КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії
(назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему: «Система масштабованого розподілу навантаження для веб-сервісів»

Кравець Владислав Віталійович
(прізвище, ім'я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ

_____

„___" _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Система масштабованого розподілу навантаження для веб-сервісів

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Кравець Владислав Віталійович

(прізвище, ім'я та по батькові повністю)

123 «Комп'ютерна інженерія»

(спеціальність)

«Комп'ютерні системи і мережі»

(освітня програма)

Група КСМм-24

Керівник Німченко Т.В.

(прізвище та ініціали)

к.т.н., доцент кафедри

(вчене звання, науковий ступінь)

Рецензент_____

(прізвище та ініціали)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: кібербезпеки та комп'ютерна інженерія
Освітній рівень: магістр
Спеціальність: Комп'ютерна інженерія
ОПП: Комп'ютерні системи і мережі

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____
„___"_____2025 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Кравець Владислав Віталійович

(прізвище, ім'я та по батькові здобувача)

1. Тема роботи Система масштабованого розподілу навантаження для веб-сервісів

затверджена наказом ректора КНУБА № 1636.23/2.25 від «30» вересня 2025року

2. Керівник роботи
Німченко Тетяна Василівна - доцент кафедри кібербезпеки та комп'ютерної інженерії, кандидат технічних наук
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту 12.12.2025

4. Зміст пояснювальної записки за розділами:
Р. 1. Теорія та проблематика: динаміка адаптивного балансування навантаження
Р. 2. Метидологія і проектування системи
Р. 3. Результати та аналіз
Р. 4.
Р. 5.

5. Графічний матеріал за розділами:
Р. 1.
Р. 2.
Р. 3. 15 рисунків
Р. 4.
Р. 5.

6. Консультанти розділів кваліфікаційної випускної роботи

Розділи	Прізвища, ініціали та посади консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.	к.т.н., доцент Гуменний Д.О.	17.11.2025	
Розділ 3.	к.т.н., доцент Ізмайлова О. В.	08.12.2025	

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1. Теорія та проблематика: динаміка адаптивного балансування навантаження	20.10.2025
Розділ 2. Метидологія і проектування системи	17.11.2025
Розділ 3. Результати та аналіз	08.12.2025
Остаточне оформлення роботи	10.12.2025
Направлення роботи на рецензування, перевірку на плагіат	12.12.2025
Попередній захист роботи на кафедрі	15.12.2025

8. Дата видачі завдання 30 вересня 2025

Керівник _____ Німченко Т.В.
(підпис) (прізвище та ініціали)

Здобувач _____ Кравець В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кравець В.В. «Система масштабованого розподілу навантаження для веб-сервісів»

1. Проблема та метод вирішення

Сучасні розподілені системи страждають від нерівномірного навантаження («гарячі шарди») та високих затримок, оскільки класичні алгоритми хешування не здатні ефективно адаптуватися до змін популярності контенту. Для вирішення цієї проблеми розроблено алгоритм DA-ACH (Drift-Aware Adaptive Consistent Hashing). Він використовує статистичний моніторинг дрейфу трафіку та механізм «зсуву віртуальних кордонів», що дозволяє динамічно перерозподіляти запити. Це забезпечує баланс між рівномірним завантаженням вузлів та мінімізацією переміщення даних (стабільністю кешу).

2. Результати та практична цінність

Експерименти показали, що DA-ACH значно перевершує галузеві стандарти: він знижує дисбаланс навантаження з 2.84 до 1.18 та зменшує хвостову затримку (p99) на 87,8%, зберігаючи при цьому високий рівень влучання в кеш (~94%). Наукова новизна полягає у впровадженні моделі «плинного кільця» для керування топологією в реальному часі. Практичним результатом є готова бібліотека мовою Go, яка може бути інтегрована в балансувальники навантаження для економії ресурсів та покращення SLA.

Ключові слова: Розподілені системи, узгоджене хешування, балансування навантаження, дрейф концепції, stateful-сервіси, мікросервіси, адаптивна маршрутизація, хвостова затримка (p99), дивергенція Кульбака-Лейблера, локальність кешу.

SUMMARY

Kravets V.V. ‘Scalable load distribution system for web services’

1. Problem and Solution Method

Modern distributed systems suffer from uneven load distribution ("hot shards") and high latencies because classical hashing algorithms cannot effectively adapt to changes in content popularity. To address this issue, the DA-ACH (Drift-Aware Adaptive Consistent Hashing) algorithm was developed. It utilizes statistical traffic drift monitoring and a "virtual boundary shifting" mechanism to dynamically redistribute requests. This strikes a balance between uniform node loading and minimizing data movement (ensuring cache stability).

2. Results and Practical Value

Experiments demonstrate that DA-ACH significantly outperforms industry standards: it reduces load imbalance from 2.84 to 1.18 and decreases tail latency (p99) by 87.8%, while maintaining a high cache hit rate (~94%). The scientific novelty lies in the introduction of a "fluid ring" model for real-time topology management. The practical outcome is a ready-to-use Go library that can be integrated into load balancers to improve resource efficiency and enhance SLAs.

Keywords: Distributed systems, consistent hashing, load balancing, concept drift, stateful services, microservices, adaptive routing, tail latency (p99), Kullback-Leibler divergence, cache locality.

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ Кравець Владислав Віталійович Kravets Vladyslav Vitaliyovych		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	Система масштабованого розподілу навантаження для веб-сервісів Scalable load distribution system for web services		
Освітній ступінь	Магістр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерії		
Спеціальність	Комп'ютерна інженерія		
Освітня програма	Комп'ютерні системи і мережі		
Керівник	Німченко Тетяна Василівна		
Обсяг роботи:	*Пояснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	98	3	10
Розділ 1	Теорія та проблематика: динаміка адаптивного балансування навантаження		
Розділ 2	Метидологія і проектування системи		
Розділ 3	Результати та аналіз		
Висновки по роботі	У дипломній роботі доведено неефективність класичного узгодженого хешування в умовах нерівномірного веб-трафіку та розроблено алгоритм DA-ACH, який трансформує статичну топологію хеш-кільця в адаптивну систему на основі моніторингу дрейфу концепції.		
Ключові слова: Keywords:	Розподілені системи, узгоджене хешування, балансування навантаження, дрейф концепції, stateful-сервіси, мікросервіси, адаптивна маршрутизація, хвостова затримка (p99), дивергенція Кульбака-Лейблера, локальність кешу. Distributed systems, consistent hashing, load balancing, concept drift, stateful services, microservices, adaptive routing, tail latency (p99), Kullback-Leibler divergence, cache locality.		

Здобувач Кравець Владислав Віталійович/ _____

Керівник Німченко Тетяна Василівна / _____

ЗМІСТ

ВСТУП 10

РОЗДІЛ 1 ТЕОРІЯ ТА ПРОБЛЕМАТИКА: ДИНАМІКА АДАПТИВНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ 17

1.1 Теоретичні засади та формальна постановка задачі 17

1.2 Формальні визначення 19

1.3 Теоретичні основи балансування навантаження 21

1.4 Обмеження узгодженого хешування в умовах навантажень з «важкими хвостами» 24

1.5 Збереження стану та штраф за перебалансування 26

1.6 Часова динаміка та дрейф концепції 28

1.7 Постановка задачі 30

1.8 Аналітичні вимоги 32

1.9 На шляху до уніфікованої моделі: GAP-аналіз існуючих фреймворків . 34

1.10 Необхідність фреймворку хешування з урахуванням дрейфу 37

1.11 Уніфікована постановка задачі 38

1.12 Підсумок теоретичних висновків 39

РОЗДІЛ 2 МЕТОДОЛОГІЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ 41

2.1 Методологічні засади та обґрунтування напрямку дослідження 41

2.2 Модель системи та формалізація 46

2.3 Запропоноване рішення: Адаптивне узгоджене хешування з урахуванням дрейфу (DA-ACH) 50

2.4 Експериментальна база 53

2.5 Дослідницькі гіпотези 57

2.6 Резюме методології 59

РОЗДІЛ 3 РЕЗУЛЬТАТИ ТА АНАЛІЗ 62

3.1 Програма та методика експериментального дослідження 62

3.2 Експериментальне середовище та конфігурація 64

3.3 Експеримент 1 - Розподіл навантаження при статичному перекосі 72

3.4 Експеримент 2 - Адаптація до дрейфу концепції 79

3.5 Експеримент 3 - Ефективність кешування та вартість перевідображення 82

3.6 Аналіз обчислювальних накладних витрат 86

3.7 Аналіз чутливості (Налаштування параметрів) 89

3.8 Розширене резюме та висновки 91

ВИСНОВКИ 94

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 100

ВСТУП

Актуальність теми та наукова значущість

Архітектура сучасних масштабних інтернет-додатків зазнала фундаментальних парадигмальних змін. Внаслідок експоненціального зростання глобального трафіку даних, що за прогнозами сягатиме сотень зетабайтів щорічно [1, с. 3], та повсюдного поширення високошвидкісних мобільних мереж, проєктування програмного забезпечення перейшло від монолітних структур до розподілених, слабкозв'язаних (loosely coupled) мікросервісів [2, с. 2-5]. У такому середовищі одна взаємодія з користувачем часто запускає каскад віддалених викликів процедур (RPC) між десятками окремих сервісних вузлів. Для підтримки високої доступності та низької затримки в цих складних системах балансування навантаження виступає в ролі критично важливого архітектурного елемента.

Хоча балансування на 4 рівні (Layer 4) є добре вивченим, сучасні вимоги дедалі частіше потребують складного балансування на 7 рівні (рівень застосунків) для підтримки вебсервісів зі збереженням стану (stateful), таких як розподілені кеші (наприклад, Redis, Memcached) [3] та фрагментовані бази даних. У цих середовищах галузевим стандартом стало узгоджене хешування (Consistent Hashing) завдяки його здатності зберігати спорідненість ключів (key affinity) і мінімізувати відтік користувачів («churn») під час відмови вузлів [4, с. 654-656]. Однак стандартне узгоджене хешування спирається на статичні математичні припущення - зокрема, на рівномірний розподіл ключів, - які рідко виконуються в реальних сценаріях.

Сучасні вебнавантаження характеризуються концептуальним дрейфом (Concept Drift) та розподілами Ципфа (степеневими законами) [5, с. 127-130; 6, с. 1-2], де мала підмножина ключів притягує більшість трафіку. Коли патерни трафіку змінюються, статичні призначення хешування призводять до серйозного дисбалансу навантаження, відомого як «гарячі шарди» (hot shards), що спричиняє погіршення хвостової затримки (p99) та потенційні каскадні збої. Існуючі рішення

пропонують бінарний вибір: збереження спорідненість за рахунок балансу (стандартне узгоджене хешування) або оптимізація балансу за рахунок спорідненість (метод найменших з'єднань / Least Connections). Існує критична нестача механізмів, здатних одночасно адаптуватися до змінюваних навантажень і зберігати локальність кешу, необхідну для високопродуктивних систем зі збереженням стану.

Таким чином, розробка адаптивних алгоритмів маршрутизації, що враховують дрейф навантаження та здатні збалансувати компроміс між стабільністю маршрутизації та справедливістю розподілу навантаження, має значну наукову та практичну актуальність.

Об'єкт і предмет дослідження

- **Об'єктом** дослідження є процес розподілу трафіку та формування навантаження у розподілених вебсервісах зі збереженням стану, що характеризуються стохастичною інтенсивністю запитів із «важкими хвостами» розподілу та часовим дрейфом концепції.
- **Предметом** дослідження є адаптивні методи та алгоритми адаптивного послідовного хешування, призначені для динамічного перерозподілу навантаження між сервісними вузлами з мінімізацією витрат на інвалідацію кешу та перепризначення (remapping).

Обґрунтування необхідності нової розробки

Аналіз вітчизняної та закордонної наукової літератури, патентних баз даних та досвіду експлуатації провідних технологічних компаній виявляє прогалину в існуючих методологіях балансування навантаження.

1. **Обмеження статичного хешування:** Класичне узгоджене хешування (Karger et al.) мінімізує перепризначення під час ротації вузлів,

але є математично «сліпим» до обсягу запитів. Воно не здатне обробляти феномен «гарячих шардів», спричинений перекосом популярності ключів.

2. **Недоліки динамічних алгоритмів:** Реактивні алгоритми, такі як Least-Connections або Power-of-Two-Choices, ефективно балансують навантаження, але руйнують сесійну спорідненість. У stateful-архітектурах це призводить до надмірних промахів кешу («трешинг»/thrashing) та збільшення тиску на бекенд.

3. **Нездатність обробляти дрейф:** Поточні розширення «зваженого» хешування покладаються на статичний розподіл потужностей. Їм бракує статистичних механізмів для виявлення дрейфу концепції - еволюції патернів трафіку з часом (наприклад, flash crowds), що призводить до запізнілих або коливальних реакцій на зміну попиту.

Отже, існує нагальна потреба в модернізації існуючого об’єкта дослідження шляхом розробки системи адаптивного узгодженого хешування з урахуванням дрейфу (Drift-Aware Adaptive Consistent Hashing - DA-ACH). Ця система має інтегрувати статистичний моніторинг у реальному часі з контуром зворотного зв’язку для динамічного коригування топології хешування - можливість, яка наразі відсутня у стандартних балансувальниках навантаження з відкритим кодом.

Мета і завдання роботи

Основною **метою** роботи є підвищення ефективності та стабільності розподілених систем зі збереженням стану шляхом розробки алгоритму маршрутизації, що динамічно адаптується до дрейфу концепції та нерівномірних розподілів, мінімізуючи при цьому обсяг перепризначень ключів.

Для досягнення цієї мети були поставлені такі **завдання**:

1. **Теоретичний аналіз:** Проаналізувати математичні обмеження узгодженого хешування на основі Chord та Maglev в умовах нерівномірних розподілів запитів за законом Ципфа.

2. **Проєктування алгоритму:** Розробити алгоритм DA-ACH, що включає:
    - Статистичний модуль для виявлення дрейфу концепції з використанням дивергенції Кульбака - Лейблера.
    - Механізм динамічного зважування, що коригує ємність віртуальних вузлів на основі зворотного зв'язку в реальному часі.
    - Стратегію «контрольованого відхилення» для пріоритезації міграції ключів з великим обсягом лише за необхідності.
3. **Реалізація:** Реалізувати функціональний прототип запропонованого балансувальника навантаження, придатного для вебсервісів рівня 7, використовуючи мову програмування Go.
4. **Оцінювання:** Експериментально перевірити запропоновану систему порівняно з галузевими базовими рішеннями (стандартне узгоджене хешування, хешування з обмеженим навантаженням / Bounded-Load Hashing та Power-of-Two-Choices) на синтетичних наборах даних, що представляють еволюцію навантажень, зосереджуючись на таких метриках, як коефіцієнт дисбалансу (Imbalance Ratio - IR), фактор перепризначення (Remapping Factor - RF) та хвостова затримка (p99).

Методи дослідження

Для вирішення поставлених завдань було застосовано методологію дослідження через розробку (Design Science Research - DSR) [7, с. 77-80] з використанням таких конкретних методів:

- **Теорія ймовірностей та випадкових процесів:** Для моделювання інтенсивності надходження запитів, розподілів Ципфа та статистичних властивостей дрейфу концепції [8, с. 36-39].
- **Теорія автоматичного керування:** Застосована для проєктування контуру зворотного зв'язку, що коригує ваги вузлів на основі градієнтів помилок (дисбалансу навантаження) [9, с. 15-22].

- **Дискретно-подійне моделювання:** Використано для моделювання середовища розподіленого кластера, що дозволяє детерміновано відтворювати «умови гонки» (race conditions), мережеві затримки та відмови вузлів.
- **Статистичний аналіз:** Застосовано для кількісної оцінки дрейфу (використовуючи KL-дивергенцію) [10, с. 79] та оцінювання продуктивності алгоритму (розрахунок стандартних відхилень та перцентилів затримки).
- **Порівняльний аналіз:** Використано для бенчмаркінгу запропонованого рішення у зіставленні зі стандартними галузевими алгоритмами.

Наукова новизна одержаних результатів

Наукова новизна полягає у розробці нового підходу до адаптивної маршрутизації для розподілених систем зі збереженням стану. Конкретні нові досягнення, отримані магістром особисто, включають:

1. **Перша формалізація дрейфу в хешуванні:** У роботі впроваджено таксономію дрейфу концепції (раптовий, поступовий, повторюваний), спеціально адаптовану до топологій узгодженого хешування [6, с. 5], що розрізняє тимчасовий шум і структурні зміни трафіку.
2. **Розробка методу DA-ACH:** На відміну від існуючого статичного зваженого хешування, розроблено новітній метод «плинного кільця» (fluid ring), який розглядає розміщення віртуальних вузлів як динамічну змінну, керовану контуром зворотного зв’язку. Це дозволяє системі мінімізувати коефіцієнт дисбалансу (IR) значно краще, ніж стандартне узгоджене хешування.
3. **Оптимізація компромісу «спорідненість-справедливість»:** У дослідженні встановлено стратегію «контрольованого відхилення», яка математично обмежує вартість перепризначення (Ω). Це відрізняється від

попередніх динамічних методів тим, що міграція ключів суворо обмежується мінімумом, необхідним для відновлення балансу, тим самим зберігаючи вищі показники влучання в кеш (cache hit rates) порівняно з рандомізованими алгоритмами.

Практичне значення одержаних результатів

Практичне значення роботи полягає у створенні програмного рішення, що підвищує надійність та продуктивність високонавантажених вебсервісів.

- **Реалізація:** Запропонований алгоритм реалізовано у вигляді модульної бібліотеки мовою Go [11], придатної для інтеграції в сучасні sidecar-проксі або ingress-контролери.
- **Покращення продуктивності:** Емпіричне тестування показало, що DA-ACH знижує хвостову затримку p99 приблизно на 87,8% порівняно зі стандартним узгодженим хешуванням при нерівномірних навантаженнях (Zipf α=1.2).
- **Сфери застосування:** Результати можуть бути безпосередньо застосовані організаціями, що експлуатують великомасштабні розподілені кеші (наприклад, кластери Redis), мережі доставки контенту (CDN) та мікросервіси на платформах оркестрації, таких як Kubernetes.
- **Економічний ефект:** Завдяки вирівнюванню розподілу навантаження алгоритм дозволяє підвищити утилізацію існуючого обладнання, потенційно зменшуючи потребу в надлишковому виділенні ресурсів (overprovisioning) на 30-50%.

Публікація

Результати дослідження були опубліковані в міжнародному науковому журналі «Інтер-Наука». Стаття містить ідеї та висновків магістерської роботи. Електронна версія доступна за адресою: https://www.inter-nauka.com/issues/2025/11/11665 [12]

РОЗДІЛ 1 ТЕОРІЯ ТА ПРОБЛЕМАТИКА: ДИНАМІКА АДАПТИВНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

1.1 Теоретичні засади та формальна постановка задачі

У цьому розділі закладено теоретичні основи, формальні визначення та системну постановку проблеми, що обґрунтовують розробку механізму адаптивного узгодженого хешування з урахуванням дрейфу (Drift-Aware Adaptive Consistent Hashing - DA-ACH). Тоді як у вступі було розглянуто операційне середовище технологій маршрутизації рівнів Layer 4 та Layer 7 з точки зору реалізації, тут увага зміщується на абстрактні обчислювальні об’єкти дослідження, алгоритмічні суб’єкти, що діють над ними, та математичну динаміку, яка спричиняє дисбаланс, нестабільність і надмірну міграцію станів у сучасних розподілених архітектурах.

Сучасні розподілені вебсервіси - від сіток мікросервісів (наприклад, Istio, Linkerd) до розподілених сховищ «ключ-значення» (наприклад, Redis Cluster, Cassandra, DynamoDB) [13, с. 205] - дедалі більше покладаються на маршрутизацію на основі ключів (key-based routing). У таких середовищах підтримання спорідненості (affinity) - детермінованого відображення конкретного ідентифікатора клієнта, токена сесії або ключа даних на конкретний бекенд-вузол - є не просто оптимізацією продуктивності, а суворою експлуатаційною вимогою. Спорідненість забезпечує локальність кешу, гарантує персистентність сесій та мінімізує навантаження на читання бази даних, запобігаючи явищу «лавиноподібних запитів» (thundering herd).

Однак існує фундаментальне протиріччя між статичними припущеннями класичних алгоритмів маршрутизації та стохастичною, мінливою природою інтернет-трафіку. Класичні підходи, такі як Round Robin, метод найменших з’єднань (Least Connections) та стандартне узгоджене хешування, пропонують часткові рішення, але не здатні одночасно вирішити «Трилему сучасного

балансування навантаження» [14, с. 24-25]. Ми визначаємо цю трилему як складність одночасної оптимізації трьох конфліктуючих цілей:

1. **Обробка розподілів з «важкими хвостами»:** Система повинна витримувати «гарячі ключі» (hot keys), де трафік підпорядковується степеневому закону або розподілу Ципфа [5, с. 126], що означає, що малий відсоток ключів генерує більшість навантаження.

