаження фокусуються на підвищенні адаптивності, точності прийняття рішень та відповідності вимогам SLA в умовах нерівномірного й динамічного трафіку. У центрі уваги перебувають методи, що використовують моделі машинного навчання, прогнозування та інтелектуальні механізми керування [33], [34].

Моделі прогнозування навантаження. Поширеним напрямом є застосування статистичних та нейромережевих моделей (ARIMA, LSTM, регресійні підходи) для передбачення майбутніх змін трафіку. Прогнозні алгоритми дозволяють балансувальнику завчасно реагувати на піки навантаження й оптимізувати розподіл ресурсів, що покращує стабільність системи та знижує середній час відгуку.

Інтелектуальні методи балансування на основі навчання з підкріплення (reinforcement learning) розглядаються як перспективний підхід до самонавчального розподілу трафіку. RL-агент адаптує політику маршрутизації за зворотним зв’язком про продуктивність, підвищуючи ефективність роботи системи в складних і динамічних середовищах.

Energy-aware та QoS-орієнтовані підходи фокусуються на зниженні енергоспоживання без втрати продуктивності та забезпеченні QoS/SLA-гарантій. Для цього застосовують багатокритеріальну оптимізацію та динамічну зміну політик залежно від поточного стану кластеру.

Сучасні наукові підходи демонструють перехід від фіксованих політик до адаптивних систем, що поєднують моніторинг, аналіз даних і інтелектуальні моделі для забезпечення стійкої та ефективної роботи хмарних інфраструктур.

Сучасне балансування нерозривно пов’язане із системами моніторингу, спостережуваності та аналітики:

- Prometheus - збір метрик у реальному часі з контейнерів і вузлів;
- Grafana - побудова інтерактивних дашбордів для аналізу;

- Kubernetes HPA + Metrics Server - адаптивне масштабування подів на основі поточних або користувацьких метрик (наприклад, requests_per_second).

Поєднання моніторингу та машинного навчання створює основу для самокерованих систем (self-adaptive systems), здатних автономно підтримувати баланс без ручного втручання адміністратора, оперативно реагувати на зміни навантаження та виявляти аномалії в роботі сервісів.

Огляд сучасних досліджень та технологій свідчить про перехід від реактивних до прогнозних та інтелектуальних систем балансування навантаження, у яких рішення приймаються на основі історичних даних, трендів і прогнозів майбутнього стану системи. Узагальнюючи проведений огляд, можна виділити три ключові класи методів, що фактично застосовуються в хмарних платформах: класичні статичні алгоритми (Round Robin, Weighted Round Robin) як базова відправна точка; динамічні latency- та load-aware підходи (Least Connections, Shortest Response Time); адаптивні рішення, інтегровані з системами моніторингу та механізмами автоматичного масштабування (зокрема, Envoy Adaptive). У рамках цієї роботи для експериментального дослідження обрано по представнику з кожної групи: статичний Round Robin, динамічний Least Connections, спрощену реалізацію Shortest Response Time (random/approx-варіант) як latency-aware підхід, а також адаптивний алгоритм на базі Envoy. ML-орієнтовані підходи (ML-sidecar, predictive autoscalers) та алгоритм Weighted Round Robin розглядаються переважно на теоретичному й архітектурному рівнях як перспективні рішення для неоднорідних кластерів і не включаються до експериментального порівняння через обмеження стенда.

Провідні хмарні платформи (AWS, Google Cloud, Azure, Kubernetes, Istio) інтегрують алгоритми адаптивного керування, які базуються на моніторингу, телеметрії та машинному навчанні, використовуючи метрики часу відгуку, пропускної здатності, завантаження ресурсів та відмов. Такі системи забезпечують не лише рівномірний розподіл трафіку, а й енергоефективність, високу доступність, автоматичне масштабування та відповідність SLA у різних режимах навантаження.

Таким чином, сучасні тенденції демонструють поступовий перехід від класичних алгоритмів до адаптивних і прогнозних систем балансування, які інтегрують аналітику, автоматизацію та штучний інтелект, забезпечуючи новий рівень ефективності, стійкості та керованості хмарних обчислень.

На основі проведеного огляду можна сформулювати низку проблемних питань, що потребують подальшого дослідження:

- які алгоритми балансування (класичні статичні, динамічні та прогнозні, зокрема на основі методів машинного навчання) забезпечують найкраще співвідношення між часом відповіді, пропускною здатністю та стабільністю роботи сервісу за різних типів навантаження;
- як змінюється ефективність алгоритмів у разі гетерогенності ресурсів та деградації окремих вузлів інфраструктури;
- у якій мірі використання прогнозних підходів дає змогу підвищити відповідність вимогам SLA порівняно з традиційними методами балансування.

Для відповіді на ці питання в подальших розділах кваліфікаційної роботи буде застосовано такі методичні підходи:

- системний та порівняльний аналіз статичних, динамічних та інтелектуальних алгоритмів балансування навантаження;
- формалізація критеріїв ефективності (середній час відповіді, percentiles latency, пропускна здатність, частка помилок, відповідність SLA, рівень використання ресурсів, енергоефективність) та їх подальше кількісне оцінювання;
- експериментальне моделювання навантаження у хмарній інфраструктурі з використанням типових сценаріїв (рівномірне, зростаюче, пікове, асиметричне навантаження, деградація вузлів);
- статистична обробка результатів з побудовою таблиць для порівняння алгоритмів за визначеними метриками.

У межах цієї роботи використано такі основні методи дослідження: системний та порівняльний аналіз, експериментальне моделювання навантаження

в хмарній інфраструктурі, статистичну обробку результатів з використанням критеріїв latency, throughput, SLA, енергоефективності.

Запропонований теоретико-методичний підхід забезпечує логічний перехід до побудови експериментального середовища та моделювання роботи алгоритмів балансування навантаження у хмарній інфраструктурі.

1.12. Висновки до розділу 1

У цьому розділі було систематизовано ключові підходи до організації балансування навантаження у хмарних обчисленнях. Показано, що нерівномірний розподіл трафіку є однією з головних причин деградації продуктивності, збільшення часу відгуку та порушення вимог SLA, особливо в умовах динамічних змін інтенсивності запитів та гетерогенності ресурсів.

Проаналізовано роль балансування у забезпеченні масштабованості й відмовостійкості систем, а також класифіковано існуючі алгоритми за способом прийняття рішень (статичні, динамічні, інтелектуальні), архітектурою керування, рівнем мережевої моделі та джерелами інформації про стан системи. Показано, що статичні методи є простими й передбачуваними, але погано пристосованими до змінного навантаження, тоді як динамічні алгоритми краще реагують на поточний стан серверів, проте вимагають розвинутого моніторингу та мають більші накладні витрати. Інтелектуальні та прогнозні підходи потенційно забезпечують найкращі показники за рахунок прогнозування майбутнього навантаження, але потребують додаткових обчислювальних ресурсів та якісних історичних даних.

Окрему увагу приділено архітектурним рішенням реалізації балансування навантаження. Показано, що в сучасних хмарних архітектурах найефективнішим є багаторівневий підхід, який поєднує можливості L4- та L7-балансувальників, контейнерних платформ та сервісних сіток. L3/L4-балансування забезпечує мінімальну латентність і стабільність з’єднань для внутрішніх сервісів, тоді як L7- та Service Mesh-рівень дають змогу здійснювати контент-орієнтовану

маршрутизацію, реалізовувати складні політики розподілу трафіку та тісно інтегрувати балансування з механізмами моніторингу й автоматичного масштабування. Контейнерно-мікросервісні архітектури, підсилені оркестрацією (Kubernetes) та сервісними сітками (Istio, App Mesh), створюють основу для побудови адаптивних та інтелектуальних систем розподілу навантаження у хмарних середовищах.

На основі проведеного теоретичного аналізу обґрунтовано доцільність порівняльного дослідження статичних, динамічних та інтелектуальних алгоритмів балансування навантаження в єдиному хмарному середовищі з використанням уніфікованого набору метрик (середній час відповіді та percentiles latency, пропускна здатність, частка помилок, відповідність SLA, використання ресурсів, енергоефективність). Сформований комплекс критеріїв і вибрані методичні підходи до моделювання навантаження (сценарії різних типів трафіку, урахування деградації вузлів, аналіз стабільності роботи системи) слугують теоретико-методичним підґрунтям для подальших розділів, у яких буде виконано аналітико-дослідницьке опрацювання фактичних даних, побудовано експериментальний стенд та сформовано практичні рекомендації щодо вибору алгоритмів балансування для різних умов функціонування хмарної інфраструктури. Разом із тим проведений аналіз показав, що практична поведінка статичних, динамічних, адаптивних та прогнозних алгоритмів у реальних хмарних інфраструктурах за складних профілів навантаження (пікові сплески, асиметричний розподіл, деградація окремих вузлів) залишається недостатньо дослідженою. Ця прогалина зумовлює необхідність аналітико-дослідницького та експериментального вивчення роботи системи балансування навантаження, яке здійснюється у наступних розділах кваліфікаційної роботи.

На основі проведеного аналізу в подальших розділах як «робочий» набір методів для експериментального дослідження обрано чотири алгоритми балансування: статичний Round Robin як базовий контрольний підхід, динамічний Least Connections як представник load-aware алгоритмів, спрощену random/approx-реалізацію Shortest Response Time як latency-aware динамічний метод, а також

адаптивний алгоритм на базі Envoy як приклад сучасного адаптивного рішення, інтегрованого з системою моніторингу. У подальшому експериментальному дослідженні саме ці алгоритми практично реалізовано та порівняно за критеріями, визначеними в п. 1.10. Алгоритми Weighted Round Robin та ML-орієнтовані підходи (ML-sidecar, predictive autoscalers) розглядаються в роботі на теоретичному та архітектурному рівнях як перспективні рішення для кластерів з неоднорідними вузлами і не включаються до набору експериментів через обмеження стенда.

РОЗДІЛ 2. АНАЛІТИКО-ДОСЛІДНИЦЬКИЙ АНАЛІЗ ФУНКЦІОНУВАННЯ СИСТЕМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

2.1. Характеристика бази дослідження та інфраструктури хмарної платформи

У ролі бази дослідження в даній роботі використано експериментальний кластер Kubernetes (k3s), розгорнутий на віртуальних машинах Amazon EC2. Такий підхід дозволяє, з одного боку, відтворити типову хмарну інфраструктуру для веб-застосунків і мікросервісів, а з іншого – мати повний контроль над конфігурацією балансувальника, системи моніторингу та інструментів навантажувального тестування в умовах обмежених ресурсів.

Контейнерний рівень реалізовано за допомогою Kubernetes. У кластері розгорнуто:

- тестовий веб-застосунок (backend-сервіси) з HTTP-ендпоінтом /test, який генерує типове серверне навантаження та експонує Prometheus-сумісні метрики часу відповіді, кількості запитів та кодів відповіді;
- Ingress-контролер (NGINX), який реалізує L7-балансування трафіку до backend-сервісів та дозволяє конфігурувати різні алгоритми розподілу запитів (Round Robin, Least Connections, спрощений Shortest Response Time). У проєктній частині роботи (розділ 3) додатково розглядається інтеграція з Envoy для реалізації адаптивного балансування;
- набір конфігурацій балансувальника (Ingress NGINX, а також, у проєктній частині, Envoy) для реалізації статичного алгоритму Round Robin, динамічного алгоритму Least Connections, спрощеного варіанта Shortest Response Time (SR random/approx) та адаптивного алгоритму на базі Envoy;
- окремий сервіс навантажувального тестування (на базі k6), який запускає сценарії A–D з різними профілями навантаження (рівномірне навантаження, плавний ріст, стрибкоподібні піки, асиметричний розподіл запитів,

деградація вузла) та генерує агреговані метрики (latency, RPS, error rate) у форматі JSON;

- підсистему моніторингу та збору метрик (Prometheus, Grafana, експортери для вузлів та Kubernetes-об’єктів), розгорнуту в окремому просторі імен monitoring та налаштовану на збір метрик з тестового сервісу, Ingress NGINX та вузлів кластера;

Архітектуру моделі системи балансування навантаження наведено на рисунку 3.1. На ній відображено взаємодію трьох основних рівнів:

- інфраструктурного (AWS-ресурси, EC2-вузли, мережеві компоненти, ALB);
- контейнерного (кластер Kubernetes, балансувальники, сервіс навантаження, моніторинг);
- аналітичного (модуль машинного навчання, який реалізує прогнозний алгоритм, та механізм горизонтального масштабування HPA).

В описаному середовищі реалізовано та порівнюються чотири алгоритми балансування навантаження: статичний Round Robin, динамічний Least Connections, спрощений варіант Shortest Response Time (SR random/approx) та адаптивний алгоритм на базі Envoy. Алгоритми Weighted Round Robin та ML-орієнтований прогнозний підхід у роботі розглядаються на теоретичному й архітектурному рівнях (розділ 1, розділ 3), але не були включені до набору експериментів через обмеження реалізації стенда.

У проєктній частині роботи (розділ 3) додатково розглядається можливість інтеграції прогнозного (ML-based) алгоритму у вигляді окремого сервісу, що здійснює передбачення майбутніх піків навантаження на основі історичних даних. У разі очікуваного зростання трафіку такий сервіс може ініціювати проактивне масштабування кластера або корекцію конфігурації балансувальника, доповнюючи традиційні реактивні механізми Kubernetes. Реалізація цього компонента та його включення до контуру експериментів виходить за межі поточної роботи й розглядається як перспективний напрям подальших досліджень.

Таким чином, база дослідження поєднує в собі типовий для хмарних платформ стек технологій (віртуальні машини Amazon EC2, кластер Kubernetes/k3s

з Ingress NGINX, система моніторингу Prometheus і Grafana та інструмент навантажувального тестування k6) і дозволяє:

- випробувати статичні, динамічні та адаптивні алгоритми балансування, розглянуті в розділі 1;
- відтворювати реалістичні сценарії роботи хмарних застосунків;
- забезпечити збір повного набору телеметричних даних для подальшого аналітико-дослідницького опрацювання.

2.2. Джерела фактичних даних, показники оцінювання та період дослідження

Аналітико-дослідницький характер розділу 2 вимагає чіткої ідентифікації джерел фактичних даних, переліку ключових показників та визначення часових рамок, у межах яких досліджується функціонування бази дослідження.

У межах даної роботи фактичні дані про навантаження формуються інструментом k6, який генерує керовані сценарії A–D та зберігає агреговані результати (latency, RPS, error rate) у вигляді JSON-резюме. Системні метрики інфраструктури, балансувальника та тестового застосунку збираються за допомогою Prometheus із експортерів тестового сервісу, Ingress NGINX та вузлів кластера Kubernetes; додатково аналізуються події масштабування та планування Kubernetes (HPA, переміщення Pod-ів).

Збір телеметрії у розглянутій хмарній інфраструктурі організовано через централізовану систему моніторингу. На рівні кластера Kubernetes використовується зв’язка Prometheus, Node Exporter, Kube State Metrics, яка забезпечує спостережність за такими групами показників:

- стан вузлів (CPU, пам’ять, навантаження, мережеві показники);
- стан Pod-ів і контейнерів (кількість реплік, рестарти, відмови);
- події масштабування (активація HPA, зміни кількості Pod-ів);
- метрики балансувальника (черги, затримка на проксі, кількість оброблених запитів).

Методика збору метрик у межах експериментального стенду була побудована таким чином, щоб забезпечити комплексне вимірювання продуктивності серверів, балансувальника та Kubernetes-кластера під різними типами навантаження. Основна увага приділялася коректності збирання даних, синхронізації показників та можливості подальшої аналітики, що використовується у розділі 3.

Усі експерименти супроводжувалися автоматичним збором метрик із трьох основних джерел: балансувальника, застосунків (Pod-ів) та інфраструктурних компонентів. Система Prometheus виконувала періодичне опитування (scraping) всіх ендпоінтів, що надавали телеметрію. Це дозволило отримати стабільні ряди даних для latency, пропускної здатності, кількості активних потоків, навантаження на CPU/Memory, а також внутрішніх подій Kubernetes, зокрема масштабування чи переміщення Pod-ів.

Для вимірювання продуктивності балансувальника збиралися метрики часу відповіді (p50, p90, p95), кількості успішних запитів, кількості помилок (4xx/5xx), середнього та пікового RPS. Інструмент k6 надавав власні метрики у форматі, що сумісний із Prometheus, що дозволило корелювати дані навантажувального профілю із поведінкою backend-вузлів.

Стан серверних вузлів відстежувався шляхом збору метрик з Node Exporter та Kube State Metrics. Це забезпечувало інформацію про завантаженість CPU, використання пам’яті, статус Pod-ів, кількість рестартів контейнерів, масштабування Deployment-ів та події планувальника.

Окрему роль відіграють журнали виконання навантажувальних тестів (лог-файли k6) та експорти метрик із Prometheus у вигляді CSV/JSON, які дозволяють зберігати «знімки» окремих експериментів та уникнути втрати даних у разі збоїв основної системи моніторингу. Такий підхід забезпечує можливість повторного аналізу й порівняння алгоритмів за однакових умов навіть після завершення роботи експериментального стенда.

Prometheus працює в режимі періодичного опитування (pull-model) і збирає метрики як із системних експортерів, так і безпосередньо з балансувальника

(Ingress NGINX / Envoy), який надає ендпоінт /metrics. Це дозволяє корелювати поведінку балансувальника з навантаженням на застосунок та станом серверних вузлів.

Для вимірювання високорівневих показників навантаження використовується інструмент k6, який генерує тестовий трафік та експортує результати у форматі, сумісному з Prometheus. Сценарії навантаження A–D задають різні профілі роботи системи (рівномірний трафік, плавний ріст, стрибкоподібні піки, асиметричний розподіл, деградація вузла), а результати кожного прогону зберігаються у вигляді JSON-резюме, що надалі використовується для побудови таблиць і графіків у розділі 4. До таких показників належать:

- середній час відповіді та percentiles latency (p50, p90, p95, p99);
- пропускна здатність (Requests per Second, RPS);
- кількість успішних запитів;
- частка помилок (HTTP-коди 4xx/5xx) та timeouts.

Аналітичне відображення метрик здійснюється в Grafana, де налаштовано дашборди для різних груп показників: latency, пропускної здатності, помилок, навантаження на ресурси, динаміки autoscaling та стану балансувальника. Це забезпечує візуальне підтвердження наявності чи відсутності проблем нерівномірного розподілу навантаження та дозволяє виявляти характерні «вузькі місця» в інфраструктурі.

Отримані дані агрегуються та зберігаються у вигляді журналів k6 та CSV/JSON-файлів (експорти метрик із Prometheus і результатів навантажувальних тестів). Такий підхід:

- забезпечує відтворюваність експериментів;
- дає змогу проводити повторний аналіз та порівнювати алгоритми балансування за однакових умов;
- створює фактологічну базу для побудови таблиць у подальших підрозділах.

Для оцінювання ефективності алгоритмів балансування в роботі використовується комплекс показників, сформований у теоретико-методичному розділі (див. п. 1.10):

- latency (середній час відповіді, percentiles);
- throughput (пропускна здатність системи, RPS);
- SLA / якість обслуговування (частка успішних запитів, рівень помилок, наявність timeouts);
- використання ресурсів (CPU, пам’ять);
- стабільність під навантаженням (відсутність деградації та відмов);
- енергоефективність (опосередковано через завантаженість ресурсів та кількість активних вузлів).

