ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизоване балансування навантаження серверів для рівномірного розподілу запитів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

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

_____	*Сергій КОРЖУК*
(підпис)	*Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Сергій КОРЖУК
Ім'я, ПРІЗВИЩЕ
Керівник: Денис ЖУЖКОВ
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
_____
«_____» _____ 2025р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коржук Сергій Миколайович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Автоматизоване балансування навантаження серверів для рівномірного розподілу запитів
керівник кваліфікаційної роботи Денис ЖУЖКОВ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56
2. Строк подання кваліфікаційної роботи «3» червня 2024р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література; NGINX, HAProxy, Traefik
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Поняття системи балансування навантаження в інформаційних системах
2. Класифікація та принципи роботи алгоритмів балансування навантаження (Round Robin, Least Connections, Least Load, IP Hash)
3. Опис інструментів, які використовуються в схемах балансування та моніторингу сервера
4. Побудова моделі інформаційної системи з реалізацією балансування навантаження
5. Перелік ілюстративного матеріалу: *презентація,схеми маршрутизації*
6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2025	Виконано
2	Обробка матеріалу	23.03-05.04.2025	Виконано
3	Розробка теоретичної частини	06.04-10.04.2025	Виконано
4	Розробка моделі мережі	11.04-20.04.2025	Виконано
5	Висновки за результатами аналізу	21.04-06.05.2025	Виконано
6	Оформлення пояснювальної записки	07.05-08.05.2025	Виконано
7	Розробка презентації	09.05-11.05.2025	Виконано
8	Передзахист	27.05-29.05.2025	Виконано
9	Здача в деканат	27.05-03.06.2025	Виконано

Здобувач вищої освіти _____ Сергій КОРЖУК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Денис ЖУЖКОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 38 стор., 7 рис., 4 табл., 12 джерел.

Мета роботи – Розробка та моделювання системи балансування навантаження для забезпечення стабільної роботи серверів в умовах змінного трафіку

Об'єкт дослідження – Процес розподілу запитів у інформаційних мережах між серверами

Предмет дослідження – Алгоритми, інструменти та технології реалізації балансування навантаження

Короткий зміст роботи: У роботі розглянуто теоретичні та практичні аспекти балансування навантаження в інформаційних мережах. Проведено аналіз сучасних алгоритмів розподілу запитів, зокрема Round Robin, Least Connections, Least Load та IP Hash. Окрему увагу приділено динамічному та адаптивному балансуванню, що враховує поточний стан серверів. Описано принципи роботи популярних інструментів — NGINX, HAProxy і Traefik, а також порівняно їхні можливості та сфери застосування. Змодельовано систему балансування навантаження з використанням обраного інструменту. У ході дослідження оцінено ефективність алгоритмів та доцільність їх застосування для забезпечення надійної роботи сервісів при змінному трафіку.

КЛЮЧОВІ СЛОВА: БАЛАНСУВАННЯ НАВАНТАЖЕННЯ, ІНФОРМАЦІЙНА МЕРЕЖА, NGINX, HAPROXY, TRAEFIC, ROUND RORIN

ЗМІСТ

ВСТУП…….. 8
1. ТЕОРЕТИЧНІ ОСНОВИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ СЕРВЕРІВ …… 9
1.1 Поняття та цілі балансування навантаження……..... 9
1.2 Типи балансувальників: апаратні та програмні……..…… 12
1.3 Мережеві рівні балансування (L4, L7)…… 14
1.4 Критерії ефективності балансування…… 16
2. МЕТОДИ РОЗПОДІЛУ НАВАНТАЖЕННЯ СЕРВЕРІВ …… 19
2.1 Round Robin та його модифікації ……. 19
2.2 Least Connections / Least Load ……..…… 21
2.3 IP Hash, Source Hash та інші детерміновані методи …….. 22
2.4 Динамічне адаптивне балансування (на основі метрик)…… 25
3. ОГЛЯД СУЧАСНИХ РІШЕНЬ……..…….. 29
3.1 Програмні засоби: NGINX, HAProxy, Traefik …….…….. 29
3.2 Хмарні сервіси: AWS ELB, Google Cloud Load Balancing ……. 32
3.3 Автоматичне масштабування як доповнення до балансування …… 34
3.4 Порівняльний аналіз рішень..……..... 36
3.5 Проблеми та обмеження сучасних систем балансування……..... 38
3.6 Забезпечення відмовостійкості та безперервної роботи……. 39
3.7 Перспективи розвитку: автоматизація, AI та самонавчальні системи… 40
ВИСНОВКИ…… 43
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ…….. 45
Додаток А. Презентація……..…..…..…..…..... 47

ВСТУП

В розвитку інформаційних технологій завжди залишається важливим ефективне функціонування серверних технологій. Через зростання кількості даних, запитів, користувачів зростає і необхідність у підвищенні якісної роботи будь-яких веб додатків та сайтів. Для цього потрібно використовувати сучасні і потужні механізми для розподілу трафіку користувачів – для цього і застосовується балансування навантаження. Це є однією з важливіших критерій для веб сервісів. Балансування навантаження трафіку має бути достатньо гнучким для того, щоб реагувати на будь-які зміни в навантаженні серверів, а також мати автоматичний моніторинг, який в будь-якій екстреній ситуації зможе збільшити ліміт навантаження за допомогою підключення додаткових серверів, адже більшість вебсайтів не мають таких технологій, і не витримують різкого зростання трафіку.

У цьому дослідженні буде здійснено ґрунтовний аналіз основних методів розподілу навантаження між серверами, а також розглянуто сучасні програмні рішення та підходи до реалізації цієї технології. До таких рішень належать як апаратні, так і програмні балансувальники навантаження, технології контейнеризації та автоматичного масштабування. Актуальність цієї теми зумовлена стрімким розвитком цифрової економіки, в якій щодня зростає попит на стабільні та масштабовані веб-сервіси, здатні обслуговувати мільйони одночасних користувачів. Саме тому розуміння принципів та практичних аспектів балансування навантаження є необхідним для фахівців у галузі розробки та адміністрування інформаційних систем.

1. ТЕОРЕТИЧНІ ОСНОВИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ СЕРВЕРІВ

1.1 Поняття та цілі балансування навантаження

Балансування навантаження — це процес розподілу вхідних запитів або обчислювальних завдань між кількома серверами з метою оптимального використання їхніх ресурсів. Такий підхід дозволяє уникати перевантаження окремих компонентів системи, забезпечити стабільну роботу сервісів і підтримувати високий рівень доступності для користувачів.

У стандартній моделі функціонування сервісів, коли запити спрямовуються на один сервер, існує ризик перевищення його обчислювальних можливостей. Це, у свою чергу, спричиняє затримки у відповідях або повне припинення обслуговування. Балансувальник навантаження усуває цю проблему, перенаправляючи трафік до кількох серверів, що працюють паралельно, та дозволяє підтримувати безперебійну роботу навіть при високій інтенсивності запитів.

Основні завдання, які вирішує балансування навантаження, можна охарактеризувати так:

– Розподіл трафіку між вузлами. Завдяки цьому запити обробляються всіма доступними серверами, що зменшує ймовірність збоїв через надмірне навантаження.

– Забезпечення стійкості до збоїв. Якщо один із серверів виходить з ладу, система продовжує функціонувати, перенаправляючи запити на інші активні вузли.

– Гнучке масштабування. Інфраструктура може легко адаптуватися до зростання навантаження за рахунок додавання нових серверів без зупинки системи.

– Покращення швидкодії. Розподіл навантаження позитивно впливає на час відповіді та загальну продуктивність сервісу.

– Реалізація додаткового захисту. Багато балансувальників дозволяють використовувати фільтрацію трафіку, керування шифруванням, кешування та інші функції.

З технічної точки зору, балансувальник є посередницьким елементом, який приймає клієнтські запити та скеровує їх на один із доступних серверів, відповідно до заданої логіки або алгоритму. Крім того, балансувальник часто виконує моніторинг стану серверів, щоб уникати перенаправлення запитів на недоступні або перевантажені машини.

З огляду на стрімке зростання вимог до продуктивності цифрових сервісів, системи балансування навантаження відіграють критично важливу роль у сучасних IT-інфраструктурах. Їх впровадження забезпечує гнучкість, стабільність і готовність до масштабування, що є фундаментальними характеристиками ефективної та надійної системи.

Окрім класичного горизонтального масштабування, балансувальники навантаження можуть реалізовувати й більш складні сценарії, як-от розподіл запитів за географічною ознакою або на основі типу запиту. Це особливо актуально для глобальних сервісів, де важливо направляти користувачів до найближчого дата-центру, щоб зменшити затримки. Географічне та контентне балансування дозволяють не лише підвищити продуктивність, а й оптимізувати використання ресурсів.

Ще одним важливим аспектом є інтеграція балансувальників із системами моніторингу та логування. Це дає змогу в реальному часі відслідковувати метрики навантаження, виявляти аномалії та швидко реагувати на інциденти. Наприклад, при виявленні зниження доступності одного з вузлів, балансувальник може автоматично виключити його з пулу активних серверів до відновлення працездатності.

Також слід зазначити роль балансування навантаження в контексті DevOps-практик і CI/CD-процесів. Під час оновлення сервісу балансувальник може здійснювати поступове перенаправлення трафіку на нову версію застосунку (blue-green deployment, canary release), що дозволяє мінімізувати ризики та забезпечити

безперервність надання послуг.

Таким чином, сучасні балансувальники — це не просто інструменти для рівномірного розподілу трафіку, а повноцінні компоненти інфраструктури, які підтримують стратегії високої доступності, продуктивності й автоматизації.

1.2 Типи балансувальників: апаратні та програмні

Системи балансування навантаження можна поділити на дві основні категорії — апаратні та програмні рішення. Обидва підходи мають свої особливості, переваги та недоліки, і вибір між ними залежить від масштабів інфраструктури, бюджету та вимог до продуктивності й гнучкості.