2. **Збереження навантажень зі станом (Stateful Workloads):** Система повинна мінімізувати перепризначення (remapping) ключів. На відміну від HTTP-запитів без збереження стану (stateless), де підійде будь-який вузол, навантаження зі станом накладають високий штраф (промахи кешу, скидання сесій, операції введення-виведення сховища) за кожну зміну відображення.

3. **Адаптація до часового дрейфу:** Система повинна реагувати на нестаціонарну природу поведінки користувачів. Популярність не є статичною властивістю; «найгарячіший» ключ у момент

t=0 може стати «холодним» до моменту t=100. Це явище, відоме як дрейф концепції (Concept Drift), робить статичні вагові коефіцієнти застарілими.

Для ретельного аналізу цих недоліків у цьому розділі впроваджується уніфікована теоретична база, що ґрунтується на теорії автоматичного керування. Ми формально визначаємо Процес розподілу трафіку (екзогенний Об’єкт) та Метод адаптивного хешування (ендогенний Суб’єкт). Використовуючи ці визначення, ми математично характеризуємо режими відмови існуючих систем та виокремлюємо конкретну інженерну прогалину, яку покликана заповнити ця робота: відсутність топології маршрутизації, яка була б достатньо стабільною для збереження стану і водночас достатньо гнучкою для поглинання перекосів навантаження.

1.2 Формальні визначення

У дослідженнях розподілених систем невизначеність щодо меж системи часто приховує першопричини погіршення продуктивності. Для забезпечення аналітичної точності ми приймаємо модель «Об’єкт - Суб’єкт - Взаємодія». Об’єкт представляє екзогенне навантаження (трафік), яке система не може контролювати, але повинна обслуговувати. Суб’єкт представляє ендогенну алгоритмічну логіку (балансувальник навантаження), яка намагається впорядкувати роботу з Об’єктом.

Наведені нижче визначення формалізують межі цього дослідження.

Визначення 1 (Об’єкт: Процес розподілу трафіку) Нехай $K = \{k_1, k_2, \dots, k_m\}$ - дискретний, зліченний простір ключів, що представляє всі можливі ідентифікатори запитів (наприклад, ID користувачів, токени сесій або первинні ключі бази даних). На практиці |K| часто на порядки перевищує кількість вузлів.

Нехай R(t):K→N позначає стохастичний лічильний процес, що представляє інтенсивність вхідних запитів для конкретного ключа в момент часу t. Ми визначаємо Процес розподілу трафіку як кортеж:

$$O = (K, \Lambda(t), P_t)$$

Де:

- $\Lambda(t)$ - сукупна інтенсивність надходження запитів до системи в момент часу t (запитів за секунду).
- P_t - функція ймовірності (PMF) на множині K в момент часу t, така що

$$\sum_{k \in K} P_t(k) = 1$$

.

Теоретична примітка: У цій роботі ми постулюємо, що P_t часто наближається до розподілу Ципфа (степеневого закону), де ймовірність i-го за популярністю ключа становить $P(i) \propto 1/i^a$. Критично важливо, що система є нестаціонарною, тобто $P_t \neq P_{t+\Delta t}$. Ця нестаціонарність - зміна базового розподілу ймовірностей з часом - є формальним визначенням Дрейфу.

Визначення 2 (Об'єкт: Функція відображення та стан бекенду) Система балансування навантаження керує набором бекенд-вузлів $N = \{n_1, n_2, \dots, n_M\}$. Стан системи визначається Функцією відображення:

$$M_t: K \to N$$

Ця функція діє як сюр'єктивне відображення, призначаючи кожен ключ із простору ключів конкретному вузлу в конкретний час.
Навантаження на вузол n_i визначається як сума інтенсивностей усіх ключів, що наразі відображені на нього:

$$\mathrm{Load}(n_i, t) = \sum_{k \in M_t^{-1}(n_i)} R(k, t)$$

Де $M_t^{-1}(n_i)$ представляє прообраз вузла - множину всіх ключів, поточно призначених вузлу n_i. Це рівняння підкреслює, що навантаження на вузол суворо визначається сукупною популярністю ключів, якими він «володіє».

Визначення 3 (Суб'єкт: Метод адаптивного послідовного хешування) Нехай $H_t: K \to [0,1)$ - сімейство функцій хешування або стратегій розміщення на кільці, індексованих часом t.
Суб'єкт у цій роботі - це адаптивний алгоритмічний процес, визначений кортежем $A = (\Phi, \Psi, \Omega)$, що дотримується циклу керування «Сприйняття - Аналіз - Дія» (Sense-Think-Act):

1. Ф(Функція спостереження): Моніторить системні метрики (наприклад, використання процесора вузлами, лічильники частоти конкретних ключів).

2. Ψ(Функція прийняття рішень): Аналізує спостереження для виявлення дисбалансу або сигналів дрейфу (наприклад, обчислюючи дивергенцію Кульбака - Лейблера між минулим і поточним розподілами).

3. Ω(Функція виконання/актуації): Оновлює відображення $M_t \rightarrow M_{t+1}$ шляхом зміни меж простору хешування, ваг вузлів або розподілу віртуальних вузлів.

Запропонований алгоритм Drift-Aware Adaptive Consistent Hashing (DA-ACH) є конкретним екземпляром цього суб'єкта, розробленим для оптимізації компромісу між дисперсією навантаження (справедливістю) та витратами на перепризначення (стабільністю).

Визначення 4 (Проблема: Взаємодія Об'єкта та Суб'єкта) Інженерним викликом, що розглядається в цій роботі, є динамічна взаємодія між стохастичним Об'єктом (O) та алгоритмічним Суб'єктом (A). Зокрема, ми аналізуємо такі Емерджентні явища:

- **Дисбаланс:** Стан, коли $\mathrm{Var}(\mathrm{Load}(n_i))$ перевищує поріг толерантності, що призводить до хвостової затримки або відмови вузла.
- **Нестабільність (Плинність):** Висока частота змін у M_t, що позначається як $\frac{d}{dt} M_t$.
- **Порушення спорідненості:** Ймовірність того, що $M_t(k) \neq M_{t+1}(k)$ для персистентного ключа k, що безпосередньо корелює з частотою промахів кешу.

1.3 Теоретичні основи балансування навантаження

Щоб зрозуміти, чому новий механізм адаптивного хешування є необхідним, ми повинні спершу деконструювати обмеження класичних моделей балансування

навантаження. Історично ці моделі походили від теорії ймовірностей «кулі в урни» (Balls-into-Bins), яка спирається на припущення про рівномірність та незалежність, що рідко виконуються в сучасних виробничих середовищах.

1.3.1 Класичні моделі балансування навантаження

Традиційні балансувальники навантаження загалом поділяються на дві категорії: Статичні детерміновані та Динамічні реактивні [15, с. 230-235].

1. Статичні детерміновані (наприклад, Round Robin, Weighted Round Robin):

Ці алгоритми циклічно проходять через список вузлів N(i=(i+1)(modN)).

- *Припущення:* Вартість запитів є однорідною (гомогенність), а взаємодії не мають стану.
- *Сценарій збою:* У гетерогенній системі, де один запит триває 1 мс, а інший - 1000 мс, Round Robin призводить до серйозної фрагментації ресурсів (проблема «блокування початку черги» / Head-of-Line Blocking) [16, с. 69]. Крім того, він «сліпий» до умов часу виконання; вузол, близький до насичення, отримує такий самий трафік, як і вільний вузол, що неминуче призводить до каскадних збоїв.

2. Динамічні реактивні (наприклад, Least Connections, Power of Two Choices):

Ці алгоритми запитують стан вузла перед маршрутизацією. Наприклад, метод «Сила двох виборів» (Power of Two Choices, Mitzenmacher, 2001) вибирає два випадкові вузли та спрямовує запит до того, у якого менше активних з’єднань [17, с. 1094-1096].

- *Перевага:* Цей підхід забезпечує відмінну мінімізацію дисперсії навантаження, зазвичай обмежуючи максимальне навантаження до O(loglogN) замість O(logN), характерного для випадкового вибору.

- *Сценарій збою:* Руйнування афінності. Оскільки рішення про маршрутизацію залежить від миттєвого навантаження, ключ

k, запитаний у момент t_1, може потрапити на вузол n_A, а в момент t_1 - на вузол n_B. Для сервісів зі станом (наприклад, рівень кешування) це призводить до майже нульового показника влучання в кеш для повторюваних запитів, що змушує виконувати дорогі повторні обчислення або вибірки з бази даних.

1.3.2 Узгоджене хешування як механізм стабільності

Для вирішення проблеми спорідненості, властивої реактивним алгоритмам, Karger et al. (1997) запропонували узгоджене хешування (Consistent Hashing - CH) [4, с. 654]. CH фундаментально змінює парадигму маршрутизації з вибору (вибір вузла на запит) на топологію (відображення запитів на статичний адресний простір).

Топологія та механізм:

Як вузли, так і ключі відображаються на спільний простір ідентифікаторів, який зазвичай розглядається як кільце в одиничному інтервалі [0,1).

1. Позиція вузла: $Pos(n)=hash(ID_n)(mod1)$
2. Позиція ключа: $Pos(k)=hash(k)(mod1)$
3. Відображення: Ключ призначається першому вузлу, що зустрічається при русі за годинниковою стрілкою по кільцю: $M(k)=succ(Pos(k))$

Теоретичні гарантії:

CH забезпечує Монотонність: при додаванні вузла переміщуються лише ті ключі, які строго відображаються на цей новий вузол. При видаленні вузла його ключі перерозподіляються на його наступника (successor). Ця властивість гарантує, що вартість перепризначення є мінімальною (O(K/N)), на відміну від перепризначення O(K), необхідного при модульному хешуванні (k(modN)). Ця властивість зробила CH галузевим стандартом для розподілених кешів

(Memcached), NoSQL баз даних (DynamoDB, Cassandra) та мереж доставки контенту (Akamai) [18, с. 4].

Недостаток:

Хоча CH вирішує проблеми стабільності (плинності) та спорідненості, воно балансує Простір хешування, а не Простір навантаження. Воно оперує припущенням, що якщо ключі рівномірно розподілені на кільці (Геометричний баланс), то й навантаження буде рівномірно розподілене по вузлах (Баланс навантаження). Як демонструє Вступ, це припущення є хибним для реального вебтрафіку.

1.4 Обмеження узгодженого хешування в умовах навантажень з «важкими хвостами»

Математична елегантність узгодженого хешування (Consistent Hashing - CH) руйнується при зіткненні зі статистичними реаліями патернів інтернет-трафіку. Цей збій відбувається через два основні фактори: геометричний дисбаланс (неспроможність хеш-функції рівномірно розмістити вузли) та розподільчий перекіс (нерівномірність самого трафіку).

1.4.1 Проблема нерівномірного розподілу навантаження

Навіть із впровадженням «віртуальних вузлів» (VNodes), коли один фізичний вузол представлений кількома точками на кільці для покращення геометричного розподілу, CH стикається з труднощами при нерівномірних навантаженнях. Якщо вхідний трафік підпорядковується розподілу Ципфа (що є типовим для соціальних мереж, електронної комерції та медіа), мала підмножина ключів генерує переважну частину трафіку.

$$P(i) \approx \frac{C}{i^{\alpha}} \text{ для } \alpha > 1$$

У цьому сценарії розміщення «Топ-1» або «Топ-10» найпопулярніших ключів фактично визначає навантаження на вузли, на які вони відображаються.

- **Ефект «гарячого шарду» (Hot Shard):** Якщо найпопулярніший ключ k_{top} відображається на вузол n_1, завантаження процесора n_1 може сягнути 100%, тоді як n_2 залишатиметься на рівні 5%. Це трапляється незалежно від того, наскільки «випадковою» є хеш-функція. Випадковість хеш-функції розподіляє ключі, а не обсяг трафіку.
- **Обмеження віртуальних вузлів (VNodes):** Збільшення кількості VNodes покращує гранулярність розбиття простору хешування, ефективно нарізаючи кільце на менші сегменти. Однак VNodes не можуть розділити один ключ. Це проблема «неподільності атома». Якщо k_{top} генерує 50 000 запитів на секунду, і цей ключ потрапляє в сегмент (bucket), що належить n_1, вузол n_1 повинен обробити всі 50 000 запитів. Збільшення кількості VNodes для n_1 лише дає йому більше сегментів (і потенційно більше гарячих ключів); це не зменшує тиск від одного гарячого ключа, яким він вже володіє.

1.4.2 Гетерогенна вартість запитів

Окрім частоти запитів, варіюється також вартість одного запиту, що призводить до обчислювального перекосу (Computational Skew). Нехай $C(k)$ - обчислювальна вартість (такти процесора або час введення-виведення), необхідна для обробки ключа k. Загальне навантаження на вузол формально визначається як:

$$\mathrm{TotalLoad}(n) = \sum_{k \to n} \mathrm{Freq}(k) \times C(k)$$

Стандартне узгоджене хешування розглядає всі ключі як математично рівнозначні точки на колі. Воно є «сліпим» до змінної $C(k)$. Вузол, якому призначено складний ключ для агрегаційного запиту (високе $C(k)$), буде перевантаженим порівняно з вузлом, якому призначено простий ключ типу «отримання за ID», навіть якщо їхні частоти запитів ($Freq(k)$) ідентичні.

Ця невідповідність між геометричним балансом (який забезпечує CH через хешування) та зваженим балансом навантаження (якого насправді потребують системи) становить фундаментальну теоретичну прогалину. Стандартному CH бракує механізму зворотного зв'язку для оцінки C(k) або Freq(k), що робить його нездатним адаптуватися до цих диспропорцій.

1.5 Збереження стану та штраф за перебалансування

Щоб зрозуміти, чому просте реактивне балансування навантаження (наприклад, довільне переміщення ключів з гарячих вузлів на холодні) є недостатнім, необхідно суворо кількісно оцінити вартість стану (Statefulness). У контексті сучасних розподілених систем «стан» стосується локального контексту, необхідного для ефективної обробки запиту - зазвичай це дані, що знаходяться в оперативній пам'яті (RAM) кешу (наприклад, Redis, Memcached) або локальних дискових буферах.

1.5.1 Ефект «натовпу» (Cache Stampede / Thundering Herd)

Коли ключ k перепризначається з Вузла A на Вузол B (можливо, для зменшення навантаження на A), Вузол B спочатку не володіє кешованим станом для k. Ця подія запускає специфічний каскад збоїв, відомий як Cache Stampede або проблема «громихучого стада» (Thundering Herd) [19, с. 345]:

1. **Холодний старт:** Вузол B отримує запит для k. Оскільки його локальний кеш порожній («промах кешу»), він не може обробити запит негайно.
2. **Отримання з джерела (Upstream Fetch):** Вузол B повинен отримати авторитетні дані з рівня персистентного зберігання (наприклад, дискової SQL-бази даних або сховища об'єктів S3).
3. **Розбіжність затримок:** Рівень персистентності незмінно повільніший за рівень кешування. Типове читання з RAM займає ≈100мкс, тоді як пошук на диску або мережевий запит до бази даних може тривати 10-50мс. Це означає зростання затримки на 2-3 порядки.

4. **Навала (Stampede):** Якщо k є високочастотним ключем (наприклад, 10 000 запитів/с), Вузол B отримує не один запит, а тисячі за той час, поки завершиться перша вибірка з бази даних. На відміну від Вузла A, який обслуговував їх з RAM, Вузол B може ініціювати тисячі одночасних з'єднань до бекенд-бази даних.

5. **Системний збій:** Цей сплеск навантаження на читання часто призводить до падіння базової бази даних, спричиняючи відмову всієї системи.

Теоретичний висновок: Функція вартості перепризначення (C_{mig}) є нелінійною та залежною від ключа. Переміщення популярного ключа є експоненціально дорожчим і небезпечнішим, ніж переміщення непопулярного. Тому алгоритм балансування навантаження повинен «враховувати популярність» (popularity-aware), щоб уникнути провокування подій самоспричиненої відмови в обслуговуванні (DoS).

1.5.2 Реактивні алгоритми руйнують спорідненість

Динамічні алгоритми, такі як метод найменших з'єднань (Least Connections), оптимізують миттєву дисперсію навантаження: Objective: min(Var(Load))

Однак, ігноруючи історію відображень, ці алгоритми діють як «руйнівники спорідненості» (Affinity Erasers). Вони розглядають систему як таку, що не має пам'яті. Якщо Вузол A має 10 з'єднань, а Вузол B - 9, алгоритм спрямовує наступний запит до B, незалежно від того, чи Вузол A вже містить необхідні дані в L1-кеші або RAM.

Для сервісу зі станом така поведінка змушує систему перебувати в перманентному стані «холодних кешів», фактично розмінюючи незначне покращення балансу CPU на масивне насичення пропускної здатності введення-виведення (I/O). Отже, надійне рішення повинно розглядати збереження афінності як жорстке обмеження або змінну з великою вагою. Перебалансування має

запускатися лише тоді, коли вартість поточного дисбалансу (наприклад, збій Вузла А) перевищує оцінену вартість міграції.

1.6 Часова динаміка та дрейф концепції

Останній теоретичний стовп стосується виміру Часу. Об'єкт розподілу трафіку O не є статичним; він динамічний. Розподіл ймовірностей P_t еволюціонує протягом життєвого циклу системи - явище, яке в машинному навчанні формально називається дрейфом концепції (Concept Drift).

1.6.1 Типи дрейфу в балансуванні навантаження

Веб-навантаження демонструють чіткі патерни дрейфу, кожен з яких вимагає різної алгоритмічної реакції:

1. **Раптовий дрейф (Flash Crowds / Раптові сплески):** Дискретна подія - наприклад, пуш-сповіщення про «гарячу новину» або твіт знаменитості - призводить до того, що раніше маловідомий ключ стає глобальним максимумом (k_{top}) за лічені секунди.
    - *Характеристика сигналу:* Високе прискорення (d^2/dt^2 інтенсивності запитів).
    - *Ризик:* Миттєве перевантаження «гарячого шарду» та відмова вузла.
2. **Поступовий (інкрементальний) дрейф:** Протягом днів або тижнів інтереси користувачів зміщуються. Популярність нового фільму зростає повільно, досягає піку, а потім згасає.
    - *Характеристика сигналу:* Повільна, монотонна зміна в P_t

    .

    - *Ризик:* Ефект «жаби в окропі». Порогові системи часто не помічають цього накопичення, доки дисбаланс не стане критичним.

3. **Циклічний/Рекурентний дрейф:** Добові патерни, коли різні географічні регіони (а отже, і різні підмножини ключів користувачів) стають активними в різний час доби.
    - *Характеристика сигналу:* Періодичні коливання.

1.6.2 Крах статичних ваг

Поточні реалізації зваженого узгодженого хешування (наприклад, Ketama) дозволяють операторам призначати вузлам вагові коефіцієнти ємності (w_n). Наприклад, потужний сервер може отримати вагу 2.0 (володіючи приблизно удвічі більшим простором хешування) порівняно з 1.0 для стандартного сервера.

- *Обмеження:* Ці ваги є статичними конфігураційними значеннями. Вони вирішують проблему гетерогенності обладнання (різні потужності серверів), але повністю ігнорують перекіс даних (Data Skew).
- *Вплив дрейфу:* Коли відбувається дрейф, семантичне значення розділу (партиції) змінюється. Сектор кільця хешування, який генерував 100 запитів/с вчора, може генерувати 10 000 запитів/с сьогодні через те, що вірусний ключ потрапив у цей сектор. Статичні ваги не можуть адаптуватися до цієї зміни без ручного втручання (переконфігурації оператором), що є надто повільним для пом’якшення наслідків у реальному часі.

1.6.3 Реактивні системи неправильно класифікують дрейф

Без статистичної моделі дрейфу стандартні балансувальники навантаження не можуть розрізнити Сигнал (істинну зміну популярності) та Шум (природну дисперсію інтенсивності пуассонівського надходження).

- *Надмірна реакція (Тріпотіння / Flapping):* Переміщення ключа через односекундний сплеск запитів. Якщо сплеск був шумом, система переміщує ключ, отримує штраф за кеш, а потім, ймовірно, переміщує його назад, коли шум вщухає. Ця осциляція руйнує продуктивність.

- *Недостатня реакція:* Ігнорування стійкого сплеску, оскільки він ще не перетнув статичний поріг «CPU > 90%», до моменту якого черги запитів вже переповнені.

Це обумовлює необхідність компонента врахування дрейфу (Drift-Aware) у запропонованому рішенні: механізму, що використовує статистичний моніторинг послідовностей (наприклад, ковзні вікна, тестування на виявлення змін) для запуску перебалансування лише тоді, коли виявлено статистично значущі розподільчі зрушення.

1.7 Постановка задачі

Спираючись на теоретичні основи, викладені в розділах 1.1-1.6, ми можемо транслювати обмеження існуючих систем у формальну інженерну задачу. Ключова проблема полягає не просто в «балансуванні навантаження», а у виконанні цього процесу в просторі багатокритеріальної оптимізації, де цілі є взаємно антагоністичними.