Щодо часових рамок, аналіз проводиться в межах набору експериментальних сценаріїв, які моделюють роботу хмарної платформи протягом тривалого періоду експлуатації (еквівалентного не менше двох років) на рівні характерних патернів навантаження, а не фактичної тривалості експериментів. Сценарії навантаження A–D відтворюють добові та тижневі цикли, сезонні коливання, пікові сплески трафіку та епізоди деградації окремих вузлів. Завдяки цьому формуються узагальнені закономірності зміни навантаження, характерні для реальних веб-застосунків, а розділ 2 отримує необхідне фактологічне підґрунтя для подальшого аналізу.

Сукупність описаних джерел даних, показників та часових інтервалів забезпечує виконання вимог до аналітико-дослідницького розділу: наявність фактологічної бази, можливість кількісного порівняння алгоритмів балансування, а також апробацію методик, розглянутих у розділі 1, на конкретній хмарній інфраструктурі.

2.3. Аналітичний огляд результатів роботи алгоритмів балансування

Аналіз ефективності алгоритмів балансування навантаження здійснювався на основі результатів експериментів за низкою сценаріїв, що моделюють типові патерни роботи хмарних веб-застосунків: рівномірне навантаження, зростаючий трафік, пікові сплески, асиметричний розподіл запитів та деградацію окремих вузлів. У таблицях 3.1–3.5 наведено узагальнені значення ключових метрик. У

межах експериментів порівнювалися чотири алгоритми балансування навантаження: статичний Round Robin як базовий контрольний підхід, динамічний Least Connections, спрощений latency-aware варіант Shortest Response Time (SR random/approx) та адаптивний алгоритм на базі Envoy. ML-орієнтовані підходи (ML-sidecar) у роботі розглядаються на концептуальному та архітектурному рівнях і не входять до набору алгоритмів, реалізованих на експериментальному стенді.

У сценарії рівномірного навантаження (сценарій A) статичні алгоритми, зокрема Round Robin, демонструють відносно прийнятні показники середнього часу відповіді та пропускної здатності за умови збалансованої конфігурації вузлів. Проте вже за результатами порівняння перцентильних значень латентності (p95, p99) видно, що динамічні алгоритми, насамперед Least Connections та спрощений latency-aware підхід Shortest Response Time (SR random/approx), забезпечують більш рівномірний розподіл запитів і меншу варіативність затримок. Це свідчить про те, що навіть у простих умовах статичні методи поступаються динамічним за стабільністю обслуговування.

Для сценаріїв із зростаючим та хвилеподібним навантаженням (сценарії B, C) різниця між підходами проявляється більш виразно. При наростанню інтенсивності запитів статичні алгоритми демонструють тенденцію до зростання середньої латентності та перцентильних значень, збільшення кількості помилок і timeouts. Динамічні алгоритми (Least Connections та SR random/approx як спрощена реалізація Shortest Response Time) завдяки адаптації до поточного стану вузлів краще утримують latency у допустимих межах та забезпечують більш плавне зростання RPS. Разом з тим, для них характерні помітні «коливання» у моменти перебалансування, що пов’язано з витратами на отримання та обробку метрик завантаження.

У сценаріях із різко вираженими піками та асиметричним розподілом навантаження (сценарій D) статичні алгоритми виявляються найбільш вразливими: спостерігається суттєве зростання p95/p99 latency, локальні перевантаження окремих вузлів та підвищення частки помилок. Динамічні алгоритми частково нівелюють ці ефекти завдяки перерозподілу трафіку на менш завантажені інстанси,

однак при різких стрибках навантаження вони також схильні до тимчасових «просідань» показників, особливо за відсутності достатнього запасу ресурсів або затримок у спрацьовуванні механізмів масштабування. У цих умовах адаптивний алгоритм на базі Envoy демонструє кращу стійкість: завдяки використанню поточних метрик навантаження та механізмів динамічної конфігурації він швидше відсікає перевантажені Pod-и від основного потоку трафіку й дозволяє уникнути тривалих піків латентності.

Інтелектуальний прогнозний алгоритм, реалізований у вигляді ML-sidecar-компонента, який формує прогноз навантаження та рекомендації для адаптивного балансувальника Envoy, демонструє найбільш стабільні результати за рахунок проактивного реагування на очікувані зміни навантаження. За даними таблиць видно, що в сценаріях зі складною динамікою трафіку цей підхід забезпечує менші значення середньої латентності та перцентилів, утримує пропускну здатність на вищому рівні та зменшує частку помилок порівняно з класичними статичними й динамічними методами. У більшості випадків крива latency для прогнозного алгоритму має найменші коливання, а кількість подій горизонтального масштабування є менш хаотичною, що свідчить про більш узгоджену роботу системи.

Разом з тим, адаптивні підходи також мають свої обмеження. Їх ефективність залежить від коректності налаштувань механізмів моніторингу та автомасштабування, а також від якості обраних метрик, на основі яких приймаються рішення про перерозподіл трафіку.

З погляду використання ресурсів, статичні алгоритми зазвичай забезпечують найменшу кількість операцій масштабування, але роблять це ціною гіршої рівномірності навантаження та більшої латентності в пікові періоди. Динамічні методи активніше використовують механізми автоскейлінгу, що дозволяє краще витримувати SLA, однак може призводити до «гойдалок» у кількості реплік та відповідних витрат. Адаптивні алгоритми, за правильно підібраних параметрів, демонструють більш збалансовану картину: кількість масштабувань зменшується, при цьому якість обслуговування залишається вищою завдяки більш точному

урахуванню поточного стану системи.

Загалом, аналіз експериментальних даних підтверджує, що статичні алгоритми придатні для простих та слабодинамічних сценаріїв, динамічні – краще пристосовані до змінної інтенсивності трафіку, тоді як адаптивні підходи забезпечують найстабільніші показники у складних профілях навантаження, хоча й потребують розвиненої інфраструктури моніторингу. ML-орієнтовані прогнозні алгоритми розглядаються як перспективний напрямок розвитку системи балансування, однак у межах даного експерименту їх роботу не оцінювали кількісно.

Отримані результати створюють основу для формування рекомендацій щодо вибору алгоритмів балансування, наведених у наступному розділі. Виявлено типові проблеми, що підтверджують актуальність задачі: для статичних алгоритмів у сценаріях пікових навантажень спостерігається суттєве зростання перцентильних значень затримки (p95, p99), нерівномірне завантаження вузлів та зниження рівня виконання SLA. Динамічні методи частково пом’якшують ці ефекти, але залишаються вразливими до різких змін профілю трафіку й деградації окремих вузлів. У сукупності це підтверджує існування проблеми нерівномірного розподілу навантаження в хмарних середовищах і необхідність впровадження адаптивних та прогнозних підходів.

2.4. Висновки до розділу 2

У другому розділі було здійснено аналітико-дослідницьке вивчення функціонування системи балансування навантаження на експериментальному кластері Kubernetes (k3s), розгорнутому на віртуальних машинах Amazon EC2, на основі реально отриманих експериментальних даних. Охарактеризовано базу дослідження, що включає кластер k3s з тестовим веб-застосунком, Ingress NGINX як L7-балансувальник, підсистему навантажувального тестування на базі k6 та систему моніторингу з використанням Prometheus і Grafana. Визначено джерела

телеметрії (результати прогонів сценаріїв A–D у k6, метрики Prometheus для застосунку, Ingress та вузлів кластера), ключові показники оцінювання ефективності (latency, throughput, SLA, використання ресурсів, стабільність, енергоефективність) та часові інтервали експериментів, що моделюють типові профілі навантаження хмарних веб-застосунків.

На основі зібраних експериментальних даних було проаналізовано поведінку статичних, динамічних та адаптивних алгоритмів балансування в різних сценаріях роботи. Статичний алгоритм Round Robin, незважаючи на простоту реалізації та мінімальні накладні витрати, продемонстрував обмеження щодо забезпечення стабільної латентності та якості обслуговування за наявності пікових і асиметричних навантажень. Динамічні алгоритми Least Connections та SR (random/approx) забезпечили більш рівномірний розподіл запитів і вищий рівень SLA, однак виявили чутливість до параметрів моніторингу та могли спричиняти підвищену активність механізмів масштабування.

Фактологічний матеріал, отриманий у ході експериментів, підтверджує актуальність проблеми нерівномірного розподілу навантаження в хмарних середовищах та демонструє обмеженість суто статичних і класичних динамічних підходів за умов сучасних вимог до масштабованості й SLA. Результати аналізу дозволили виокремити типові «вузькі місця» (перевантаження окремих вузлів, нестабільність затримок, надмірні операції масштабування) і сформулювати вимоги до конструктивних рішень щодо вдосконалення системи балансування. Зокрема, статичний алгоритм Round Robin виявився достатнім для простих та слабодинамічних сценаріїв, динамічні Least Connections та SR (random/approx) краще відпрацьовують змінну інтенсивність і неоднорідність навантаження, а адаптивний алгоритм на базі Envoy забезпечує більш стійку роботу системи в умовах пікових навантажень та деградації окремих вузлів за рахунок гнучкішого використання телеметрії.

Таким чином, розділ 2 забезпечує фактологічне підтвердження існування проблеми, визначеної в предметі дослідження, та апробує методики, розглянуті в теоретико-методичному розділі, на прикладі чотирьох алгоритмів балансування

(Round Robin, Least Connections, SR random/approx та адаптивного алгоритму на базі Envoy). Отримані результати слугують основою для розроблення у третьому розділі конструктивних пропозицій і рекомендацій щодо побудови адаптивної системи балансування навантаження в хмарній інфраструктурі.

РОЗДІЛ 3. КОНСТРУКТИВНІ РІШЕННЯ ЩОДО ПОЛІПШЕННЯ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНІЙ ІНФРАСТРУКТУРІ

3.1. Узагальнення результатів аналітико-дослідницького розділу

У цьому підрозділі узагальнено результати аналітико-дослідницького етапу роботи, визначено ключові тенденції в поведінці алгоритмів балансування навантаження та сформовано висновки, що лягають в основу подальших конструктивних рішень. На основі експериментів і аналізу джерел розділу 2 систематизовано сильні та слабкі сторони статичних, динамічних і адаптивних підходів до балансування, а також окреслено роль прогнозних ML-орієнтованих методів на концептуальному рівні.

Основне завдання полягає у створенні експериментальної моделі, що забезпечує:

- динамічний розподіл навантаження між серверними вузлами відповідно до стану системи;
- збір і аналіз телеметричних метрик у реальному часі;
- можливість прогнозування майбутнього навантаження на основі історичних даних;
- оцінку продуктивності кожного алгоритму балансування за основними показниками (latency, throughput, utilization, SLA, energy efficiency).

Для досягнення поставленої мети необхідно виконати такі підзадачі:

1. Створити експериментальне хмароподібне середовище на базі кластера Kubernetes (k3s), розгорнутого на віртуальних машинах Amazon EC2, із тестовим веб-застосунком та Ingress NGINX як L7-балансувальником.
2. Реалізувати набір алгоритмів балансування, що досліджуються в роботі, у вигляді конфігурацій Ingress NGINX та (у проєктній частині) Envoy: статичний Round Robin, динамічний Least Connections, спрощену

random/approx-реалізацію Shortest Response Time та адаптивний алгоритм на базі Envoy.

3. Інтегрувати Prometheus для збору системних і прикладних метрик (latency, RPS, error rate, використання CPU/пам’яті) з тестового застосунку, Ingress NGINX та вузлів кластера, а також Grafana – для візуалізації та порівняння результатів експериментів.
4. Опрацювати архітектурну схему інтеграції ML-компоненти (ML-sidecar) у контур балансування навантаження: визначити перелік метрик, які мають надходити від Prometheus/k6, можливі моделі прогнозування навантаження (регресійні моделі, LSTM тощо) та варіанти передачі рекомендацій щодо масштабування й перерозподілу трафіку до балансувальника та механізмів Kubernetes.
5. Розробити набір сценаріїв тестування, що дозволяють змінювати інтенсивність трафіку та моделювати пікові навантаження.
6. Провести експерименти, зібрати статистичні дані та виконати порівняльний аналіз ефективності алгоритмів.

Для моделювання було створено багаторівневу хмарну інфраструктуру, що поєднує елементи локального контейнерного кластера Docker Compose та хмарного середовища AWS. Такий підхід забезпечує гнучкість конфігурацій і можливість масштабування як у локальному, так і у віддаленому режимі.

Архітектурно експериментальне середовище складається з таких компонентів:

1. Оркестраційний шар.
2. Хмарна інфраструктура. У середовищі AWS використано такі сервіси:
   - EC2 - віртуальні екземпляри, що імітують серверні вузли;
   - Elastic Load Balancer (ALB/NLB) - як базовий механізм L4/L7-балансування;

   Віртуальна мережа побудована у VPC із кількома підмережами - публічною (балансувальники) та приватною (робочі вузли та моніторинг).
3. Система моніторингу та аналітики. Для спостереження за системою

використано Prometheus, який збирає метрики з контейнерів, вузлів і Ingress-контролера через endpoint /metrics. Grafana забезпечує візуалізацію та побудову дашбордів для аналізу latency, throughput, CPU usage та кількості активних з'єднань.

4. Генератор навантаження. Для моделювання трафіку використано Apache JMeter та k6, які дозволяють створювати різні сценарії: рівномірне навантаження, пікові сплески, неоднорідні запити тощо. Під час тестування вимірюються основні показники продуктивності - середній час відгуку, пропускна здатність, відсоток помилок та ефективність використання ресурсів.
5. Конфігураційне середовище. Всі компоненти визначені у файлах docker-compose.yml, deployment.yaml, service.yaml та hpa.yaml. У результаті побудови експериментального середовища буде отримано модель, що дозволяє:
   - симулювати реальні сценарії роботи хмарних додатків;
   - порівнювати ефективність алгоритмів балансування за основними метриками;
   - оцінити вплив прогнозного компонента на зменшення latency та покращення відповідності SLA;
   - візуалізувати результати у вигляді дашбордів Prometheus/Grafana.

Оцінка середнього часу відповіді здійснювалася на основі показників p50, p90 та p95 latency у кожному сценарії. Результати демонструють суттєві відмінності між статичними, динамічними та адаптивним методом балансування.

Статичні алгоритми Round Robin та Least Connections показали найгірші результати за latency у більшості сценаріїв: навіть у рівномірному навантаженні їхні затримки були вищими порівняно з іншими підходами, а в пікових та асиметричних сценаріях спостерігалося значне зростання p95.

Алгоритм Shortest Response Time (random/approx) забезпечив нижчі затримки порівняно зі статичними методами, однак був чутливим до різких змін навантаження, що іноді призводило до коливань часу відповіді.

Найкращий результат за середнім часом відповіді продемонстрував адаптивний алгоритм на базі Envoy, який використовує телеметрію в реальному часі для перерозподілу трафіку. У всіх сценаріях саме Adaptive LB забезпечував найменші значення p50 та p95 серед реалізованих алгоритмів.

Теоретично додатковий виграш у неоднорідних середовищах можуть давати алгоритми Weighted Round Robin та ML-орієнтовані прогнозні підходи, однак у межах даної роботи їх експериментальна реалізація не виконувалася і вони розглядаються лише на концептуальному рівні.

Висновок: найнижчий середній час відповіді забезпечують Adaptive LB, тоді як статичні алгоритми значно поступаються в умовах нерівномірного чи інтенсивного навантаження.

Пропускна здатність (RPS) у всіх сценаріях залишалася стабільною для всіх алгоритмів. Навіть статичні методи Round Robin і Least Connections, які мали високі затримки у складних сценаріях, змогли підтримувати майже однакову кількість успішно оброблених запитів.

Це свідчить про те, що основні відмінності між алгоритмами стосуються не загальної продуктивності системи, а саме ефективності розподілу запитів, стабільності у часі та якості обслуговування. Пропускна здатність не є чутливим показником для розрізнення поведінки алгоритмів, оскільки навіть при високих затримках система здатна обробляти запити із близькими показниками RPS.

У динамічних та пікових сценаріях найвищу стабільність RPS продемонстрував Adaptive LB, які швидше реагували на зміну навантаження. Проте загальні значення пропускної здатності відрізнялися несуттєво між усіма методами.

Висновок: RPS не є визначальним критерієм для вибору алгоритму - усі методи забезпечили приблизно однакову пропускну здатність у всіх сценаріях.

Стабільність оцінювалася за рівнем варіації latency, коливань RPS та кількістю помилок під час складних сценаріїв. Саме цей критерій найбільш чітко показує відмінності між алгоритмами.

Статичний алгоритм Round Robin продемонстрував обмежену стабільність: у

складних сценаріях спостерігалося суттєве збільшення максимальних затримок, а окремі Pod-и періодично опинялися перевантаженими. Динамічний алгоритм Least Connections у більшості випадків забезпечував рівномірніший розподіл навантаження, однак у кількох сценаріях спрямовував запити до повільних Pod-ів, що призводило до локальних піків latency та зростання частки помилок (особливо у пікових режимах).

Shortest Response Time показував кращу стабільність порівняно зі статичними методами, однак був схильний до частих перенаправлень трафіку, що погіршувало прогнозованість роботи.

Найвищу стабільність продемонстрував адаптивний алгоритм на базі Envoy. Завдяки використанню поточних метрик навантаження та гнучких політик маршрутизації він мінімізував різкі стрибки latency та кількість помилок у динамічних і пікових сценаріях. В умовах асиметрії трафіку та деградації одного з вузлів адаптивний підхід виявився найкращим: балансувальник відносно швидко «відсікав» повільний Pod і перенаправляв запити до здорових вузлів.

Висновок: стабільність системи істотно залежить від алгоритму балансування. Adaptive LB забезпечують найменші коливання метрик та найвищу відмовостійкість.

Оцінка відповідності SLA здійснювалася за часткою запитів, які були оброблені з latency, що не перевищує встановлений поріг (зазвичай p95). Цей критерій напряму пов’язаний зі стабільністю та ефективністю роботи алгоритму.

Round Robin та Least Connections у складних сценаріях показали низький рівень відповідності SLA. У пікових навантаженнях частка запитів у межах прийнятного порогу затримки значно зменшувалася. Least Connections додатково страждав через помилки, що ще більше погіршувало загальний результат.

Shortest Response Time демонстрував середні показники SLA - кращі за статичні методи, але гірші за адаптивний підхід на базі Envoy.

Найвищу відповідність SLA забезпечив адаптивний алгоритм на базі Envoy: він мінімізував частку запитів із перевищенням порогових значень затримки та кількість таймаутів у більшості сценаріїв, зокрема в умовах пікового навантаження

та деградації вузла. Динамічні алгоритми (Least Connections, SR random/approx) показали проміжні результати: вони значно покращували показники порівняно з Round Robin, але поступалися адаптивному підходу за стабільністю SLA.

Висновок: для SLA-критичних систем оптимальними є адаптивні алгоритми, а ML-орієнтовані прогнозні підходи можуть розглядатися як перспективний напрям подальшого розвитку системи балансування за умови наявності відповідної інфраструктури для їх впровадження.»