Апаратні балансувальники — це спеціалізовані пристрої, які виконують функцію розподілу трафіку на рівні мережі. Такі пристрої часто використовуються у великих дата-центрах або в критично важливих системах, де потрібна максимальна продуктивність, надійність і захист від відмов. Вони забезпечують високу пропускну здатність, низьку затримку обробки запитів і додаткові функції, як-от фільтрація трафіку, виявлення атак або SSL-термінація.

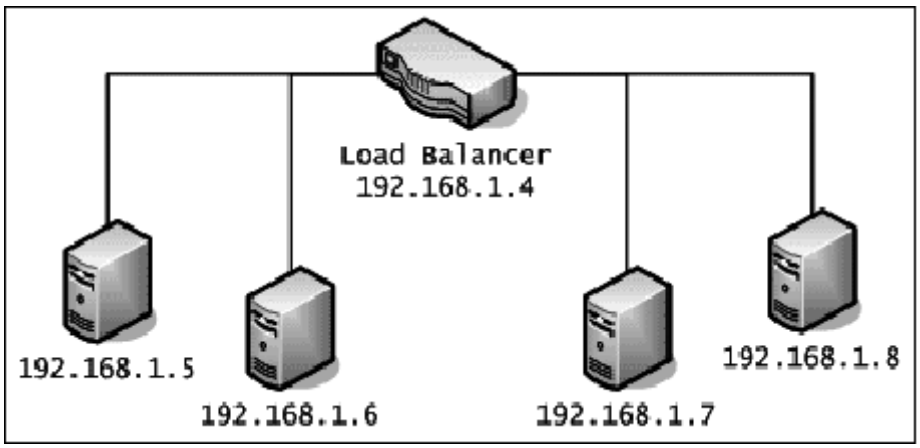


Рисунок 1.1 – Приклад схеми підключення апаратного балансувальника між користувачами та серверним кластером.

Однак апаратні рішення мають низку обмежень: вони коштовні, менш гнучкі щодо налаштувань та масштабування, а також прив’язані до конкретного постачальника. Серед популярних виробників таких рішень можна виділити F5 Networks, Citrix ADC, Cisco тощо.

Програмні балансувальники є гнучкішими й економічно вигіднішими. Вони працюють на звичайних серверах або у віртуальному середовищі, й можуть бути інтегровані у хмарну інфраструктуру. Такі рішення часто підтримують налаштування через конфігураційні файли або інтерфейси керування, що робить їх привабливими для DevOps-команд і хмарних архітектур.

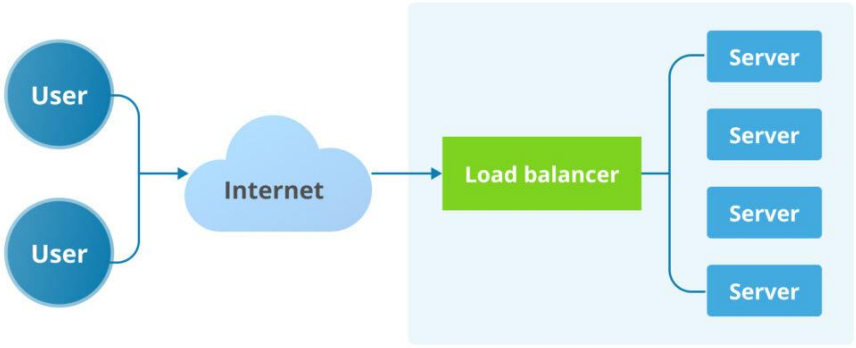


Рисунок 1.2 – Схема використання програмного балансувальника в середовищі мікросервісів або контейнерів.

До найпопулярніших програмних рішень належать NGINX, HAProxy, Traefik, які можуть функціонувати як окремі балансувальники або як частина більших оркестраційних систем (наприклад, Kubernetes Ingress).

Таблиця 1.1 – Порівняльна характеристика апаратних і програмних балансувальників.

Характеристика	Апаратні балансувальники	Програмні балансувальники
Продуктивність	Висока	Залежить від ресурсів
Вартість	Висока	Низька або відсутня (open source)
Масштабованість	Обмежена	Висока
Гнучкість конфігурації	Обмежена	Висока
Залежить від постачальника	Так	Ні (особливо з відкритим кодом)

З огляду на розвиток хмарних обчислень і контейнеризації, програмні балансувальники останніми роками зазнали суттєвої еволюції. Багато сучасних систем підтримують інтеграцію з DevOps-інструментами, що дозволяє автоматизувати процеси маршрутизації, масштабування і безперервного розгортання сервісів. Наприклад, при оновленні версії мікросервісу можна налаштувати програмний балансувальник так, щоб частина трафіку скеровувалась на нову версію, а решта — на стабільну. Це дає змогу безперервно тестувати нові релізи без ризику для всіх користувачів.

Водночас апаратні рішення залишаються незамінними у ситуаціях, де важливі не лише продуктивність, а й сертифікація безпеки, відповідність стандартам або централізоване керування трафіком на фізичному рівні. Наприклад, у банківських установах або дата-центрах, що обслуговують державні ресурси, вимоги до сертифікації FIPS, PCI DSS або ISO можуть змусити обирати саме сертифіковані апаратні рішення, що пройшли відповідні перевірки.

Ще однією відмінністю між апаратними та програмними балансувальниками є принцип оновлення та масштабування. Якщо програмне рішення можна швидко оновити, замінити чи доповнити віртуально, то апаратні пристрої потребують фізичного втручання. Це ускладнює масштабування в реальному часі, особливо у випадках пікового навантаження або під час аварійного перенесення навантаження на інші вузли.

Варто також відзначити, що деякі провайдери та підприємства обирають гібридний підхід — поєднання апаратних і програмних рішень. Такий підхід дає змогу поєднати надійність і високу швидкість апаратного рівня з гнучкістю та автоматизацією програмного. Наприклад, апаратний балансувальник може розподіляти трафік між різними зонами доступності, тоді як програмні — керувати розподілом усередині окремих кластерів або контейнерних хостів.

Таким чином, різниця між апаратними і програмними балансувальниками полягає не лише у типі реалізації, а й у підходах до керування, масштабування, інтеграції в інфраструктуру та вартості.

1.3. Мережеві рівні балансування (L4, L7)

Балансування навантаження може здійснюватися на різних рівнях мережевої моделі OSI, серед яких найчастіше використовують транспортний рівень (L4) та рівень прикладних протоколів (L7). Вибір рівня, на якому виконується розподіл трафіку, визначає глибину аналізу запиту та можливості маршрутизації.

Балансування на рівні L4 (Transport Layer). На цьому рівні трафік розподіляється на основі базових характеристик з'єднання, таких як IP-адреса джерела, порт призначення та протокол (TCP, UDP). Балансувальник не аналізує вміст самого запиту — рішення про маршрутизацію приймається лише на основі мережевих заголовків.

Це забезпечує швидку обробку та низьку затримку, оскільки не потрібно аналізувати додаткові дані. Такий підхід добре підходить для простих служб (FTP, SMTP, VoIP) та ситуацій, де критичною є продуктивність.

Балансування на рівні L7 (Application Layer). У цьому випадку балансувальник аналізує повний вміст HTTP(S)-запиту або іншого прикладного протоколу. Це дозволяє здійснювати інтелектуальний розподіл — наприклад, переспрямовувати запити залежно від URL-шляху, заголовків, параметрів запиту або навіть типу браузера.

L7-балансування активно використовується у веб-застосунках, де є потреба направити різні типи запитів (наприклад, /api та /static) на різні сервери чи мікросервіси.

Таблиця 1.2 – Порівняння L4 і L7 балансування

Параметр	L4 балансування	L7 балансування
Аналіз трафіку	IP/порт, TCP/UDP	HTTP, HTTPS, DNS, FTP тощо
Гнучкість маршрутизації	Обмежена	Висока (URL, cookies, headers)
Продуктивність	Вища	Трохи нижча через аналіз запиту
Підтримка сесій	Обмежена	Можлива (через cookie/persistence)
Типові задачі	FTP, SMTP, VoIP	Веб-додатки, API, REST, GraphQL

Балансування навантаження на рівні L4 зазвичай реалізується за допомогою простих, але ефективних алгоритмів, таких як Round Robin, Least Connections або Source IP Hash. У такій конфігурації балансувальник не витрачає час на аналіз вмісту запиту — він просто переадресовує трафік на доступні сервери відповідно до встановленої логіки. Це дозволяє забезпечити максимальну швидкість обробки, особливо у високонавантажених середовищах, де навіть затримка в кілька мілісекунд може мати значення. Наприклад, у фінансових системах або ігрових платформах, де критична продуктивність, L4 є оптимальним вибором.

Натомість балансування на рівні L7 надає значно більше гнучкості. Завдяки можливості аналізувати заголовки HTTP, тіла запитів та інші параметри, балансувальник може спрямовувати трафік з урахуванням конкретного контексту. Наприклад, запити до /admin можуть направлятись на окремий кластер серверів з посиленою безпекою, тоді як статичні файли з /images — оброблятись окремим сервером або кешуватись. Це дає змогу не лише оптимізувати використання ресурсів, а й покращити масштабованість та безпеку системи.

Ще однією перевагою L7 є підтримка сесій. Завдяки аналізу cookie або інших елементів HTTP-запиту, балансувальник може забезпечити так зване “прилипання сесій” (session stickiness) — коли користувач завжди потрапляє на той самий сервер. Це корисно для додатків, де важливо зберігати контекст або стан користувача на одному сервері, наприклад, у кошику покупок чи панелі адміністратора.

Варто зазначити, що L7-балансування має вищу вимогу до ресурсів, оскільки потребує глибшого аналізу запитів. Проте сучасні програмні рішення та апаратні засоби дозволяють оптимізувати цей процес, використовуючи кешування, асинхронну обробку та розподілені алгоритми. Такі системи здатні обробляти мільйони запитів за секунду, не втрачаючи при цьому гнучкості.