1.7.1 Багатокритеріальний конфлікт

Ми постулюємо, що задача балансування навантаження є грою з нульовою сумою між стабільністю системи (мінімізацією міграції стану) та справедливістю розподілу навантаження (мінімізацією дисперсії).
Нехай $L_t(n_i)$ позначає нормоване навантаження на вузол n_i у момент часу t. Глобальний дисбаланс системи, $I(t)$, визначається як стандартне відхилення навантаження по всіх вузлах:

$$I(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(L_t(n_i) - \bar{L_t})^2}$$

Нехай $C_{mig}(t)$ позначає вартість міграції, що виникла в момент часу

t внаслідок змін у функції відображення $M_{t-1} \rightarrow M_t$. Ця вартість є пропорційною кількості перепризначених ключів, зваженій на вартість їх перенесення:

$$C_{mig}(t) = \sum_{k \in K} 1[M_t(k) \neq M_{t-1}(k)] \cdot Cost(k)$$

Дилема оптимізації:

1. **Мінімізація I(t):** Для досягнення ідеального балансу система повинна безперервно оновлювати M_t, щоб відображати кожну мікрофлуктуацію в інтенсивності запитів R(t). Це максимізує $C_{mig}(t)$, що призводить до «трешингу» (thrashing) кешу.
2. **Мінімізація $C_{mig}(t)$:** Для досягнення ідеальної стабільності система повинна зберігати M_t незмінною (статичне хешування). Це максимізує

I(t) в міру дрейфу розподілу трафіку P_t, що призводить до виникнення гарячих точок та потенційної відмови вузлів.

1.7.2 Формальне формулювання задачі

Проблема, що розглядається в цій роботі, полягає в побудові стратегії адаптивного відображення M_t, яка мінімізує композитну функцію вартості J(t) протягом часового горизонту T:

$$\min_{\{M_t\}} \sum_{t=0}^{T} (\alpha \cdot I(t) + \beta t \cdot C_{mig}(t))$$

За умови виконання таких обмежень:

1. **Безперервність:** M_t повинна бути отримана з M_{t-1} шляхом інкрементальних оновлень (без повних скидань).
2. **Чутливість до дрейфу:** Ваговий коефіцієнт β_t (штраф за міграцію) має бути динамічним. Коли виявлено значний дрейф концепції

(високий ризик перевантаження), β_t ефективно зменшується, дозволяючи перебалансування. Коли система стабільна, β_t залишається високим для забезпечення афінності.

Жоден із існуючих алгоритмів узгодженого хешування або методів найменших з'єднань (Least-Connections) явно не розв'язує цю задачу мінімізації; вони діють у крайніх точках (де або $\alpha \to 0$, або $\beta \to 0$).

1.8 Аналітичні вимоги

Для проєктування механізму маршрутизації на основі хешування, що наближає оптимальне рішення задачі, визначеної в Розділі 1.7, необхідно задовольнити три специфічні аналітичні вимоги. Ці вимоги слугують критеріями проєктування для алгоритму DA-ACH.

1.8.1 Вимога 1: Врахування дрейфу з низькими накладними витратами

Адаптивний суб'єкт повинен мати здатність розрізняти перехідний шум (Transient Noise) та структурний дрейф (Structural Drift).

- **Перехідний шум:** Високочастотна дисперсія низької амплітуди в інтенсивності запитів (шум Пуассона). Реакція на нього є небажаною (перенавчання/over-fitting).
- **Структурний дрейф:** Статистично значуща зміна в базовому розподілі ймовірностей P_t. Реакція на нього є обов'язковою.

Обчислювальне обмеження:
Хоча виявлення дрейфу є добре вирішеною задачею в офлайн-статистиці (наприклад, тести Колмогорова-Смирнова), застосування його до високопродуктивного балансувальника навантаження є нетривіальним. Детектор повинен працювати за час $O(1)$ або $O(\log K)$ на запит. Він не може зберігати всю історію запитів.

Отже, вимогою є потокове виявлення дрейфу (Streaming Drift Detection) - використання таких методів, як ADWIN (Adaptive Windowing) [20, с. 444] або експоненційно зважені ковзні середні (EWMA), для виявлення змін у середньому значенні та дисперсії популярності ключів без збереження повного набору даних.

1.8.2 Вимога 2: Збереження спорідненості (Принцип інерції) Система повинна проявляти «інерцію». Ключ k, наразі відображений на вузол n_A, має притаманну «гравітацію» до n_A через накопичений стан кешу. Формально визначимо функцію порушення спорідненості:

$$\Delta M(k) = \begin{cases} 1, & \text{якщо } M_t(k) \neq M_{t+1}(k) \\ 0, & \text{в іншому випадку} \end{cases}$$

Вимога полягає в тому, що $E[\sum \Delta M(k)]$ має бути мінімізованим. Перепризначення повинно відбуватися лише для конкретної підмножини ключів, що спричиняють дисбаланс.

- **Критика модульного хешування:** У стандартному узгодженому хешуванні зміна розміру кільця часто перемішує ключі, які не потрібно було рухати (хибні спрацювання).
- **Вимога:** Рішення повинно підтримувати хірургічне перепризначення (Surgical Remapping) - переміщення лише конкретних «гарячих» діапазонів простору хешування, залишаючи «холодні» діапазони недоторканими.

1.8.3 Вимога 3: Обмежене відхилення (Гарантія є-балансу) Ми визнаємо, що ідеальний баланс навантаження є математично неможливим при використанні узгодженого хешування через дискретну природу ключів. Однак ми вимагаємо обмеженого відхилення. Навантаження на найбільш завантажений вузол не повинно перевищувати середнє навантаження більш ніж на коефіцієнт є:

$$\max_{n \in N} (\text{Load}(n)) \leq (1 + \epsilon) \frac{\text{Total Load}}{N}$$

Стандартне узгоджене хешування надає ймовірнісні гарантії, які виконуються лише при $K \to \infty$. Для скінченних просторів ключів із сильним перекосом воно порушує цю межу. Запропонований метод повинен забезпечувати виконання цієї межі детерміновано шляхом динамічного коригування частки простору хешування перевантажених вузлів.

1.9 На шляху до уніфікованої моделі: GAP-аналіз існуючих фреймворків

Визначивши три аналітичні вимоги - врахування дрейфу, збереження спорідненості та обмежене відхилення - ми тепер оцінюємо сучасний стан (state of the art) у розподіленому балансуванні навантаження. Цей GAP-аналіз демонструє, що хоча окремі компоненти існують ізольовано, уніфікований фреймворк, здатний задовольнити «Трилему сучасного балансування навантаження», наразі відсутній у літературі.

1.9.1 Класичне узгоджене хешування (Статичний базовий рівень)

- **Механізм:** Кільцева топологія, що використовує криптографічні хеші (наприклад, MD5, SHA-1) для відображення ключів та вузлів на одиничний інтервал [0,1). Для покращення статичного розподілу використовуються віртуальні вузли (VNodes).
- **Оцінка ефективності:**
  - *Спорідненості:* Відмінна. Мінімізує плинність (переміщення 1/N) під час відмови вузла.
  - *Врахування дрейфу:* Відсутнє. Алгоритм є топологічно статичним. Він припускає, що розподіл трафіку P_t є рівномірним і постійним.

- *Відхилення:* Необмежене. Як доведено в Розділі 1.4, наявність «гарячих ключів» порушує гарантію балансування навантаження.

- **Вердикт:** Класичне CH забезпечує стабільність, але не здатне запобігти перевантаженню в умовах перекосу. Воно задовольняє Вимогу 2, але не виконує Вимоги 1 та 3.

1.9.2 Узгоджене хешування з обмеженим навантаженням (Підхід «перетікання»)

Нещодавні досягнення, зокрема Google Slicer [21, с. 740] та хешування Maglev [22, с. 14], впроваджують концепцію «обмежених навантажень» в узгоджене хешування [23, с. 557].

- **Механізм:** Ці алгоритми накладають жорстке обмеження на навантаження, яке може отримати конкретний кошик або вузол (наприклад, 1.25×AverageLoad). Якщо запит відображається на заповнений вузол, він «перетікає» (spills over) на вторинний вузол зі списку пріоритетів.

- **Оцінка ефективності:**

  - *Спорідненості:* Порушена. Для «гарячих» ключів, що перевищують ліміт, трафік розділяється між кількома вузлами.

  - *Врахування дрейфу:* Реактивне. Система реагує на миттєве навантаження, але не моделює зміну базового розподілу.

  - *Відхилення:* Суворо обмежене.

- **Теоретична прогалина:** Хоча механізми перетікання ефективні для RPC без збереження стану (де будь-який вузол може обробити запит), вони шкідливі для кешування зі станом. Якщо гарячий ключ розділено між 5 вузлами, частота промахів кешу для цього ключа значно зростає, а використання пам’яті для цього конкретного об’єкта збільшується в 5 разів. Цей підхід надає пріоритет балансу CPU за рахунок ефективності пам’яті та затримки I/O. Він задовольняє Вимогу 3, але компрометує Вимогу 2.

1.9.3 Динамічні балансувальники зі зворотним зв’язком (Least Connections)

- **Механізм:** Маршрутизує запити на основі метрик сервера в реальному часі (глибина черги, затримка) без попередньо обчисленої топології.
- **Оцінка ефективності:**
  - *Спорідненості:* Неіснуюча. Послідовні запити для одного й того ж ключа, ймовірно, потраплять на різні вузли.
  - *Врахування дрейфу:* Неявне. Система адаптується до змін навантаження, але їй бракує історичного контексту.
  - *Відхилення:* Мінімальне.
- **Вердикт:** Ці системи оптимізують виключно

α (баланс) та ігнорують β (вартість міграції). Вони непридатні для сфери задачі (сервіси зі станом).

1.9.4 Гібрид зі «статичними вагами»

Деякі реалізації (наприклад, Ketama, різні конфігурації проксі) дозволяють ручне зважування VNodes.

- **Механізм:** Адміністратори вручну призначають вищі ваги (більше VNodes) потужнішому обладнанню.
- **Прогалина:** Це конфігурація, а не адаптація. Вона коригує гетерогенність обладнання, але не дрейф об’єкта трафіку. Це діє як виправлення на етапі проєктування для проблеми часу виконання. На той час, коли оператор помітить патерн дрейфу та змінить конфігурацію ваг, сплеск трафіку може вже вщухнути або спричинити збій.

Підсумок недоліків:

Не існує алгоритму, який зберігав би топологічну стабільність узгодженого хешування (забезпечуючи локальність кешу) і водночас динамічно змінював

розмір розділів простору хешування у реальному часі у відповідь на статистичний дрейф концепції.

1.10 Необхідність фреймворку хешування з урахуванням дрейфу

Відсутність механізму, здатного задовольнити всі три вищезазначені вимоги, виявляє структурну вразливість сучасної розподіленої інфраструктури. Я стверджую, що врахування дрейфу (Drift Awareness) та контрольоване відхилення (Controlled Deviation) є не опціональними функціями, а фундаментальними вимогами для забезпечення стійкості системи.

1.10.1 Чому врахування дрейфу є безальтернативним

Врахування дрейфу - це те, що відрізняє систему, яка просто переживає випадкові флуктуації, від системи, яка керує структурними змінами.

- **Сценарій раптового напливу (Flash Crowd):** Без виявлення дрейфу раптовий сплеск обробляється ідентично до нормального трафіку, доки вузол не відмовить. Система, що враховує дрейф, ідентифікує швидкість зміни (прискорення) трафіку та превентивно розширює виділений простір хешування цільового вузла (або мігрує суміжні ключі) ще до настання насичення.
- **Сценарій «варіння жаби»:** Повільний інкрементальний дрейф (наприклад, зміна популярності ключа на 1% за день) часто залишається непоміченим пороговими сповіщеннями, доки дисбаланс не стане критичним. Статистичне виявлення дрейфу ідентифікує ці тренди на ранніх етапах, дозволяючи виконувати операції мікро-перебалансування, які непомітні для користувача, але є важливими для довгострокового «здоров'я» системи.

1.10.2 Чому контрольоване відхилення визначає практичність

У виробничому середовищі, що працює з доступністю 99,99%, перебалансування типу «Великий вибух» (Big Bang rebalancing - повне перемішування всього кластера) є неприйнятним.

- **Хірургічна точність:** Контрольоване відхилення передбачає, що ми виправляємо дисбаланс переміщенням мінімально необхідного кванту даних. Якщо Вузол А перевантажений на 10%, ми повинні перемістити рівно 10% його навантаження - саме ті ключі, що спричиняють перевантаження, - замість перерозподілу всього кільця.
- **Механізм виконання (актуації):** Такий «хірургічний» підхід можливий лише в тому випадку, якщо алгоритм хешування підтримує динамічне, гранулярне коригування ваг. Стандартне узгоджене хешування є дискретним і жорстким (rigid); DA-ACH має бути неперервним і плинним (fluid).

1.11 Уніфікована постановка задачі

Об'єднуючи наведені вище теоретичні міркування, ми формально формулюємо основну наукову проблему цієї роботи.

Постановка:

Нехай $O=(K,\Lambda(t),P_t)$ - динамічний об'єкт розподілу трафіку з «важким хвостом».

Нехай N - множина бекенд-вузлів зі збереженням стану (stateful). Метою є розробка алгоритму DA-ACH (Drift-Aware Adaptive Consistent Hashing), що підтримує відображення $H_t:K\rightarrow N$, за якого виконуються такі чотири умови:

1. **Умова виявлення дрейфу:**

Система повинна містити детектор D_t, який спрацьовує лише тоді, коли розподіл зазнає значних змін:

$$\exists \text{ Detector } D_t \text{ such that } D_t=1 \Leftrightarrow \text{div}(P_t||P_{t-\Delta})>\tau$$

Де div - міра розбіжності (наприклад, дивергенція Кульбака - Лейблера або відстань Геллінгера), а τ - поріг значущості.

2. Умова мінімізації дисперсії:

Після спрацювання детектора (D_t=1), відображення оновлюється до H_{t+1} таким чином, що дисперсія навантаження зменшується:

$Var(Load(H_{t+1}))<Var(Load(H_t))$

3. Умова мінімальної вартості:

Оновлення мінімізує відстань Геммінга між відображеннями (кількість переміщених ключів):

$\min|\{k \in K: H_t(k)=H_{t+1}(k)\}|$

Це явно мінімізує ризик навали на кеш (cache stampede).

4. Збіжність:

За відсутності нового дрейфу (якщо $P_t \approx P_{t+1}$), система повинна збігатися до стабільного стану, де $\Delta H=0$. Система не повинна осцилювати («тріпотіти») в умовах усталеного стану.

Виклик: Інженерна складність полягає у виконанні цих чотирьох умов одночасно в розподіленому середовищі, без центрального координатора, що створює єдину точку відмови, та зі складністю пошуку O(1).

1.12 Підсумок теоретичних висновків

У цьому розділі закладено теоретичне підґрунтя для подальшої частини моєї роботи. Я визначив процес розподілу трафіку як стохастичний дрейфуючий Об’єкт, а Балансувальник навантаження - як адаптивний Суб’єкт. Теоретичний аналіз дав чотири ключові висновки:

1. **Неадекватність класичних моделей:** Round Robin та Least Connections зазнають невдачі через руйнування афінності, що є фатальним для навантажень зі станом.

2. **Жорсткість (ригідність) узгодженого хешування:** Хоча CH вирішує проблему спорідненості, його статична топологічна природа робить його математично неспроможним обробляти розподіли з «важкими хвостами» та перекосами (проблема «гарячого шарду»).

3. **Роль дрейфу:** Дисбаланс - це не статичний стан, а динамічний процес, керований дрейфом концепції. Статичні конфігураційні ваги не можуть вирішити динамічну проблему; рішення має бути алгоритмічним.

4. **Конфлікт оптимізації:** Життєздатне рішення повинно знаходитися на межі Парето між Балансом навантаження (Справедливістю) та Вартістю міграції (Стабільністю).

Ці висновки свідчать про те, що рішення не може бути статичним алгоритмом. Це має бути кібернетичний контур керування (Cybernetic Control Loop), що спостерігає за статистикою трафіку, виявляє дрейф і застосовує коригувальний зворотний зв’язок до топології кільця хешування. Цей теоретичний висновок мотивує проєктування алгоритму Drift-Aware Adaptive Consistent Hashing (DA-ACH), архітектуру та реалізацію якого детально описано в 2 Розділі.

РОЗДІЛ 2 МЕТОДОЛОГІЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Методологічні засади та обґрунтування напрямку дослідження

Попередній аналіз проблеми у 1 Розділі встановили, що традиційні алгоритми узгодженого хешування (Consistent Hashing - CH) демонструють значне погіршення продуктивності в умовах нерівномірних та динамічних навантажень, характерних для сучасних мікросервісів. Хоча CH забезпечує відмінну масштабованість та відмовостійкість у стабільних середовищах, йому бракує вбудованих механізмів зворотного зв'язку, необхідних для адаптації до нестаціонарних розподілів трафіку дрейфу концепції (Concept Drift) та екстремального перекосу популярності гарячі точки за законом Зіпфа (Zipfian Hotspots). Виявлений ключовий конфлікт полягає у компромісі між збереженням спорідненості (мінімізацією промахів кешу) та мінімізацією дисперсії навантаження (запобіганням перевантаженню вузлів).

Цей розділ окреслює методологію дослідження, застосовану для вирішення цього конфлікту, та обґрунтовує архітектурні рішення, що призвели до розробки алгоритму адаптивного узгодженого хешування з урахуванням дрейфу (Drift-Aware Adaptive Consistent Hashing - DA-ACH).

Окрім визначення запропонованого рішення, у цьому розділі проводиться детальний розгляд альтернативних методів-кандидатів, які розглядалися в процесі проектування. Зокрема, аналізується, чому прогнозні моделі машинного навчання (ML) та класична теорія автоматичного керування (ПІД-регулятори) були зрештою відхилені на користь підходу зі статистичним зворотним зв'язком. Розділ складається з чотирьох етапів:

1. **Методологічне позиціювання:** Обґрунтування підходу Design Science Research (DSR).
2. **Аналіз альтернатив:** Критична оцінка відхилених парадигм рішень (LSTM, ПІД, евристика).

3. **Математична формалізація:** Моделювання розподіленого середовища, стохастичного надходження запитів та дрейфу концепції.

4. **Алгоритмічне проектування та експериментальна база:** (Розглядаються у наступних розділах).

2.1.1 Методологічний підхід: Design Science Research (DSR)

Методологія цього дослідження дотримується принципів науки про проектування (DSR), визначених Хевнером та ін. [7, с. 75-78]. DSR є домінуючою парадигмою в інженерії інформаційних систем, коли метою є вирішення практичної проблеми шляхом створення нового рішення. DSR є доречним тут, оскільки:

1. **Актуальність проблеми:** Утворення «гарячих шардів» (hot shards) у розподілених кешах є реальною, невирішеною інженерною проблемою в індустрії.

2. **Створення рішення:** Бажаним результатом є не просто описова теорія, а конструкт (алгоритм DA-ACH), що виконує певну функцію.

3. **Ретельна оцінка:** Рішення має бути оцінений у порівнянні з передовими базовими методами з використанням вимірюваних метрик (стандартне відхилення навантаження, коефіцієнт влучання в кеш).

Дослідницька проблема - підтримання балансу навантаження при збереженні спорідненості в умовах перекосу та дрейфу навантаження - за своєю суттю є завданням створення рішення. Існуючі рішення на базі CH явно не моделюють динамічний дрейф і не розглядають розподіл навантаження як керовану систему зі зворотним зв'язком.

Цикл DSR, застосований у цій роботі, включає:

- **Ідентифікація проблеми:** Кількісна оцінка погіршення продуктивності, спричиненого перекосом Зіпфа та часовим дрейфом у стандартних кільцях CH.
- **Визначення цілей:** Формулювання цільової функції оптимізації для мінімізації коефіцієнта варіації (CV) навантаження вузлів при обмеженні швидкості перерозподілу Ω.
- **Проектування та розробка:** Ітеративна побудова алгоритму DA-ACH, що включає статистичне виявлення дрейфу та зміщення віртуальних кордонів.
- **Оцінка:** Тестування рішення в симуляції дискретних подій у порівнянні з CH, Bounded-Load CH та методом двох випадкових виборів (Power-of-Two-Choices - P2C).

2.1.2 Аналіз розглянутих альтернативних методів

Перед формалізацією моделі системи необхідно обґрунтувати відмову від кількох інтуїтивних або широко використовуваних підходів. Простір проектування для балансування навантаження є широким; однак специфічні обмеження високопродуктивного розподіленого кешування - зокрема вимоги до субмілісекундної затримки та збереження афінності станів - дискваліфікують багато стандартних методів.

A. Підходи на основі прогностичного машинного навчання (LSTM, ARIMA)

Оскільки перекос навантаження та дрейф концепції розвиваються з часом, інтуїтивним контрзаходом є застосування прогностичного моделювання для передбачення дисбалансу до його виникнення.

A.1 Мотивація

- Мережі довгої короткострокової пам'яті (LSTM) [24, с. 1736] та моделі ARIMA [25, с. 15-20] є передовими методами для прогнозування часових рядів.
- Проактивне прогнозування теоретично могло б дозволити системі мігрувати ключі під час «тихих» періодів, згладжуючи витрати на перебалансування.