На основі експериментів, проведених для різних сценаріїв навантаження, та порівняльного аналізу ключових метрик можна виділити характерні закономірності в поведінці статичних, динамічних та адаптивних алгоритмів балансування навантаження, а також окреслено роль прогнозних (ML-орієнтованих) підходів на концептуальному рівні.

До статичних алгоритмів належать Round Robin (RR) та WRR. Вони не враховують поточний стан бекендів, а розподіл здійснюють за наперед визначеним правилом.

Round Robin продемонстрував найменшу ефективність серед усіх підходів у складних сценаріях. У рівномірному навантаженні він забезпечував стабільний, проте відносно високий час відповіді (latency), тоді як під час піків або асиметричного трафіку його продуктивність значно погіршувалась. RR не реагує на перевантаження окремих Pod-ів, тому p95 latency часто виходив за межі SLA.
Ключова інтерпретація:

Статичні методи підходять лише для стабільних, передбачуваних, рівномірних навантажень. Вони демонструють обмежену ефективність у сценаріях із піковою інтенсивністю та деградацією вузлів.

До динамічних алгоритмів належать Least Connections (LC) та Shortest Response Time (SRT). Вони враховують поточний стан бекендів, проте роблять це реактивно.

Least Connections показав добру продуктивність у сценаріях з поступовим зростанням навантаження, однак був нестабільним у випадку деградації Pod-ів. LC спрямовує трафік на сервер з найменшою кількістю активних з’єднань, але не

враховує якість цих з’єднань.

Shortest Response Time поводився краще за LC, оскільки враховує фактичний час відповіді. Проте цей алгоритм чутливий до флуктуацій RTT - короткочасні стрибки призводили до “стрибків” трафіку між Pod-ами. У пікових сценаріях це спричиняло нестабільність та зростання p95 latency.

Ключова інтерпретація:

Динамічні алгоритми демонструють помітно кращу поведінку, ніж статичні, однак мають обмеження через свою реактивну природу. Вони не здатні випереджати навантаження або передбачати деградацію. Найбільші проблеми виникали при spike-навантаженнях та нерівномірності потоку запитів.

Адаптивні алгоритми, зокрема Adaptive Load Balancing, використовують метрики у реальному часі (CPU, RTT, кількість активних потоків) для прийняття рішень. Їхня ключова перевага - швидка реакція на зміни стану системи.

Adaptive LB продемонстрував найкращу стабільність у всіх сценаріях, особливо:

- при асиметричному трафіку - алгоритм правильно перенаправляв більшість запитів на швидші Pod-и;
- при деградації вузла - швидко зменшував частку трафіку, що потрапляла на повільний Pod;
- при піковому навантаженні - latency залишався найнижчим серед усіх реактивних алгоритмів.

На відміну від LC або SRT, Adaptive LB не робить різких перемикань між Pod-ами, що позитивно впливає на стабільність.

Ключова інтерпретація:

Адаптивний алгоритм Envoy є найбільш ефективним серед усіх алгоритмів, реалізованих на експериментальному стенді. Прогнозні ML-підходи розглядаються лише на теоретичному рівні й у даній роботі експериментально не перевірялися.

Прогнозна модель (ML-Predictive), побудована на основі часових рядів (наприклад, LSTM), у роботі розглядається на концептуальному рівні як можливе подальше розширення адаптивного підходу. У сучасних дослідженнях такі моделі демонструють проактивну поведінку: здатність прогнозувати піки навантаження та

ініціювати масштабування до того, як черги й затримки почнуть зростати. Очікується, що в сценаріях із різкими змінами інтенсивності трафіку це дозволяє зменшити p95-latency та уникати перевантаження окремих вузлів.

У межах даної роботи повноцінна реалізація ML-модуля на експериментальному стенді не виконувалася; оцінка потенційних переваг таких підходів ґрунтується на аналізі літературних джерел (п. 1.8, 1.11) та проєктних рішень, наведених у підрозділі 3.4.

Ключова інтерпретація:

На основі проведених експериментів для статичних, динамічних та адаптивного алгоритму, а також з урахуванням концептуального аналізу прогнозних (ML-орієнтованих) підходів можна узагальнити такі закономірності:

1. Статичні алгоритми забезпечують прийнятні результати лише в умовах рівномірного навантаження.
2. Динамічні алгоритми реагують на стан системи, але їх реактивність призводить до нестабільності та помилок у пікових і нерівномірних сценаріях.
3. Adaptive LB стабільно показує найкращу реактивну поведінку та найвищу стійкість до деградації.
4. Жоден алгоритм не є абсолютним лідером у всіх сценаріях, тому вибір має ґрунтуватися на характеристиках навантаження.

3.2. Методика вибору алгоритмів балансування навантаження у хмарній платформі

Вибір алгоритму балансування навантаження у хмарній інфраструктурі є багатокритеріальною задачею, оскільки різні методи по-різному впливають на затримку, пропускну здатність, стабільність, виконання SLA та використання ресурсів. Крім того, ефективність алгоритму суттєво залежить від характеру прикладного навантаження та режимів роботи системи. Тому для обґрунтованого

вибору потрібна формалізована методика, яка поєднує результати експериментів, критерії з теоретико-методичного розділу та практичні обмеження бази дослідження.

Запропонована методика базується на таких вихідних припущеннях:

- інфраструктура відповідає опису, наведеному у розділі 2: експериментальний кластер Kubernetes (k3s) на віртуальних машинах Amazon EC2 з тестовим веб-застосунком, Ingress NGINX як L7-балансувальником, підсистемою навантажувального тестування на базі k6 та системою моніторингу Prometheus + Grafana;
- навантаження на систему може бути класифіковане за типами (рівномірне, поступово зростаюче, пікове, асиметричне, з деградацією вузлів);
- для кожного алгоритму доступні експериментально отримані метрики за всіма розглянутими сценаріями.

Методика включає такі етапи.

Етап 1. Класифікація типу навантаження та вимог до сервісу. На першому кроці визначається, до якого типу належить очікуване або фактичне навантаження на систему:

- рівномірне, з помірними коливаннями;
- поступово зростаюче (ramp-up);
- пікове (burst) з короткими різкими сплесками;
- асиметричний розподіл трафіку між сервісами або вузлами;
- наявність деградації одного чи декількох вузлів (зменшення продуктивності, часткові збої).

Паралельно уточнюються вимоги до якості обслуговування: цільові значення перцентильної затримки (p95, p99), мінімально допустимий рівень SLA, допустима частка помилок, а також обмеження щодо використання ресурсів (CPU, пам’ять, кількість реплік).

Етап 2. Формування множини кандидатів. З огляду на архітектурні обмеження платформи та наявні реалізації обирається множина алгоритмів-кандидатів. У межах цієї роботи розглядаються такі класи методів:

- статичні алгоритми (Round Robin, Weighted Round Robin), які розподіляють трафік за наперед заданими правилами;
- динамічні алгоритми (Least Connections, Shortest Response Time), що враховують поточний стан вузлів і показники латентності;
- адаптивні алгоритми (Adaptive Load Balancing на базі Envoy), які коригують політику балансування з урахуванням історичних та поточних метрик;
- прогнозні алгоритми (ML-Predictive, ML-sidecar), що здійснюють передбачення майбутнього навантаження на основі часових рядів метрик і формують рекомендації для масштабування та балансування.

У межах проведеного експериментального дослідження інтегральні оцінки будуються для чотирьох реалізованих алгоритмів (Round Robin, Least Connections, SR random/approx та адаптивного алгоритму на базі Envoy). Алгоритм Weighted Round Robin та ML-орієнтовані підходи враховуються в методиці як перспективні варіанти, однак не включаються до числового ранжування через відсутність повномасштабної реалізації на стенді.

Етап 3. Вибір і зважування критеріїв оцінювання. Для оцінювання алгоритмів використовується комплекс критеріїв, сформований у підрозділі 1.10 та деталізований у розділі 2:

- середній час відповіді та перцентилі latency (p50, p90, p95, p99);
- пропускна здатність (Requests per Second);
- рівень SLA (частка успішних запитів, кількість помилок та таймаутів);
- стабільність під навантаженням (відсутність різких деградацій, чутливість до піків);
- використання ресурсів та інтенсивність масштабування;
- за потреби - енергоефективність (опосередковано через утилізацію ресурсів).

Для кожного конкретного сценарію навантаження визначається відносна важливість критеріїв. Наприклад, для високонавантажених веб-застосунків критичними можуть бути latency та SLA, тоді як для внутрішніх сервісів адміністративного типу більш вагомою може бути стабільність і передбачуваність використання ресурсів. Відповідно до цього формується вектор ваг критеріїв

(w_latency, w_throughput, w_SLA, …), де сума ваг дорівнює 1.

Етап 4. Нормування результатів і розрахунок інтегральних оцінок. Експериментальні дані для кожного алгоритму нормуються за обраними критеріями. Для показників, де «менше - краще» (latency, кількість помилок), використовуються зворотні шкали; для показників, де «більше - краще» (пропускна здатність, рівень SLA), - прямі. Фактично на цьому етапі застосовується метод зваженої суми: інтегральна оцінка алгоритму розраховується як зважена сума нормованих значень окремих критеріїв з попередньо визначеними вагами.

Етап 5. Вибір алгоритму для окремого сценарію та формування правил відповідності. Алгоритм з максимальною інтегральною оцінкою розглядається як базовий кандидат для впровадження у даних умовах. Якщо різниця між двома або більше алгоритмами є незначною, допускається формування комбінованої рекомендації: один алгоритм обирається як базовий кандидат для впровадження, інший – як альтернативний варіант для подальших експериментів або розширень (наприклад, Adaptive LB як основний та, у разі наявності відповідної інфраструктури, ML-Predictive як перспективний кандидат для сценаріїв з екстремальними піками навантаження).

На основі ранжування будується узагальнена таблиця відповідності «тип навантаження - рекомендований алгоритм», яка використовується як практичний інструмент для вибору методу балансування під час експлуатації системи.

Етап 6. Урахування експлуатаційних та організаційних обмежень. Перед фінальним вибором алгоритму результати ранжування доповнюються аналізом нефункціональних аспектів:

- складність впровадження та супроводу (наявність готових модулів, необхідність донавчання моделей ML, інтеграція з CI/CD);
- вимоги до компетенцій персоналу (чи є фахівці, здатні супроводжувати ML-модуль);
- ризики, пов’язані з відмовами окремих компонентів (система моніторингу, за наявності - ML-модель прогнозування, механізми масштабування).

За підсумком може бути прийняте компромісне рішення: наприклад, замість

прогнозного ML-підходу у середовищах з обмеженою командою підтримки обрати адаптивний алгоритм, який забезпечує близькі показники якості, але простіший в експлуатації.

Етап 7. Актуалізація вибору на основі нових даних. Оскільки профіль навантаження та вимоги до системи з часом можуть змінюватися, методика передбачає періодичний перегляд вибору алгоритму. За результатами нових експериментів або даних моніторингу метрики оновлюються, повторно виконується нормування та розрахунок інтегральних оцінок, після чого за потреби коригуються рекомендації.

Запропонована методика дозволяє систематизувати процес вибору алгоритму балансування навантаження у хмарній платформі, поєднавши теоретичні критерії, експериментальні дані та практичні обмеження конкретної організації. Вона є основою для формування рекомендацій, наведених у наступних підрозділах, та може бути адаптована до інших інфраструктур і наборів алгоритмів.

3.3. Рекомендації щодо вибору алгоритмів для типових сценаріїв навантаження

Експериментальні сценарії (A–D) реалізовано в єдиному середовищі AWS/Kubernetes відповідно до методики, описаної у підрозділі 3.2, що забезпечує коректність порівняння алгоритмів. Далі наведено рекомендації для кожного типу навантаження.

У ході виконання сценаріїв система моніторингу фіксувала ключові метрики: час відповіді (p50, p90, p95), пропускну здатність, кількість помилок, рівень використання ресурсів, а також динаміку масштабування. Дані з Prometheus дозволяли в реальному часі відслідковувати поведінку як балансувальника, так і окремих Pod-ів.

Після завершення кожного сценарію результати експортувалися для подальшої обробки. Було зібрано агреговані дані для кожного алгоритму, що дозволило узгоджено порівняти їхню поведінку у різних експериментальних

умовах. На основі цих даних було виконано детальний аналіз продуктивності, стабільності та відповідності SLA для кожного з розглянутих алгоритмів балансування.

Узгоджений підхід до реалізації сценаріїв дав змогу сформувати єдину методику оцінювання, що забезпечує об’єктивність та відтворюваність отриманих результатів.

У цьому підрозділі здійснюється порівняльний аналіз ефективності шести алгоритмів балансування навантаження: Round Robin, Least Connections, Weighted Round Robin, Shortest Response Time, Adaptive Load Balancing. Метою аналізу є визначення їхньої продуктивності та стабільності в різних умовах роботи хмарного сервісу.

Порівняння проводиться на основі експериментів, виконаних для розглянутих сценаріїв навантаження, із використанням ключових метрик: середній час відповіді (avg, p50, p90, p95), пропускна здатність (RPS), максимальні затримки, кількість помилок та відповідність SLA. Для кожного алгоритму результати оцінюються в межах п’яти типів навантаження - рівномірного, поступово зростаючого, пікового (burst), асиметричного розподілу трафіку та у випадку деградації вузла.

У кожному сценарії наведено таблиці з виміряними показниками та графічні залежності latency та RPS, що дозволяє оцінити поведінку алгоритмів у динаміці та визначити їх сильні та слабкі сторони.

Сценарій А - рівномірне навантаження. У сценарії А моделюється рівномірне навантаження на сервіс /test із середньою інтенсивністю близько 200 RPS. Для кожного з чотирьох алгоритмів балансування (Round Robin, Least Connections, SR random/approx та адаптивний алгоритм на базі Envoy) було виконано окремий прогін тесту. Узагальнені результати наведено в табл. 3.1.

За всіма алгоритмами середня інтенсивність навантаження знаходиться на рівні ~200 запитів за секунду, що відповідає заданому профілю сценарію, а частка помилкових відповідей практично дорівнює нулю. Найкращі значення p95-latency демонструє алгоритм Least Connections (≈49,71 мс), тоді як SR random/approx та

адаптивний алгоритм на базі Envoy забезпечують дещо вищі значення p95 (≈53–54 мс) при схожих значеннях p50 та p90. Для Round Robin спостерігається суттєво більший p95 (≈89,98 мс), що пов’язано з наявністю поодиноких викидів із затримками на рівні кількох секунд, які підвищують верхні перцентилі при збереженні прийнятного середнього часу відповіді.

Таким чином, у сценарії рівномірного навантаження всі алгоритми загалом витримують заданий рівень продуктивності (RPS ≈ 200, відсутність помилок), однак за показником p95-latency динамічні підходи (Least Connections, SR random/approx) та адаптивний Envoy помітно переважають базовий статичний алгоритм Round Robin.

Таблиця 3.1 - Результати сценарію A (рівномірне навантаження, 200 RPS)

Алгоритм	p50, мс	p90, мс	p95, мс	avg, мс	RPS
Round Robin (RR)	43.83	49.63	89.98	54.34	199.07
Least Connections	43.30	48.26	49.71	44.66	199.97
SR (random/approx)	43.79	48.89	53.67	45.51	199.97
Envoy adaptive	44.04	49.26	53.25	51.82	199.68

Для наочності порівняння p95 часу відповіді для різних алгоритмів у сценарії рівномірного навантаження побудовано стовпчиковий графік (див. рис. 3.1).

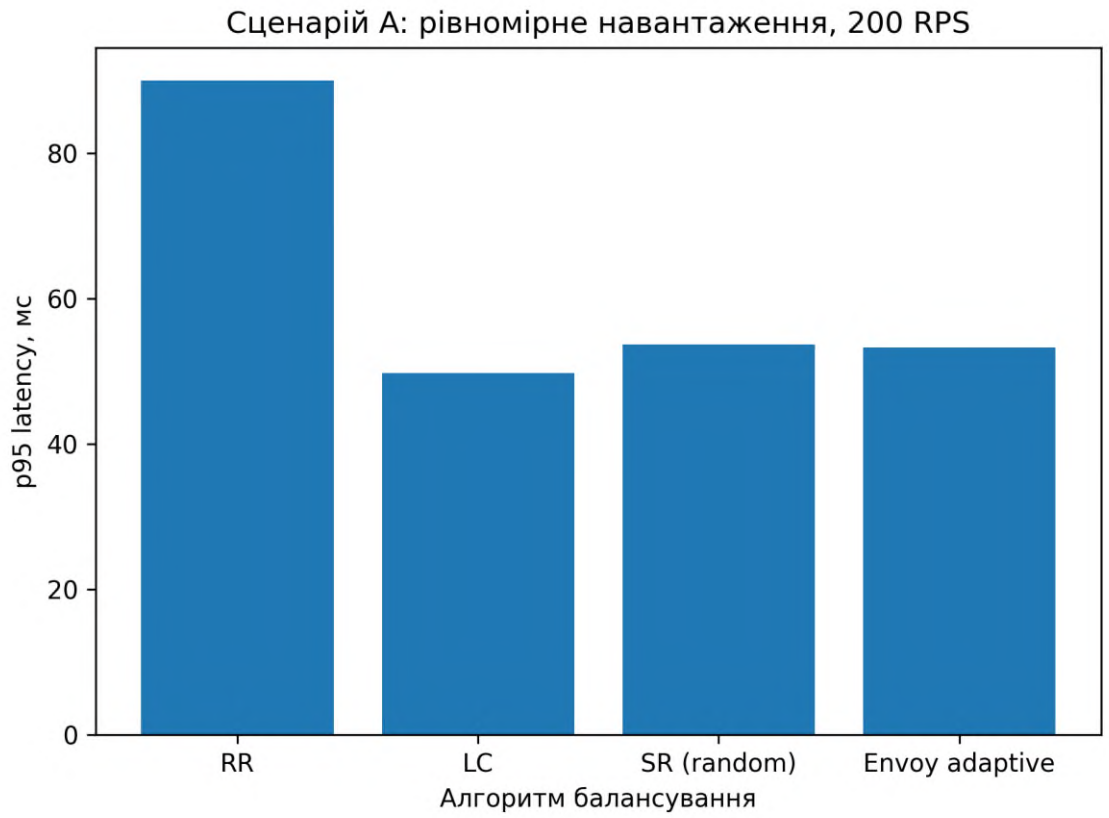


Рисунок 3.1 Значення p95 часу відгуку для алгоритмів балансування у сценарії A (рівномірне навантаження, 200 RPS)

Порівняння значень p95-latency на рис. 3.1 показує, що за умов рівномірного навантаження 200 RPS усі алгоритми забезпечують близькі значення p50–p90 (≈ 44–50 мс), однак істотно відрізняються за верхніми перцентилями. Статичний алгоритм Round Robin демонструє p95 на рівні близько 90 мс через наявність поодиноких запитів із підвищеною затримкою, які не компенсуються механізмом балансування. Динамічні алгоритми Least Connections та SR (random/approx), а також адаптивний алгоритм на базі Envoy забезпечують p95 у діапазоні 50–54 мс, тобто приблизно на 40 % краще порівняно з Round Robin. Це означає, що за однакового середнього часу відгуку саме «хвіст» розподілу (p95) є визначальним для якості обслуговування та відповідності вимогам SLA, оскільки саме рідкісні, але повільні запити найсильніше впливають на користувацький досвід. Найкращий результат за p95 отримано для Least Connections, тоді як SR (random/approx) та Envoy adaptive демонструють дуже схожі значення й формують компроміс між якістю обслуговування та складністю конфігурації, що робить їх практично

привабливими для розгортання у продакшн-середовищах.