У багатьох сучасних рішеннях обидва підходи комбінуються: спочатку L4-балансування розподіляє запити на певні вузли, а всередині вузлів L7-балансування керує конкретними маршрутами в межах мікросервісної архітектури.

1.4. Критерії ефективності балансування

Оцінка ефективності системи балансування навантаження ґрунтується на ряді технічних та експлуатаційних показників. Правильно налаштований балансувальник має не лише рівномірно розподіляти навантаження між серверами, а й забезпечувати стабільну, надійну та масштабовану роботу всієї інфраструктури.

1. Рівномірність розподілу запитів. Один із головних параметрів — наскільки рівно навантаження розподіляється між серверами. Якщо певні вузли систематично перевантажуються, а інші — майже не використовуються, це вказує на недосконалий алгоритм балансування або неправильні налаштування.

2. Затримка та час відповіді. Ефективне балансування сприяє зменшенню часу обробки запитів. Важливо оцінювати як середню, так і пікову затримку при зростанні кількості запитів. У системах реального часу цей критерій критично важливий.

3. Пропускна здатність. Цей показник відображає, скільки запитів або обсягу даних система здатна обробляти за одиницю часу. Ефективне балансування має забезпечувати максимальну продуктивність без втрат або черг.

4. Стійкість до відмов. Надійна система повинна залишатися працездатною навіть при відмові одного чи кількох серверів. Балансувальник повинен автоматично виявляти такі збої та виключати проблемні вузли з обслуговування до їх відновлення.

5. Масштабованість. Система має легко масштабуватись — як вертикально (заміна серверів на потужніші), так і горизонтально (додавання нових вузлів). Балансувальник повинен динамічно враховувати зміну кількості серверів.

6. Адаптивність до змін навантаження. У реальних умовах навантаження змінюється залежно від часу доби, подій чи активності користувачів. Добре налаштований балансувальник адаптується до змін, використовуючи динамічні алгоритми та моніторинг.

7. Простота адміністрування. Останній, але не менш важливий критерій —

зручність налаштування, моніторингу та обслуговування балансувальника. Хороше рішення повинно мати логічний інтерфейс, підтримку логів, алертів і можливість інтеграції з системами моніторингу (Prometheus, Grafana тощо).

Ще одним важливим критерієм є відновлюваність (recovery) системи після аварій або пікових навантажень. Ефективне балансування повинно не тільки витримувати навантаження під час інцидентів, а й швидко повертати систему до стабільного стану після їх завершення. Наприклад, коли трафік повертається до нормального рівня після сплеску активності, балансувальник має забезпечити поступове зменшення кількості серверів без різких змін і втрати даних сесій. Це особливо актуально в інфраструктурах, де використовується автоматичне масштабування — невдале налаштування може призвести до повторного перевантаження одразу після стабілізації.

Також варто враховувати гнучкість інтеграції з іншими компонентами інфраструктури. Сучасні системи часто складаються з десятків або сотень сервісів, що взаємодіють між собою. Балансувальник має підтримувати різні протоколи, методи аутентифікації, шифрування трафіку та інтеграцію з CI/CD-середовищем. Здатність безперешкодно взаємодіяти з системами обліку, безпеки та логування значно полегшує супровід і масштабування проєкту в майбутньому.

Не менш важливим є спостереження за метриками в реальному часі. Навіть найкращий алгоритм балансування не зможе гарантувати ефективну роботу без постійного моніторингу. Інструменти типу ELK-стек дозволяють візуалізувати трафік, аналізувати помилки, будувати алерти та приймати рішення на основі фактів. У великих системах, де важлива кожна секунда затримки, наявність повноцінного моніторингу — не просто перевага, а обов’язкова вимога.

Крім технічних характеристик, варто розглядати й економічну ефективність системи. Використання надто складних або ресурсомістких алгоритмів може забезпечити високу якість обслуговування, але при цьому значно збільшити витрати на інфраструктуру. Тому при оцінці ефективності важливо знайти баланс між продуктивністю, стійкістю до навантаження та вартістю обслуговування. Наприклад, у деяких випадках достатньо простих методів балансування у

поєднанні з резервними вузлами, замість повноцінної адаптивної системи з машинним навчанням.

Нарешті, слід звернути увагу на поведінку балансувальника при аномаліях у трафіку або нетиповій активності. У випадках DDoS-атак, внутрішніх помилок клієнтського ПЗ або сторонніх інтеграцій, які генерують надмірну кількість запитів, система повинна вміти розпізнавати подібні шаблони і відповідно реагувати: наприклад, обмежувати швидкість запитів, блокувати окремі IP-адреси або тимчасово змінювати маршрути. Таким чином, система балансування виконує не лише технічну, а й частково захисну функцію, забезпечуючи цілісність та безперебійність сервісу.

Ці критерії використовуються як індивідуально, так і в сукупності при виборі балансувальника або при порівнянні кількох рішень. На їх основі можна проводити тестування та оптимізацію системи.

2. МЕТОДИ РОЗПОДІЛУ НАВАНТАЖЕННЯ СЕРВЕРІВ

2.1 Round Robin та його модифікації

Round Robin — один із найпростіших алгоритмів розподілу навантаження, що широко використовується в мережевих системах. Його принцип полягає в послідовному скеруванні вхідних запитів до доступних серверів за циклічним порядком. Припустимо, у системі є три сервери: перший запит надсилається на перший сервер, другий — на другий, третій — на третій, а четвертий знову повертається до першого, і так далі по колу.

Основна перевага цього методу — його простота та низькі обчислювальні витрати. У середовищах із однорідними за продуктивністю серверами та короткочасними запитами він може забезпечити досить ефективний розподіл. Водночас, у випадках, коли сервери мають різну потужність або якщо характер запитів варіюється за складністю й тривалістю, базовий алгоритм може призвести до нерівномірного навантаження.

Щоб компенсувати недоліки класичного Round Robin, використовуються його вдосконалені варіанти.

Weighted Round Robin (зважений метод). Цей підхід дозволяє призначити кожному серверу певну вагу відповідно до його обчислювальних можливостей або доступних ресурсів. Сервер з більшою вагою отримує більше запитів у межах одного циклу, ніж сервер з меншою вагою. Наприклад, якщо одному серверу задано вагу 3, а іншому — вагу 1, перший отримає три запити підряд, а другий — лише один, після чого цикл повториться. Такий механізм забезпечує кращий розподіл ресурсів у гетерогенних середовищах.

Dynamic Round Robin (динамічний підхід). На відміну від фіксованого зважування, динамічний варіант використовує метрики реального часу — наприклад, поточне навантаження, кількість активних з’єднань або середній час

відповіді. Залежно від цих параметрів вага серверів коригується автоматично, що дозволяє більш ефективно адаптуватися до змін у навантаженні.

IP-based Round Robin (на основі IP-адреси) — це варіант модифікації класичного алгоритму Round Robin, який враховує IP-адресу клієнта для визначення послідовності скерування запитів. Такий підхід дозволяє уникати безперервної зміни серверів для кожного нового запиту, забезпечуючи певну сталість маршрутизації. Завдяки цьому можна значно зменшити навантаження на сервери збереження сесійної інформації, а також покращити загальну продуктивність і зручність користувацького досвіду. Такий підхід особливо актуальний для веб-застосунків, де збереження стану взаємодії з клієнтом має вирішальне значення.

Переваги методу Round Robin:

1. Простота реалізації. Не потребує складних обчислень або моніторингу стану серверів;
2. Швидка маршрутизація. Алгоритм майже не має затримок на ухвалення рішення.
3. Добре працює в однорідних середовищах, де всі сервери мають приблизно однакову продуктивність.

Недоліки методу Round Robin:

1. Не враховує реальне навантаження серверів.
2. Може призвести до нерівномірного розподілу в гетерогенних системах (сервери з різною потужністю або швидкістю відповіді).
3. Не підходить для довготривалих або важких запитів (один сервер може бути перевантажений, інші — простоюють).

Алгоритм Round Robin і його модифікації є універсальними та часто застосовуються в системах з розподіленою архітектурою. Вибір конкретного варіанту залежить від вимог до масштабованості, гнучкості та характеру навантаження в цільовій інфраструктурі.

2.2 Least Connections / Least Load

Алгоритми, засновані на оцінці поточного стану серверів, є наступним логічним кроком після простих циклічних методів. Їхня головна ідея полягає не в рівномірному чергуванні запитів, а у врахуванні реального навантаження на кожен із серверів у момент розподілу.

Алгоритм Least Connections (найменша кількість з'єднань) передбачає, що новий запит скеровується на той сервер, який має найменшу кількість активних клієнтських підключень на поточний момент. Це дозволяє більш рівномірно розподіляти навантаження в умовах, коли запити мають нерівномірну тривалість обробки.

Переваги Least Connections:

1. Підходить для систем із довготривалими з'єднаннями (наприклад, HTTP Keep-Alive або WebSocket);
2. Зменшує ймовірність перевантаження окремих серверів.
3. Підвищує загальну стабільність обробки запитів.

Недоліки Least Connections:

1. Не враховує обсяг навантаження, яке кожне з'єднання створює;
2. Вимагає від балансувальника підтримки інформації про активні сесії, що ускладнює реалізацію в деяких середовищах.

Least Load (або Least Resource Usage). Цей підхід розширює ідею алгоритму Least Connections, беручи до уваги не тільки кількість активних з'єднань, а й інші показники навантаження: використання CPU, оперативної пам'яті, мережевого трафіку тощо. Таким чином, вибір сервера для обробки нового запиту ґрунтується на оцінці загального рівня його завантаження.

Типові метрики, що використовуються:

- CPU load average;
- Відсоток використання оперативної пам'яті;
- Кількість активних потоків або процесів;

– Пропускна здатність мережевого інтерфейсу.