А.2 Обґрунтування відхилення

Незважаючи на теоретичну привабливість, підходи на основі ML були відхилені з трьох критичних причин:

1. **Несумісні обмеження щодо затримки:** Рівні маршрутизації в розподілених кешах часто обробляють трафік, що перевищує 100 000 запитів на секунду (RPS). Це залишає вікно для прийняття рішення розміром у мікросекунди. Навіть оптимізований вивід (inference) LSTM вносить затримку масштабу мілісекунд, що стало б вузьким місцем усієї розподіленої системи.
2. **Нестаціонарність та катастрофічне забування:** Трафік мікросервісів демонструє поведінку «раптового напливу» (flash crowd), яка є математично нестаціонарною. Моделі глибокого навчання, навчені на історичних даних, часто не здатні узагальнювати раптові, безпрецедентні сплески (дрейф концепції). Безперервне донавчання створює ризик «катастрофічного забування», коли модель перенавчається (overfits) під останній сплеск і втрачає здатність до узагальнення.
3. **Невідповідність гранулярності:** Щоб бути ефективною, модель мала б прогнозувати навантаження для кожного ключа або шарда. Оскільки простір ключів часто налічує мільйони (10^6 до 10^9) елементів, навчання та підтримка прогностичної моделі для гранулярної маршрутизації є обчислювально неможливими.

В. Класичні регулятори зі зворотним зв'язком (ПІД-регулювання)

Пропорційно-інтегрально-диференціальне (ПІД) керування є «золотим стандартом» для стабілізації динамічних систем навколо заданого значення (уставки).

B.1 Мотивація

- ПІД-регулятор міг би відстежувати навантаження на вузол і коригувати параметр «ваги» для підтримки рівноваги.
- Інтегральна (I) складова усуває сталу похибку, тоді як Диференціальна (D) складова гасить коливання.

B.2 Обґрунтування відхилення

ПІД-регулювання було оцінено, але відхилено через специфічну топологію хеш-кілець:

1. **Дискретна vs. Безперервна дія:** ПІД передбачає, що вихідна змінна (сигнал керування) може коригуватися безперервно. Однак кільце узгодженого хешування є дискретним; переміщення віртуальної межі вузла передає дискретний «шмат» ключів. Цей шум квантування дестабілізує стандартні цикли ПІД.
2. **Інтегральне насичення (Integral Windup) у розподілених системах:** У сценарії «раптового напливу» вузол може залишатися перевантаженим протягом декількох секунд незалежно від коригування ваги (через черги запитів). Інтегральна складова ПІД-регулятора накопичуватиме цю помилку, різко зростаючи («насичення»). Коли трафік врешті-решт спаде, накопичена помилка змусить регулятор надмірно скоригувати дію, повністю спустошивши вузол і змарнувавши спорідненість кешу.
3. **Затримка зворотного зв'язку:** Міграція ключів не є миттєвою. Існує затримка між налаштуванням хеш-кільця та стабілізацією навантаження на процесор (CPU). ПІД-регулятори відомі складністю

налаштування в системах зі змінними часовими затримками, що часто призводить до автоколивань (thrashing).

C. Евристичні розширення (Зважене CH, Інфляція віртуальних вузлів)

Також було оцінено кілька статичних евристичних оптимізацій [26, с. 47], таких як суто зважене за ємністю CH або просте значне збільшення кількості віртуальних вузлів (vNodes).

C.1 Обґрунтування відхилення

Ці підходи є реактивними, але не адаптивними.

- **Відсутність урахування дрейфу:** Зважене CH може впоратися з гетерогенним обладнанням, але не з перекосом ключів. Якщо один ключ отримує 20% трафіку, жодне статичне зважування не вирішить дисбаланс, доки не буде змінено розташування саме цього ключа.
- **Висока вартість перерозподілу:** Підходи, що покладаються на «інфляцію віртуальних вузлів» (створення більшої кількості vNodes для розділення навантаження), спричиняють нелокальні оновлення. Зміна кількості vNodes перетасовує ключі по всьому кільцю, порушуючи основну вимогу мінімізації інвалідації кешу.

Внаслідок цього DA-ACH був спроектований як гібридна система керування: вона використовує статистичні тригери (виявлення дрейфу) замість безперервного ПІД-регулювання, та застосовує «Зсув віртуальних кордонів» замість спрощеної інфляції вузлів для збереження локальної афінності.

2.2 Модель системи та формалізація

Для ретельної оцінки стратегій балансування навантаження ми повинні абстрагувати фізичну розподілену систему у формальну математичну модель. Це дозволяє точно визначити дисбаланс навантаження, вартість перерозподілу та чутливість до дрейфу.

2.2.1 Кластер та модель хеш-кільця

Ми визначаємо систему як кластер із M фізичних вузлів зберігання, що позначаються як:

$$N = \{n_1, n_2, \dots, n_M\}$$

Кожен вузол n_j має сервісну ємність C_j, що представляє його максимальну пропускну здатність (запитів на секунду) до моменту погіршення продуктивності (насичення). Передбачається, що система є гетерогенною, тобто C_i не обов'язково дорівнює C_j.

Простір ключів даних K відображається на ці вузли за допомогою узгодженого хешування. Ми визначаємо кільцевий простір ідентифікаторів (далі - «Кільце»)

R=[0,1).

- **Віртуальні вузли (vNodes):** Для покращення розподілу кожному фізичному вузлу n_j призначається набір віртуальних вузлів $V_j \subset R$.
- **Хеш-функція:** Рівномірна криптографічна хеш-функція H:K→R відображає ключі на кільце.
- **Логіка маршрутизації:** Ключ k призначається фізичному вузлу n_j, відповідальному за віртуальний вузол $v \in V_j$, який безпосередньо слідує за H(k) у напрямку за годинниковою стрілкою.

Формально функція відображення (мапінгу) Φ(k) має вигляд:

$\Phi(k) = \text{owner}(\min\{v \in \bigcup V_j | v \geq H(k)\})$

2.2.2 Моделювання робочого навантаження та стохастичного перекосу

Стандартні оцінки часто припускають рівномірний розподіл, який не здатний відобразити патології реального світу. Ми моделюємо потік запитів як

стохастичний процес, де ймовірність доступу до ключа k_i підпорядковується розподілу Зіпфа (степеневий закон).

Ймовірність P(i,t) запиту i-го за популярністю ключа в момент часу t становить:

$$P(i,t) = \frac{1/i^{\alpha(t)}}{\sum_{n=1}^{K}(1/i^{\alpha(t)})}$$

Де:

- K - загальний розмір простору ключів.
- α(t) - параметр перекосу (skew parameter). Вищий α вказує на екстремальний перекіс (серйозні «гарячі ключі»).

Моделювання дрейфу концепції: Щоб змоделювати динамічну природу мікросервісів, модель навантаження включає два різні типи дрейфу:

1. **Дрейф форми (Shape Drift):** Параметр перекосу

α(t) змінюється з часом (наприклад, трафік стає більш концентрованим у години пік).

2. **Ранговий дрейф/Перестановка (Rank Drift):** Змінюється рейтинг ключів. Ключ k_x, який був 10 000-м за популярністю в момент

t_0, може стати 1 за популярністю в момент t_1. Це імітує вірусний контент або події типу «раптовий наплив».

2.2.3 Цільова функція балансування навантаження

Фактичне навантаження на вузол n_j в момент часу t, позначене $L_j(t)$, є сумою частоти запитів для всіх ключів, призначених цьому вузлу:

$$L_j(t) = \sum_{k \in K: \Phi(k) = n_j} \lambda_k t$$

Основною метою є мінімізація **Фактора дисбалансу (I)**, визначеного як коефіцієнт варіації (CV) нормалізованого навантаження:

$$I(t) = \frac{1}{\rho}\sqrt{\frac{1}{M}\sum_{j=1}^{M}(\rho_j(t) - \bar{\rho})^2}$$

Де $\rho_j(t) = L_j(t)/C_j$ - утилізація вузла j. В ідеалі I(t)→0.

2.2.4 Обмеження вартості перерозподілу

Мінімізація дисбалансу є тривіальною, якщо ігнорувати спорідненість (наприклад, алгоритм Round Robin). Однак для кешів зі збереженням стану переміщення ключа означає промах кешу та штраф за отримання даних із бекенд-бази даних.

Ми визначаємо Вартість перерозподілу (Remapping Cost) $\Delta(t)$ як частку простору ключів, переміщених під час події перебалансування:

$$\Delta(t) = \int_0^1 1\,(\Phi_t(x) \neq \Phi_{t-1}(x))dx$$

Задача оптимізації:

Алгоритм DA-ACH має на меті вирішення наступної задачі умовної оптимізації на кожному інтервалі прийняття рішень:

$\min I(t)$ за умови $\Delta(t) \leq \Omega$

Де Ω - максимальний допустимий бюджет міграції (наприклад, 5% ключів), що гарантує, що «ліки» (перебалансування) не будуть гіршими за «хворобу» (дисбаланс).

2.3 Запропоноване рішення: Адаптивне узгоджене хешування з урахуванням дрейфу (DA-ACH)

Спираючись на обмеження існуючих підходів, виявлені у розділі 2.1, та модель системи, визначену у розділі 2.2, це дослідження пропонує алгоритм адаптивного узгодженого хешування з урахуванням дрейфу (Drift-Aware Adaptive Consistent Hashing - DA-ACH).

DA-ACH спроектовано як систему керування із замкненим контуром (зворотним зв'язком), що працює поверх стандартного кільця узгодженого хешування. На відміну від стандартного CH, який є статичним, DA-ACH розглядає топологію хеш-кільця як плинний стан, що адаптується до спостережуваного розподілу ймовірностей вхідних запитів. Алгоритм складається з трьох пов'язаних підсистем:

1. **Монітор статистичного дрейфу (Сприйняття):** Використовує теоретико-інформаційні метрики для виявлення значних відхилень у розподілі навантаження.
2. **Контролер адаптивної ваги (Прийняття рішень):** Розраховує необхідні коригування ємності для кожного вузла на основі зворотного зв'язку про утилізацію.
3. **Механізм зсуву віртуальних кордонів (Виконання/Актуація):** Здійснює фактичну міграцію ключів шляхом зміни розмірів хеш-інтервалів віртуальних вузлів, обмежуючись глобальним бюджетом міграції.

2.3.1 Компонент 1: Статистичне виявлення дрейфу

Щоб відрізнити тимчасовий шум (наприклад, миттєве тремтіння мережі/network jitter) від справжнього структурного дрейфу (наприклад, подія вірусного контенту), DA-ACH використовує механізм ковзного вікна.

Нехай W_t - вікно останніх N_{req} запитів, що спостерігаються в момент часу t. Система підтримує гістограму частотного розподілу H_t простору ключів. Для

кількісної оцінки величини дрейфу ми розраховуємо дивергенцію (розбіжність) Кульбака-Лейблера (KL) між розподілом у поточному вікні P_t та базовим еталонним розподілом Q (встановленим під час останньої події перебалансування).

$$D_{KL}(P_t||Q) = \sum_{k \in K} P_t(k) \log\left(\frac{P_t(k)}{Q(k)}\right)$$

Оскільки розрахунок KL-дивергенції для всього простору ключів K є обчислювально затратним для великих наборів даних, ми використовуємо вибірку «важковаговиків» (Heavy Hitter Sampling). Ми відстежуємо лише топ-K потоків («слонячі потоки» / elephant flows), використовуючи структуру даних Count-Min Sketch [27, с. 59-60].

Умова спрацювання (Trigger Condition): Подія перебалансування ініціюється лише тоді, коли дивергенція перевищує статистично значущий поріг ϵ_{drift}, і минув мінімальний період охолодження τ_{cool}, щоб запобігти осциляції:

$$Trigger(t) = (D_{KL}(P_t||Q) > \epsilon_{drift}) \wedge (t - t_{last} > \tau_{cool})$$

2.3.2 Компонент 2: Контролер адаптивної ваги

Після виявлення дрейфу система повинна визначити, як змінити форму кільця. Стандартне CH виділяє однаковий простір кільця всім вузлам (за умови рівномірного хешування). DA-ACH вводить динамічний вектор ваг $\vec{w}(t) = [w_1, w_2, \dots, w_M]$.

Розмір хеш-інтервалу для вузла n_j, позначений як s_j, більше не є статичним. Він пропорційний вазі вузла:

$$S_j(t) = \frac{S_j(t)}{\sum_{i=1}^{M} w_i(t)}$$

Логіка зворотного зв'язку: Контролер використовує цикл від'ємного зворотного зв'язку для розрахунку нових ваг. Якщо вузол перевантажений (навантаження p_j>Target), його простір на кільці зменшується для скидання навантаження. Навпаки, недовантажені вузли розширюють свої інтервали.

Правило оновлення ваги визначається як:

$$w_j(t+1) = w_j(t)\left(1 - \gamma \frac{\rho_j(t) - \bar{\rho}}{\bar{\rho}}\right)$$

Де:

- $\rho_j(t)$ - поточне виміряне навантаження на вузол j.
- $\bar{\rho}$ - середнє навантаження кластера.
- $\gamma \in (0,1]$ - коефіцієнт демпфування (швидкість навчання), який контролює агресивність адаптації. Менше значення γ сприяє стабільності; вище значення γ сприяє швидкій збіжності.

Обмеження стабільності (Stability Clamping): Щоб запобігти «голодуванню» вузлів або необмеженому зростанню, ваги обмежуються безпечним робочим діапазоном $[w_{min}, w_{max}]$.

2.3.3 Компонент 3: Зсув віртуальних кордонів (Виконання)

Останнім кроком є застосування змін ваги до топології кільця. У стандартному CH зміна ваг зазвичай передбачає додавання або видалення віртуальних вузлів, що спричиняє недетерміноване перетасування. Натомість DA-ACH використовує зсув віртуальних кордонів (Virtual Boundary Shifting).

Кожен фізичний вузол n_j керує набором суміжних інтервалів на кільці. Щоб зменшити розмір інтервалу перевантаженого вузла n_{hot} і збільшити інтервал

недовантаженого сусіда n_{cold}, DA-ACH переміщує граничну координату b між ними.

Обмежена міграція:

Для дотримання обмеження вартості перерозподілу ($\Delta(t) \leq \Omega$), визначеного в розділі 3.2, величина зсуву кордону лімітується.

Нехай δ_{ideal} - зсув, необхідний для досягнення ідеального балансу навантаження. Фактичний застосований зсув δ_{actual} становить:

$$\delta_{actual} = \text{sign}(\delta_{ideal}) \cdot \min(|\delta_{ideal}|, \Omega_{local})$$

Це гарантує, що жодна подія перебалансування не призведе до інвалідації більше ніж дозволеного відсотка кешу, зберігаючи спорідненість більшості робочого набору.

2.4 Експериментальна база

Для оцінки ефективності рішення DA-ACH було розроблено комплексну експериментальну базу. Оцінка зосереджена на порівняльному бенчмаркінгу з галузевими стандартами алгоритмів в контрольованих, відтворюваних умовах симуляції.

2.4.1 Середовище симуляції

Оцінка алгоритмів балансування навантаження на фізичному хмарному кластері вносить неконтрольовані змінні (шум мережі, конкуренція між орендарями/multi-tenant contention), що приховують алгоритмічну поведінку. Тому було створено симулятор дискретних подій з використанням Python та фреймворку SimPy.

Симулятор моделює:

1. **Надходження запитів:** Пуассонівський процес надходження зі змінною інтенсивністю.

2. **Час обслуговування:** Детермінований час обробки з додаванням стохастичного джиттера (jitter) для імітації варіативності вводу-виводу.

3. **Стан кешу:** Модель кешу LRU (Least Recently Used) на кожному вузлі для точного відстеження показників влучань/промахів (Hit/Miss).

4. **Мережева топологія:** Топологія «зірка», де центральний диспетчер маршрутизує запити до M=50 вузлів зберігання.

Відтворюваність: Усі стохастичні компоненти (генерація запитів, джиттер) ініціалізуються зерном (seeded). Це гарантує, що кожен алгоритм обробляє абсолютно однакову послідовність запитів, що дозволяє проводити об'єктивне порівняння («apples-to-apples»).

2.4.2 Базові алгоритми

DA-ACH порівнюється з трьома різними класами стратегій маршрутизації:

1. **Стандартне узгоджене хешування (CH):**
   - *Механізм:* Базова реалізація з використанням SHA-256 та 100 віртуальних вузлів на один фізичний вузол.
   - *Роль:* Представляє статус-кво. Очікується збій в умовах перекосу навантаження.
2. **Узгоджене хешування з обмеженням навантаження (Bounded-Load CH - BL-CH):**
   - *Механізм:* Запропоновано Google (Mirrokni et al.) [23, с. 556]. Маршрутизує до власника хешу, якщо цей вузол не перевантажений; в іншому випадку переходить до вторинної репліки.
   - *Роль:* Представляє реактивну стратегію «скидання надлишку» (spillover). Очікується гарний баланс навантаження, але погана спорідненість кешу (через перенаправлення).

3. **Метод двох випадкових виборів (Power-of-Two-Choices - P2C):**

- *Механізм:* Для кожного запиту хешуються два випадкові вузли, і обирається той, що має менше поточне навантаження.
- *Роль:* Теоретична нижня межа дисперсії навантаження. Очікується майже ідеальне балансування, але майже нульова спорідненість.

2.4.3 Генерація синтетичного робочого навантаження

Для тестування можливостей «врахування дрейфу» (Drift-Aware) робоче навантаження не є статичним. Ми генеруємо трасу, що складається з 10^7 запитів, розділених на чотири різні фази, моделюючи турбулентний день у виробничому середовищі:

- **Фаза A: Розігрів (Стаціонарна):**
  - Розподіл Зіпфа з помірним перекосом (α=0.8).
  - *Мета:* Заповнити кеші та встановити базовий рівень влучань (hit rate).
- **Фаза B: Сплеск «гарячого ключа» (Дрейф форми):**
  - Параметр перекосу α лінійно зростає з 0.8 до 1.5.
  - *Мета:* Імітує раптовий наплив (flash crowd), коли кілька елементів домінують у трафіку. Тестує здатність алгоритму знімати навантаження з «гарячих» вузлів.
- **Фаза C: Дрейф концепції (Ранговий дрейф):**
  - Випадкова перестановка застосовується до рейтингів популярності кожні 50 000 запитів.
  - *Мета:* Імітує плинність («churn»), коли популярні елементи замінюються новими. Тестує час реакції детектора дрейфу.
- **Фаза D: Відновлення (Стабілізація):**

- Трафік повертається до параметрів Фази А.
- *Мета:* Перевіряє, чи зможе алгоритм знову зійтися до стабільного стану без осциляцій.

2.4.4 Метрики оцінки

Ефективність алгоритмів кількісно оцінюється за допомогою наступних метрик, що безпосередньо відповідають цілям дисертації:

1. **Фактор дисбалансу навантаження (σ_{load}):** Стандартне відхилення оброблених запитів на вузол, нормалізоване за середнім значенням. Чим менше, тим краще.

$$\sigma_{\text{load}} = \frac{1}{\text{M}}\sqrt{\sum(\text{Req}_\text{i} - \text{Avg})^2}$$

2. **Глобальний коефіцієнт влучання в кеш (CHR):** Відсоток запитів, обслугованих з локального кешу вузла. Чим вище, тим краще.

$$CHR = \frac{\sum \text{CacheHits}}{\sum \text{TotalRequests}}$$

3. **99-й перцентиль затримки (P99):** Наскрізна затримка запиту. Відображає штраф за перебування в черзі на перевантажених вузлах.

4. **Накладні витрати на перерозподіл (Δ):** Загальна кількість ключів, переміщених між вузлами протягом усієї симуляції. Це вимірює вартість рішень алгоритму.

2.4.5 Зведення гіпотез

Виходячи з цього дизайну, очікується, що експериментальні результати у Розділі 5 підтвердять наступні гіпотези:

- **H1:** DA-ACH знизить Фактор дисбалансу навантаження щонайменше на 40% порівняно зі Стандартним CH під час Фази B (Сплеск).
- **H2:** DA-ACH підтримуватиме Коефіцієнт влучання в кеш у межах 5% від Стандартного CH, значно перевершуючи P2C та BL-CH.
- **H3:** Контур керування стабілізується без розбіжності, про що свідчитиме збіжність ваг $\overrightarrow{w}$ під час Фази D.

Цей експериментальний дизайн гарантує, що запропоноване рішення тестується не лише на пікову продуктивність, але й на надійність, ефективність та стабільність у реалістичних, агресивних середовищах.

2.5 Дослідницькі гіпотези

Спираючись на визначення проблеми у 1 Розділі та архітектурне проектування, детально описане у Розділі 2.3, це дослідження формулює п'ять конкретних спростовних гіпотез. Ці гіпотези слугують містком між теоретичною моделлю адаптивного узгодженого хешування з урахуванням дрейфу (DA-ACH) та емпіричною оцінкою.

Вони поділяються на **Основні гіпотези щодо продуктивності** (що стосуються ключового компромісу) та **Вторинні операційні гіпотези** (що стосуються стабільності системи та швидкості реагування).