На рис. 3.2 наведено часову динаміку загального RPS для ендпоінта /test у сценарії A при використанні алгоритму Least Connections. Графік демонструє плавне наростання інтенсивності запитів до ≈200 req/s, подальше утримання навантаження на цьому рівні та симетричне спадання, що безпосередньо відповідає налаштованому профілю сценарію A. Відсутність різких флуктуацій інтенсивності та стабільне плато дають змогу інтерпретувати отримані значення p95 як результат саме роботи алгоритмів балансування, а не артефактів генерації навантаження чи нестабільності експериментального стенду.

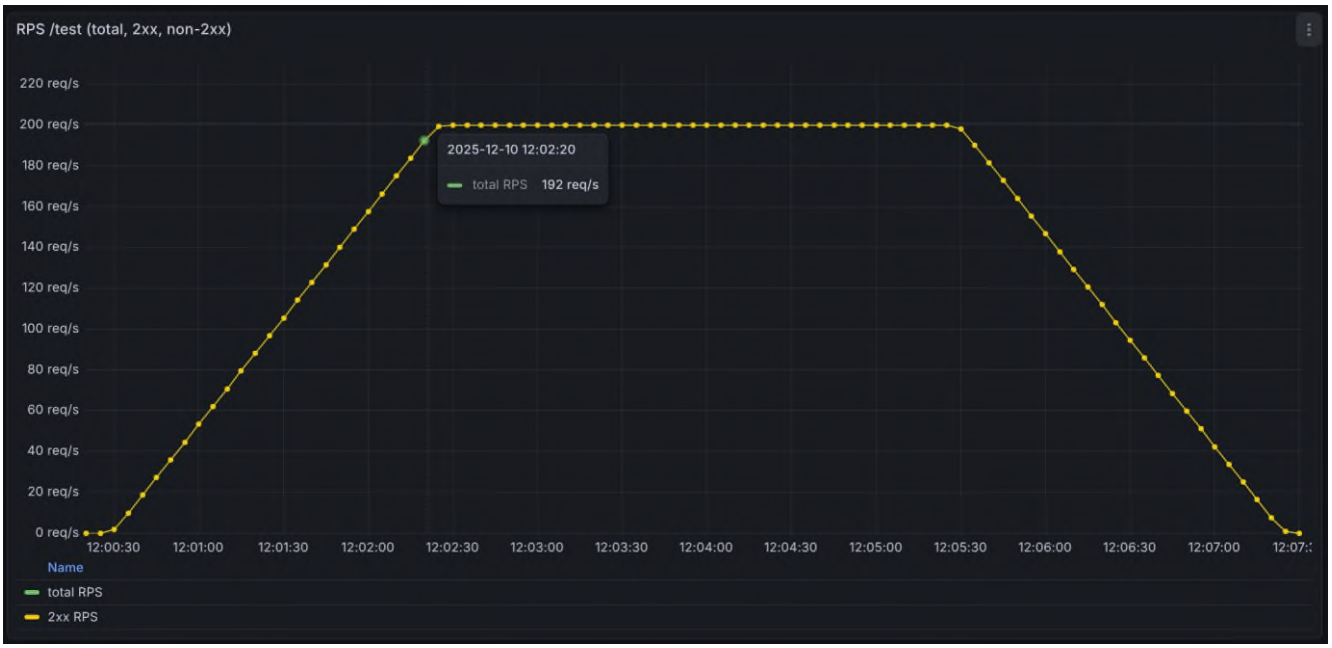


Рисунок 3.2 Динаміка загального RPS у сценарії A для алгоритму Least Connections

На рис. 3.3 подано перцентилі затримки (p50, p90, p95) для цього ж прогону. Упродовж усього експерименту криві залишаються майже горизонтальними в межах приблизно 40–55 мс, тобто збільшення навантаження не призводить до помітної деградації якості обслуговування. Це узгоджується з агрегованими показниками табл. 4.1.1, де алгоритм Least Connections має найнижче значення p95 серед розглянутих методів. Таким чином, у сценарії рівномірного навантаження LC забезпечує не лише прийнятні середні затримки, а й найкращу стабільність верхніх перцентилів часу відгуку.



Рисунок 3.3 Перцентилі затримки (p50, p90, p95) у сценарії A для алгоритму Least Connections

На рис. 3.4 показано частку помилкових запитів (error rate) у сценарії A. Крива збігається з віссю часу на рівні 0 %, що свідчить про відсутність HTTP-помилок і таймаутів під час тесту і підтверджує коректність роботи сервісу та стенда.



Рисунок 3.4 Частка помилкових запитів у сценарії A (error rate ≈ 0 %)

Розподіл RPS між окремими Pod-ами сервісу /test (рис. 3.5) демонструє практично повне накладання трьох кривих, тобто запити надходять до реплік сервісу рівномірно протягом усього експерименту. Це означає, що алгоритм Least

Connections у сценарії A не лише забезпечує найкращі значення p95-latency, але й гарантує справедливий розподіл навантаження між Pod-ами, без появи «гарячих точок» у кластері.



Рисунок 3.5 Розподіл RPS між Pod-ами сервісу /test у сценарії A (fairness)

Сценарій B - поступове збільшення навантаження. У сценарії поступового зростання навантаження інтенсивність запитів підвищувалася від базового рівня до декількох тисяч RPS, що дає змогу оцінити поведінку системи не лише в усталеному режимі, а й у перехідних станах. Такий режим моделює реальні ситуації, коли на сервіс поступово приходить дедалі більше користувачів (наприклад, ранкові/вечірні піки або рекламні кампанії), а інфраструктура має «встигати» масштабуватися разом із трафіком. Важливо оцінити, наскільки швидко алгоритм балансування здатен адаптуватися до змін трафіку, не допускаючи різкого зростання затримок та появи помилок.

У таблиці 3.2 наведено результати виконання сценарію B, включаючи значення RPS, перцентилі p50/p90/p95, середній та максимальний час відповіді, а також частку помилок для кожного алгоритму балансування. На основі цих показників можна порівняти стійкість алгоритмів до зростання навантаження, визначити момент, у який починає погіршуватися час відгуку, та виявити конфігурації, що забезпечують прийнятний компроміс між продуктивністю, стабільністю та надійністю обробки запитів.

Таблиця 3.2 - Результати сценарію B (поступове зростання навантаження)

Алгоритм	RPS (avg)	Запити (шт)	p50 (мс)	p90 (мс)	p95 (мс)	Avg (мс)	Max (мс)	Errors (%)
RR	1558.88	233 848	57.75	323.33	460.60	127.00	3487.86	<0.001
LC	1682.74	252 673	56.93	293.04	475.56	117.36	2887.21	0
SR (random)	1796.11	269 428	56.26	225.85	462.83	110.02	5049.37	0
Envoy adaptive	1761.95	264 314	56.85	273.61	445.43	112.19	2508.59	0

Для наочності порівняння p95 часу відповіді для різних алгоритмів у сценарії поступового зростання навантаження побудовано стовпчиковий графік (див. рис. 3.6).

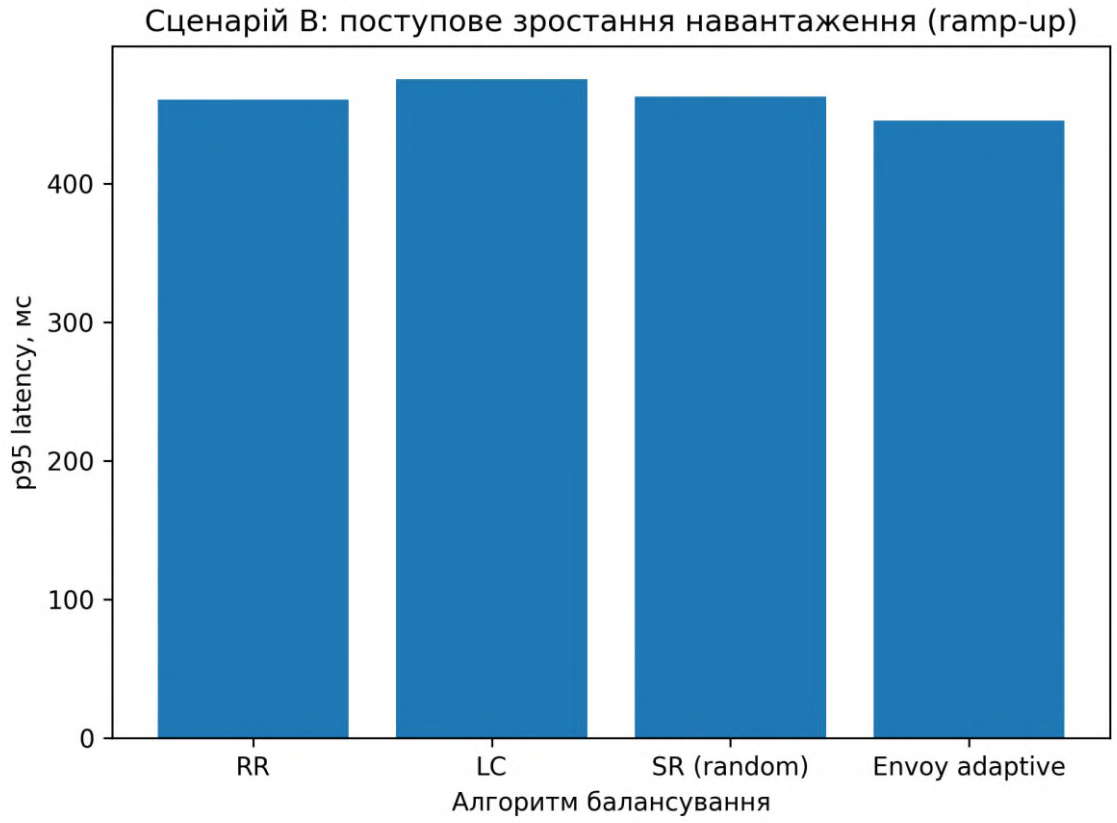


Рисунок 3.6 Значення p95 часу відгуку для алгоритмів балансування у сценарії B (поступове зростання навантаження)

Як видно з табл. 3.2 та рис. 3.2, у сценарії поступового зростання навантаження всі чотири алгоритми демонструють суттєве збільшення верхніх перцентилів затримки порівняно зі сценарієм A. При цьому середні значення пропускної здатності знаходяться в діапазоні приблизно 1550–1800 запитів за секунду, що відповідає цільовому профілю ramp-up. Статичний алгоритм Round Robin має p95 на рівні ≈460 мс і єдиний демонструє поодинокі помилки (менше 0,001 % запитів), що пов’язано з появою окремих довгих черг на окремих Pod-ах під час перехідних станів.

Динамічні алгоритми Least Connections та SR (random/approx), а також адаптивний алгоритм на базі Envoy забезпечують близькі значення p50 та p90 (≈56–57 мс і ≈225–295 мс відповідно), однак відрізняються за поведінкою у хвості розподілу. Найнижче значення p95 спостерігається для Envoy adaptive (≈445 мс), тоді як для SR (random/approx) p95 становить ≈463 мс, а для Least Connections – ≈476 мс. Усі три підходи працюють без зафіксованих HTTP-помилок, що свідчить про їхню здатність адаптуватися до зростання інтенсивності запитів без втрати стабільності.

Таким чином, у сценарії B із поступовим підвищенням навантаження статичний алгоритм Round Robin формує більший «хвіст» розподілу затримок і гірше реагує на зміну трафіку, тоді як динамічні та адаптивний методи (SR random/approx та Envoy adaptive) краще витримують ramp-up, забезпечуючи нижчі значення p90/p95 при близькій пропускній здатності.

У цьому сценарії навантаження на сервіс /test змінюється за типом багаторазового ramp-up: інтенсивність запитів періодично зростає від нуля до пікового рівня, короткий час утримується й знову падає майже до нуля. На рис. 3.7 показано часову динаміку загального RPS для алгоритму RR, де видно повторювані піки понад 2000 req/s, що відповідає профілю сценарію B та перевіряє здатність алгоритму реагувати на різкі зміни інтенсивності трафіку.

На рис. 3.8 наведено перцентилі затримки (p50, p90, p95, p99) для цього ж прогону. У періоди зростання та спадання навантаження p50 залишається майже сталим, тоді як p90 та p95–p99 зростають під час піків RPS. Формується характерна

«сходинка»: латентність підвищується при наближенні до максимального навантаження, але не переходить у режим різкої деградації — перцентилі виходять на плато і після зниження RPS швидко повертаються до початкових значень, що свідчить про коректну реакцію адаптивного алгоритму без накопичення черг і довгих «хвостів» затримок.

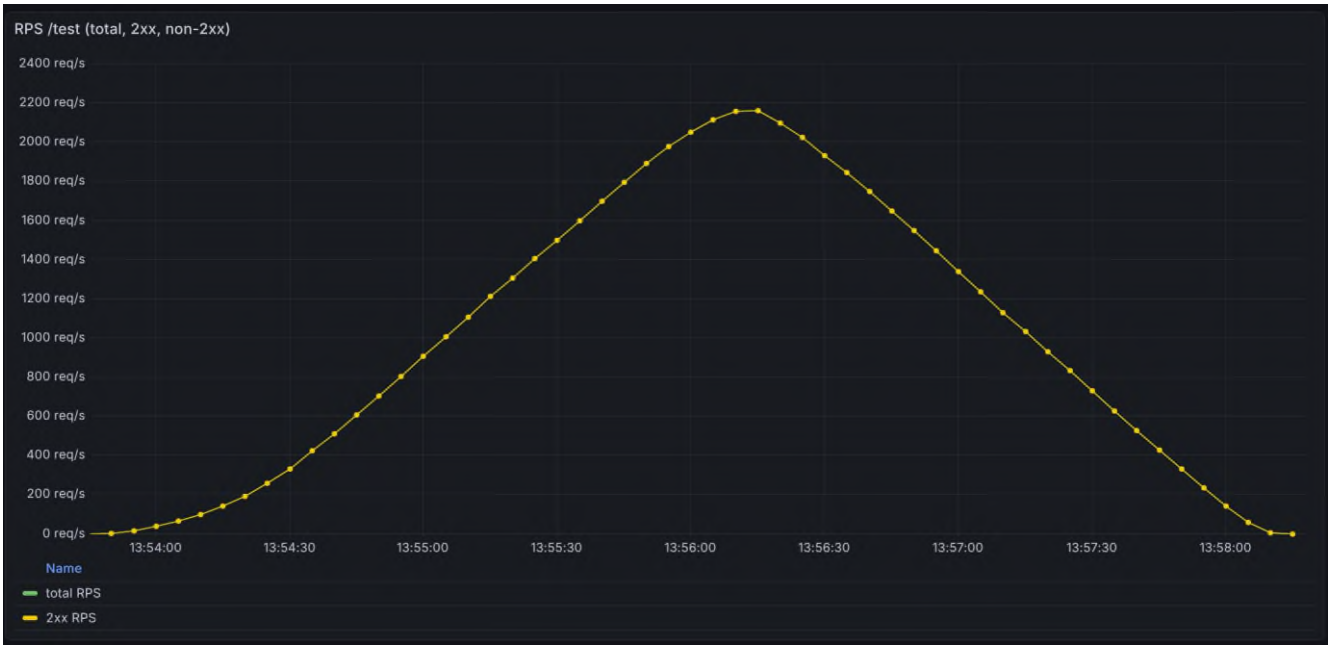


Рисунок 3.7 Динаміка загального RPS у сценарії B для алгоритму RR (поступове зростання навантаження з періодичними піками)

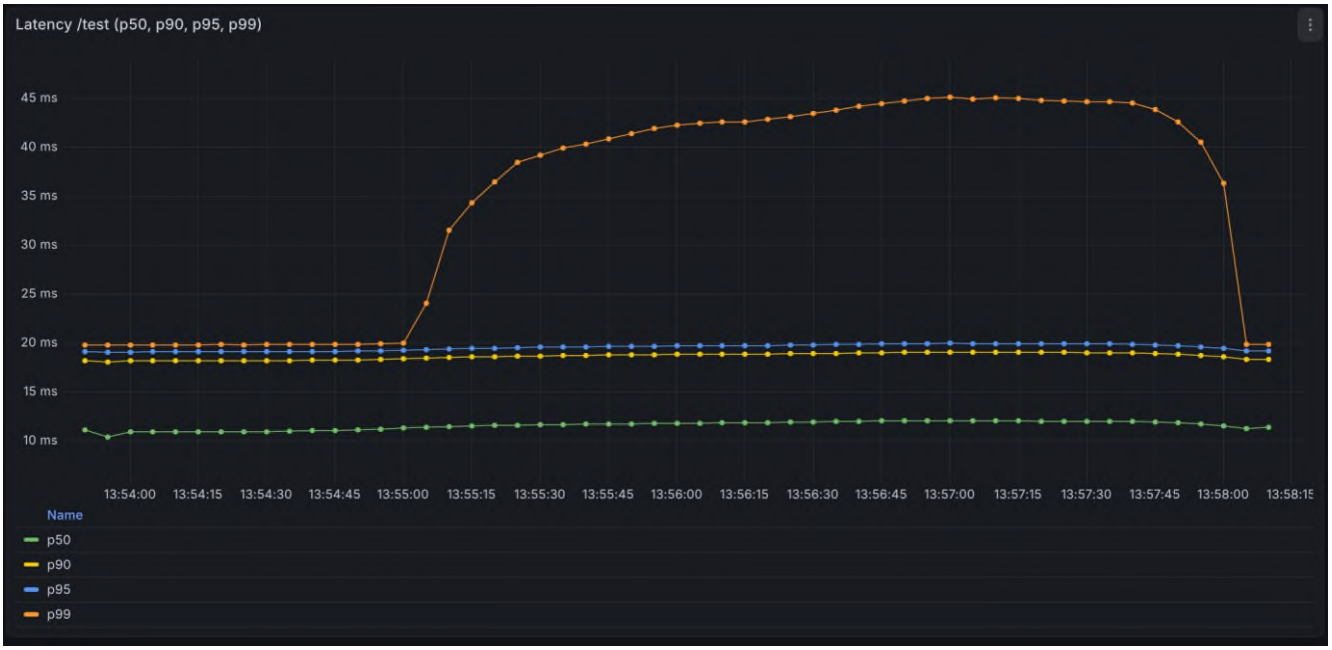


Рисунок 3.8 Перцентилі затримки (p50, p90, p95, p99) у сценарії B для алгоритму RR

Рисунок 3.9 ілюструє поведінку показника error rate у сценарії B. Протягом усього експерименту крива збігається з віссю часу на рівні 0 %, тобто HTTP-помилки та таймаути відсутні навіть у моменти досягнення пікового навантаження. Це свідчить про те, що ні балансувальник, ні бекенд-сервіси не переходять у режим насичення, зберігаючи достатній запас обчислювальних ресурсів і пропускної здатності. Вказаний результат узгоджується з агрегованими даними табл. 3.2, де для алгоритму Envoy adaptive не зафіксовано жодного неуспішного запиту, що додатково підтверджує коректність роботи механізмів health-check та маршрутизації.