Переваги Least Road:

1. Підходить для систем із довготривалими з’єднаннями (наприклад, HTTP Keep-Alive або WebSocket);
2. Найточніше відображає реальний стан серверів.
3. Оптимальний вибір у складних або критичних системах, що потребують високої стабільності.

Недоліки Least Road:

1. Потребує інтеграції з системами моніторингу (наприклад, Prometheus, Zabbix).
2. Залежить від точності та швидкості збору метрик.
3. Може створювати додаткове навантаження на систему балансування.

Таблиця 2.1 – Порівняння методів Least Load та Least Connections

Параметр	Least Connections	Least Load
Основна метрика	Кількість активних з’єднань	Загальне системне навантаження
Простота реалізації	Відносно проста	Складна, потребує додаткових інструментів
Актуальність метрик	Висока	Залежить від швидкості збору метрик
Застосування	Вебсервери, проксі	Високонавантажені та критичні системи

Методи на основі оцінки навантаження дозволяють досягти більшої ефективності та надійності при обробці запитів, особливо в умовах високої варіативності навантаження. У складних інфраструктурах вони часто використовуються як основа або складова частина гібридних стратегій балансування.

2.3 IP Hash, Source Hash та інші детерміновані методи

Детерміновані методи балансування навантаження — це такі підходи, при яких рішення, на який сервер надіслати запит, приймається на основі чіткого

правила (наприклад, IP-адреси користувача). Ці методи часто використовують, коли потрібно, щоб один і той самий користувач завжди потрапляв на той самий сервер. Це називається збереженням сесії або стійкістю сесії.

Один із найпоширеніших способів — IP Hash. У цьому методі IP-адреса користувача "перетворюється" на число за допомогою спеціальної математичної функції (хешування), і це число використовується для вибору сервера.

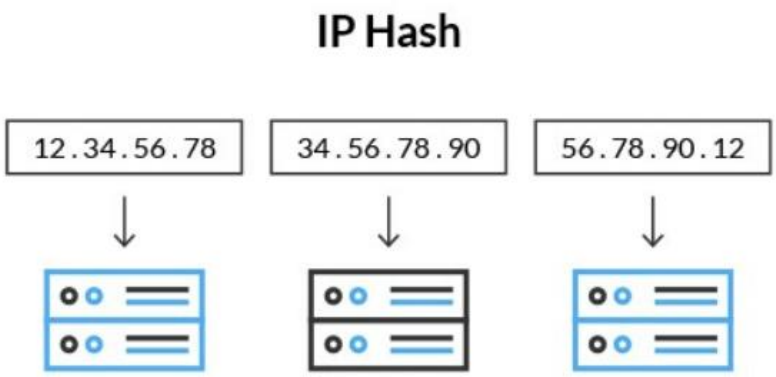


Рисунок 2.1 – Приниципп роботи IP-Hash

Окрім IP-адреси, можна також використовувати інші частини запиту:

- Source port — порт, з якого прийшов запит;
- URI (шлях у запиті) — наприклад, /images/, /api/data;
- Cookies або User-Agent — наприклад, браузер користувача.

Це дозволяє гнучко розділяти навантаження: наприклад, всі запити на /api/ можна направити на одну групу серверів, а запити на /static/ — на іншу.

Consistent Hashing (стійке хешування). Це більш розумна версія IP Hash. У звичайному IP Hash, якщо один сервер відключається або додається новий, — всі клієнти можуть почати направлятися на інші сервери. А стійке хешування дозволяє мінімізувати такі зміни. Його часто використовують у великих розподілених системах, де важливо не втратити збережені сесії чи кеші, наприклад у Redis або Cassandra.

Переваги IP Hash:

1. Простота реалізації;
2. Забезпечує постійне обслуговування користувача одним сервером без додаткових налаштувань;
3. Ідеально підходить для застосунків, які зберігають сесію на сервері.

Недоліки IP Hash:

1. Якщо один сервер виходить з ладу або додається новий — клієнти можуть перепризначитись;
2. Не враховується поточне навантаження на сервери — один може бути перевантажений, інші — вільні.

У контексті детермінованих методів варто окремо виділити ситуації, коли ці підходи особливо ефективні. Наприклад, у банківських системах або CRM-рішеннях, де кожна сесія містить важливу тимчасову інформацію, прив'язку користувача до одного й того ж сервера потрібно зберігати протягом усього періоду взаємодії. У таких випадках навантаження може бути розподілено нерівномірно, проте стабільність та передбачуваність поведінки клієнта мають вищий пріоритет, ніж рівна зайнятість усіх вузлів.

Разом із тим, детерміновані методи мають обмежену гнучкість в умовах змінного або непередбачуваного навантаження. У системах, де активність користувачів нерівномірна — наприклад, один клієнт надсилає 10 запитів на секунду, а інший — 1 запит за хвилину — фіксоване призначення до одного сервера може призвести до перевантаження одного вузла. У таких випадках доцільно комбінувати IP Hash або інші методи з обліком поточного стану системи — наприклад, за допомогою вагового розподілу або черг.

Ще один аспект — стійкість до змін у конфігурації. Як було зазначено, додавання або видалення сервера при класичному IP Hash може спричинити масове “перепризначення” клієнтів, що є небажаним для критичних застосунків. Саме тому в системах високої доступності дедалі частіше застосовується стійке хешування, яке використовує віртуальні вузли для зменшення коливань у маршрутизації. Це особливо актуально для кеш-систем або розподілених баз даних,

де важлива збереженість даних між вузлами.

Детерміновані методи також можуть бути використані для впровадження сегментації трафіку. Наприклад, можна розподіляти запити з мобільних додатків на окремі сервери, тоді як запити з браузера — на інші. Це дозволяє адаптувати інфраструктуру під специфіку різних типів клієнтів і поліпшити загальну ефективність. Подібна логіка іноді вбудовується у балансувальники рівня L7, однак її можна реалізувати і у вигляді правил маршрутизації на основі хеш-функцій.

Нарешті, варто згадати про важливість вибору правильної хеш-функції. Від її якості залежить рівномірність розподілу клієнтів. Недостатньо “випадкова” функція може призводити до того, що велика кількість IP-адрес потрапляє на один і той самий сервер. У сучасних реалізаціях зазвичай використовуються криптографічно стійкі функції або функції з гарною ентропією, які дозволяють зменшити кількість колізій. У деяких випадках система може додатково контролювати баланс, аналізуючи розподіл і вручну налаштовуючи ваги серверів або маскуючи частину IP-адреси.

Методи хешування (IP Hash, Source Hash тощо) — це корисні інструменти для стабільного розподілу користувачів у системі, особливо якщо важливо зберігати їхні сесії. Проте їх краще використовувати разом з іншими методами або в умовах, коли всі сервери мають приблизно однакову продуктивність.

2.4 Динамічне адаптивне балансування (на основі метрик)

У сучасних системах навантаження на сервери може сильно змінюватися залежно від часу доби, днів тижня чи поведінки користувачів. Саме тому виникла потреба у динамічному, тобто гнучкому підході до балансування, де рішення приймаються в режимі реального часу — на основі актуального стану кожного сервера.

На відміну від статичних методів (як Round Robin або IP Hash), адаптивне

балансування постійно аналізує поточну ситуацію: наскільки кожен сервер завантажений, скільки в нього запитів, яка затримка у відповіді тощо. Потім система направляє нові запити туди, де зараз найбільше вільних ресурсів.

Таблиця 2.2 – Метрики, що враховуються

Метрика	Значення
CPU Usage	Скільки % процесора зайнято
RAM Usage	Скільки оперативної пам’яті використано
Network traffic	Який трафік входить / виходить із сервера
Response Time	Середній час відповіді на запит
Active Connections / Threads	Скільки одночасних запитів обробляється

Ці показники збираються спеціальними програмами моніторингу — наприклад, Prometheus**,** Zabbix**,** Grafana Agent — і передаються у систему балансування.

Як працює на практиці:

1. Балансувальник постійно перевіряє, який сервер має менше навантаження;
2. Якщо один із серверів перевантажений — він тимчасово виключається з пулу.
3. Нові запити автоматично перенаправляються на менш зайняті сервери.
4. Коли навантаження стабілізується — сервер знову додається до списку.

Як приклад, ввечері, коли багато користувачів заходить на сайт, система може тимчасово перенести трафік на більш потужні або менш завантажені машини, навіть якщо вранці все працювало інакше.

Переваги:

1. Висока ефективність — завжди використовується найбільш вільний сервер;
2. Стабільна робота навіть при пікових навантаженнях.
3. Можна швидко масштабувати систему при потребі.

Недоліки:

1. Потребує складнішого налаштування та інструментів моніторингу;
2. Необхідна швидка передача метрик — щоб рішення були актуальними.

3. Вимагає добре організованої інфраструктури (наприклад, хмара або Kubernetes).

Адаптивне балансування на основі метрик — це сучасний і ефективний спосіб підтримувати стабільність системи в умовах змінного навантаження. Воно особливо корисне для великих проектів, де кількість запитів може швидко змінюватися, і важливо забезпечити якісне обслуговування для всіх користувачів.

Адаптивне балансування тісно пов’язане з концепцією спостережливістю — здатністю системи не просто відображати поточні дані, а й надавати повну картину її стану в будь-який момент часу. Це передбачає використання не лише базових метрик, а й складніших індикаторів, таких як частота помилок, середній час життя запиту, нестандартна поведінка користувачів чи несподівані «сплески» трафіку. Усе це дозволяє балансувальнику приймати більш обґрунтовані рішення та виявляти потенційні проблеми ще до того, як вони вплинуть на користувача.

Окремо варто згадати, що динамічні алгоритми можуть бути як досить простими — наприклад, на основі лише одного параметра (завантаженість процесорів), так і складними — з урахуванням багатьох змінних, пріоритетів, і навіть прогнозування майбутнього навантаження. Деякі сучасні рішення застосовують елементи машинного навчання, щоб «навчитися» характерним патернам у використанні ресурсу та заздалегідь підготувати резерви. Наприклад, якщо щодня о 18:00 система відчуває пік навантаження, балансувальник може ще за годину активувати додаткові вузли.