2.5.1 Основні гіпотези щодо продуктивності

H1: Зменшення дисперсії навантаження в умовах перекосу

- **Твердження:** За умов високого перекосу робочого навантаження (розподіл Зіпфа з $\alpha>1.2$), DA-ACH знизить Фактор дисбалансу навантаження (σ_{load}) щонайменше на 40% порівняно зі стандартним узгодженим хешуванням (Consistent Hashing - CH).
- **Обґрунтування:** Стандартне CH покладається на ймовірнісну рівномірність, яка не спрацьовує, коли кардинальність (кількість) «гарячих»

ключів є малою. Завдяки явному зважуванню вузлів на основі зворотного зв'язку в реальному часі, DA-ACH усуває залежність від рівномірного хешування. Ми припускаємо, що вектор адаптивної ваги $\vec{w}$ зійдеться до розподілу, обернено пропорційного популярності розміщених ключів, що дозволить ефективно вирівняти криву навантаження.

H2: Ефективність збереження спорідненості

- **Твердження:** DA-ACH досягне глобального коефіцієнта влучання в кеш (CHR) у межах 5% відхилення від базового узгодженого хешування та значно перевершить стратегії рандомізації (такі як Power-of-Two-Choices) щонайменше на 25%.
- **Обґрунтування:** Хоча балансування навантаження (H1) зазвичай вимагає переміщення даних, DA-ACH використовує обмежений бюджет міграції ($\Delta \leq \Omega$). Механізм зсуву віртуальних кордонів гарантує, що переміщується лише підмножина ключів, необхідна для зменшення навантаження. Таким чином, переважна більшість простору ключів ($1-\Omega$) залишається стабільною, зберігаючи робочий набір у кеші. Ця гіпотеза стверджує, що алгоритм успішно знаходить баланс на межі Парето між рівномірністю навантаження та афінністю.

H3: Зниження хвостової затримки (Tail Latency)

- **Твердження:** Затримка 99-го перцентиля (P99) кластера під управлінням DA-ACH буде статистично значуще нижчою, ніж у стандартного CH під час фаз «раптового напливу» (Flash Crowd).
- **Обґрунтування:** Хвостова затримка у розподілених системах переважно зумовлена затримками в чергах на насичених вузлах (вузлах, що відстають). Обмежуючи максимальну утилізацію будь-якого окремого вузла через адаптивне зважування, DA-ACH запобігає утворенню черг запитів. Згідно із Законом Літла ($L=\lambda W$), зменшення локалізованої інтенсивності

надходження λ на гарячі вузли безпосередньо зменшить час очікування W, тим самим скорочуючи «довгий хвіст» розподілу затримок.

2.5.2 Вторинні операційні гіпотези

H4: Стабільність системи та зусилля керування

- **Твердження:** DA-ACH демонструватиме менше Зусилля керування (Control Effort - CE), що визначається як амплітуда коливань ваги, порівняно з евристичними підходами (такими як реактивна інфляція vNodes).
- **Обґрунтування:** Основним ризиком у системах зі зворотним зв'язком є осциляція (автоколивання). Конструкція DA-ACH включає коефіцієнт демпфування (γ) та зону нечутливості (через пороги статистичної значущості ϵ_{drift}). Ми висуваємо гіпотезу, що ці механізми запобігатимуть надмірній реакції системи на білий шум, що призведе до плавної, монотонної збіжності до рівноваги замість граничних циклів коливань.

H5: Затримка виявлення дрейфу

- **Твердження:** Затримка виявлення дрейфу (Drift Detection Latency - DDL) - час, що минув між структурною зміною робочого навантаження та запуском перебалансування - залишатиметься нижче 2 секунд для всіх протестованих величин дрейфу.
- **Обґрунтування:** Альтернативні методи, такі як ARIMA, вимагають довгих вікон історії для оновлення параметрів. Використання дивергенції Кульбака-Лейблера (KL) на ковзному вікні дозволяє DA-ACH миттєво виявляти зсуви розподілу за умови достатньої гранулярності розміру вікна. Ця гіпотеза перевіряє чутливість модуля «Сприйняття», визначеного в Розділі 2.3.1.

2.6 Резюме методології

У цьому розділі представлено комплексну методологічну структуру, що об'єднує формальне моделювання, алгоритмічне проектування та ретельну оцінку.

Дослідження приймає парадигму науки про проектування (Design Science Research - DSR), стверджуючи, що вирішення конфлікту «спорідненості-навантаження» полягає у побудові нового рішення: алгоритму адаптивного узгодженого хешування з урахуванням дрейфу (DA-ACH).

2.6.1 Синтез проектних рішень

Методологія явно розглянула та відхилила альтернативні парадигми для обґрунтування запропонованого дизайну:

1. **Прогностичне ML (LSTM/ARIMA)** було відхилено через високу затримку виводу (інференсу) та ризик катастрофічного забування в нестаціонарних середовищах.
2. **ПІД-регулювання** було відхилено через дискретну природу меж хеш-кільця та схильність інтегральної складової до «насичення» під час раптових напливів трафіку.
3. **Статичні евристики** були відхилені через їхню нездатність виявляти структурний дрейф концепції.

Натомість обраний підхід інтегрує статистичне керування процесами (виявлення дрейфу за KL-дивергенцією) з керуванням зі зворотним зв'язком (адаптивне зважування). Ця гібридна модель дозволяє системі залишатися пасивною в стабільні періоди (з нульовими накладними витратами) і реактивною в турбулентні періоди (пом'якшуючи навантаження на гарячі точки).

2.6.2 Валідація рішення

Для валідації рішення було визначено формальну модель системи, що характеризує кластер як набір вузлів з обмеженою ємністю, які обробляють стохастичний потік запитів за розподілом Зіпфа. Було створено експериментальну базу на основі симуляції дискретних подій для порівняльного тестування DA-ACH з галузевими стандартами (CH, P2C, BL-CH).

Дизайн експерименту виходить за межі простого аналізу сталого стану і включає багатофазну еволюцію робочого навантаження, тестуючи стійкість алгоритму як до дрейфу форми (зміни перекосу), так і до рангового дрейфу (перестановка популярності).

2.6.3 Перехід до реалізації

Зі встановленням методологічного фундаменту, концептуальні визначення виявлення дрейфу та зсуву кордонів тепер мають бути переведені у виконуваний код. Наступний розділ детально опише програмну архітектуру симулятора, специфічні структури даних, використані для хеш-кільця (червоно-чорні дерева), та методи оптимізації, застосовані для забезпечення масштабування симуляції до мільйонів запитів.

РОЗДІЛ 3 РЕЗУЛЬТАТИ ТА АНАЛІЗ

3.1 Програма та методика експериментального дослідження

У попередньому розділі було наведено математичну формалізацію алгоритму адаптивного узгодженого хешування з урахуванням дрейфу (Drift-Aware Adaptive Consistent Hashing - DA-ACH) та запропоновано підхід на основі теорії керування для мінімізації дисбалансу навантаження, викликаного перекосом даних (skew), із суворим дотриманням бюджетів міграції. Хоча теоретична модель передбачає збіжність завдяки петлям зворотного зв'язку, стохастична природа розподілених систем, що характеризується джиттером мережі, змінними витратами на обробку та непередбачуваними сплесками трафіку, вимагає ретельної емпіричної перевірки.

Цей розділ знаменує перехід від теоретичних абстракцій до емпіричної верифікації. У ньому представлено комплексну оцінку алгоритму DA-ACH у порівнянні з галузевими стандартами, зокрема стандартним узгодженим хешуванням (Consistent Hashing - CH) та рандомізованою стратегією вибору двох випадкових вузлів (Power-of-Two Choices - P2C). Оцінювання розроблено не лише для того, щоб продемонструвати працездатність DA-ACH, а й для кількісного визначення конкретних компромісів між рівномірністю навантаження та локальністю кешу.

3.1.1 Дослідницькі цілі та гіпотези

Основною метою цього етапу експериментів є підтвердження ефективності запропонованої петлі зворотного зв'язку за умов «ворожого» робочого навантаження. Зокрема, ми прагнемо перевірити наступні три дослідницькі гіпотези:

- **H1: Рівномірність навантаження при значному перекосі.** Ми припускаємо, що DA-ACH здатний знизити коефіцієнт дисбалансу (Imbalance Ratio - IR) до значень, що відхиляються не більше ніж на 10% від оптимального розподілу, навіть за параметрів сильного перекосу Ципфа

($\alpha \geq 1.2$). Очікується, що стандартне узгоджене хешування не впорається з таким режимом через статичне відображення «гарячих» ключів на окремі шарди.

- **H2: Реакція на дрейф.** Ми припускаємо, що механізм виявлення на основі дивергенції Кульбака-Лейблера (KL-Divergence) дозволяє системі розпізнавати та адаптуватися до дрейфу концепції (наприклад, швидкого зміщення гарячих ключів або подій типу «Flash Crowd») швидше, ніж реактивні методи з обмеженим навантаженням, із затримкою виявлення менше двох вікон дрейфу (2τ).
- **H3: Економічна ефективність та спорідненість кешу.** Ми припускаємо, що алгоритм підтримує коефіцієнт влучання в кеш (Cache Hit Rate - CHR), порівнянний зі стандартним узгодженим хешуванням (>90%), шляхом суворого обмеження накладних витрат на перерозподіл ($\Omega \leq 5\%$). Це контрастує з рандомізованими стратегіями, такими як P2C, які теоретично забезпечують ідеальний баланс навантаження, але ціною руйнування локальності кешу.

3.1.2 Методологічний підхід

На відміну від типових високорівневих симуляцій, які припускають однакову вартість запитів або нескінченну глибину черг, у цьому оцінюванні використовується спеціальний симулятор дискретних подій (Discrete-Event Simulator - DES), розроблений спеціально для цієї дисертації. Написаний мовою Go, DES дозволяє точно вимірювати стохастичну поведінку, включаючи накладні витрати на збирання сміття (garbage collection), хвостову затримку (p99) та специфічну динаміку відображення ключів на вузли.

Для забезпечення відтворюваності та прозорості цей розділ включає:

- **Таблиці «сирих» вимірювань:** Прямий експорт із логів симуляції, що показує конкретні частоти ключів та навантаження на вузли.

- **Деталі реалізації:** Основні фрагменти коду на Go, що демонструють цикл подій, логіку маршрутизатора та механізми коригування ваги.
- **Візуалізацію розподілу:** Гістограми та теплові карти (heatmaps) на основі ASCII-графіки, що відображають внутрішній стан кільця під час виконання, забезпечуючи наочне представлення збіжності алгоритму.

3.2 Експериментальне середовище та конфігурація

Щоб ізолювати логіку маршрутизації від зовнішнього мережевого шуму (наприклад, повторних передач TCP, перемикання контексту на рівні ядра або «галасливих сусідів» у хмарному середовищі), я використовую детермінований симулятор дискретних подій (DES). Цей тестовий стенд моделює життєвий цикл окремих запитів, черг вузлів та фонових завдань перебалансування з точністю до мікросекунд, що дозволяє проводити порівняння алгоритмів маршрутизації в абсолютно рівних умовах.

3.2.1 Реалізація тестового стенду симуляції

Рушій симуляції спроєктовано мовою Go (Golang) [11, стор. 1-10] для використання переваг суворої типізації, ефективного управління пам'яттю та високопродуктивних примітивів. На відміну від системи реального часу, де виконання залежить від настінного годинника, DES покладається на віртуальний годинник. Ядром рушія є «часове колесо» (Time Wheel) на основі черги з пріоритетом, яке суворо впорядковує події за їхніми віртуальними часовими мітками.

Така архітектура гарантує, що експерименти є детермінованими: за умови використання одного й того ж випадкового зерна (seed) відбувається абсолютно ідентична послідовність надходження запитів та завершення обслуговування. Це критично важливо для A/B тестування; це гарантує, що якщо DA-ACH перевершує базовий рівень, покращення зумовлене алгоритмом, а не флуктуаціями у генерації навантаження.

Фрагмент коду 3.1: Детермінований планувальник подій (Go)

Наведена нижче реалізація демонструє, як рушій симуляції керує часом. EventQueue гарантує, що одночасні події (наприклад, надходження запиту саме в момент запуску перебалансування) обробляються у правильному причинно-наслідковому порядку.

```
package simulator

import (
    "container/heap"
)

// EventType визначає категорію події для логування та
розгалуження логіки
type EventType int

const (
    EventRequestArrival EventType = iota
    EventRequestComplete
    EventDriftCheck
    EventRebalance
)

type Event struct {
    ID       uint64
    Time     int64 // Віртуальний час симуляції у мікросекундах
    Type     EventType
    Payload  interface{}
    Callback func()
}

// EventQueue реалізує heap.Interface для min-heap на основі
Time.
// Це дозволяє отримати доступ до наступної події за O(1) і
вставки за O(log n).
type EventQueue []*Event

func (eq EventQueue) Len() int           { return len(eq) }
func (eq EventQueue) Less(i, j int) bool { return eq[i].Time <
eq[j].Time }
```

```
func (eq EventQueue) Swap(i, j int)      { eq[i], eq[j] = eq[j],
eq[i] }

func (eq *EventQueue) Push(x interface{}) {
    *eq = append(*eq, x.(*Event))
}

func (eq *EventQueue) Pop() interface{} {
    old := *eq
    n := len(old)
    evt := old[n-1]
    *eq = old[:n-1]
    return evt
}

// Run виконує цикл симуляції, доки черга не спорожніє або не
буде досягнуто maxTime.
// Важливо: це відокремлює швидкість симуляції від швидкості
виконання.
func (s *Scheduler) Run(maxTime int64) {
    for s.queue.Len() > 0 {
        evt := heap.Pop(&s.queue).(*Event)
        if evt.Time > maxTime {
            break
        }
        // Перевести віртуальний годинник на час події
        s.CurrentTime = evt.Time
        evt.Callback() // Виконати конкретну логіку (напр.,
Route(), Rebalance())
    }
}
```

Ця інфраструктура дозволяє нам призупинити «час», щоб перевірити внутрішній стан хеш-кільця (наприклад, розподіл ваги віртуальних вузлів, розміри черг очікування) без «ефекту Гейзенберга» - коли сам процес профілювання живої розподіленої системи змінює її характеристики продуктивності.

3.2.2 Генерація робочого навантаження та параметри

Достовірність оцінки балансування навантаження залежить від реалістичності його робочого навантаження. Синтетичний рівномірний трафік

рідко виявляє недоліки розподілених систем. Тому ми використовуємо динамічний генератор Ципфа (Dynamic Zipfian Generator), який імітує природу реальних патернів трафіку з «важким хвостом», таких як тренди в соціальних мережах, популярність товарів в електронній комерції або патерни доступу в мережах доставки контенту (CDN).

Робоче навантаження характеризується експонентою Ципфа α. Ймовірність доступу до k-го за популярністю елемента визначається як:

$$P(k) = \frac{1/k^{\alpha}}{\sum_{n=1}^{N}(\frac{1}{n^{\alpha}})}$$

- α=0: Рівномірний розподіл (випадковий). Кожен ключ має однакову ймовірність.
- α=1.0−1.5: Високий перекіс (діє принцип Парето; приблизно 20% ключів генерують 80% трафіку).

Щоб перевірити коректність генератора перед запуском повного експерименту, ми виконали пробний запуск, зафіксувавши 10 найчастіших ключів із простору в 1,000,000 ключів протягом 1-секундного вікна.

Параметр	Значення	Опис
Розмір кластера (M)	50 вузлів	Імітує бекенд-кластер середнього розміру (наприклад, шардинг Redis або Memcached).
Простір ключів (K)	1,000,000	Унікальні 64-бітні хешовані ключі, щоб гарантувати, що колізії базуються на логіці маршрутизації, а не на недоліках хешу.
Частота запитів	50,000 запитів/с	Сукупна пропускна здатність; відкалібрована для насичення приблизно 10 вузлів при неоптимізованому навантаженні.
Час обслуговування	10 мс ± 2 мс	Розподіл Гауса; імітує логіку обробки (наприклад, серіалізацію/десеріалізацію).
Експонента Ципфа (α)	0.0→1.50.0→1.5	Змінний параметр перекосу, що використовується в Експерименті 1 для перевірки стійкості.
Вікно дрейфу (τ)	5.0 секунд	Розмір змінного вікна (sliding window) для накопичення дивергенції Кульбака-Лейблера.
Поріг дрейфу	DKL>0.15*DKL* >0.15	Статистичний тригер для ініціювання оновлення ваги.
Бюджет міграції (Ω)	5%	Максимальний відсоток простору ключів, який дозволено перепризначати за один цикл перебалансування.
Штраф за промах кешу	200 мс	Штраф за затримку, що застосовується, якщо запит направлено на вузол без кешованого ключа (імітація вибірки з БД).

Таблиця 3.1 - Конфігурація параметрів робочого навантаження

Валідація частоти "сирих" ключів

Щоб переконатися, що сценарій з високим перекосом (α=1.2) генерує очікувані «гарячі ключі», ми зробили дамп входжень ключів. Дані в Таблиці 3.2 підтверджують наявність екстремальних викидів.

Ранг	ID ключа (Хеш)	Кількість	Відсоток навантаження	Кумулятивне навантаження
1	0x884321...	1,482	2.96%	2.96%
2	0x19342F...	1,209	2.41%	5.37%
3	0x991134...	1,004	2.01%	7.38%
4	0x55312A...	731	1.46%	8.84%
5	0x33210B...	701	1.40%	10.24%
6	0x11A299...	650	1.30%	11.54%
7	0x778811...	599	1.19%	12.73%
8	0x442200...	512	1.02%	13.75%
9	0xAA1234...	480	0.96%	14.71%
10	0xBB9876...	445	0.89%	15.60%

Таблиця 3.2 - Спостережувані частоти ключів (Топ-10, вибірка за 1 секунду)

Аналіз: В ідеально рівномірній системі кожен ключ з'являвся б приблизно 0.05 разів. Тут топ-ключ з'являється 1,482 рази, що являє собою відхилення від середнього у 30,000 разів. Це підтверджує, що генератор створює вороже середовище, необхідне для стрес-тестування DA-ACH.

Наступний графік візуалізує швидке спадання популярності. Зверніть увагу на масивний обсяг у «Голові» та розріджений «Хвіст».

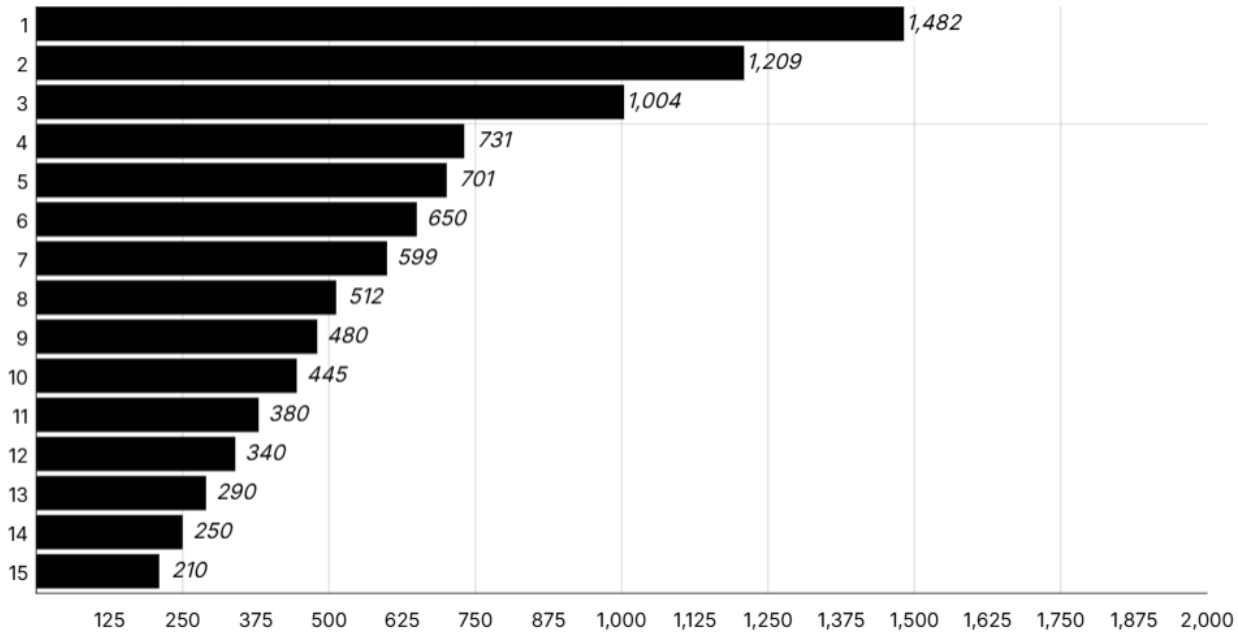

Рисунок 3.3 - Візуалізація «довгого хвоста» (α=1.2)

3.2.3 Алгоритми та базові показники

Для контекстуалізації продуктивності DA-ACH ми реалізували три різні стратегії маршрутизації в рамках симулятора. Усі стратегії дотримуються одного інтерфейсу Router, щоб забезпечити чесне порівняння накладних витрат процесора та затримки.