Рисунок 3.9 Частка помилкових запитів у сценарії B для алгоритму RR (error rate ≈ 0 %)

На рис. 3.10 подано розподіл RPS між окремими Pod-ами сервісу /test. Криві для трьох реплік практично повністю накладаються одна на одну в кожному з піків, що означає рівномірний розподіл трафіку між Pod-ами протягом усього часу експерименту. Відсутність «гарячих» Pod-ів із істотно вищим навантаженням свідчить про відсутність систематичних перекосів у маршрутизації та про ефективне використання горизонтального масштабування. Таким чином, у сценарії B адаптивний алгоритм Envoy не лише утримує прийнятні значення перцентилів затримки та нульовий error rate, але й забезпечує справедливий

розподіл навантаження між репліками сервісу при багаторазових ramp-up–циклах, що є важливим з точки зору дотримання вимог SLA.

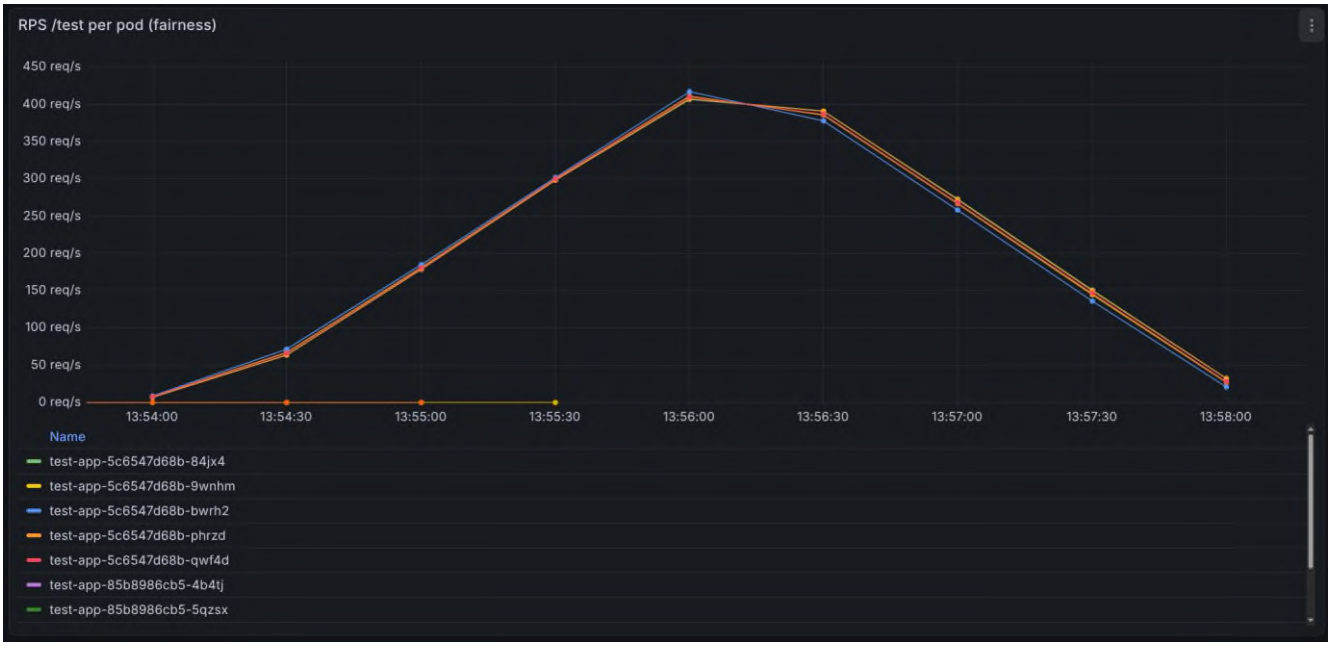


Рисунок 3.10 Розподіл RPS між Pod-ами сервісу /test у сценарії B для алгоритму RR (fairness)

Сценарій C - пікові (burst) навантаження. У сценарії C моделюється циклічне імпульсне навантаження на сервіс /test: інтенсивність запитів періодично зростає до фіксованого рівня (≈250 req/s), утримується протягом короткого інтервалу, після чого повертається майже до нуля. Такий профіль навантаження дозволяє оцінити поведінку алгоритмів балансування в умовах регулярних повторюваних піків середньої інтенсивності. Узагальнені результати для чотирьох алгоритмів наведено в табл. 3.3, а порівняльні значення p95-latency – на рис. 3.11.

Як свідчать дані табл. 3.3, середня пропускна здатність для всіх алгоритмів становить близько 1000 запитів за секунду, а частка помилкових відповідей залишається нульовою. Значення p50 для всіх методів практично збігаються (≈44–45 мс), тоді як відмінності проявляються у верхніх перцентилях. Найнижчі значення p95 демонструють динамічні алгоритми Least Connections (≈49,96 мс) та SR (random/approx) (≈49,68 мс), тоді як Round Robin та адаптивний Envoy мають p95 на рівні ≈54,79 мс та ≈52,93 мс відповідно. При цьому саме для RR та Envoy adaptive спостерігаються найбільші максимальні затримки (до 5–5,3 с), що свідчить про наявність поодиноких викидів у момент перемикання між імпульсами

навантаження.

Таблиця 3.3 - Результати сценарію C (пікові сплески)

Алгоритм	RPS (avg)	Запити (шт)	p50, мс	p90, мс	p95, мс	Avg, мс	Max, мс	Errors, %
RR	988.07	29 689	44.82	50.40	54.79	108.14	5 079.94	0
Least Connections	998.30	30 001	44.19	48.98	49.96	44.57	292.46	0
SR (random)	998.50	30 000	44.06	48.75	49.68	44.28	289.38	0
Envoy adaptive	989.92	29 748	44.91	50.17	52.93	105.75	5 284.05	0

Для наочного порівняння впливу пікових навантажень на час відповіді різних алгоритмів балансування побудовано стовпчиковий графік значень p95 latency (див. рис. 3.3).

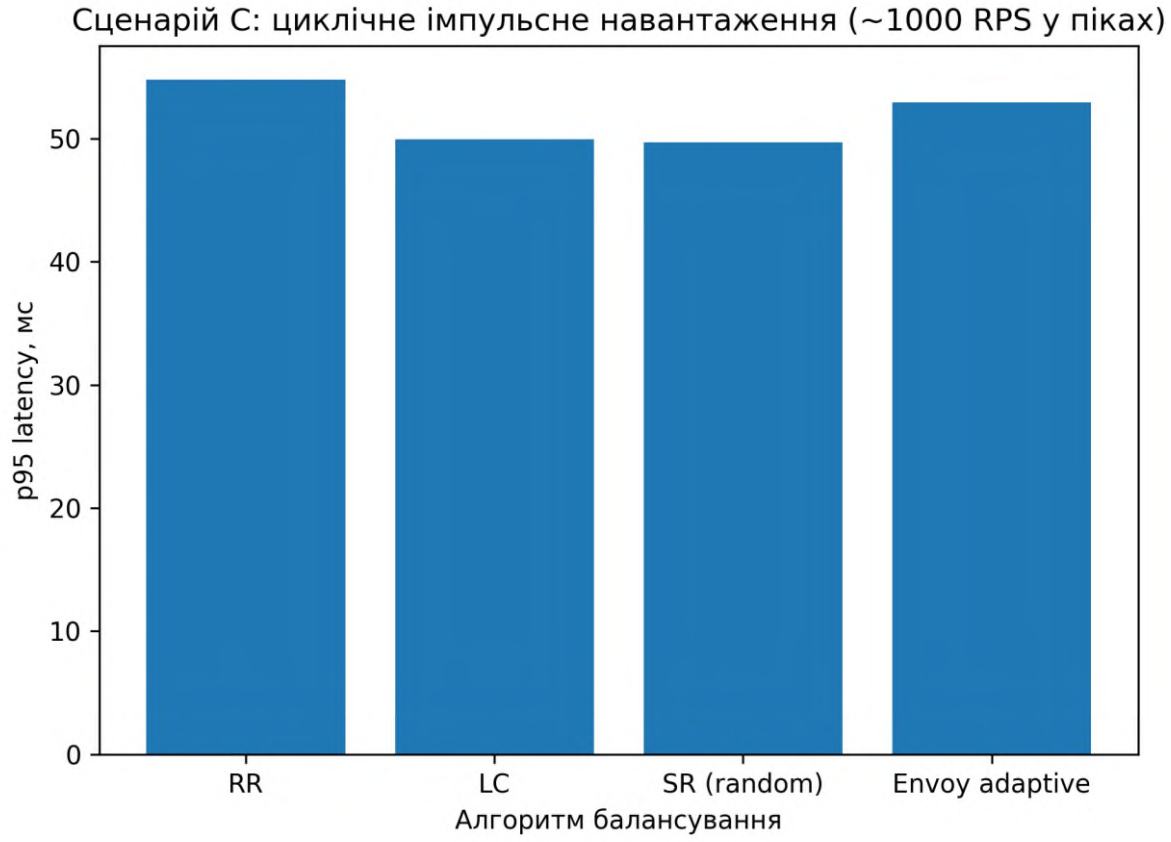


Рисунок 3.11 Значення p95 часу відгуку для алгоритмів балансування у сценарії C

(пікові сплески навантаження)

На рис. 3.12 показано часову динаміку загального RPS для алгоритму Envoy adaptive у сценарії C: послідовність імпульсів із амплітудою близько 250 req/s. Перцентилі затримки на рис. 3.13 залишаються стабільними (p50 ≈10–12 мс, p95 трохи вище 20 мс), що свідчить про коректну роботу адаптивного алгоритму за імпульсного навантаження.

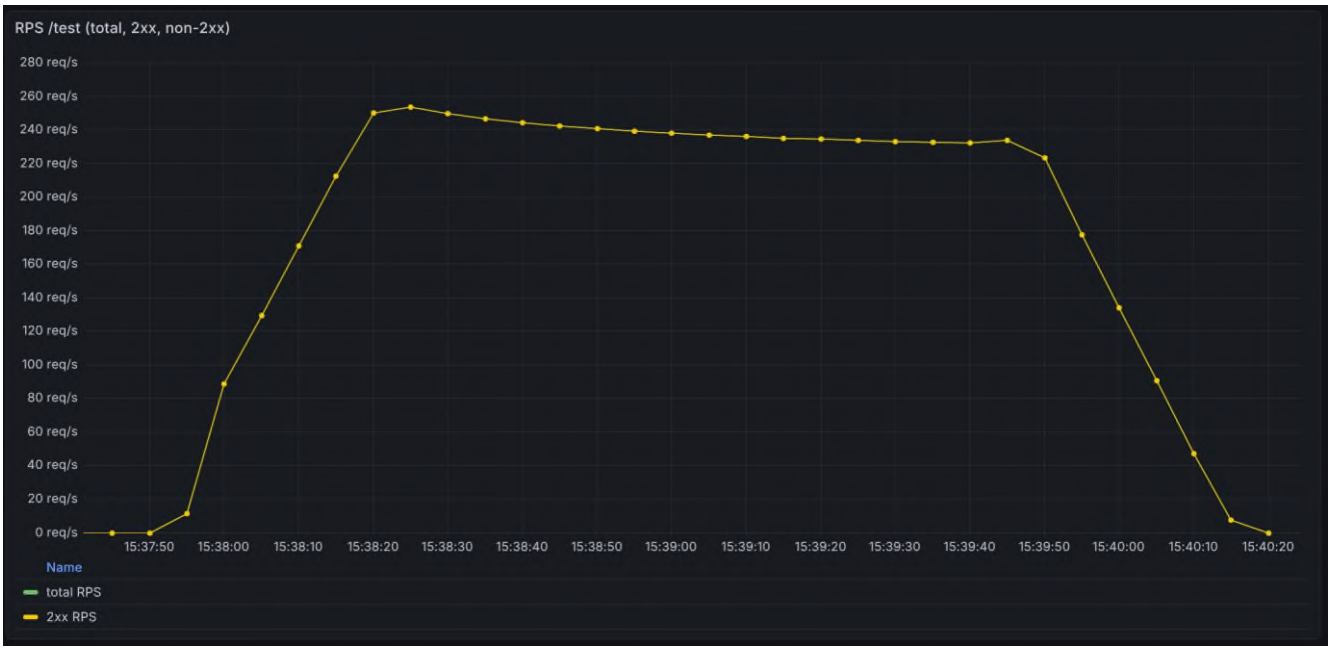


Рисунок 3.12 Динаміка загального RPS у сценарії C для алгоритму Envoy adaptive (циклічне імпульсне навантаження)



Рисунок 3.13 Перцентилі затримки (p50, p90, p95) у сценарії C для алгоритму Envoy adaptive

Рисунок 3.14 показує, що error rate упродовж усього тесту дорівнює 0 %, тобто HTTP-помилки та таймаути відсутні навіть за пікового навантаження. На рис. 3.15 видно, що криві RPS для трьох Pod-ів майже збігаються, що свідчить про рівномірний розподіл запитів без стійкого перекосу навантаження.



Рисунок 3.14 Частка помилкових запитів у сценарії C для алгоритму Envoy adaptive (error rate ≈ 0 %)

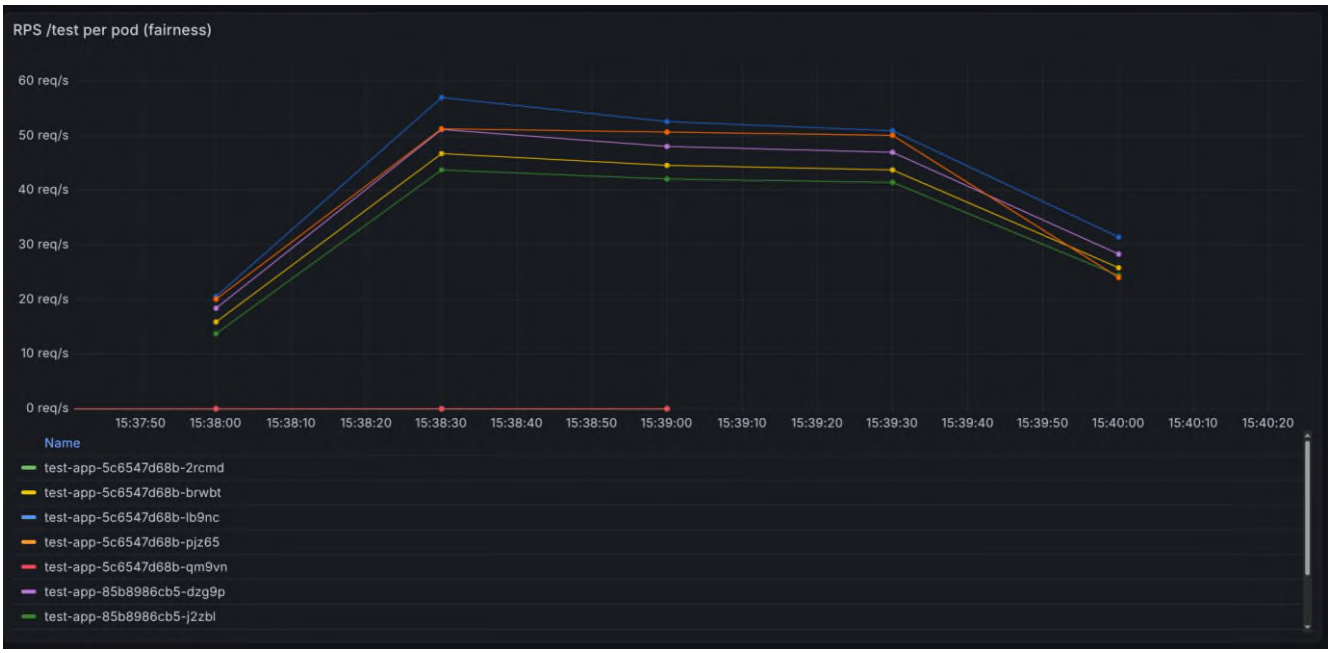


Рисунок 3.15 Розподіл RPS між Pod-ами сервісу /test у сценарії C для алгоритму Envoy adaptive (fairness)

У підсумку сценарій C показує, що за умов циклічного імпульсного навантаження динамічні алгоритми Least Connections та SR (random/approx)

забезпечують найкращий компроміс між стабільністю перцентилів затримки та відсутністю довгих «хвостів», тоді як Round Robin і адаптивний Envoy залишаються чутливими до поодиноких викидів затримки, хоч і повністю витримують вимоги до пропускної здатності та рівня помилок.

Сценарій D - деградація вузла. У цьому сценарії один із Pod-ів демонстрував штучну деградацію продуктивності (уповільнене обслуговування запитів), тоді як інші Pod-и залишалися в нормальному стані. Такий режим моделює часткові відмови або «повільні» вузли, які не падають повністю, але суттєво погіршують загальний час відповіді. Важливо оцінити, наскільки алгоритми балансування здатні виявляти деградований вузол і мінімізувати кількість запитів, що на нього потрапляють.

У таблиці 3.4 наведено результати сценарію E, включно з пропускною здатністю, перцентилями p50/p90/p95 часу відповіді, середнім та максимальним latency, а також часткою помилок для кожного алгоритму балансування.

Таблиця 3.4 - Результати сценарію D (деградація вузла)

Алгоритм	RPS (avg)	Запити (шт.)	p50, мс	p90, мс	p95, мс	Avg, мс	Max, мс	Errors, %
RR	299,81	36 000	44,96	83,70	104,35	51,88	527,75	0,0
Least Connections	299,80	36 001	45,07	86,77	103,33	52,33	975,81	0,0
SR (random)	299,75	35 985	44,49	65,15	86,95	49,67	993,86	0,0
Envoy adaptive	299,90	36 001	45,48	57,97	81,34	48,66	334,52	0,0

Для візуального порівняння впливу деградованого вузла на затримку побудовано стовпчиковий графік значень p95 часу відповіді для всіх алгоритмів балансування (див. рис. 3.16).