Завдяки тісній інтеграції з платформами на зразок Kubernetes, адаптивне балансування може працювати на рівні окремих подів або мікросервісів. Це особливо зручно для архітектур, де кожен сервіс обробляє різні типи запитів або має різну критичність. Наприклад, запити до платіжного модуля можуть отримувати вищий пріоритет, а інші — оброблятись у звичайному режимі. У такій системі балансувальник стає частиною загального механізму SLA-контролю, дотримуючись вимог до якості обслуговування.

Нарешті, варто зазначити, що впровадження адаптивного балансування — це не лише технічний, а й організаційний крок. Щоб така система працювала надійно,

необхідно узгодити роботу декількох компонентів: сервісів моніторингу, балансувальників, хмарної платформи, команд DevOps. Крім того, варто передбачити ситуації, коли метрики можуть бути неточними або затримуватись — у таких випадках потрібно мати резервні алгоритми (наприклад, fallback на Round Robin), щоб уникнути простоїв.

3. ОГЛЯД СУЧАСНИХ РІШЕНЬ

3.1 Програмні засоби: NGINX, HAProxy, Traefik

Для реалізації балансування навантаження найчастіше використовують програмні інструменти, які працюють як посередники між користувачем і серверами. Вони отримують вхідні запити, аналізують їх і направляють на відповідний сервер згідно з обраною логікою. Найвідоміші з таких рішень — це NGINX, HAProxy та Traefik.

NGINX — це один із найпопулярніших веб-серверів, який також виконує роль балансувальника навантаження. Його перевага — швидкість, простота налаштування та підтримка як базових, так і складніших методів балансування (Round Robin, IP Hash, Least Connections тощо).

NGINX добре підходить для проєктів, де потрібна швидка і стабільна робота без складної конфігурації. Його часто використовують не тільки як балансувальник, а й як зворотний проксі, кешуючий сервер або навіть статичний веб-сервер. Завдяки своїй простоті, він ідеально підходить для невеликих та середніх проєктів. Також він легко масштабується, тож при зростанні навантаження його можна без проблем використовувати у складніших інфраструктурах.



Рисунок 3.1 – Схема роботи NGINX

HAProxy — це спеціалізований інструмент для балансування навантаження, який часто використовують у великих проєктах. Він підтримує високі навантаження, дозволяє детально налаштовувати логіку балансування та надає зручний веб-інтерфейс для моніторингу.

HAProxy зазвичай обирають тоді, коли потрібна максимальна контрольованість. Наприклад, у фінансових системах або великих платформах, де важливо точно контролювати, як розподіляються запити, відстежувати статистику, будувати звіти, HAProxy надає всі ці можливості. Його конфігурація може бути складнішою, ніж у NGINX, але це компенсується гнучкістю і підтримкою великої кількості функцій — наприклад, балансування на рівні L4 і L7 одночасно, SSL-термінація, логування в реальному часі.

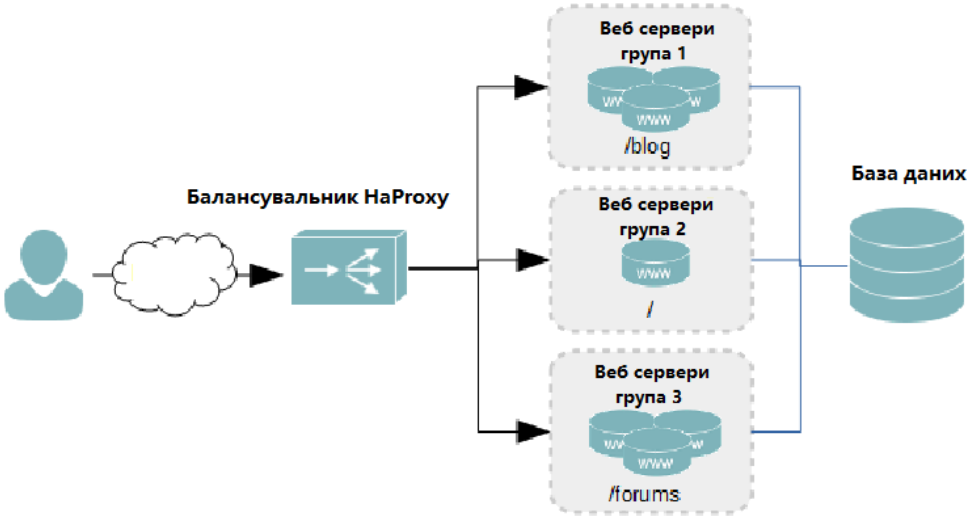


Рисунок 3.2 – Приклад роботи HAProxy

Traefik — відносно новий балансувальник, який створено спеціально для хмарних середовищ та контейнеризованих додатків (наприклад, у Docker або Kubernetes). Його головна особливість — автоматичне виявлення сервісів і проста інтеграція з популярними платформами.

Traefik, на відміну від попередніх двох, спочатку розроблявся з урахуванням контейнеризації. Тому він дуже добре взаємодіє із Docker, Kubernetes та іншими сучасними платформами. Основна його перевага — автоматизація. Traefik сам виявляє нові сервіси і починає обслуговувати їх без перезавантаження конфігурації. Це дуже зручно в динамічних середовищах, де сервіси постійно додаються, оновлюються або видаляються.

Ще одна корисна особливість Traefik — це вбудована підтримка автоматичної генерації та оновлення SSL-сертифікатів. Це значно полегшує налаштування безпечного з'єднання, особливо в невеликих або тестових проєктах. Завдяки простому веб-інтерфейсу або API, адміністратори можуть швидко контролювати стан усіх сервісів, перевіряти трафік і реагувати на збої.

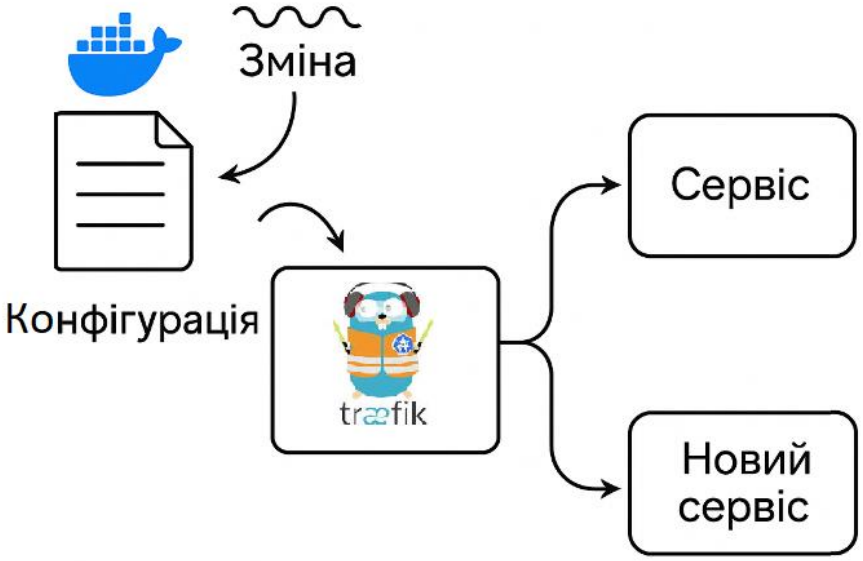


Рисунок 3.3 – Приниципадаптації Traefik при підключенні нових сервісів

На схемі видно, як Traefik вловлює зміни в конфігурації (наприклад, додавання нового сервісу в Docker), і автоматично починає на нього спрямовувати запити.

Усі ці три інструменти — NGINX, HAProxy і Traefik — виконують схожу функцію, але орієнтовані на різні потреби. Вибір між ними залежить від типу проєкту, масштабів, потреб у гнучкості й автоматизації. У деяких випадках їх навіть комбінують. Наприклад, HAProxy може працювати як зовнішній балансувальник, а всередині кожного вузла — використовуватиметься NGINX або Traefik для внутрішнього маршрутизації. Такий підхід дозволяє використовувати сильні сторони кожного з інструментів залежно від конкретних задач

3.2 Хмарні сервіси: AWS ELB, Google Cloud Load Balancing

З розвитком хмарних технологій з'явилася потреба у спеціалізованих рішеннях балансування навантаження, які б були інтегровані безпосередньо в хмарну інфраструктуру. Такі сервіси мають низку переваг: автоматичне

масштабування, простоту інтеграції з іншими сервісами платформи та високий рівень доступності.

Amazon Web Services (AWS) пропонує Elastic Load Balancing (ELB) як масштабоване рішення для автоматичного розподілу вхідного трафіку між декількома цільовими інстансами (EC2, Lambda, контейнерами тощо). ELB працює на різних рівнях моделі OSI:

Application Load Balancer (ALB) — працює на рівні L7, оптимізований для HTTP/HTTPS;

Network Load Balancer (NLB) — L4-рівень, орієнтований на низьку затримку;

Gateway Load Balancer (GWLB) — для глибокої інтеграції з мережевими пристроями типу firewall або IDS/IPS.

Ці типи можуть бути об'єднані для складніших архітектур.