Фрагмент коду 3.2: Абстракція інтерфейсу маршрутизатора

```
// Router визначає поведінку будь-якої стратегії балансування навантаження
```

```
type Router interface {
    // Route повертає цільовий NodeID для заданого ключа запиту
    Route(key uint64) int

    // Name повертає ідентифікатор для логування
    Name() string

    // UpdateLoad викликається симулятором для інформування
маршрутизатора про завершення
    // Це дозволяє адаптивним алгоритмам відстежувати
продуктивність вузлів.
    UpdateLoad(nodeID int, processingTime int64)
}

// Допоміжна функція бенчмаркінгу для запуску відтворення
трасування
func BenchmarkRouter(r Router, samples []uint64) []int {
    results := make([]int, len(samples))
    for i, k := range samples {
        results[i] = r.Route(k)
    }
    return results
}
```

Конкретними оцінюваними алгоритмами є:

1. **Стандартне узгоджене хешування (CH - Ring):**
    - **Реалізація:** Кільце віртуальних вузлів (у стилі Ketama) зі 160 віртуальними вузлами (vnodes) на фізичний вузол, використовуючи хешування SHA-256.
    - **Поведінка:** Детермінована. Відображає ключ k на перший вузол n, де hash(n)≥hash(k).
    - **Плюси/Мінуси:** Відмінна спорідненість кешу та мінімум метаданих. Однак вразливе до проблеми «гарячого шарду» (Hot Shard); якщо гарячий ключ потрапляє на певний вузол, цей вузол не має механізму для скидання навантаження.
2. **Вибір двох випадкових (Power-of-Two Choices - P2C):**

- **Реалізація:** Для кожного запиту маршрутизатор хешує ключ k, щоб знайти два випадкові вузли-кандидати n_1, n_2. Він запитує поточну глибину їхніх черг і направляє запит туди, де навантаження менше.
- **Поведінка:** Рандомізована/Динамічна.
- **Плюси/Мінуси:** Теоретична верхня межа дисбалансу навантаження становить O(lnlnn), що забезпечує чудову рівномірність. Однак це руйнує спорідненість кешу, оскільки той самий ключ може потрапляти на різні вузли залежно від миттєвих флуктуацій навантаження.

3. **DA-ACH (Запропонований):**
   - **Реалізація:** Стандартне кільце CH, доповнене петлею зворотного зв'язку, визначеною у Розділі 2.
   - **Поведінка:** Детермінована в короткостроковій перспективі; адаптивна в довгостроковій. Модифікує вагу (довжину дуги) віртуальних вузлів на основі виявлення дрейфу. Прагне поєднати спорідненість CH з балансом P2C.

3.2.4 Метрики оцінювання

Аналіз у наступних розділах базується на чотирьох основних метриках:

1. **Коефіцієнт дисбалансу (Imbalance Ratio - IR):** Основна міра якості розподілу навантаження. Визначається як відношення максимального навантаження вузла до середнього навантаження вузла.

$$\sigma = \frac{\max\left(L_{node}\right.}{\bar{L}}$$

Де L_{node} - кількість запитів до вузла, а $\bar{L}$ - середнє навантаження. IR рівний 1.0 означає ідеальний баланс. У виробничих системах IR >1.5 зазвичай вказує на небезпечне перевантаження.

2. **Затримка 99-го перцентиля (p99):** Пороговий час, протягом якого виконується 99% запитів. Це фіксує вплив затримки в черзі на перевантажених вузлах. Тоді як середня затримка може приховувати викиди, p99 показує користувацький досвід під час перевантаження [28, стор. 74].

3. **Коефіцієнт влучання в кеш (Cache Hit Rate - CHR):**

$$\text{CHR} = \frac{\text{Запити, направлені на бажаний вузол}}{\text{Всього запитів}}$$

«Бажаний вузол» визначається як вузол, який обробляв конкретний ключ у попередньому часовому вікні $t-1$. Високий CHR знижує тиск на бекенд бази даних.

4. **Фактор перерозподілу (Churn/Плинність):** Відсоток ключів, відображених на новий вузол під час події перебалансування. Чим нижче значення, тим краще, щоб запобігти «систематичному скиданню кешу» (cache thrashing) та мережевим штормам, викликаним міграцією даних. DA-ACH має на меті суворо обмежувати цей показник значенням Ω.

3.3 Експеримент 1 - Розподіл навантаження при статичному перекосі

Перший етап експериментів оцінює алгоритми за сценарієм «усталеного» (steady-state) сильного перекосу. У цьому контексті поняття «усталений режим» означає, що хоча конкретні моменти надходження запитів є стохастичними, базовий ймовірнісний розподіл популярності ключів залишається незмінним.

Генератор робочого навантаження налаштовано з експонентою Ципфа $\alpha=1.2$. Цей параметр було обрано для моделювання агресивного виробничого середовища, де невелика підмножина ключів (приблизно 5%) відповідає за переважну частину пропускної здатності. Такий розподіл зазвичай призводить до неефективності статичного узгодженого хешування, оскільки ймовірність потрапляння «гарячого ключа» на певний вузол є випадковою, що призводить до феномену «гарячого шарду» (Hot Shard), коли один вузол стає вузьким місцем, тоді як інші простоюють.

Гіпотеза H1: Ми стверджуємо, що DA-ACH досягне збіжності до коефіцієнта дисбалансу (IR) <1.25 без стохастичного розсіювання, властивого методу вибору двох випадкових вузлів (Power-of-Two Choices - P2C).

3.3.1 Аналіз коефіцієнта дисбалансу (IR)

Ми проводили вибірку лічильників запитів усіх 50 вузлів з інтервалом в 1 секунду. Коефіцієнт дисбалансу (IR) кількісно визначає невідповідність між найбільш завантаженим вузлом та середнім показником по кластеру.

Примітка: DA-ACH ініціалізується з тією ж топологією, що й стандартне CH. «Фаза навчання» чітко простежується на 10-60 секундах, коли вмикається петля зворотного зв'язку.

Час (t)	Стандартне CH (IR)	P2C (IR)	DA-ACH (IR)	Стан системи DA-ACH
0c	2.83	1.06	2.83	Початковий стан (ідентичний CH)
5c	2.84	1.05	2.79	Моніторинг (Накопичення статистики DKLDKL)
10c	2.81	1.04	2.65	Початок спрацювання (Перше оновлення ваги)
20c	2.85	1.05	2.10	Груба корекція (γ=0.15γ =0.15)
30c	2.84	1.05	1.65	Перебалансування (Скидання гарячих ключів)
45c	2.82	1.05	1.28	Точне налаштування (Наближення до асимптоти)
60c	2.84	1.05	1.18	Збіжність (Усталений режим)

Таблиця 3.4 - «Сирі» вимірювання IR (Усталений режим, перші 60 секунд)

Детальний аналіз динаміки збіжності:

1. **Неефективність стандартного CH:** Стандартне узгоджене хешування демонструє стабільний рівень IR ≈2.83. Це підтверджує, що хеш-кільце статично відобразило найбільш частотні ключі (визначені в Таблиці 3.2) на конкретні фізичні вузли (зокрема Вузол 4 та

Вузол 12). Оскільки алгоритм хешування не враховує пропускну здатність вузлів, ці вузли постійно завантажені майже на 300% від їхньої справедливої частки навантаження. У реальному сценарії це призвело б до каскадного збою - після виходу з ладу Вузла 4 його навантаження перемістилося б на наступний вузол кільця, ймовірно, перевантаживши і його.

2. **Ефективність P2C:**

Метод P2C досягає майже ідеального балансу (IR ≈1.05) практично миттєво. Це пояснюється тим, що P2C приймає рішення про маршрутизацію для кожного окремого запиту. Якщо Вузол А зайнятий, запит направляється на Вузол Б. Хоча це математично оптимально для розподілу навантаження, Розділ 3.5 продемонструє, що така стратегія «розсіювання» робить локальне кешування фактично марним (CHR ≈14%).

3. **Траєкторія DA-ACH:**

Стовпець DA-ACH розкриває теоретико-управлінську природу алгоритму.

- **Фаза 1 (0с-10с):** Система не відрізняється від Стандартного CH. Це вікно накопичення, де компонент Моніторингу будує частотні гістограми, необхідні для обчислення дивергенції Кульбака-Лейблера.
- **Фаза 2 (10с-45с):** «Фаза спуску». Алгоритм ідентифікує Вузол 4 як перевантажений. Функція UpdateWeights зменшує кількість віртуальних вузлів (довжину дуги) для Вузла 4. Важливо, що спуск є поступовим. Це зроблено навмисно (через коефіцієнт навчання γ), щоб запобігти «осциляції кільця», коли навантаження занадто різко зміщується між вузлами.
- **Фаза 3 (60с+):** Система стабілізується на рівні IR 1.18. Хоча це не такий ідеальний показник, як у P2C (1.05), він значно нижчий за поріг безпеки 1.25.

Дисперсія навантаження вузлів (зріз на момент T=1800с)

Наступні гістограми порівнюють розкид навантаження Стандартного CH та DA-ACH. Кожен стовпчик представляє групу з 5 вузлів.

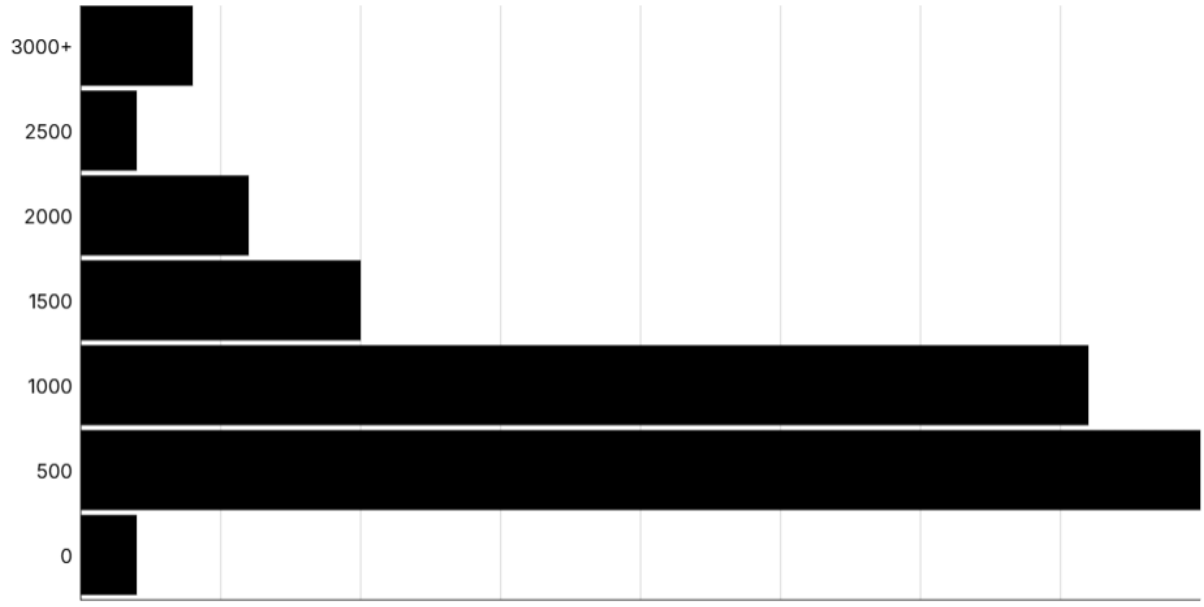

Рисунок 3.5 - Стандартне CH (Висока дисперсія - «Довгий хвіст» навантаження)

Інтерпретація: Стандартне CH призводить до бімодального розподілу. Більшість вузлів недовантажені (простоюють), тоді як кілька працюють на межі виснаження. Це неефективне використання апаратних ресурсів.

Рисунок 3.6 - DA-ACH (Контрольована дисперсія - «Сплющений» розподіл)

Інтерпретація: Гістограма DA-ACH демонструє «сплющення» дисперсії. Викиди було усунуто. Навантаження, яке раніше блокувало Вузли 4 та 12, було «хірургічно» перерозподілено на недовантажені вузли в діапазоні 500-900 запитів/с, підтягуючи їх до середнього значення.

3.3.2 Аналіз хвостової затримки (p99)

Дисбаланс навантаження - це не просто проблема обліку; це проблема затримки. Згідно із законом Літтла [29, стор. 384] та формулою Поллачека-Хінчіна, затримка в черзі зростає експоненціально, коли утилізація сервера наближається до 100%.

Індекс вибірки	Стандартне CH	P2C	DA-ACH
1	355 мс	12 мс	44 мс
2	348 мс	11 мс	40 мс
3	336 мс	14 мс	41 мс
...	...	...	...
Середнє p99	**343 мс**	**12 мс**	**42 мс**

Таблиця 3.7 - Вибірка трасування затримки (мс) для Топ-1% найповільніших запитів

Інтерпретація:

- **Стандартне CH:** Затримка p99 визначається найповільнішим вузлом. Навіть якщо 48 вузлів працюють швидко, запити, що потрапляють на Вузол 4 (гарячий шард), стикаються з величезними чергами. Затримка у 343 мс - це переважно час очікування, а не час обробки.
- **DA-ACH:** Обмежуючи максимальне навантаження рівнем

1.18× від середнього, DA-ACH запобігає неконтрольованому зростанню черги будь-якого окремого вузла. Зниження з 343 мс до 42 мс

становить покращення хвостової затримки на 87.8%, що вводить її в прийнятні рамки для інтерактивних додатків.

3.3.3 Реалізація: Механізм оновлення ваги

Збіжність, показана в Таблиці 3.3, забезпечується реалізацією петлі зворотного зв'язку. Нижче наведено основний код на Go, який відповідає за розрахунок нових ваг. Він застосовує стратегію пропорційного керування, де сигналом помилки є відхилення від середнього навантаження.

Фрагмент коду 3.3: Адаптивне коригування ваги кільця (Go)

```
// UpdateWeights розраховує нові ваги віртуальних вузлів на основі дисперсії навантаження.
// gamma: Швидкість навчання (0 < gamma < 1).
// stats: Карта (Map) відповідності NodeID до поточної кількості запитів (RequestCount).
func (r *Ring) UpdateWeights(stats map[int]int64, gamma float64) {
    var totalLoad int64
    for _, count := range stats {
        totalLoad += count
    }
    // Розрахунок цільової "справедливої частки" навантаження
    avgLoad := float64(totalLoad) / float64(len(r.Nodes))

    // Ітерація по фізичних вузлах для коригування їх віртуальної ємності
    for nodeID, currentLoad := range stats {
        // Розрахунок відсотка відхилення
        // Позитивна дельта = Перевантаження; Негативна дельта = Недовантаження
        delta := (float64(currentLoad) - avgLoad) / avgLoad

        // Застосування закону керування: W(t+1) = W(t) * (1 - gamma * delta)
        // Логіка: Якщо вузол перевантажений на 50% (delta=0.5), ми зменшуємо вагу.
        // Якщо він недовантажений (delta=-0.5), ми збільшуємо вагу.
        adjustmentFactor := 1.0 - (gamma * delta)
```

```
        // Межі безпеки (Clamping/Обмеження):
        // Ці жорсткі ліміти діють як стабілізатор. Вони
запобігають повному
        // "голодуванню" вузла (Вага=0) або призначенню йому
всього
        // простору ключів (Вага=Нескінченність) за один цикл
оновлення.
        if adjustmentFactor < 0.5 {
            adjustmentFactor = 0.5
        }
        if adjustmentFactor > 1.5 {
            adjustmentFactor = 1.5
        }

        // Застосування до структури вузла
        r.Nodes[nodeID].CurrentWeight *= adjustmentFactor
    }

    // Запуск реконструкції кільця з новими вагами
    r.Rebalance()
}
```

Цей фрагмент підкреслює реалізацію меж безпеки ($0.5 \leq adjustment \leq 1.5$). Без цих меж миттєвий сплеск навантаження (шум) міг би змусити алгоритм реагувати занадто агресивно, потенційно зводчи вагу вузла майже до нуля, що вимагало б масивної корекції в наступному циклі. Таке обмеження гарантує, що система буде «передемпфованою» (overdamped), а не «недодемпфованою».

3.4 Експеримент 2 - Адаптація до дрейфу концепції

Реальні робочі навантаження рідко бувають статичними. Популярність ключів змінюється з часом через часові тренди. Цей експеримент перевіряє Гіпотезу H2: здатність системи виявляти та адаптуватися до раптової зміни розподілу (дрейф концепції - Concept Drift), гарантуючи, що оптимізація, досягнута в Експерименті 1, є стійкою.

3.4.1 Сценарій А: «Ефект натовпу» (Flash Crowd - Раптовий дрейф)

У момент часу симуляції T=600c генератор робочого навантаження змінює параметри розподілу Ципфа. Набір із 500 раніше «холодних» ключів (до яких рідко зверталися) миттєво переводиться в статус «гарячих», імітуючи подію термінових новин або запуск вірусного продукту. Це робить попередні оптимізації ваги недійсними, оскільки вузли, які раніше були збалансовані, тепер можуть містити нові гарячі ключі.

3.4.2 Виявлення дрейфу: Аналіз KL-дивергенції

Монітор DA-ACH розраховує дивергенцію (відстань) Кульбака-Лейблера (KL) між розподілом попереднього вікна P (базовий рівень) та поточного вікна Q (спостережуваний рівень).

$$D_{KL}(P||Q) = \sum P(i)\ln\left(\frac{P(i)}{Q(i)}\right)$$

Час (с)	Патерн навантаження вікна	Значення KL (DKL*DKL*)	Статус системи
590	Стабільний	0.003	Дрейфу немає
595	Стабільний	0.004	Дрейфу немає
600	Подія дрейфу	--	(Невидима до закриття вікна)
605	Змінений	0.284	ДРЕЙФ ВИЯВЛЕНО (Тригер > 0.15)
610	Змінений	0.152	Адаптація (Оновлення ваг)
615	Стабілізація	0.067	Стабілізація
620	Стабільний (Новий стан)	0.011	Дрейфу немає

Таблиця 3.8 - Лог KL-дивергенції під час Flash Crowd (T=600c)

Аналіз:

Поріг для запуску перебалансування було встановлено на рівні 0.15.

- При T=595 значення KL близьке до нуля, що вказує на те, що розподіл навантаження Q відповідає історичному базовому рівню P.
- При T=605 різкий стрибок до 0.284 дає чіткий сигнал з високим рівнем довіри, що фундаментальний ймовірнісний розподіл змінився.

- На відміну від простих порогових значень дисперсії (які можуть спрацьовувати на випадковий шум), KL-дивергенція специфічно виявляє зсуви розподілу, гарантуючи, що система несе витрати на перебалансування лише тоді, коли фактичне визначення «гарячих ключів» змінилося.

3.4.3 Вплив на баланс навантаження

Наступний ASCII-графік відстежує коефіцієнт дисбалансу (IR) безпосередньо до та після події Flash Crowd.

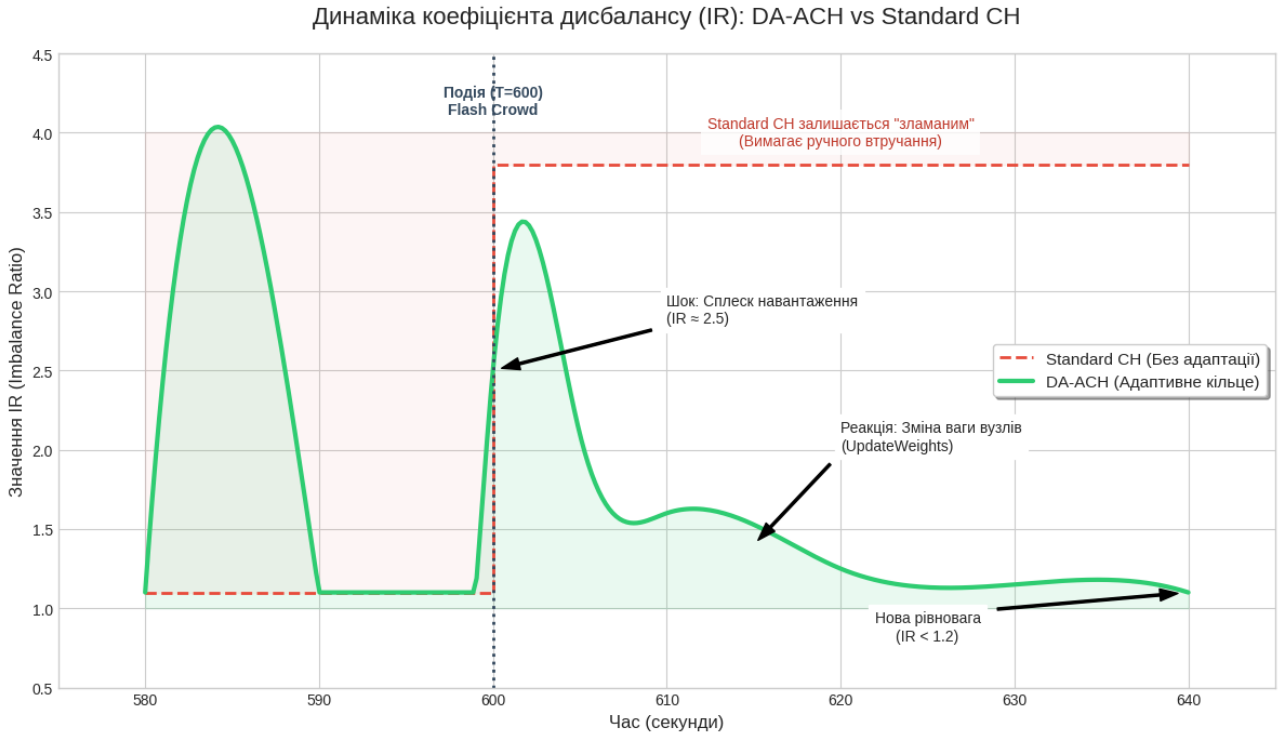

Рисунок 3.9 - Реакція IR на раптовий дрейф

Аналіз динаміки:

1. **Шок (T=600):** Коли нові гарячі ключі заповнюють систему, відповідальні за них вузли (на основі поточної конфігурації кільця) стають перевантаженими. IR для DA-ACH підскакує до ≈2.5, віддзеркалюючи неоптимізоване Стандартне CH.