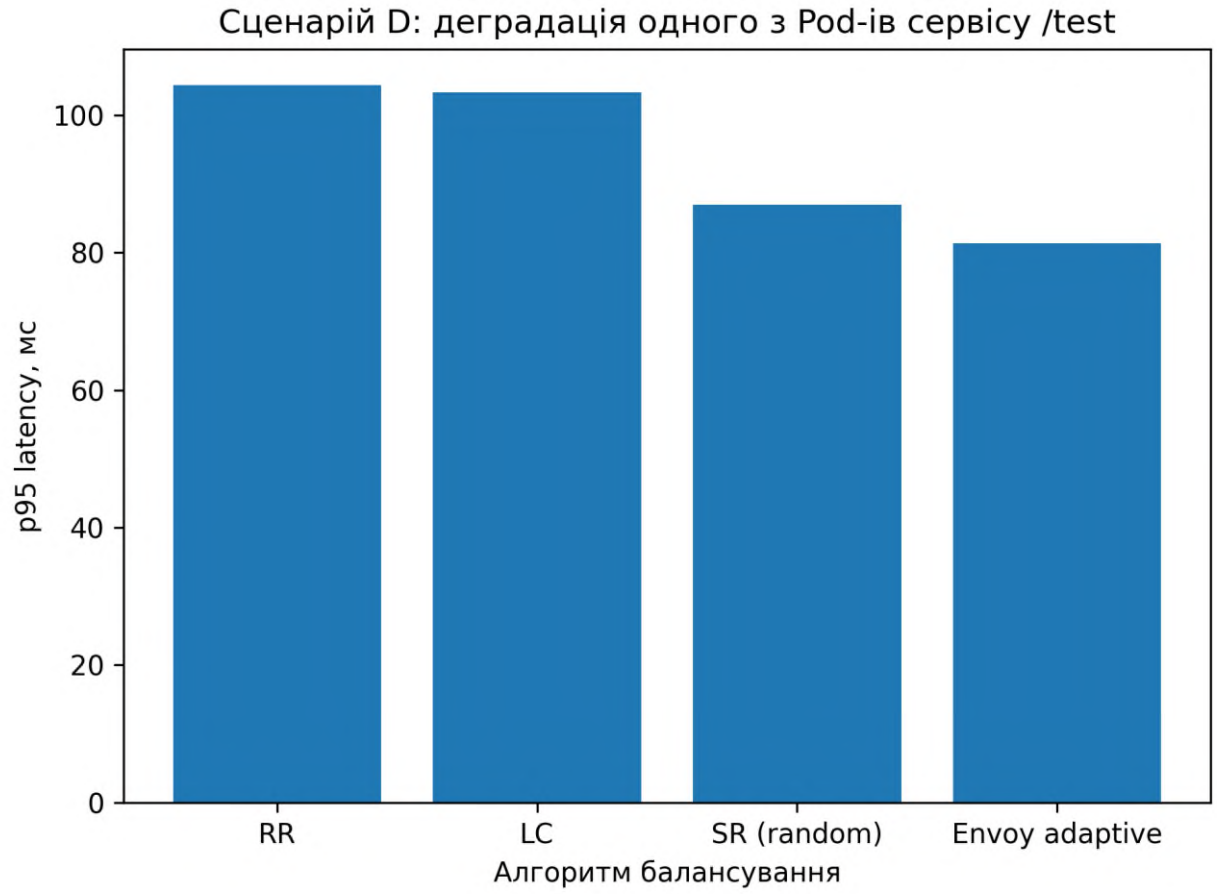


Рисунок 3.16 Значення p95 часу відгуку для алгоритмів балансування у сценарії D (деградація вузла)

У сценарії D моделюється деградація одного з Pod-ів сервісу /test: усередині вибраного Pod-а запускається фоновий процес, який інтенсивно навантажує CPU, тоді як інші репліки працюють у штатному режимі. Навантаження на сервіс зростає від нуля до приблизно 300 запитів за секунду, після чого плавно зменшується, що дозволяє оцінити поведінку алгоритмів балансування у перехідному режимі з частковою втратою продуктивності одного вузла. Узагальнені результати наведено в табл. 3.4, а порівняльні значення p95-latency – на рис. 3.16.

Середня інтенсивність навантаження для всіх алгоритмів становить близько 300 запитів за секунду, а частка помилкових відповідей дорівнює нулю, тобто деградація Pod-а проявляється насамперед у зростанні затримки, а не в HTTP-помилках. Найбільші значення p95-latency фіксуються для статичних і класичних динамічних алгоритмів: Round Robin (≈104,35 мс) та Least Connections (≈103,33 мс), причому для них також спостерігаються поодинокі запити з максимальною

затримкою понад 500–900 мс. Алгоритм SR (random/approx) зменшує верхні перцентилі до ≈86,95 мс, але окремі викиди затримки все ще досягають ≈990 мс. Найкращий результат демонструє адаптивний алгоритм на базі Envoy: p95-latency ≈81,34 мс, а максимальна затримка не перевищує ≈335 мс, що свідчить про більш ефективну реакцію на уповільнений Pod.

На рис. 3.17 показано часову динаміку загального RPS у сценарії D для алгоритму Envoy adaptive. Інтенсивність запитів лінійно зростає від нуля до ≈300 req/s, після чого так само плавно знижується, що відповідає профілю навантаження сценарію деградації вузла. Попри наявність одного перевантаженого Pod-а, сервіс у цілому підтримує заданий рівень пропускної здатності без провалів по RPS.

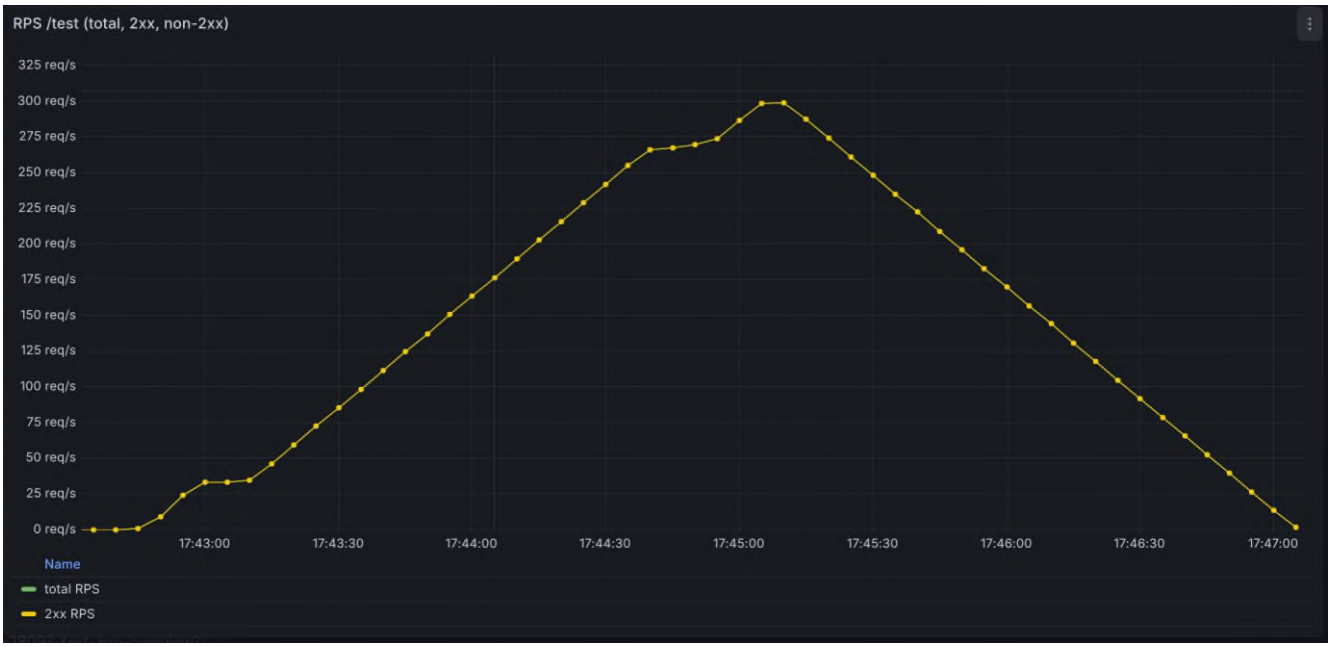


Рисунок 3.17 Динаміка загального RPS у сценарії D для алгоритму Envoy adaptive (деградація одного з Pod-ів сервісу /test)

Перцентилі затримки, наведені на рис. 3.18, демонструють характерний перехідний режим: на початку тесту значення p90–p99 підвищені, але в міру стабілізації навантаження крива p90 знижується і утримується в діапазоні десятків мілісекунд, тоді як p50 залишається майже горизонтальною. Це означає, що більшість запитів обробляється з відносно невеликою латентністю, а вплив деградованого Pod-а проявляється переважно у верхніх перцентилях, що узгоджується з числовими значеннями p95 з табл. 3.4.



Рисунок 3.18 Перцентилі затримки (p50, p90, p95, p99) у сценарії D для алгоритму Envoy adaptive

На рис. 3.19 показано, що показник error rate упродовж усього експерименту дорівнює 0 %, тобто HTTP-помилки та таймаути не виникають навіть за наявності повільного Pod-а.



Рисунок 3.19 Частка помилкових запитів у сценарії D для алгоритму Envoy adaptive (error rate ≈ 0 %)

На рис. 3.20 криві RPS для трьох Pod-ів значною мірою збігаються, що свідчить про рівномірний розподіл трафіку; відмінності в латентності зумовлені саме деградацією одного Pod-а, а не перекосом навантаження.

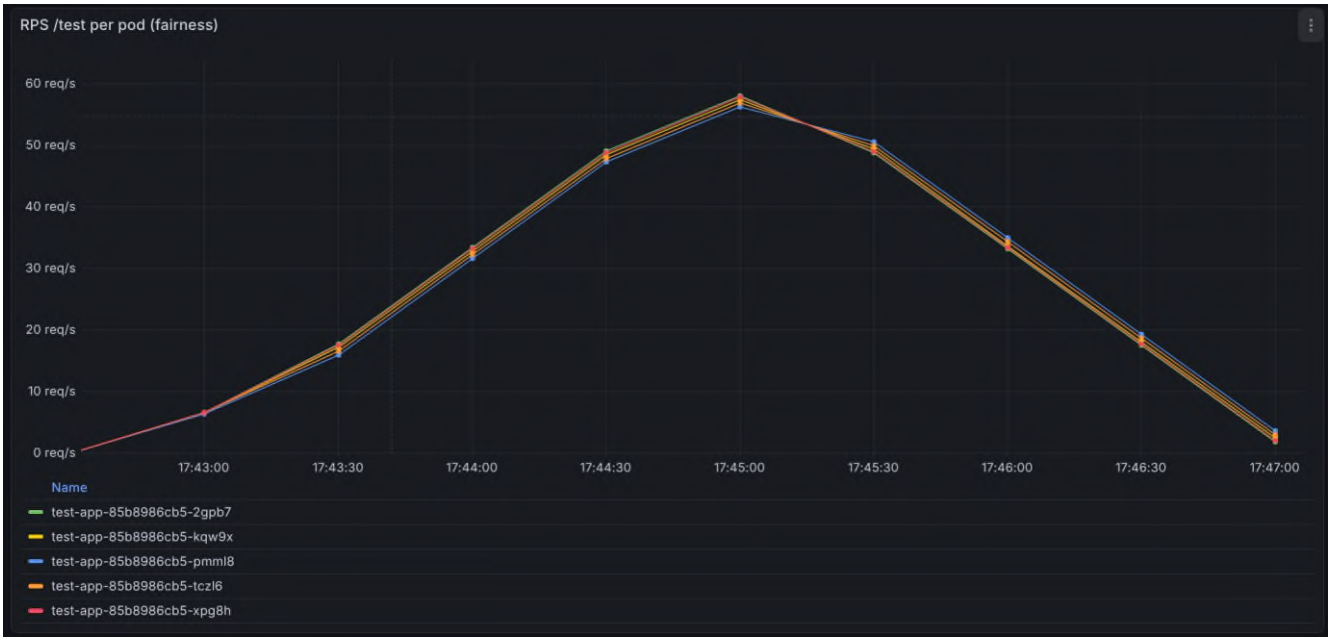


Рисунок 3.20 Розподіл RPS між Pod-ами сервісу /test у сценарії D для алгоритму Envoy adaptive (fairness)

У сукупності ці спостереження підтверджують, що в умовах часткової деградації вузла адаптивний алгоритм Envoy забезпечує стабільну пропускну здатність, рівномірний розподіл запитів між репліками та відносно низькі верхні перцентилі затримки, що робить його доцільним вибором для сценаріїв із можливими локальними відмовами. Це особливо важливо для сервісів, чутливих до «хвостів» латентності, де навіть одиничні повільні запити можуть суттєво погіршувати користувацький досвід і призводити до порушення вимог SLA. Фактично Envoy adaptive підтримує роботу кластера в режимі «м'якої деградації», не допускаючи переходу системи в стан різкої втрати продуктивності.

Таким чином, у сценарії деградації вузла усі розглянуті методи зберігають необхідну пропускну здатність і нульовий рівень помилок, проте адаптивний Envoy і SR (random/approx) формують суттєво коротший «хвіст» розподілу затримок порівняно з Round Robin та Least Connections. Це підтверджує перевагу підходів, які враховують поточний стан системи, у ситуаціях часткової деградації окремих реплік сервісу, оскільки вони дають змогу мінімізувати вплив проблемного Pod-а на загальний час відгуку та підтримувати більш стабільну якість обслуговування. У практичних умовах це означає підвищену стійкість до збоїв окремих компонентів і більшу надійність сервісу для кінцевих користувачів.

3.4. Проєкт адаптивної системи балансування навантаження на базі AWS та Kubernetes

Архітектура розробленої моделі системи балансування навантаження побудована за принципом модульності та багаторівневої організації. Вона поєднує компоненти контейнеризації, хмарної інфраструктури, моніторингу, що спільно забезпечують адаптивний розподіл навантаження та оцінку ефективності алгоритмів. Структура проєкту безпосередньо ґрунтується на висновках попередніх розділів щодо доцільності використання адаптивних та прогнозних алгоритмів у Kubernetes-кластері та потреби багаторівневого балансування в хмарній інфраструктурі AWS.

Мета архітектури - створити експериментальне середовище, у якому можна досліджувати роботу різних типів балансувальників (статичних, динамічних, інтелектуальних), моделювати зміну трафіку, збирати метрики й аналізувати продуктивність системи.

Загальна структура системи. Модель реалізовано як багаторівневу систему, що включає такі основні рівні:

1. Рівень інфраструктури (Infrastructure Layer). Складається з обчислювальних вузлів, розгорнутих у середовищі Amazon Web Services (AWS). Основу інфраструктури становлять:
    - EC2-інстанси, на яких розгортаються вузли Kubernetes або контейнерні середовища Docker Compose;
    - Elastic Load Balancer (ALB/NLB) - первинний балансувальник, який розподіляє вхідний трафік між вузлами кластера;
2. Рівень контейнеризації (Container Layer). У середовищі Kubernetes розгорнуто набір сервісів, кожен із яких представлений контейнерами:
    - Balance Service - балансувальник навантаження, що реалізує алгоритми Round Robin, Least Connection, Adaptive через NGINX Ingress

Controller;

- Application Pods - контейнери, які імітують серверні вузли або застосунки, що приймають запити від клієнтів;
- Load Generator - контейнер із інструментом k6 або JMeter, який генерує запити з різною інтенсивністю;
- Monitoring Stack - окремі контейнери Prometheus і Grafana, що збирають і візуалізують метрики.

3. Рівень балансування (Load Balancing Layer). Реалізує кілька рівнів розподілу навантаження:
    - L4-балансування (Network Load Balancer або kube-proxy) - базовий розподіл TCP/UDP-трафіку між вузлами;
    - L7-балансування (Ingress Controller, NGINX, HAProxy) - маршрутизація HTTP-запитів за URI або контентом;
    - Predictive Load Balancer - модуль, що отримує прогнозні значення навантаження від ML-моделі та коригує політику балансування.
4. Рівень моніторингу та аналітики (Monitoring and Analytics Layer)
    - Prometheus збирає метрики з балансувальників, застосунків і контейнерів через endpoint /metrics;
    - Grafana візуалізує ключові показники - CPU, memory, latency, throughput, error rate;
    - Метрики також експортуються у Python-модуль машинного навчання для прогнозу майбутнього навантаження.
5. Рівень візуалізації та звітності (Visualization Layer). Для підсумкового аналізу результати експериментів - метрики latency, throughput, utilization, SLA compliance - виводяться на дашборди Grafana.

На рівні проєктних рішень у роботі сформовано архітектуру модуля машинного навчання, який у перспективі має виконувати прогнозування навантаження на основі часових рядів метрик, що збираються Prometheus. Передбачається можливість реалізації таких підходів, як лінійна регресія та LSTM-мережі. Важливо підкреслити, що у межах експериментальної частини даної

роботи ML-модуль не був розгорнутий і не брав участі у вимірюваннях; його роль обмежується концептуальним описом і обґрунтуванням доцільності впровадження. Послідовність взаємодії компонентів у системі описується так:

1. Користувацький або симульований запит надходить через Elastic Load Balancer (L4) у Ingress Controller (L7) кластера Kubernetes.
2. Балансувальник визначає цільовий Pod за обраним алгоритмом (Round Robin, Least Connection або Predictive).
3. Application Pod обробляє запит і повертає відповідь користувачу.
4. Prometheus збирає метрики запитів, навантаження на CPU, пам’ять та час відповіді з усіх вузлів.
5. Дані з Prometheus надходять до Grafana для візуалізації і до ML-модуля для навчання або прогнозу.
6. Прогнозна модель формує оцінку майбутнього навантаження і передає її в Kubernetes HPA, який змінює кількість Pod-ів.
7. Новий стан кластера враховується балансувальником, і трафік автоматично перенаправляється відповідно до оновлених ресурсів.

Ця взаємодія утворює замкнений цикл зворотного зв’язку, у якому моніторинг і прогнозування впливають на рішення балансувальника, а зміни конфігурації відображаються у нових метриках.

На рисунку 3.1 схематично зображено загальну архітектуру моделі системи балансування навантаження в хмарному середовищі.

Архітектура моделі забезпечує інтеграцію традиційних та інтелектуальних підходів до балансування навантаження. Її багаторівнева структура дозволяє:

- гнучко змінювати алгоритми балансування без модифікації інфраструктури;
- забезпечувати реальний моніторинг і прогнозування в єдиному циклі;
- досліджувати вплив алгоритмів на ключові показники продуктивності системи.

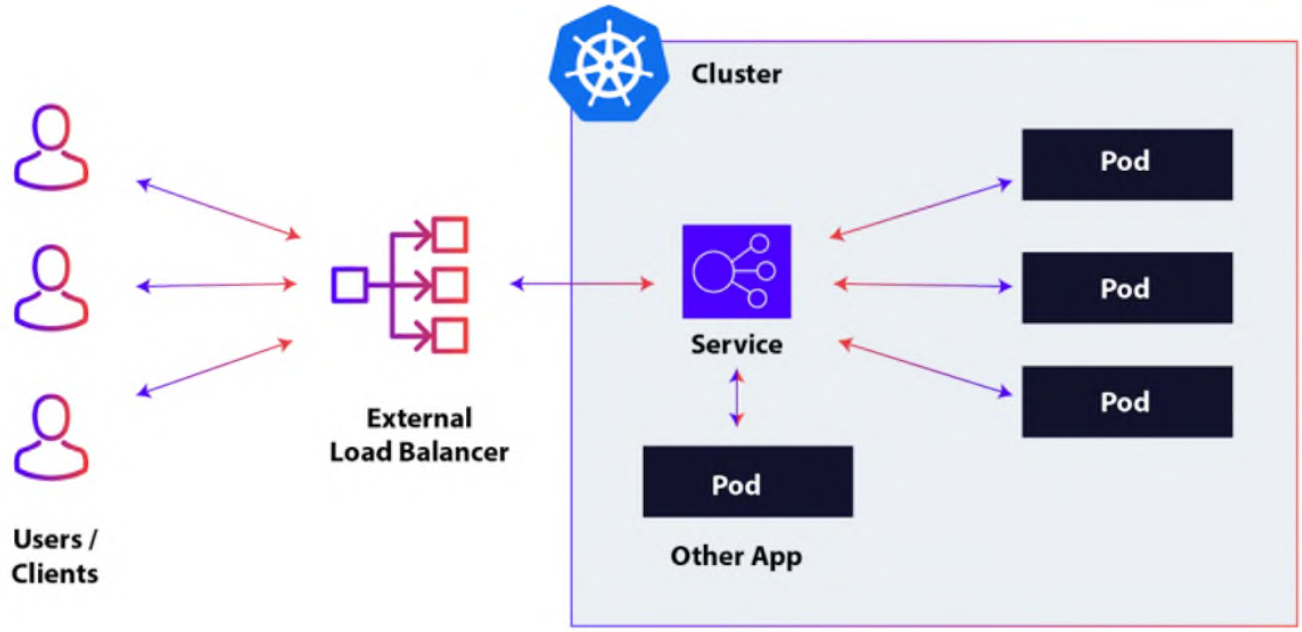


Рисунок 3.21 Архітектура моделі системи балансування навантаження в хмарному середовищі

Така архітектура є універсальною платформою для подальших експериментів із методами оптимізації, машинного навчання та енергоефективного розподілу ресурсів у хмарних обчисленнях.