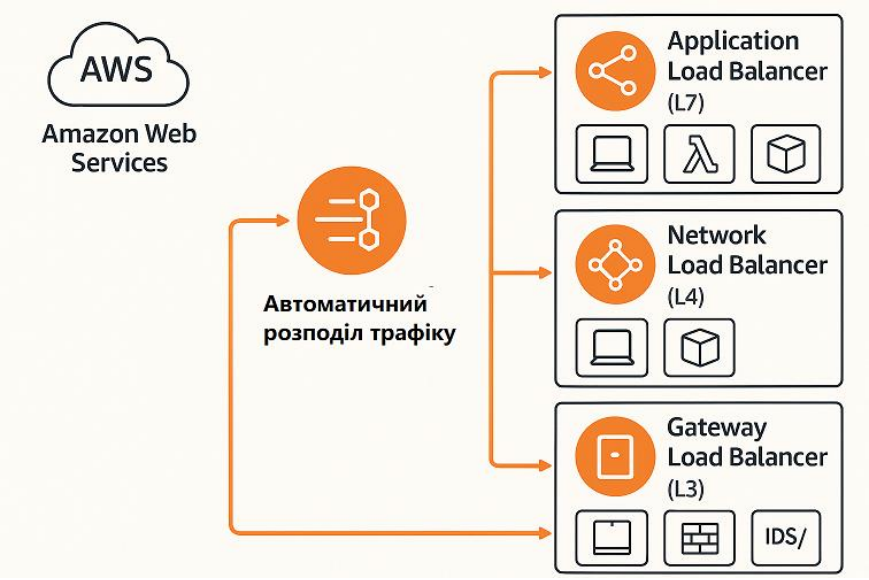


Рисунок 3.4 – Архітектура розподілу трафіку за допомогою AWS ELB

Особливістю цього сервісу є його глибока інтеграція з інфраструктурою Google — він може напряму взаємодіяти з Kubernetes Engine, Cloud Run, App Engine тощо.

Окрім типів, ELB забезпечує автоматичну діагностику, шифрування трафіку, а також інтеграцію з AWS Auto Scaling. Це дозволяє автоматично додавати або

видаляти інстанси в залежності від навантаження.

Google Cloud також пропонує власне рішення — Google Cloud Load Balancing, яке є повністю керованим і глобальним. Це означає, що балансувальник може обслуговувати клієнтів з усього світу за допомогою єдиного IP-адресу, розподіляючи трафік на основі геолокації, навантаження або пріоритету.

Основні типи балансування в GCP:

– *HTTP(S) Load Balancer (L7)* — для вебзастосунків;

– *TCP/SSL Load Balancer* — для нетипових сервісів або приватних мереж;

– *Internal Load Balancer* — для трафіку всередині VPC;

– *Global Load Balancer* — забезпечує доступність на глобальному рівні з використанням anycast-адрес.

Особливістю цього сервісу є його глибока інтеграція з інфраструктурою Google — він може напряму взаємодіяти з Kubernetes Engine, Cloud Run, App Engine тощо.

Таким чином, обидва хмарні сервіси надають потужні можливості для автоматизованого балансування навантаження, проте відрізняються підходами до інтеграції, глобальності й масштабування.

3.3 Автоматичне масштабування як доповнення до балансування

Автоматичне масштабування — це процес динамічного збільшення або зменшення кількості серверних ресурсів залежно від поточного навантаження на систему. Його роль у системах балансування навантаження стає дедалі важливішою, особливо в хмарних та мікросервісних архітектурах.

Балансування навантаження саме по собі вирішує проблему розподілу вхідного трафіку між доступними ресурсами. Проте, коли кількість запитів значно перевищує можливості наявних серверів, балансувальник просто не має куди направляти трафік. Тут і вступає в дію автомасштабування, який додає нові

екземпляри серверів, коли навантаження перевищує встановлені пороги.

У типовій інфраструктурі масштабування та балансування працюють у зв’язці. Коли новий сервер запускається автоматично, балансувальник (NGINX, HAProxy або хмарний — типу AWS ELB) додає його до списку доступних вузлів. Таким чином, балансування залишається ефективним навіть при зміні кількості обчислювальних ресурсів у реальному часі.

Цей підхід особливо цінний для сценаріїв з різкою зміною навантаження — наприклад, під час флешмобів у соціальних мережах, великих розпродажів чи запуску нових продуктів. Автоматичне масштабування забезпечує стабільну роботу сервісу без ручного втручання адміністратора. Основні типи:

Горизонтальне масштабування — додає нові вузли (сервери або контейнери).

Вертикальне масштабування — збільшує ресурси вже існуючих вузлів (CPU, RAM).

У більшості випадків у хмарних середовищах застосовується саме горизонтальне масштабування, оскільки воно краще підходить для контейнеризованих або мікросервісних додатків.

Хоча автоматичне масштабування дає змогу значно підвищити адаптивність системи, воно не завжди підходить для систем з дуже коротким життєвим циклом запитів, де затримка запуску нових інстансів грає критичну роль. Крім того, неправильна конфігурація порогів масштабування може призвести до «гойдалок» — надмірного збільшення й зменшення кількості серверів.

Однією з ключових переваг автоматичного масштабування є економія ресурсів. У хмарному середовищі, де обчислювальні потужності оплачуються залежно від використання, можливість зменшити кількість активних серверів у періоди низького навантаження дозволяє значно знизити витрати. Наприклад, уночі або у вихідні, коли кількість користувачів зменшується, система самостійно вимикає непотрібні екземпляри серверів. Коли ж навантаження зростає — додаткові сервери знову автоматично вмикаються в роботу.

Сам механізм масштабування зазвичай працює за заздалегідь визначеними

правилами. Це можуть бути такі показники, як завантаження процесора, використання оперативної пам’яті, кількість одночасних підключень або середній час відповіді системи. Наприклад, якщо завантаження процесора перевищує 75% протягом кількох хвилин, система додає ще один сервер до пулу. А якщо навантаження стабілізується — цей додатковий сервер автоматично вимикається.

Сучасні інфраструктури, як-от ті, що побудовані на контейнерах, підтримують автоматичне масштабування на декількох рівнях. Окремі елементи програми можуть масштабуватись незалежно один від одного, що дає змогу гнучко реагувати на різні типи навантаження. Наприклад, частина системи, яка обробляє зображення або відео, може масштабуватись незалежно від тієї, що працює з базами даних чи обробкою запитів користувачів.

Масштабування тісно пов’язане з балансуванням навантаження: нові сервери після запуску мають автоматично підключатися до балансувальника, щоб він міг рівномірно розподіляти запити між усіма доступними вузлами. Цей процес зазвичай автоматизується: система слідкує за тим, які сервери активні, і оновлює список доступних для обробки трафіку. Таким чином, навантаження рівномірно розподіляється навіть у динамічному середовищі, де кількість серверів постійно змінюється.

Однак автоматичне масштабування має і свої обмеження. Якщо запуск нового сервера займає багато часу (наприклад, кілька десятків секунд), це може бути проблемою для систем, де критично важлива швидкість реакції. Тому для складних сервісів часто застосовують комбінований підхід — наприклад, фіксовану мінімальну кількість постійно працюючих серверів та динамічну групу, яка активується лише при потребі.

3.4 Порівняльний аналіз рішень

У сучасній IT-інфраструктурі навантаження на сервери зростає щодня, а

вимоги до надійності, масштабованості та швидкодії сервісів стають дедалі жорсткішими. Це змушує інженерів ретельно підходити до вибору інструментів для балансування навантаження, враховуючи як функціональні можливості, так і контекст конкретної системи. Серед основних груп рішень виділяють локальні програмні балансувальники, спеціалізовані хмарні сервіси, а також більш гнучкі рішення, орієнтовані на мікросервіси та Kubernetes-середовища.

Вибір конкретного рішення багато в чому залежить від типу середовища. У традиційній архітектурі, де сервіси розгортаються на фізичних чи віртуальних серверах у межах дата-центру, часто використовуються програмні балансувальники. Вони забезпечують гнучке конфігурування, підтримують різні рівні протоколів (L4, L7), і дають змогу здійснювати тонке налаштування логіки маршрутизації.

Натомість у хмарних платформах головний акцент робиться на автоматизацію, інтеграцію з іншими сервісами та зменшення ручної роботи. Такі сервіси, як AWS ELB або Google Cloud Load Balancing, автоматично виявляють доступні екземпляри, інтегруються з системами моніторингу й масштабування, та мінімізують ризики відмови. Вони надають інтерфейси для спостереження за трафіком і пропонують глобальний розподіл трафіку без необхідності вручну керувати маршрутами.

У мікросервісних архітектурах особливої популярності набули балансувальники, вбудовані у систему сервісної сітки. Traefik, Istio чи Linkerd не просто розподіляють трафік — вони також забезпечують безпеку (мутуальна TLS-автентифікація), збір телеметрії, трасування та динамічне оновлення маршрутів. Це дає змогу будувати системи з високою адаптивністю до змін, що важливо в умовах CI/CD-процесів.

Підхід до маршрутизації. Ще один аспект, який варто враховувати при аналізі — це логіка прийняття рішень при маршрутизації. Наприклад, NGINX дозволяє реалізовувати складні сценарії через конфігураційні файли та LUA-скрипти. HAProxy підтримує різні політики балансування, а також детальні перевірки. Це дозволяє адаптувати балансування до особливостей навантаження —

наприклад, направляти частину трафіку на менш завантажені сервери.

У свою чергу, хмарні сервіси часто абстрагують цю логіку від користувача, пропонуючи спрощений, але надійний механізм. Це спрощує адміністрування, але водночас обмежує контроль над деталями. Наприклад, в Google Cloud Load Balancing вибір між регіонами, зонами та окремими екземплярами відбувається автоматично на основі алгоритмів, які недоступні для прямого налаштування.

У випадках аварій або збоїв важливо, щоб балансувальник міг швидко відреагувати. Тут програми типу HAProxy або Traefik надають більше контролю: можна вказати резервні вузли, періодичність перевірок, правила таймаутів тощо. Хмарні ж сервіси беруть на себе відповідальність за відмовостійкість: вони розгорнуті на інфраструктурі з вбудованим резервуванням та геореплікацією.

Це дозволяє системам у хмарі працювати навіть при втраті одного з дата-центрів або збої мережевих компонентів, тоді як локальні балансувальники зазвичай потребують ручного втручання або додаткових налаштувань кластера високої доступності.