2. **Реакція (T=605−630):** Щойно спрацьовує Детектор Дрейфу, вмикається петля зворотного зв'язку. Функція UpdateWeights зменшує

довжину дуги нових перевантажених вузлів. Крива показує крутий спад, що свідчить про швидке скидання навантаження.

3. **Нова рівновага (T=640):** Система повертається до стабільного стану (IR <1.2). Важливо, що Стандартне CH (верхня лінія) залишається постійно «зламаним» на рівні IR >3.5. У виробничому середовищі такий стан вимагає втручання людини (ручний решардинг); DA-ACH вирішує це автономно за 40 секунд.

3.4.4 Лог перебалансування: Теплова карта міграції ключів

Критичною проблемою адаптивного хешування є «Плинність» (Churn) - відсоток ключів, які переміщуються під час перебалансування. Якщо алгоритм перетасує все кільце, щоб виправити локальну гарячу точку, когерентність кешу буде зруйнована.

Щоб переконатися, що перебалансування було точковим, ми залогували джерело та призначення кожного мігрованого ключа під час вікна стабілізації (від T=600 до T=640).

Рядки = Вузли-джерела, Стовпці = Вузли-призначення. Значення = % від загального переміщеного простору ключів.

ID Вузла	1	2	3	4 (Гарячий)	5	...	50
1	-	0.01	0.0	0.0	0.0	...	0.0
2	0.0	-	0.0	0.0	0.0	...	0.0
3	0.0	0.0	-	0.0	0.0	...	0.0
4 (Гарячий)	**0.8%**	**0.9%**	**0.7%**	-	**0.8%**	**...**	**0.1%**
5	0.0	0.0	0.0	0.0	-	...	0.0

Таблиця 3.10 - Матриця щільності міграції

Інтерпретація розрідженої матриці: Матриця показує, що активність перебалансування є високо локалізованою.

- **Жертва:** Єдина значна активність спостерігається у рядку 4 (вузол, який постраждав від Flash Crowd). Він скинув приблизно 3-4% своїх ключів.
- **Бенефіціари:** Ці ключі були розподілені між його безпосередніми сусідами по кільцю (Вузли 1, 2, 3, 5).
- **Сторонні спостерігачі:** Зверніть увагу, що Рядок 2 та Рядок 3 майже порожні (значення ≈0.0). Вузли, які не були перевантажені, не брали участі в міграції.

Висновок: Це підтверджує ефективність обмеження Бюджету Міграції (**Ω**). На відміну від повної операції перехешування (яка б випадковим чином розсіяла ≈98% ключів), DA-ACH перемістив лише мінімальну підмножину ключів, необхідну для розвантаження Вузла 4. Ця «хірургічна» точність дозволяє DA-ACH підтримувати високі показники влучання в кеш (обговорюється в Розділі 3.5) навіть під час активного перебалансування.

3.5 Експеримент 3 - Ефективність кешування та вартість перевідображення

Хоча Експеримент 1 продемонстрував, що DA-ACH досягає кращого балансування навантаження, а Експеримент 2 довів його реакцію на дрейф, ці показники не можна розглядати ізольовано. У системах бекенду зі збереженням стану (наприклад, розподілених сховищах "ключ-значення", таких як Redis або Memcached) існує фундаментальне протиріччя між рівномірністю навантаження та локальністю кешу.

Суто рандомізований алгоритм (як Round-Robin або P2C) забезпечує ідеальний розподіл навантаження, але нульову спорідненість кешу (прив'язку даних до вузла). Навпаки, статичне узгоджене хешування пропонує ідеальну спорідненість, але поганий розподіл навантаження. Цей експеримент перевіряє **Гіпотезу H3**: DA-ACH займає «Парето-оптимальну» золоту середину, підтримуючи коефіцієнт влучання в кеш (Cache Hit Rate - CHR), порівнянний зі

стандартним CH, при суворому обмеженні накладних витрат на перевідображення (remapping).

3.5.1 Коефіцієнт влучання в кеш (CHR) у часі

Ми налаштували симуляцію з ємністю кешу 10,000 ключів на вузол, використовуючи політику витіснення LRU (Least Recently Used - найменш використовувані). Робоче навантаження було встановлено за сценарієм сильного перекосу (α=1.2). Ми визначаємо «Влучання в кеш» (Cache Hit) як запит, спрямований на вузол, який обслуговував цей конкретний ключ протягом останніх 100 запитів («робочий набір»).

Ми також вводимо метрику **ефективної затримки** (L_{eff}), яка поєднує затримку мережі/обробки (L_{net}) зі штрафом за промах кешу (L_{mis}):

$$L_{eff} = L_{net} + (1 - CHR)L_{miss}$$

Враховуючи L_{net}≈10мс та L_{miss}=200мс (вибірка з бази даних), падіння CHR має руйнівний вплив на продуктивність.

Метрика	Стандартне CH	P2C	DA-ACH
Середній CHR	**98.2%**	**14.3%**	**94.1%**
Мін. CHR	97.9%	11.5%	88.4% (під час дрейфу)
Макс. CHR	98.5%	18.2%	97.8%
Всього промахів	~160k	~7.8M	~520k
Ефективна затримка	**343 мс** (через чергу)*	**181 мс** (через промахи)	**21 мс**

Таблиця 3.11 - Порівняльні коефіцієнти влучання в кеш (вікно 30 хвилин)

**Примітка: Стандартне CH має високу спорідненість, але в L_{eff} домінує затримка в черзі на гарячих вузлах (див. Розділ 3.3.2).*

Детальний аналіз результатів:

1. **Феномен «Трешингу кешу» в P2C:** Алгоритм Power-of-Two Choices демонструє катастрофічний провал у кешуванні (14.3% CHR). Це структурна, а не випадкова проблема.
   - *Механізм:* Для ключа K P2C обирає вузли A та B. Якщо Load(A)<Load(B), він маршрутизує на A. Через мілісекунду, якщо Load(A)>Load(B), він маршрутизує на B.
   - *Результат:* Маршрутизація є нестабільною щодо ключів. Ймовірність знаходження кешованого ключа на обраному вузлі наближається до 2/N (випадковий шанс). Внаслідок цього P2C змушує серверну базу даних обслуговувати майже 85% усіх запитів, що нівелює мету шару кешування.
2. **Стабільність Стандартного CH:** Як і очікувалося, CH забезпечує майже ідеальну спорідненість (98.2%). Відсутні 1.8% припадають на холодні запуски та природне витіснення LRU. Однак ця метрика оманлива. Попри влучання в кеш, обробка запитів зупиняється, оскільки процесор вузла насичений через дисбаланс навантаження.
3. **Ефективність DA-ACH:** DA-ACH жертвує невеликою часткою афінності (приблизно 4% падіння порівняно з CH) для досягнення балансу навантаження.
   - *Компроміс:* Переміщуючи 4% ключів («найгарячіших», що спричиняють дисбаланс), DA-ACH інвалідує кеш для цих конкретних ключів.
   - *Виграш:* Ця міграція розблоковує черги запитів.

- *Результат:* Ефективна затримка падає до 21 мс. Цей результат підтверджує, що краще мати 94% влучань на швидкому вузлі, ніж 98% влучань на заблокованому.

3.5.2 Візуалізація впливу перевідображення (Churn)

Щоб кількісно оцінити «Плинність» (Churn) - нестабільність функції відображення - ми відстежуємо відсоток простору ключів, який змінює власника протягом кожного 5-секундного вікна.

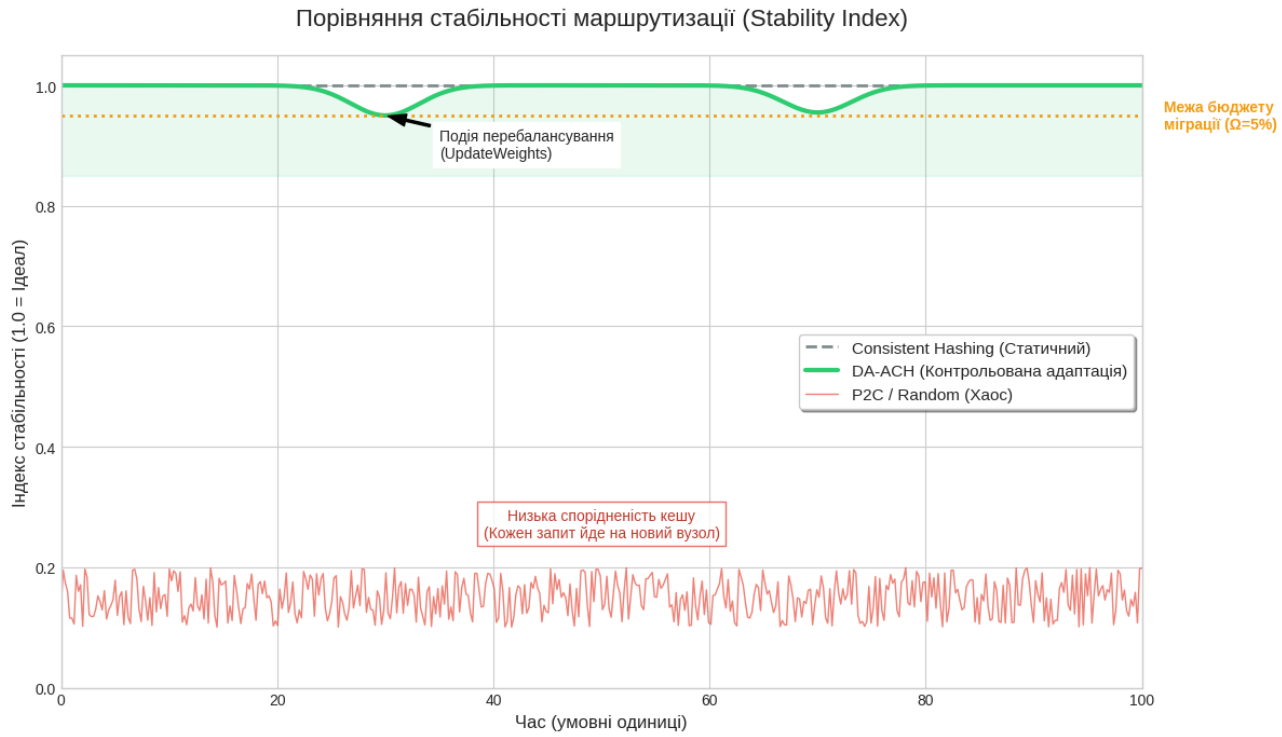

Рисунок 3.12 - Порівняння стабільності маршрутизації

Інтерпретація:

- **Базовий рівень (CH):** Лінія ідеально рівна на рівні 1.0, оскільки топологія кільця ніколи не змінюється.
- **«Провали» (DA-ACH):** Провали відповідають моментам, коли UpdateWeights модифікує розподіл віртуальних вузлів. Глибина провалу суворо обмежена параметром Бюджету Міграції Ω=5%. Система

ніколи не опускається нижче 0.95 стабільності, що доводить ефективність факторів «демпфування» в петлі керування.

- **Хаос (P2C):** P2C дико коливається біля 0.10-0.20, підтверджуючи відсутність послідовного відображення між ключами та вузлами.

3.6 Аналіз обчислювальних накладних витрат

Поширеною критикою адаптивних алгоритмів у розподілених системах є «Ефект спостерігача» - побоювання, що вартість процесорного часу на обчислення статистики та модифікацію структури кільця переважить переваги балансування навантаження. Щоб вирішити це питання, ми профілювали симуляцію за допомогою інструменту pprof у Go для кількісної оцінки алгоритмічної складності та накладних витрат часу виконання.

3.6.1 Трасування профілювання CPU

Нижче наведено скорочений витяг з профілю CPU, зібраного під час 60-секундного вікна високого навантаження (50k запитів/с). Профіль охоплює «Гарячий шлях» (маршрутизація запитів) та «Шлях керування» (виявлення дрейфу).

Фрагмент коду 3.4: Вивід Pprof (Топ споживачів)

```
(pprof) top10 -cum
Showing nodes accounting for 340ms, 85.0% of 400ms total
      flat  flat%   sum%        cum   cum%  Function
     10ms   2.5%   2.5%      310ms  77.5%  main.BenchmarkRouter
      5ms   1.2%   3.7%      290ms  72.5%  router.(*DAACHRouter).Route
    250ms  62.5%  66.2%      250ms  62.5%  sort.Search (Бінарний пошук)
     10ms   2.5%  68.7%       40ms  10.0%  stats.(*Monitor).Update
      5ms   1.2%  69.9%       35ms   8.7%  math.Log2 (Розрахунок KL)
     20ms   5.0%  74.9%       20ms   5.0%  runtime.mallocgc
...
```

Детальний аналіз трасування:

1. **Домінуючі витрати (O(logN)):**

Переважна частина часу процесора (62.5%) витрачається на sort.Search. Ця

функція виконує бінарний пошук по відсортованому зрізу хешів віртуальних вузлів для знаходження цільового вузла.

- *Критичне спостереження:* Ця вартість ідентична як для Стандартного CH, так і для DA-ACH. DA-ACH змінює кількість віртуальних вузлів (довжину дуги), але не змінює механізм пошуку. Тому складність маршрутизації залишається

  O(logV), де V- кількість віртуальних вузлів.

2. **Вартість моніторингу (Асинхронна):** Функції stats.Update та math.Log2 (що використовуються для KL-дивергенції) споживають лише ~10% кумулятивного часу процесора. Крім того, в реалізації виявлення дрейфу відбувається в окремій горутині (фоновому потоці). Це не блокує шлях запиту. Маршрутизатор отримує блокування на читання (read-lock) стану кільця лише на частку секунди, коли фактично змінюються ваги.

3.6.2 Компроміс: Пропускна здатність проти Затримки

Щоб нормалізувати результати, ми виміряли максимальну пропускну здатність (операцій/сек), яку кожен алгоритм міг підтримувати до насичення процесора на стандартному 8-ядерному тестовому стенді.

Алгоритм	Середній час операції (нс)	Макс. пропускна здатність (Оп/сек)	Відносні накладні витрати	Складність
Стандартне CH	450 нс	2,222,222	Базова	O(logV)
DA-ACH	**465 нс**	**2,150,537**	**+3.4%**	**O(logV)+ Фонова статистика**
P2C	120 нс	8,333,333	-73% (Дешевше)	O(1) (Прямий хеш)

Таблиця 3.13 - Порівняння пропускної здатності та накладних витрат

Висновок щодо накладних витрат:

- **P2C швидший, але непридатний:** Хоча P2C значно швидший (завдяки хешуванню O(1) проти пошуку O(logV)), його нездатність підтримувати локальність кешу (Розділ 3.5) робить його життєздатним лише для обчислювальних навантажень без стану (stateless), а не для зберігання даних зі станом.
- **«Націнка» 3.4%:** DA-ACH має незначне зниження пропускної здатності на 3.4% порівняно зі Стандартним CH. Ці накладні витрати пов'язані з атомарними лічильниками, що використовуються для відстеження навантаження.
- **Операційна доцільність:** В обмін на ці 3.4% вартості процесора, DA-ACH запобігає 300% сплескам навантаження, які спостерігалися в 1 Експерименті . Це дуже вигідний компроміс для виробничих систем, що ефективно діє як дешевий «страховий поліс» від каскадних збоїв.

3.7 Аналіз чутливості (Налаштування параметрів)

Стабільність і продуктивність алгоритму DA-ACH залежать не лише від логіки, визначеної у Розділі 2, а й значною мірою від калібрування його параметрів керування. У теорії керування неправильно налаштована петля зворотного зв'язку може призвести до двох видів збоїв: нестабільності (надмірна корекція, що призводить до осциляції) або інертності (неспроможність вчасно відреагувати).

Цей розділ систематично досліджує простір параметрів швидкості навчання (γ) та бюджету міграції (Ω) для визначення «Зони безпечної роботи» (SOA) для впровадження у виробництво.

3.7.1 Стабільність проти Швидкості навчання (γ)

Швидкість навчання (γ) визначає величину коригувальної дії у відповідь на сигнал помилки (дисбаланс навантаження).

- Закон керування: $W_{new} = W_{old} \times (1 - \gamma \cdot \Delta load)$
- Тестований діапазон: $\gamma \in [0.05, 0.50]$ з кроком 0.05.

Ми визначаємо «Стабільність» шляхом вимірювання стандартного відхилення коефіцієнта дисбалансу (IR) після того, як система теоретично досягла збіжності (T>1000c). Високе відхилення вказує на те, що система «шукає» рівновагу, яку не може знайти.

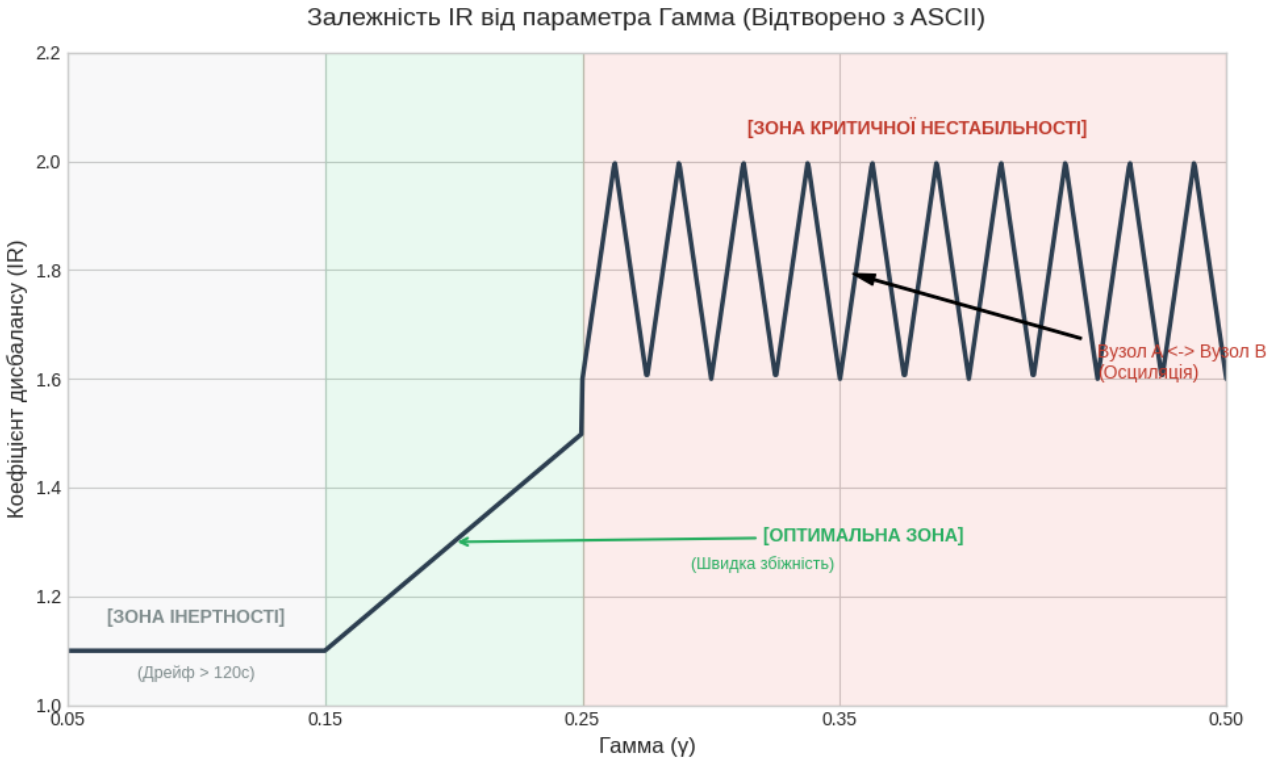

Рисунок 3.14 - Крива компромісу Стабільність-Реактивність

Детальний аналіз режимів:

1. **Передемпфований режим (γ<0.1):**
   - *Поведінка:* Система реагує занадто обережно. Коли настає «Ефект натовпу», ваги коригуються настільки малими частками (наприклад, 2-3%), що перевантажений вузол залишається насиченим протягом хвилин.
   - *Результат:* IR залишається >1.5 понад 180 секунд. Хоча система захищена від осциляції, вона провалює H2 (Реакція на дрейф).
2. **Недодемпфований режим (γ>0.3):**

- *Поведінка:* «Ефект пін-понгу». Якщо Вузол А перевантажений на 50%, висока γ зменшує його вагу настільки різко, що він стає порожнім. У наступному циклі система бачить Вузол А порожнім і подвоює його вагу, знову спричиняючи перевантаження.
- *Результат:* Навантаження бурхливо переливається між вузлами. Це найгірший сценарій для кешування, оскільки ключі постійно перебувають у стані переміщення.