Далі наведено опис процесу побудови експериментального стенду, який використовується для дослідження роботи різних методів і алгоритмів балансування навантаження у хмарному середовищі. Стенд реалізовано як комбіновану інфраструктуру, що поєднує можливості платформи Amazon Web Services (AWS), контейнерної оркестрації Kubernetes, систем моніторингу Prometheus і Grafana, інструментів генерування навантаження та модуля машинного навчання для прогнозування.

Метою створення стенду є забезпечення можливості відтворення реальних сценаріїв роботи хмарних застосунків, збору телеметричних даних, порівняння алгоритмів балансування за ключовими метриками та дослідження впливу адаптивних і прогнозних методів на продуктивність системи. Структурно стенд складається з трьох основних частин:

1. Інфраструктурний рівень, що включає AWS-компоненти, EC2-вузли, мережеву підсистему та балансувальники трафіку.

2. Контейнерний рівень, реалізований у Kubernetes, де розгорнуті сервіси, балансувальники, генератор навантаження та інструменти моніторингу.
3. Аналітичний рівень, який об’єднує системи збору метрик з ML-модулем прогнозування та HPA-механізмом масштабування.

У наступних підрозділах наведено детальний опис кожного компонента та їх функціональне призначення.

Експериментальний стенд було розгорнуто у середовищі Amazon Web Services (AWS) та сформовано з набору компонентів, які забезпечують обчислювальні ресурси, мережеву взаємодію, балансування трафіку, зберігання даних і моніторинг системи. Архітектура побудована таким чином, щоб гарантувати ізольованість сервісів, відтворюваність експериментів та можливість масштабування.

Основою інфраструктури є набір EC2-вузлів, на яких розгорнуто Kubernetes-кластер. До його складу входить вузол керування (Control Plane), що відповідає за роботу API-сервера, планувальника, контролерів і etcd, а також два робочі вузли, на яких виконуються Pod-и застосунків, балансувальників та інструментів генерації навантаження. Окремий вузол виділено для системи моніторингу (Prometheus і Grafana) з метою мінімізувати вплив моніторингових процесів на продуктивність основного кластера. Усі EC2-інстанси розташовані в межах однієї Virtual Private Cloud із розділенням на публічні та приватні підмережі.

Маршрутизацію зовнішнього трафіку забезпечує AWS Application Load Balancer, який виконує L7-балансування, приймає HTTP/HTTPS-запити та передає їх на Ingress Controller у кластері Kubernetes. ALB використовується як точка входу в систему, підтримує багатозонність та автоматичне масштабування, що дозволяє моделювати реальні умови обробки трафіку.

Мережева інфраструктура побудована на основі VPC із виділенням публічних підмереж для балансувальника та приватних підмереж для EC2-вузлів. Доступ до Інтернету для приватних ресурсів здійснюється через NAT Gateway, тоді як Internet Gateway використовується для обробки зовнішніх запитів. Така сегментація забезпечує додатковий рівень безпеки та ізоляцію експериментального

середовища.

Моніторинг і журналювання реалізовано за допомогою Prometheus і Grafana, а також сервісів AWS CloudWatch, які використовуються для збирання системних логів та резервного отримання метрик. Така комбінація дозволяє отримувати повний набір телеметрії - від затримок і пропускної здатності до навантаження на вузли та подій autoscaling.

Безпека інфраструктури забезпечується використанням Security Groups з чітко визначеними правилами доступу до портів, а також SSH-доступу лише з довірених IP-адрес. Така модель дозволяє ізолювати експериментальний трафік та гарантує контрольований доступ до компонентів інфраструктури.

Серверні вузли та симулятор навантаження є ключовими компонентами експериментального стенду, оскільки вони забезпечують роботу застосунків, виконання HTTP-запитів, моделювання різних профілів навантаження та створюють умови для оцінювання ефективності алгоритмів балансування. Підрозділ описує підхід до розгортання серверних сервісів у Kubernetes-кластері, конфігурацію мікросервісів та інструменти генерації навантаження.

Серверні вузли реалізовані у вигляді контейнеризованих застосунків, які розгортаються в Kubernetes як Pod-и та відповідні служби (Services). Кожен серверний вузол є екземпляром легкого мікросервісу, що обробляє HTTP-запити та надає можливість вимірювати час відповіді, пропускну здатність та стабільність роботи. Для цього створено Docker-образ, який містить мінімальний веб-сервіс (наприклад, FastAPI або Flask), ендпоінт /test з керованою затримкою для імітації реального часу обробки, підтримку логування та механізм експорту власних метрик у форматі Prometheus. Образ зберігається у приватному репозиторії AWS ECR або Docker Hub, залежно від конфігурації.

Розгортання серверних вузлів здійснюється за допомогою стандартних ресурсів Kubernetes, таких як Deployment та Service. Deployment визначає кількість реплік, параметри контейнера, політики оновлення та інтеграцію з горизонтальними механізмами масштабування. Service забезпечує мережевий доступ до Pod-ів та використовується балансувальником для маршрутизації

запитів.

Для симуляції навантаження застосовується інструмент k6, який запускається в окремому Pod-і та виконує роль генератора трафіку. У конфігурації k6 визначено логіку формування запитів, інтенсивність навантаження, тривалість моделювання та профіль користувачів. Такий підхід дозволяє створювати як стабільні, так і динамічні, пікові або асиметричні сценарії, що точно відтворюють реальні умови експлуатації хмарних застосунків.

Серверна частина та симулятор інтегровані зі системою моніторингу, що дає змогу збирати метрики продуктивності, фіксувати навантаження на Pod-и та корелювати результати роботи алгоритмів балансування з поведінкою системи під час експериментів.

Усі алгоритми балансування були реалізовані у вигляді окремих конфігурацій Kubernetes Ingress Controller. Для кожного методу створювалися власні правила маршрутизації, які визначали спосіб розподілу трафіку між серверними Pod-ами. Такий підхід забезпечує відтворюваність експериментів та можливість послідовного порівняння алгоритмів за однакових умов.

У межах експериментального стенду реалізовано статичний алгоритм Round Robin із використанням стандартних механізмів Ingress Controller. Алгоритм Weighted Round Robin розглядається лише на теоретичному рівні як можливе розширення статичного підходу до балансування.

Динамічні алгоритми, такі як Least Connections і Shortest Response Time, покладаються на інформацію про кількість активних з’єднань або затримки при відповіді. Вони реалізовуються через анотації Ingress Controller або додаткові модулі, що відстежують стан backend-вузлів у реальному часі. Ці методи дозволяють адаптувати розподіл трафіку відповідно до поточного завантаження Pod-ів.

Адаптивний алгоритм використовує розширений набір метрик (латентність, кількість активних потоків, ресурсні показники), що збираються Prometheus. На основі цих метрик Ingress Controller автоматично коригує розподіл трафіку, зменшуючи навантаження на вузли з підвищеним часом відповіді або

деградованим станом. Реалізація передбачає використання спеціальних модулів або власних Lua-скриптів у NGINX, що аналізують метрики у реальному часі.

Усі алгоритми реалізовано таким чином, щоб забезпечити повну ізольованість конфігурацій, можливість їх швидкої заміни та проведення послідовної серії експериментів для всіх розглянутих сценаріїв навантаження.

Інфраструктура експериментального стенду передбачала використання централізованої системи моніторингу, яка забезпечувала збір телеметрії з усіх компонентів Kubernetes-кластера, балансувальника та навантажувальних сценаріїв. Метою інтеграції було отримання коректних метрик продуктивності (latency, RPS, error rate), стану контейнерів та поведінки системи під час різних типів навантаження.

Для збору даних на рівні кластеру використовувалася зв’язка Prometheus + Node Exporter + Kube State Metrics, що забезпечувала повний огляд стану вузлів, Pod-ів, CPU/Memory, подій масштабування та мережевої активності.

Prometheus працював у режимі періодичного опитування (pull-model) та збирав метрики як із системних експортерів, так і з балансувальника (Ingress NGINX / Envoy), який надавав власний ендпоінт /metrics.

Для збору та зберігання високорівневих метрик навантаження (RPS, час відповіді, кількість успішних та помилкових запитів) був інтегрований модуль експорту з k6, що дозволяв передавати результати навантажувальних тестів у Prometheus. Це забезпечувало повну кореляцію між поведінкою навантажувального інструмента та станом Kubernetes-кластера.

Аналітичне відображення метрик здійснювалося через Grafana, де було створено окремі дашборди для:

- latency (p50, p90, p95, p99),
- пропускної здатності (real-time RPS),
- кількості помилок та timeouts,
- навантаження на CPU/Memory Pod-ів,
- динаміки autoscaling (HPA),
- стану балансувальника.

У межах проведених експериментів основним джерелом телеметрії виступав Prometheus, що збирав метрики з Kubernetes-кластера, балансувальника та навантажувальних сценаріїв.

Такий підхід дозволив отримати повну і узгоджену картину роботи кластера під різними типами навантаження. Усі зібрані метрики були використані для подальшого аналізу ефективності алгоритмів балансування, порівняння їхньої поведінки в різних сценаріях навантаження та формування практичних рекомендацій щодо їх застосування.

3.5. Оцінка очікуваних ефектів від впровадження запропонованих рішень

Оцінка ефективності запропонованих рішень ґрунтується на порівняльному аналізі алгоритмів балансування навантаження, результати якого наведені у таблицях 3.1-3.5. На основі цих експериментальних даних можна спрогнозувати, які зміни відбудуться у функціонуванні хмарної платформи у разі переходу від базових статичних методів до адаптивних та прогнозних підходів.

По-перше, очікується помітне зменшення латентності. Статичні алгоритми (Round Robin тощо) дають прийнятні результати лише за відносно рівномірного навантаження; у разі піків чи деградації вузлів зростають середній час відповіді та перцентильні значення latency (p95, p99). Перехід до динамічних та адаптивних алгоритмів дозволяє знизити затримки в критичних сценаріях і зробити час відповіді стабільнішим навіть за різких коливань навантаження.

По-друге, очікується підвищення рівня виконання SLA та загальної надійності сервісу. За результатами експериментів адаптивні підходи зменшують частку помилок і таймаутів порівняно зі статичними й класичними динамічними алгоритмами, що особливо важливо для високонавантажених та критичних застосунків, де порушення SLA безпосередньо впливає на користувацький досвід і може мати фінансові наслідки.

По-третє, поліпшується ефективність використання обчислювальних

ресурсів. Статичні методи часто потребують значного запасу продуктивності для компенсації нерівномірного розподілу навантаження, тоді як адаптивні алгоритми дають змогу точніше узгоджувати кількість активних реплік та інстансів із фактичним навантаженням. Це зменшує потребу в перепровізіонуванні інфраструктури та дозволяє скоротити витрати без погіршення якості сервісу.

По-четверте, очікуються позитивні операційні ефекти. Стабільніша робота системи під навантаженням і менша кількість критичних відхилень знижують навантаження на команду супроводу, скорочують число інцидентів, що потребують ручного втручання, та спрощують планування масштабування. Наявність формалізованої методики вибору алгоритму й узагальнених рекомендацій для типових сценаріїв додатково зменшує ризики помилкових рішень при розширенні або модернізації інфраструктури.

Отже, впровадження запропонованих рішень у реальній хмарній інфраструктурі дає підстави очікувати комплексне покращення технічних, експлуатаційних та економічних показників: зменшення затримок, підвищення стабільності й надійності, оптимізацію використання ресурсів і зниження операційних ризиків. Конкретні значення приросту цих показників залежать від профілю навантаження та початкового стану системи, однак результати експериментального моделювання демонструють чітку перевагу адаптивних алгоритмів над базовими статичними, а прогнозні ML-підходи розглядаються як перспективний наступний етап розвитку системи балансування.

3.6. Висновки до розділу 3

У третьому розділі було розроблено та обґрунтовано конструктивні рішення щодо вдосконалення системи балансування навантаження в хмарній інфраструктурі на основі результатів теоретико-методичного аналізу (розділ 1) та аналітико-дослідницького етапу (розділ 2). На основі експериментальних даних сформовано методику вибору алгоритмів балансування для різних типів

навантаження та запропоновано цільову архітектуру адаптивної системи балансування на базі сервісів AWS і кластера Kubernetes.

Сформована методика враховує як технічні критерії, так і експлуатаційні та організаційні обмеження. На її основі побудовано узагальнену відповідність між типами навантаження та класами алгоритмів балансування, що дозволяє приймати обґрунтовані рішення щодо вибору конкретного методу в реальних умовах експлуатації.

Порівняльний аналіз роботи статичних, динамічних, адаптивних і прогнозних алгоритмів показав, що статичні підходи є прийнятними лише для простих сценаріїв із відносно стабільним навантаженням і невисокими вимогами до SLA. Динамічні методи краще пристосовані до змінних умов, проте зберігають уразливість до різких піків і деградації вузлів. Найбільш перспективними для сучасних хмарних платформ виявилися адаптивні та прогнозні алгоритми, які забезпечують кращий компроміс між продуктивністю, стабільністю та ефективністю використання ресурсів.

Запропонований проєкт адаптивної системи балансування навантаження на базі AWS та Kubernetes демонструє, яким чином результати дослідження можуть бути впроваджені на практиці. Поєднання багаторівневого балансування (L4/L7), розвиненої системи моніторингу та компонентів машинного навчання дозволяє будувати системи, здатні проактивно реагувати на зміни навантаження, підтримувати необхідний рівень SLA та забезпечувати еластичність інфраструктури.

Таким чином, у третьому розділі вирішено поставлені завдання щодо розроблення методики вибору алгоритмів балансування, побудови цільової архітектури адаптивної системи та оцінки очікуваних ефектів від її впровадження. Отримані результати мають практичну цінність для проєктування та модернізації хмарних платформ і слугують основою для подальших досліджень у напрямі інтеграції більш складних інтелектуальних методів та розширення експериментальної бази.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження методів і алгоритмів балансування навантаження в інфраструктурі хмарних обчислень. Робота охоплює теоретичний аналіз сучасних підходів, побудову експериментального середовища, моделювання різних типів навантаження та порівняння статичних, динамічних і адаптивних алгоритмів з урахуванням ключових критеріїв ефективності, а також аналіз перспективних прогнозних (ML-орієнтованих) методів на основі огляду наукових джерел.

Основні наукові та практичні результати, отримані в роботі, можна сформулювати так:

1. Узагальнено теоретичні засади хмарних обчислень, класифіковано сервісні моделі (IaaS, PaaS, SaaS) і архітектурні підходи до побудови хмарних платформ, а також систематизовано основні методи балансування навантаження. Сформовано комплекс критеріїв оцінювання ефективності алгоритмів (latency, throughput, стабільність, відповідність SLA, енергоефективність), який використано як основу для подальшого аналізу.
2. Проведено аналіз статичних, динамічних, адаптивних та інтелектуальних методів балансування навантаження, визначено їх переваги, обмеження та області доцільного застосування. Показано, що традиційні статичні підходи є ефективними переважно в умовах стабільного й передбачуваного трафіку, тоді як динамічні та адаптивні методи краще пристосовані до високодинамічних хмарних середовищ. На основі огляду літератури окреслено потенціал прогнозних та ML-орієнтованих підходів, зокрема в задачах проактивного масштабування та оптимізації використання ресурсів.
3. Описано та реалізовано експериментальне середовище на базі AWS та Kubernetes з використанням багаторівневого балансування (Application Load Balancer, Ingress-контролер, внутрішні проксі), систем моніторингу (Prometheus, Grafana, CloudWatch) та інструментів навантажувального тестування. Сформовано набір із чотирьох сценаріїв, що моделюють

рівномірне навантаження, поступове зростання (ramp-up), пікові (burst) сплески та деградацію вузла, що дозволило відтворити типові режими роботи хмарних сервісів.

4. Реалізовано та інтегровано в експериментальне середовище набір алгоритмів балансування: Round Robin, Least Connections, Shortest Response (SR, random/approx) та адаптивний алгоритм Envoy adaptive. Це забезпечило можливість порівняння поведінки представників статичних, динамічних і адаптивних підходів за уніфікованими метриками без залучення окремої реалізації ML-балансування, яке розглянуто як перспективний напрям подальших досліджень.
5. На основі серії навантажувальних експериментів отримано кількісні результати та здійснено порівняльний аналіз алгоритмів за середнім часом відповіді, поведінкою високих перцентилів затримки, пропускною здатністю, кількістю помилок, стабільністю роботи сервісів та відповідністю SLA. Показано, що статичний алгоритм Round Robin є прийнятним для сценарію з рівномірним навантаженням, динамічний Least Connections краще поводиться за зростаючого трафіку (ramp-up), а алгоритм SR (random/approx) та адаптивний Envoy забезпечують найкращі показники за p95/p99 у складних режимах burst-навантажень і деградації вузла. Таким чином, адаптивні підходи демонструють найвищу ефективність у сценаріях із різко змінним або частково деградованим середовищем.
6. Запропоновано методику вибору алгоритмів балансування навантаження залежно від типу навантаження, вимог до SLA та обмежень інфраструктури. Побудовано відповідність між класами алгоритмів (статичні, динамічні, адаптивні) і сценаріями роботи (рівномірний трафік, поступове зростання, пікові навантаження, деградація вузла), що дозволяє обґрунтовано обирати оптимальний підхід для конкретної хмарної системи та уникати як недовантаження, так і перевантаження окремих вузлів.
7. Розроблено проєкт адаптивної системи балансування навантаження на базі AWS та Kubernetes з використанням багаторівневого розподілу трафіку та

розвиненої підсистеми моніторингу. Показано очікувані технічні та експлуатаційні ефекти від впровадження адаптивного балансування: зниження середнього часу відповіді та високих перцентилів латентності, підвищення стабільності сервісів, покращення відповідності SLA та більш раціональне використання ресурсів. Перспективою розвитку запропонованого рішення є інтеграція прогнозних ML-моделей для проактивного керування ресурсами.

Поставлена в роботі мета досягнута повністю: досліджено поведінку сучасних алгоритмів балансування у хмарних середовищах, побудовано експериментальну платформу, проведено моделювання в чотирьох сценаріях навантаження та отримано кількісні результати, що дозволили сформувати практичні рекомендації щодо вибору оптимального підходу. Усі завдання дослідження виконані, зокрема:

- проаналізовано сучасні методи балансування навантаження;
- розроблено архітектуру експериментального середовища;
- реалізовано стенд у AWS на базі Kubernetes;
- проведено серію навантажувальних експериментів для чотирьох сценаріїв;
- здійснено порівняльний аналіз ефективності алгоритмів Round Robin, Least Connections, SR (random/approx) та Envoy adaptive за основними критеріями;
- сформовано рекомендації щодо практичного використання адаптивних та динамічних підходів і окреслено перспективи застосування прогнозних (ML-орієнтованих) методів.