Важливо розуміти, що немає універсального рішення для балансування — кожен інструмент вирішує специфічне коло задач. У деяких випадках доцільно навіть комбінувати кілька балансувальників одночасно: наприклад, використовувати зовнішній хмарний сервіс для маршрутизації на рівні регіонів, а всередині кожного — окремий локальний балансувальник для управління внутрішнім трафіком.

3.5 Проблеми та обмеження сучасних систем балансування

Попри активний розвиток технологій балансування навантаження, ці системи залишаються вразливими до ряду технічних і організаційних проблем. Однією з ключових є затримки в обробці запитів, які виникають через перевантаження балансувальників або недостатньо оптимальні алгоритми маршрутизації. У

великомасштабних середовищах навіть незначні затримки можуть накопичуватися, негативно впливаючи на загальну продуктивність сервісу.

Ще одним важливим викликом є поява так званих «одиничних точок відмови». Якщо балансувальник виходить з ладу, доступ до всіх серверів за ним може бути втрачено. У таких випадках втрачається не лише доступність, а й довіра користувачів. Для зниження ризиків впроваджуються дублюючі системи або кластери балансувальників, однак це підвищує складність інфраструктури та потребує додаткових витрат.

Масштабування — ще одна сфера, де проявляються обмеження. І хоча горизонтальне масштабування (додавання нових вузлів) є популярною стратегією, деякі алгоритми балансування не здатні ефективно враховувати динаміку змін у реальному часі. Наприклад, простий метод Round Robin може однаково розподіляти запити, ігноруючи різницю в навантаженні між серверами, що призводить до дисбалансу.

У сфері безпеки балансувальники можуть бути ціллю для атак типу DoS або навіть використовуватися для прихованого перенаправлення трафіку. Без належного захисту вони можуть стати вразливим місцем у ланцюгу побудови системи. Проблема також ускладнюється у випадках, коли балансування відбувається на прикладному рівні (L7), оскільки тут залучаються більш складні механізми аналізу запитів. Такі ситуації вимагають впровадження більш гнучких і самонавчальних алгоритмів, здатних реагувати в реальному часі.

3.6 Забезпечення відмовостійкості та безперервної роботи

Стабільна та безперебійна робота сервісів — один із головних пріоритетів у сучасних мережевих інфраструктурах. У разі виходу з ладу навіть одного компонента можуть виникнути суттєві збої, що впливають на бізнес-процеси, безпеку та взаємодію з користувачами. Саме тому впровадження механізмів

відмовостійкості в системах балансування навантаження є необхідним, а не опціональним рішенням.

Одним з основних підходів до забезпечення надійності є дублювання балансувальників. У цьому випадку один пристрій працює як активний, а інший – у режимі очікування (standby), готовий перебрати на себе функціональність у разі збою. Існують також конфігурації, де обидва пристрої працюють одночасно, ділячи між собою навантаження, забезпечуючи як відмовостійкість, так і підвищену продуктивність.

Ще однією поширеною практикою є георезервування — створення кількох незалежних дата-центрів, розташованих у різних географічних регіонах. Такий підхід дозволяє переадресовувати трафік на альтернативний центр обробки даних у разі аварії або катастрофічного збою в основному. Для цього використовуються глобальні системи балансування, які динамічно перенаправляють запити на доступний вузол.

Нарешті, важливим елементом підтримки безперервності є інтеграція з системами моніторингу. Такі інструменти, як Prometheus або Zabbix, дозволяють оперативно виявляти проблеми та автоматизовано переключати навантаження на резервні вузли ще до того, як користувачі відчують неполадки.

Таким чином, сучасний підхід до побудови стійких до збоїв балансувальних систем передбачає комбінацію різних стратегій — дублювання, кластеризацію, георезервування та використання хмари — кожна з яких відіграє важливу роль у забезпеченні безперервної роботи сервісів.

3.7 Перспективи розвитку: автоматизація, AI та самонавчальні системи

У сучасному світі зростаючої складності IT-інфраструктур традиційні підходи до балансування навантаження поступово поступаються місцем автоматизованим та інтелектуальним рішенням. Зі збільшенням обсягу даних,

кількості користувачів та мінливості навантаження все більшої актуальності набуває застосування штучного інтелекту, машинного навчання і концепцій AIOps у сфері управління трафіком.

Один з ключових напрямів розвитку — це використання алгоритмів машинного навчання для адаптивного балансування. Замість фіксованих правил (наприклад, Round Robin), самонавчальні системи можуть аналізувати історичні дані про навантаження, активність користувачів, мережеві затримки, час відповіді серверів та інші метрики, щоб самостійно приймати рішення про розподіл запитів у реальному часі.

Такі підходи дають змогу не лише оптимізувати використання ресурсів, але й автоматично адаптуватися до змін — наприклад, у разі пікового навантаження або виникнення «гарячих точок» у трафіку. Системи з підтримкою ML здатні передбачати потенційні проблеми, аналізуючи тренди та аномалії, і діяти на випередження, запобігаючи зниженню продуктивності.

Іншою важливою тенденцією є інтеграція балансувальників з системами моніторингу та спостереження. Використання рішень на кшталт Prometheus, Grafana або Datadog дозволяє не лише візуалізувати стан систем у реальному часі, а й автоматично передавати дані в механізми прийняття рішень. Завдяки цьому балансування стає частиною загального циклу DevOps — динамічного, автоматизованого й контрольованого.

Також активно розвивається концепція AIOps (Artificial Intelligence for IT Operations) — підхід, що поєднує AI, великі дані й автоматизацію для оптимізації процесів управління IT. У контексті балансування навантаження AIOps може забезпечувати виявлення причин збоїв, прийняття рішень на основі контексту та навіть автоматичне масштабування систем залежно від прогнозованого попиту.

Поява таких інструментів дає можливість створювати самодостатні системи, які не лише реагують на події, а й приймають проактивні рішення — наприклад, переведення трафіку в інший дата-центр ще до того, як основний вийде з ладу.

У майбутньому ми можемо очікувати злиття рішень для балансування з іншими компонентами інфраструктури — контейнерними оркестраторами

(наприклад, Kubernetes), CI/CD-пайплайнами та автоматизованими системами керування політиками доступу.

Отже, перспективи розвитку систем балансування навантаження тісно пов’язані з автоматизацією, інтелектуальною обробкою даних і глибокою інтеграцією з іншими компонентами сучасних IT-систем. Такий підхід відкриває нові горизонти для створення гнучких, самонавчальних і стійких до збоїв інфраструктур.

ВИСНОВКИ

У процесі дослідження теми автоматизованого балансування навантаження серверів було виявлено, що рівномірний розподіл запитів у сучасних розподілених системах є критично важливим для забезпечення стабільної роботи, високої доступності та масштабованості веб-сервісів. Усе частіше системи працюють у режимі пікового навантаження, обробляють велику кількість одночасних з'єднань, і саме балансування є тією ланкою, що дозволяє підтримувати працездатність і запобігати відмовам.

Перш за все, було сформульовано поняття балансування навантаження, його основні цілі та функціональне значення в інфраструктурі. Також було розглянуто апаратні та програмні рішення, що застосовуються у цій сфері. Хоча апаратні балансувальники і мають високу продуктивність, програмні інструменти забезпечують гнучкість і легкість інтеграції у вже наявні архітектури, особливо у випадках з мікросервісами та контейнерними середовищами.

Особливу увагу було приділено різним алгоритмам розподілу навантаження, зокрема: Round Robin, Least Connections, IP Hash, а також динамічним методам, які базуються на аналізі поточних метрик. Кожен з методів має свою специфіку використання, і оптимальний вибір залежить від особливостей застосунку, типу навантаження та очікуваної поведінки клієнтів.

Окремий розділ був присвячений сучасним інструментам і технологіям: як відкритим програмним засобам (NGINX, HAProxy, Traefik), так і хмарним сервісам (AWS ELB, Google Cloud Load Balancing). Важливим доповненням до класичного балансування є автоматичне масштабування, яке дозволяє динамічно змінювати кількість обчислювальних ресурсів у системі відповідно до навантаження. Це суттєво підвищує ефективність роботи серверів та оптимізує витрати на інфраструктуру.

Порівняльний аналіз рішень показав, що не існує універсального підходу, який був би найкращим для всіх ситуацій. Ефективність обраного методу чи

інструменту значною мірою залежить від характеру системи, вимог до її масштабованості, очікуваного трафіку, обмежень безпеки та бюджету.

Таким чином, можна зробити висновок, що автоматизоване балансування навантаження — це не лише технічний компонент, а й невід’ємна частина загальної архітектурної стратегії системи. Його правильне впровадження дозволяє підвищити якість обслуговування користувачів, зменшити час відгуку, уникнути перевантаження та забезпечити безперервність роботи навіть у найскладніших умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Load balancing - Wikipedia. URL: https://en.wikipedia.org/wiki/Load_balancing_(computing) (дата звернення: 10.05.2025).

2. Layer 4 vs Layer 7 Load Balancing: What's the Difference? | Kemp. Kemp Technologies. URL: https://kemptechnologies.com/load-balancer/load-balancing-techniques/layer-4-vs-layer-7/ (дата звернення: 10.05.2025).

3. HAProxy – The Reliable, High Performance TCP/HTTP Load Balancer. HAProxy Technologies. URL: https://www.haproxy.org/ (дата звернення: 10.05.2025).

4. NGINX Load Balancing. NGINX Documentation. URL: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/ (дата звернення: 11.05.2025).

5. What is Round Robin DNS Load Balancing? Cloudflare. URL: https://www.cloudflare.com/learning/performance/what-is-round-robin-dns/ (дата звернення: 11.05.2025).

6. Least Connections Load Balancing Method – Citrix. Citrix Documentation. URL: https://docs.citrix.com/en-us/citrix-adc/current-release/load-balancing/load-balancing-algorithms/least-connections.html (дата звернення: 14.05.2025).

7. Load balancing algorithms | DigitalOcean. DigitalOcean Docs. URL: https://docs.digitalocean.com/products/load-balancer/how-to/choose-algorithm/ (дата звернення: 17.05.2025).