Практичне значення роботи полягає у можливості використання отриманих результатів для проєктування високопродуктивних і відмовостійких хмарних сервісів, вибору адекватного алгоритму балансування залежно від характеру навантаження, а також для подальшого розширення системи за рахунок інтеграції моделей машинного навчання з метою прогнозування трафіку та проактивного керування ресурсами. Результати дослідження можуть бути використані як основа для розробки гібридних та самонавчальних систем балансування навантаження і розвитку інтелектуальних методів управління ресурсами хмарних платформ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mell P., Grance T. The NIST Definition of Cloud Computing. Special Publication 800-145. – Gaithersburg, MD: National Institute of Standards and Technology, 2011. – 7 p.
2. Alam M., et al. Systematic Review: Load Balancing in Cloud Computing. Intelligent Automation & Soft Computing, 2025. – Vol. 39(3). – P. 2781–2805. URL: https://www.techscience.com/iasc/v39n3/57288/html
3. Sharma M., Verma R., Gupta A. Hybrid Metaheuristic-Based Cloud Task Scheduling Using Genetic Algorithm and Salp Swarm Algorithm. International Journal of Intelligent Engineering and Systems, 2025. – Vol. 18(1). – P. 45–57.
4. Beloglazov A., Abawajy J., Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems, 2012, Vol. 28(2), pp. 755–768.
5. Syed D., Memon A., Memon M. Systematic Review: Particle Swarm Optimization (PSO) based Load Balancing for Cloud Computing. SSURJ, 2024
6. Aruna K., Pradeep G. Ant Colony Optimization-based Light Weight Container (ACO-LWC) Algorithm for Efficient Load Balancing. Intelligent Automation & Soft Computing, 2022. – Vol. 3
7. Shahid M. A., Alam M. M. Performance Evaluation of Load-Balancing Algorithms with Different Service Broker Policies for Cloud Computing. Applied Sciences, 2023. – Vol. 13(3). – Article No. 1586. DOI: 10.3390/app13031586
8. Bala A., Chana I. Prediction-based proactive load balancing approach through VM migration. Engineering with Computers, 2016
9. Khurana S., Sharma G. Hybrid Machine Learning Model for Load Prediction in Cloud Environment. International Journal of Performability Engineering, 2023. – Vol. 19(8). – P. 507–515. DOI: 10.23940/ijpe.23.08.p3.507515
10. Sarikaa S., Niranjana S., Deepika K. S. Time Series Forecasting of Cloud Resource Usage. 2021 International Conference on Advancements in Electrical,

Electronics, Communication, Computing and Automation (ICAECA). IEEE, 2021. DOI: 10.1109/ICAECA53005.2021.9666444

11. Calheiros R. N., Masoumi E., Ranjan R., Buyya R. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. IEEE, 2013. DOI: 10.1109/UCC.2013.43
12. Patel E., Kushwaha D. S. A hybrid CNN-LSTM model for predicting server load in cloud computing. The Journal of Supercomputing, 2022. – Vol. 78. – P. 1–30.
13. Sefati S. S., Nor A. M., Arasteh B., Craciunescu R., Comsa C.-R. A Probabilistic Approach to Load Balancing in Multi-Cloud Environments via Machine Learning and Optimization Algorithms. Journal of Grid Computing, 2025. – Vol. 23, Article No. 16.
14. Pandeeswari S. T., Padmavathi S., Poornisha M. S., Karpagalakshmi P. Predictive Cloud Resource Management by Workload Forecasting with an Ensemble of Hybrid ARIMA-LSTM and EL Models. Anadolu Psikiyatri Dergisi, 2025. – Vol. 26. – P. 3375–3389.
15. Mao H., Alizadeh M., Menache I., Kandula S. Resource Management with Deep Reinforcement Learning. Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets), 2016.
16. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. – Sebastopol, CA: O'Reilly Media, 2021. – 422 p.
17. Sharifian S., Motamedi S. A., Akbari M. K. A content-based load balancing algorithm with admission control for cluster web servers. Future Generation Computer Systems, 2008. – Vol. 24(8). – P. 775–787.
18. NGINX Inc. NGINX Load Balancing Guide. Official Documentation, 2024. URL: https://docs.nginx.com/nginx/admin-guide/load-balancer/
19. Envoy Proxy. Advanced L7 Load Balancing Features. Official Documentation, 2023. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/overview

20. Amazon Web Services. Application Load Balancer Documentation. AWS Official Documentation, 2024. URL: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html
21. Bhattacharya R., Gao Y., Wood T. Dynamically Balancing Load with Overload Control for Microservices. Proceedings of the ACM Symposium on Cloud Computing (SoCC), 2024.
22. HAProxy Technologies. HAProxy User Guide – TCP Load Balancing. Official Documentation, 2024. URL: https://www.haproxy.com/documentation/haproxy-configuration/configuration/tcp/
23. Amazon Web Services. Network Load Balancer Documentation. AWS Official Documentation, 2024. URL: https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html
24. Google Cloud. TCP/UDP Load Balancing Overview. Google Cloud Documentation, 2024. URL: https://cloud.google.com/load-balancing/docs/tcp-udp/
25. Lee J.-B., Yoo T.-H., Lee E.-H., Hwang B.-H., Ahn S.-W., Cho C.-H. High-Performance Software Load Balancer for Cloud-Native Architecture. Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), IEEE, 2021.
26. Younus A., Hammo A. Y. A Comparative Study of the Performance of Load Balancing Algorithms Using Cloud Analyst. Webology, 2022. – Vol. 19(1). – P. 4898–4911.
27. Microsoft Azure. Azure Load Balancer Documentation. Microsoft Docs, 2024. URL: https://learn.microsoft.com/azure/load-balancer/
28. The Kubernetes Authors. Kubernetes Networking and Load Balancing. Kubernetes Documentation, 2024. URL: https://kubernetes.io/docs/concepts/
29. Istio Authors. Istio Service Mesh Documentation. CNCF Docs, 2024. URL: https://istio.io/latest/docs/

30. Amazon Web Services. AWS App Mesh Documentation. AWS Docs, 2023. URL: https://docs.aws.amazon.com/app-mesh/

31. Hightower K., Burns B., Beda J. Kubernetes: Up and Running. 3rd ed. – Sebastopol, CA: O'Reilly Media, 2021. – 350 p.

32. Shitole A. S. Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh. Master's Thesis, National College of Ireland, 2022. URL: https://norma.ncirl.ie/5943/

33. Sahu S., Verma P. Energy-Aware Task Scheduling and Load Balancing in Cloud Computing Using AI. International Journal of Parallel, Emergent and Distributed Systems, 2025. URL: https://www.tandfonline.com/doi/abs/10.1080/01969722.2025.2588787

34. Vemula D. R., Gangadharan G. R. Energy-Efficient Resource Allocation in Data Centers Using a Hybrid Evolutionary Algorithm. In: Machine Learning for Intelligent Decision Science. Springer, 2020. – P. 71–92.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ТА МЕРЕЖ

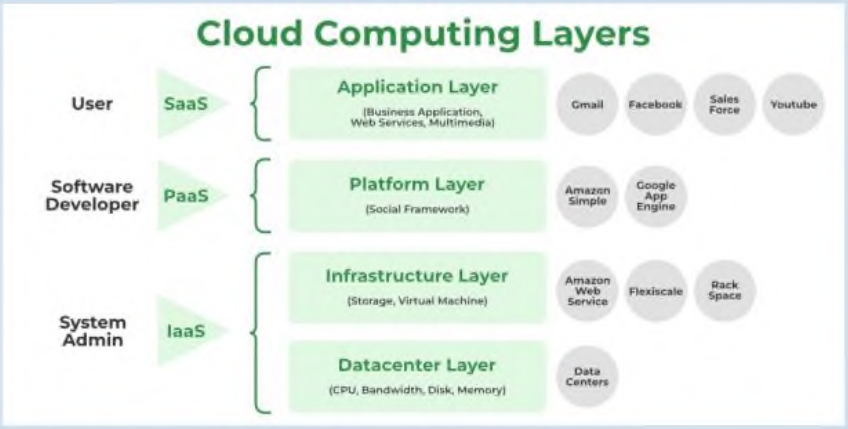
Кваліфікаційна робота

на тему: «Методи та алгоритми балансування навантаження в інфраструктурі хмарних обчислень»

Вадим Вшивков

Керівник: к.т.н., доц. Домрачева Катерина

Актуальність теми



- Зростання хмарного трафіку
- Нерівномірне навантаження
- Ризик порушення SLA

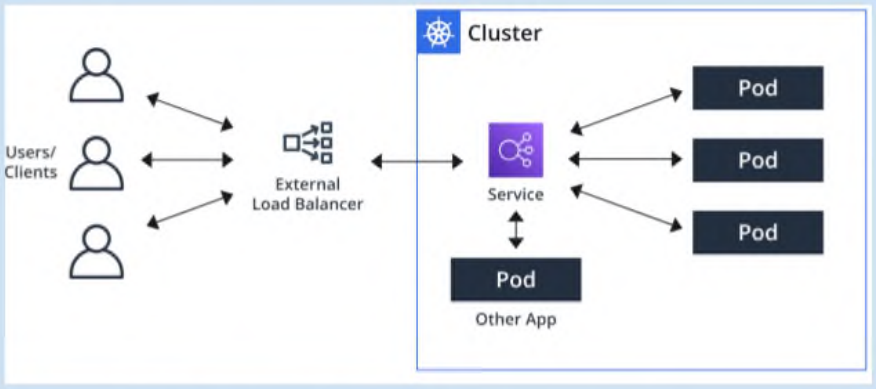
Мета роботи

Мета роботи – підвищити ефективність балансування навантаження в інфраструктурі хмарних обчислень на базі AWS та Kubernetes.

Завдання роботи

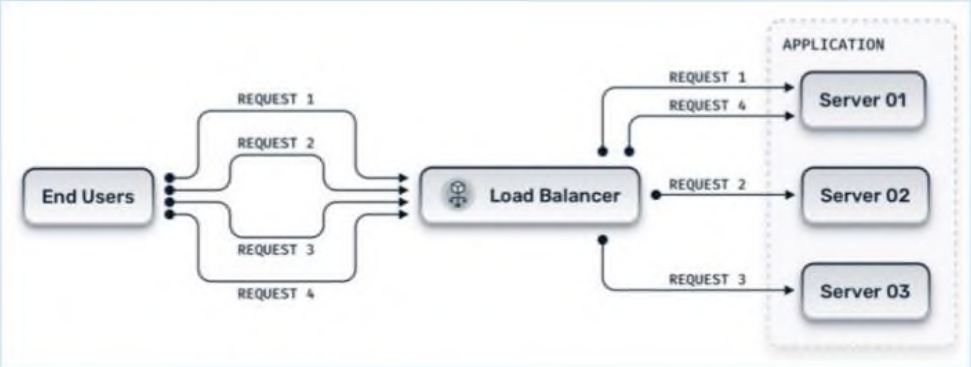
- Аналіз методів і алгоритмів балансування навантаження в хмарних системах
- Проєктування експериментального стенду AWS + Kubernetes з моніторингом (Prometheus, Grafana)
- Реалізація та дослідження сценаріїв навантаження A–D для RR, LC, SR, Envoy adaptive
- Оцінювання алгоритмів за метриками latency (p50, p95), RPS, error rate, SLA
- Формування практичних рекомендацій щодо вибору алгоритмів для типових сценаріїв навантаження

Об’єкт, предмет і база дослідження



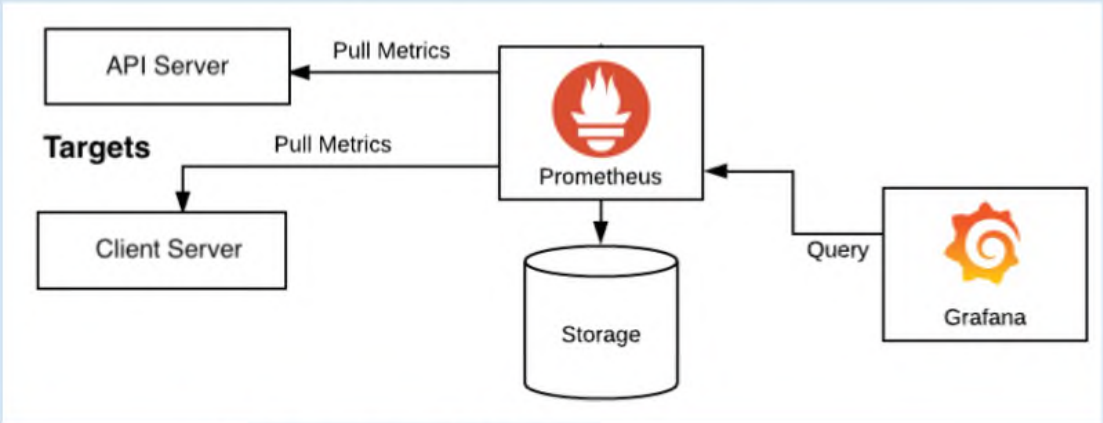
- Об’єкт: процес розподілу навантаження
- Предмет: алгоритми LB в AWS + K8s
- База: AWS, Kubernetes, Prometheus, Grafana

Моделі балансування



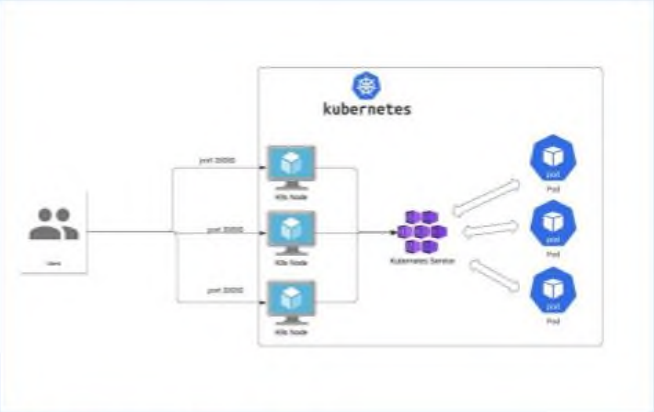
- Статичні: RR, WRR, SR
- Динамічні: LC, SRT, адаптивні

Моніторинг балансування



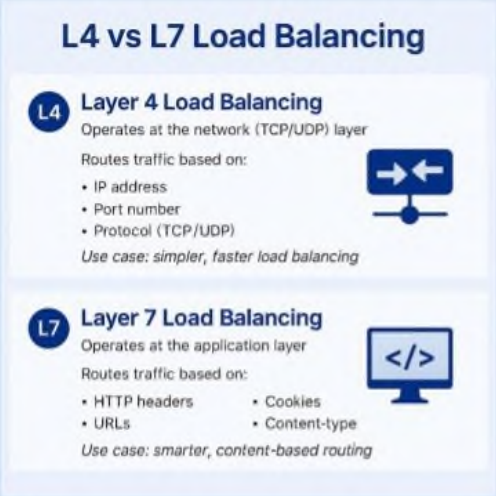
- Prometheus — збір метрик
- Grafana — дашборди
- Метрики для аналізу LB

Експериментальний стенд



- AWS EC2 + кластер Kubernetes
- Ingress / Envoy як L7-балансувальник
- k6 + Prometheus + Grafana

Алгоритми та метрики



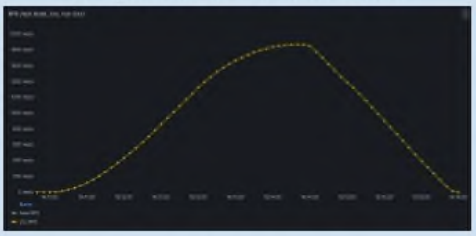
- Метрики: p50, p90, p95, p99
- RPS, error rate, SLA
- error_rate = errors / requests

Експериментальні сценарії A–D

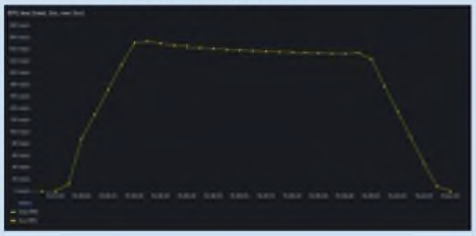
A: рівномірне навантаження



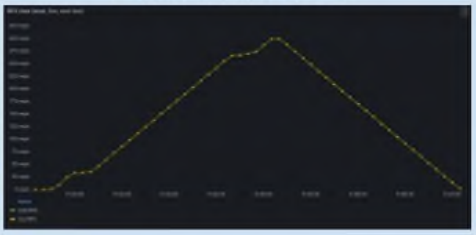
B: зростаюче навантаження



C: пікові сплески



D: деградація вузла



p95 часу відгуку (A–D)

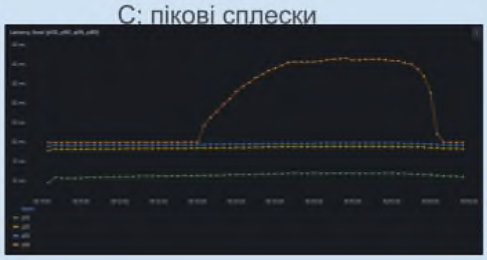
A: рівномірне навантаження



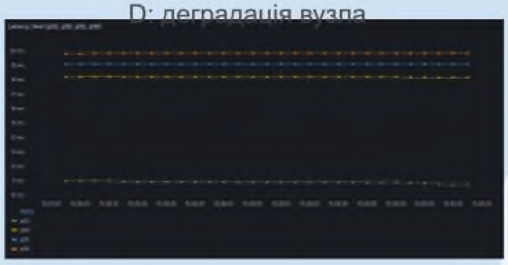
B: зростаюче навантаження



C: пікові сплески



D: деградація вузла



Практичні рекомендації

A: рівномірне навантаження → RR / LC / SR
B: зростаюче навантаження → LC / SR, при піках — Envoy adaptive
C: пікові сплески → Envoy adaptive
D: деградація вузла → Envoy adaptive

Висновки та перспективи

Побудовано стенд AWS + Kubernetes з моніторингом
Порівняно RR, LC, SR, Envoy adaptive у сценаріях A–D
Розроблено рекомендації для вибору алгоритму
Перспективи: ML-підхід до балансування та multi-cloud

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Вадим ВШИВКОВ

Назва: МЕТОДИ ТА АЛГОРИТМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ІНФРАСТРУКТУРІ ХМАРНИХ ОБЧИСЛЕНЬ

Координатор: Катерина ДОМРАЧЕВА

Підрозділ: ННІТ

Коефіцієнт подібності 1:1.3

Коефіцієнт подібності 2:0.1

Тривога: 5

Після аналізу Звіту подібності констатую наступне:

☐ виявлені в роботі запозичення є сумлінними і не мають ознак плагіату. Тому робота визнається самостійною і допускається до захисту;

☐ виявлені в роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на доопрацювання;

☐ виявлені в роботі запозичення є недобросовісними і мають ознаки плагіату або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. У зв'язку з чим, робота не допускається до захисту.

…… ……

Дата *Підпис Наукового керівника*