8. AWS Elastic Load Balancing. Amazon Web Services. URL: https://aws.amazon.com/elasticloadbalancing/ (дата звернення: 14.05.2025)

9. Google Cloud Load Balancing. Google Cloud Platform. URL: https://cloud.google.com/load-balancing (дата звернення: 16.05.2025).

10. Traefik Documentation - Modern HTTP reverse proxy and load balancer. Traefik. URL: https://doc.traefik.io/traefik/ (дата звернення: 17.05.2025)

11. Load Balancing Architecture and Algorithms – GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/load-balancing-architecture-and-algorithms/ (дата звернення: 17.05.2025).

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій
Кафедра Комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
на тему: «Автоматизоване балансування навантаження серверів для рівномірного розподілу запитів»

Виконав студент групи КІД-41
Коржук Сергій Миколайович

Керівник кваліфікаційної роботи:
Жужков Денис Ігорович, викладач кафедри КІ

Мета — дослідити принципи роботи автоматизованого розподілу запитів на сервери.

Об`єкт дослідження – методи балансування навантаження в інформаційних системах.

Предмет дослідження – алгоритми та інструменти, що забезпечують розподіл запитів між серверами.

Актуальність обраної теми

В розвитку інформаційних технологій завжди залишається важливим ефективне функціонування серверних технологій. Через зростання кількості даних, запитів, користувачів зростає і необхідність у підвищенні якісної роботи будь-яких веб додатків та сайтів. Для цього потрібно використовувати сучасні і потужні механізми для розподілу трафіку користувачів – для цього і застосовується балансування навантаження. Це є однією з важливіших критерій для веб сервісів.

3

Апаратні та програмні балансувальники трафіку

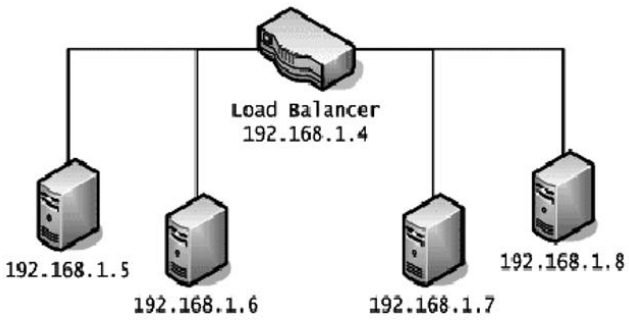


Рисунок 1 – Апаратний балансувальник

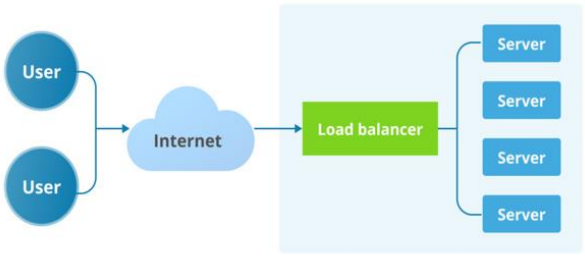


Рисунок 2 – Програмний балансувальник

4

Рівні балансування L4 і L7

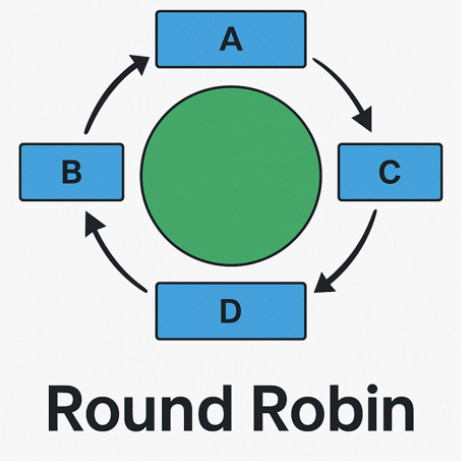
L4-балансування (транспортний рівень) розподіляє трафік на основі IP-адрес, портів і протоколів (TCP/UDP), без аналізу вмісту запиту. Він забезпечує високу продуктивність і низьку затримку, що робить його ефективним для простих служб і високонавантажених систем.

L7-балансування (рівень прикладних протоколів) аналізує повний HTTP(S)-запит, дозволяючи здійснювати інтелектуальну маршрутизацію залежно від URL, заголовків чи параметрів. Такий підхід забезпечує високу гнучкість, підтримку сесій і більше можливостей для безпеки та масштабування.

5

Алгоритм балансування Round Robin

Round Robin — це базовий алгоритм балансування навантаження, що працює за принципом циклічного розподілу запитів між доступними серверами. Кожен новий запит надсилається до наступного сервера у списку, після чого цикл повторюється. Такий підхід простий у реалізації, не потребує моніторингу стану серверів і добре підходить для однорідних середовищ з приблизно однаковою потужністю вузлів. Однак він не враховує реальне навантаження на сервери, що може призводити до дисбалансу в продуктивності



6

Least Connections та Least Load

Least Connections — спрямовує нові запити на сервер з найменшою кількістю активних підключень. Підходить для систем із довготривалими сесіями, забезпечує кращий розподіл у разі нерівномірного часу обробки, але не враховує інтенсивність навантаження кожного з'єднання.
Least Load — розширює попередній підхід, орієнтуючись на загальне навантаження сервера (CPU, RAM, трафік тощо). Дає точніше уявлення про стан інфраструктури, але потребує систем моніторингу і точного збору метрик, що ускладнює реалізацію.

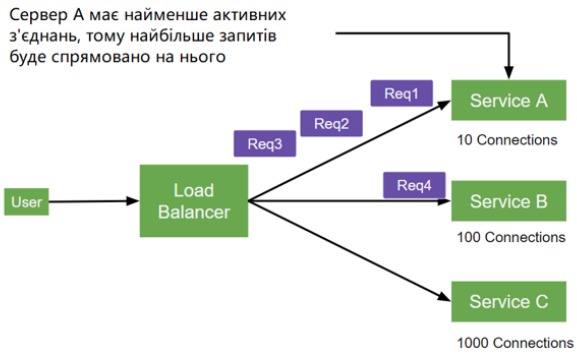


Рисунок 3 – Почерговість з'єднань в Least Connections

7

Розподіл трафіку за IP HASH

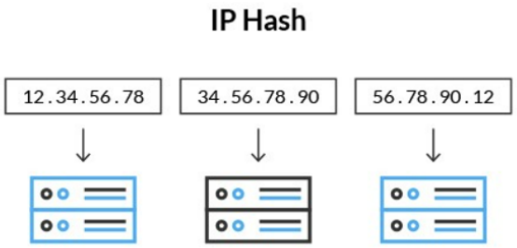


Рисунок 4 – Принцип роботи IP Hash

У цьому методі IP-адреса користувача перетворюється на число за допомогою спеціальної математичної функції (хешування), і це число використовується для вибору сервера.
Окрім IP-адреси, можна також використовувати інші частини запиту, такі як порт, з якого прийшов запит, шлях у запиті, Cookies або браузер користувача.

8

Динамічне адаптивне балансування

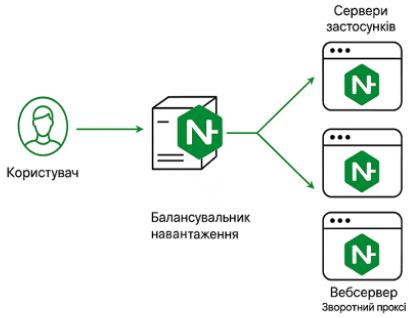


Рисунок 5 – Приклад інтерфейсу для моніторингу метрик

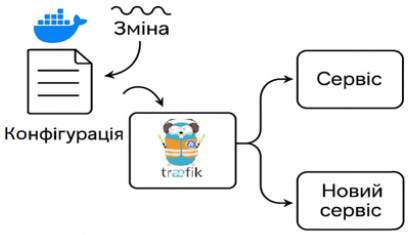
Динамічне адаптивне балансування — це метод розподілу запитів на основі реальних метрик стану серверів, таких як завантаженість CPU, пам’яті, мережевий трафік і час відповіді. На відміну від статичних підходів (Round Robin, Least Connections), він враховує повну картину навантаження і дозволяє більш точно визначити, який сервер зараз найменш завантажений.

9

Програмні засоби балансування трафіку: NGINX, HAProxy, Traefik







10

Етапи впровадження балансувальника трафіку

1. Аналіз потреб системи — визначити обсяг трафіку, тип запитів, критичність сервісів і очікувані навантаження.
2. Вибір рівня балансування — вирішити, чи потрібне балансування на рівні L4 (TCP/UDP) чи L7 (HTTP/HTTPS).
3. Оцінка інфраструктури — врахувати, чи використовується хмара, Docker, Kubernetes, віртуальні чи фізичні сервери.
4. Порівняння інструментів — обрати між NGINX, HAProxy, Traefik або іншими, виходячи з гнучкості, автоматизації та масштабованості.
5. Налаштування моніторингу — інтегрувати інструменти спостереження (Prometheus, Grafana тощо) для збору метрик.

11

Висновки

У даній роботі проведено аналіз та моделювання систем балансування навантаження в інформаційних мережах. Оцінено сучасні підходи до розподілу запитів між серверами. Основними результатами роботи є: досліджено принципи роботи популярних інструментів балансування, зокрема NGINX, HAProxy та Traefik; порівняно їхні можливості та особливості використання в різних інфраструктурах; розглянуто ключові алгоритми балансування — від простого Round Robin до адаптивних методів на основі метрик навантаження. Проведено моделювання інформаційної системи з динамічним балансуванням запитів, що дозволило оцінити ефективність обраних рішень. Встановлено, що адаптивні підходи забезпечують вищу стабільність та продуктивність у змінних умовах навантаження. Отримані результати підтверджують доцільність використання сучасних балансувальників навантаження для забезпечення безперебійної роботи критичних сервісів.

12