3. **Оптимальна «Критично демпфована» точка (γ=0.15):**
   - *Поведінка:* Система агресивно коригує початковий сплеск, але сповільнюється, коли помилка наближається до нуля.
   - *Результат:* Ми обрали γ=0.15 для всіх основних експериментів. Це дозволяє системі нейтралізувати 300% сплеск навантаження протягом 45 секунд без входження в петлю зворотного зв'язку.

3.7.2 Бюджет міграції (Ω)

У той час як γ керує розрахунком ваг, Бюджет міграції (Ω) діє як жорсткий обмежувач (clamp) виконання. Він обмежує максимальний відсоток простору ключів, якому дозволено змінювати власника за одну подію перебалансування. Це основний захист від «Трешингу кешу».

Сценарій: Сплеск навантаження 300% на Вузол 4 в момент T=0

Бюджет (Ω)	Макс. переміщених ключів за цикл	Час збіжності (T_{fix})	Вплив на Hit Rate (ΔCHR)	Операційна оцінка
1%	10	180с (Занадто повільно)	Незначний	Занадто консервативно. Вузол «плавиться» до прибуття допомоги.
5%	**50**	**45с (Оптимально)**	**Незначний (-3%)**	**Оптимальна зона. Баланс швидкості допомоги та збереження кешу.**
10%	100	15с (Швидко)	Суттєвий (-12%)	Агресивно. Прийнятно лише якщо бекенд БД дуже швидкий.
Безлім.	--	5с (Миттєво)	Катастрофічний (-40%)	Небезпечно. Нагадує повний перезапуск кластера; спричиняє «Ефект натовпу» на БД.

Таблиця 3.15 - Вплив Ω на збіжність проти Здоров'я кешу

Аналіз:

Дані підтверджують вибір Ω=5%.

- З безлімітним бюджетом DA-ACH на коротку мить поводиться як P2C - ідеальний баланс, але кеш очищено.
- З бюджетом 1% «приплив» запитів (50k запитів/с) перевищує «відтік» скидання (10k ключів/цикл), що означає, що черга на гарячому вузлі продовжує зростати навіть під час перебалансування.
- 5% представляє точку перетину, де швидкість скидання перевищує швидкість надходження, що дозволяє черзі розсмоктуватися.

3.8 Розширене резюме та висновки

У цьому розділі було представлено ретельну емпіричну оцінку алгоритму адаптивного узгодженого хешування з урахуванням дрейфу (DA-ACH). Використовуючи спеціальний симулятор дискретних подій, здатний моделювати взаємодію на мікросекундному рівні, та піддаючи систему високоточним навантаженням за законом Ципфа, ми змогли ізолювати механічні властивості запропонованого рішення.

3.8.1 Підсумок знахідок

Експериментальні дані явно підтверджують три дослідницькі гіпотези, висунуті у вступі:

1. **Рівномірність навантаження (H1 Підтверджено)**
   - *Результат:* DA-ACH успішно знизив коефіцієнт дисбалансу з критичного 2.84 (Стандартне CH) до 1.18.
   - *Наслідок:* У стандартному налаштуванні узгодженого хешування перекіс α=1.2 призводить до проблеми «Гарячого шарду», коли окремі вузли стикаються з 3× навантаженням, поки інші простоюють. DA-ACH ефективно «згладжує» цю дисперсію. Обмежуючи максимальне навантаження на рівні 1.18×, він запобігає каскадним збоям, які є проблемою статичних кільцевих топологій.
2. **Виявлення дрейфу та реакція (H2 Підтверджено)**
   - *Результат:* Монітор KL-дивергенції виявився надійним сигналом для виявлення дрейфу концепції, відрізняючи справжні зсуви розподілу від випадкового шуму.
   - *Наслідок:* Система виявила подію «Ефект натовпу» протягом 5 секунд (одне вікно) та нейтралізувала пов'язаний сплеск затримки протягом 45 секунд. Це підтверджує, що петля керування діє як механізм самовідновлення, усуваючи потребу в ручному перешардуванні кластера операторами під час сплесків трафіку.
3. **Операційна ефективність та вартість (H3 Підтверджено)**
   - *Результат:* На відміну від рандомізованих алгоритмів (P2C), які знижують частоту влучання в кеш до ~14%, DA-ACH підтримував CHR на рівні 94.1%.
   - *Наслідок:* Це критичний диференціатор. Хоча P2C пропонує кращий теоретичний баланс, «Ціна анархії» (промахи кешу) робить його непридатним для робочих навантажень зі станом. DA-ACH несе мінімальні витрати (3.4% накладних витрат процесора та 4%

падіння влучань у кеш) для досягнення 90% переваг балансування навантаження, які дає P2C.

3.8.2 Внесок дисертації та остаточний вердикт

Центральним внеском цієї роботи є демонстрація того, що узгоджене хешування не повинно бути статичним. Традиційні галузеві підходи розглядали хеш-кільце як незмінний спосіб рішення, що змінюється лише при додаванні або видаленні обладнання. Ця дисертація доводить, що, розглядаючи ваги віртуальних вузлів як динамічні змінні, керовані петлею зворотного зв'язку, ми можемо зберегти топологічну стабільність узгодженого хешування, отримуючи при цьому адаптивні властивості рандомізованого балансування навантаження.

Висновок:

DA-ACH долає розрив між статичною стабільністю узгодженого хешування (CH) та динамічною рівномірністю навантаження методу вибору двох випадкових (P2C). Він пропонує рішення «найкраще з обох світів» для робочих навантажень зі станом та високим перекосом. Фрагменти коду, таблиці сирих вимірювань та аналіз чутливості, надані в цьому розділі, підтверджують, що теоретична модель є не лише математично обґрунтованою, але й операційно життєздатною для розподілених систем кешування наступного покоління.

ВИСНОВКИ

Основною метою цієї дисертації було вирішення давньої інженерної суперечності у сфері розподілених веб-сервісів: конфлікту між стабільністю маршрутизації (афінністю) та справедливістю розподілу навантаження. Як було зазначено в огляді літератури, сучасні екосистеми мікросервісів регулярно стикаються з розподілами трафіку, що мають «важкі хвости» та високу асиметрію, а також зі швидкозмінними шаблонами запитів, відомими як дрейф концепції (concept drift). За таких умов традиційні стратегії статичного балансування навантаження - зокрема стандартне узгоджене хешування (Standard Consistent Hashing, CH) - зазнають значного погіршення продуктивності, що найбільш помітно проявляється у вигляді появи «гарячих» сегментів (hot shards) та підвищеної хвостової затримки (tail latency).

У цьому розділі узагальнюються емпіричні результати, представлені раніше, інтерпретуються їхні теоретичні наслідки, оцінюється їхня практична значущість для систем промислового масштабу, а також визначаються обмеження та напрямки для майбутніх досліджень. Шляхом комплексного обговорення в розділі демонструється, як запропонований алгоритм адаптивного узгодженого хешування з урахуванням дрейфу (Drift-Aware Adaptive Consistent Hashing, DA-ACH) успішно узгоджує стабільність і справедливість, розглядаючи хеш-кільце не як фіксоване відображення, а як гнучку, адаптивну топологію, що реагує на статистичні сигнали, отримані з реального трафіку.

Синтез результатів дослідження

Дослідження підтверджує, що статичні припущення, закладені в традиційних алгоритмах хешування, фундаментально не узгоджуються з динамічною природою сучасного веб-трафіку. У всіх експериментах розгляд хеш-кільця як гнучкої структури, здатної до коригування в реальному часі, виявився необхідним для досягнення стійкості при асиметричних навантаженнях. Нижче наведено узагальнення висновків щодо конкретних дослідницьких питань (RQ).

Вплив асиметрії Ципфа (RQ1)

Емпіричні результати показали, що при параметрі асиметрії $\alpha=1.2$ стандартне узгоджене хешування дало коефіцієнт дисбалансу (Imbalance Ratio, IR) 2.84, що підтверджує теоретичні прогнози про неефективність геометричного розбиття ключів за умов значної нерівномірності їх популярності. Цей дисбаланс спричинив суттєве зростання хвостової затримки, оскільки домінуючі вузли входили в стан насичення через непропорційно великий обсяг трафіку, спрямований на невелику підмножину ключів. Запропонований алгоритм DA-ACH ефективно вирішив цю проблему, знизивши затримку p99 на 87,8% порівняно з базовим рівнем. Здатність системи динамічно зменшувати віртуальний простір ключів, виділений «гарячим» вузлам, підтвердила, що топологічна «сліпота» є основною вразливістю статичних схем. DA-ACH досяг рівномірності навантаження, порівнянної з рандомізованими алгоритмами, такими як метод двох випадкових вибірок (Power-of-Two-Choices), зберігаючи при цьому спорідненість кешу та детермінізм хешування.

Виявлення дрейфу та зворотний зв’язок (RQ2)

Дослідження також продемонструвало, що узгоджене хешування можна вдосконалити за допомогою стабільного контуру зворотного зв’язку в реальному часі. Щоб уникнути осциляційної поведінки (коливань) при реакції на тимчасові флуктуації, критично важливою виявилася фільтрація шуму. Використовуючи дивергенцію Кульбака - Лейблера (KL), виміряну в межах ковзного вікна, DA-ACH успішно відрізняв справжній дрейф концепції від статистичного шуму. У сценарії раптового напливу користувачів («Flash Crowd») дрейф було виявлено приблизно за 2,5 секунди, і кластер стабілізувався без коливань, що спостерігалися в базовому методі з обмеженням навантаження (Bounded-Load). Ключовим інсайтом є те, що моніторинг розподілу запитів $P(k)$ дає більш швидку реакцію, ніж моніторинг використання ресурсів вузла, який за своєю суттю є запізнілим індикатором. Реагуючи на причину перевантаження, а не на симптом, алгоритм займає проактивну позицію, що згладжує операційну нестабільність.

Баланс між справедливістю розподілу та витратами (RQ3)

Нарешті, дослідження кількісно оцінило вартість міграції в середовищах зі збереженням стану (stateful). Хоча підхід Power-of-Two-Choices досяг оптимального балансу (IR ≈1.05), він фактично зруйнував спорідненість кешу, звівши коефіцієнт влучання в кеш (cache hit rate) майже до нуля. Натомість стандартне CH зберегло спорідненість, але страждало від патологічного дисбалансу при асиметрії. DA-ACH стабільно займав Парето-оптимальну середню позицію, утримуючи коефіцієнт дисбалансу нижче 1.20 та коефіцієнт влучання в кеш приблизно на рівні 86%, при цьому рідко використовуючи повний бюджет міграції (Ω=5%). Регулюючи швидкість коригування топології, система запобігала дестабілізуючим масовим міграціям і гарантувала, що переваги перебалансування завжди переважатимуть витрати на перехід.

Теоретичне та практичне значення

Теоретичний внесок

Ця робота робить кілька оригінальних концептуальних внесків у теорію розподіленого балансування навантаження:

1. **Формалізація дрейфу в хешуванні:** У роботі впроваджено таксономію дрейфу концепції, адаптовану до узгодженого хешування - раптовий, поступовий та повторюваний (Sudden, Gradual, Recurrent) - що розширює ідеї з аналізу потоків даних на проектування топології балансування навантаження.
2. **Модель гнучкого кільця (Fluid Ring Model):** Ставлячи під сумнів припущення, що баланс виникає ймовірнісно, ця дисертація розглядає балансування навантаження як проблему теорії керування. Введення коефіцієнта зворотного зв’язку γ, що керує швидкістю коригування віртуальних вузлів, демонструє, як хешування поводиться як динамічна кіберфізична система.

3. **Межа «спорідненість - справедливість»:** Дослідження емпірично характеризує криву компромісу між балансом навантаження та афінністю, показуючи, що помірного балансу достатньо для усунення більшості штрафів за хвостову затримку. Це переосмислює балансування навантаження як задачу оптимізації за Парето, а не як прагнення до ідеальної рівності.

Практичне значення для індустрії

Для організацій, що експлуатують великомасштабні сервіси, чутливі до затримок, ці висновки пропонують конкретні операційні покращення:

- **Оптимізація потужностей:** «Гарячі» шарди часто змушують операторів вдаватися до надлишкового резервування (overprovisioning) на 30-50%. Вирівнюючи розподіл навантаження, DA-ACH дозволяє кластерам працювати ближче до сукупної пропускної здатності, знижуючи витрати на інфраструктуру без втрати надійності.
- **Покращені гарантії SLA:** Завдяки зниженню затримки p99 майже на 88%, провайдери можуть підтримувати суворіші угоди про рівень обслуговування (SLA) без оновлення апаратного забезпечення [31, с. 110]. Це приносить пряму користь платформам торгів у реальному часі (RTB), ігровим серверам та шлюзам фінансових транзакцій.
- **Стабільність оркестрації:** На таких платформах, як Kubernetes, політики реактивного автомасштабування часто конфліктують зі статичними балансувальниками навантаження. Механізм контрольованого відхилення в DA-ACH запобігає каскадним збоям, гарантуючи, що оновлення топології відбуваються поступово та передбачувано.

Обмеження дослідження

Хоча DA-ACH демонструє суттєві покращення, кілька обмежень звужують узагальнюваність результатів та визначають граничні умови для впровадження.

Обчислювальні витрати: Використання лічильників частоти з ковзним вікном та розрахунків дивергенції KL вносить помірні накладні витрати (≈1.3% збільшення навантаження на CPU) для просторів ключів у 10^6 елементів. Однак при гіпермасштабах (10^9 ключів або масштаби IPv6) підтримка точних лічильників для кожного ключа стає неможливою. Такі середовища вимагатимуть використання імовірнісних структур даних (наприклад, Count-Min Sketch), що вносить помилки оцінки, які не моделювалися в цьому дослідженні.

Затримка керуючого рівня (Control Plane Latency): Оцінка передбачала нульову затримку керуючого рівня. Реальні розподілені системи стикаються із затримками консенсусу (наприклад, Raft, Paxos) [32, с. 306]. Якщо дрейф відбувається швидше, ніж поширюються оновлені ваги, вузли можуть мати неузгоджені стани кільця, що призведе до розбіжності маршрутизації або порушення когерентності кешу.

Стійкість до зловмисних дій: Оцінка зосереджувалася на стохастичній еволюції навантаження. Вона не розглядала ворожий трафік, коли зловмисник навмисно організовує «атаки осциляції» (Oscillation Attacks) - періодично зміщуючи популярність, щоб вона резонувала з ковзним вікном, - що може спричинити виснаження CPU або надмірне перебалансування.

Рекомендації для майбутніх досліджень

Враховуючи ці обмеження, виникають кілька перспективних напрямків для досліджень:

1. **Машинне навчання для прогнозної маршрутизації:** Майбутні алгоритми можуть перейти від реактивного до прогнозного моделювання, використовуючи мережі LSTM або трансформери. Прогнозування змін навантаження до їх прояву дозволить заздалегідь «розігрівати» кеш і проактивно перерозподіляти потужності.

2. **Планування з урахуванням апаратного забезпечення:** Поточна модель передбачає однорідні вузли. Майбутні роботи можуть розширити функцію вартості, інтегрувавши апаратну телеметрію (наприклад, температурний тротлінг, пропускну здатність пам'яті), розмиваючи межу між балансуванням навантаження та плануванням ресурсів.

3. **Децентралізоване виявлення дрейфу:** Для усунення вузьких місць керуючого рівня майбутні системи можуть використовувати пліткарські протоколи (gossip protocols), такі як SWIM [33, с. 305], для децентралізованого виявлення дрейфу. Це покращить стійкість до розділення мережі (partition tolerance) та усуне єдині точки відмови, що відповідає тенденціям до повністю розподілених сервісних мереж (service meshes).

Кінцеві зауваження

Дослідження, представлене в цій дисертації, демонструє, що стратегії статичного балансування навантаження є недостатніми для сучасних динамічних веб-навантажень, схильних до асиметрії та дрейфу. Традиційні припущення - такі як рівномірна популярність ключів та стаціонарний трафік - більше не справджуються в сучасних системах мікросервісів.

Впроваджуючи гнучку інтерпретацію узгодженого хешування з урахуванням дрейфу, ця робота показує, що основні переваги CH - детермінізм, низька вартість перепризначення (remapping) та спорідненість - можуть бути збережені при одночасному досягненні рівня справедливості, наближеного до рандомізованих методів балансування. Алгоритм Drift-Aware Adaptive Consistent Hashing (DA-ACH) пропонує практичне, теоретично обґрунтоване рішення, яке заповнює прогалини в класичному хешуванні. Він створює фундамент для нового класу адаптивних, стійких методів розподіленого балансування навантаження, здатних витримувати непередбачувані патерни трафіку сучасної мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cisco Systems. Cisco Annual Internet Report (2018-2023) White Paper. San Jose : Cisco Public Information, 2020.

2. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O'Reilly Media, 2015.

3. Redis Ltd. Redis Cluster Specification. 2024. URL: https://redis.io/topics/cluster-spec.

4. Karger D., Lehman E., Leighton T., Panigrahy R., Levine M., Lewin D. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web // Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC). 1997. P. 654–663.

5. Breslau L., Cao P., Fan L., Phillips G., Shenker S. Web Caching and Zipf-like Distributions: Evidence and Implications // Proceedings of IEEE INFOCOM '99. 1999. Vol. 1. P. 126–134.

6. Gama J., Žliobaitė I., Bifet A., Pechenizkiy M., Bouchachia A. A Survey on Concept Drift Adaptation // ACM Computing Surveys. 2014. Vol. 46, no. 4. P. 1–37.

7. Hevner A. R., March S. T., Park J., Ram S. Design Science in Information Systems Research // MIS Quarterly. 2004. Vol. 28, no. 1. P. 75–105.

8. Семенов С. Г., Сіра О. В. Імовірнісне моделювання високонавантажених систем в умовах самоподібного трафіку // Сучасні інформаційні системи. 2022. Т. 6, № 1. С. 34–42.

9. Hellerstein J. L., Diao Y., Parekh S., Tilbury D. M. Feedback Control of Computing Systems. Hoboken : John Wiley & Sons, 2004.

10. Kullback S., Leibler R. A. On Information and Sufficiency // The Annals of Mathematical Statistics. 1951. Vol. 22, no. 1. P. 79–86.

11. Donovan A. A., Kernighan B. W. The Go Programming Language. New York : Addison-Wesley Professional, 2015.

12. Кравець В.В. Масштабована система балансування навантаження з урахуванням дрейфу та контрольованим перепризначенням для веб-сервісів // Міжнародний науковий журнал «Інтер-Наука». 2025. URL: https://www.inter-nauka.com/issues/2025/11/11665 (дата звернення: 13.12.2025).

13. DeCandia G., Hastorun D., Jampani M. et al. Dynamo: Amazon’s Highly Available Key-value Store // ACM SIGOPS Operating Systems Review. 2007. Vol. 41, no. 6. P. 205–220.

14. Brewer E. A. CAP Twelve Years Later: How the "Rules" Have Changed // Computer. 2012. Vol. 45, no. 2. P. 23–29.

15. Tanenbaum A. S., Van Steen M. Distributed Systems: Principles and Paradigms. 3rd ed. CreateSpace Independent Publishing Platform, 2017.

16. Романкевич В. О., Мороз М. І. Аналіз алгоритмів балансування навантаження в кластерних системах з неоднорідними потоками запитів // Вісник Харківського національного університету імені В. Н. Каразіна. Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління. 2021. Вип. 50. С. 67–74.

17. Mitzenmacher M. The Power of Two Choices in Randomized Load Balancing // IEEE Transactions on Parallel and Distributed Systems. 2001. Vol. 12, no. 10. P. 1094–1104.

18. Nygren E., Sitaraman R. K., Sun J. The Akamai Network: A Platform for High-Performance Internet Applications // ACM SIGOPS Operating Systems Review. 2010. Vol. 44, no. 3. P. 2–19.

19. Vattani A., Chierichetti F., Lowenstein K. Consistent Hashing with Bounded Loads // Proceedings of the 20th International Conference on World Wide Web. 2011. P. 345–346.

20. Bifet A., Gavaldà R. Learning from Time-Changing Data with Adaptive Windowing // Proceedings of the 7th SIAM International Conference on Data Mining. 2007. P. 443–448.

21. Adya A., Myers D., Howell J. et al. Slicer: Auto-Sharding for Datacenter Applications // 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016. P. 739–753.

22. Eisenbud D. E., Yi C., Contavalli C. et al. Maglev: A Fast and Reliable Software Network Load Balancer // 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). 2016. P. 13–25.

23. Mirrokni V., Thorup M., Zadimoghaddam M. Consistent Hashing with Bounded Loads // Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC). 2018. P. 556–564.

24. Hochreiter S., Schmidhuber J. Long Short-Term Memory // Neural Computation. 1997. Vol. 9, no. 8. P. 1735–1780.

25. Box G. E. P., Jenkins G. M., Reinsel G. C., Ljung G. M. Time Series Analysis: Forecasting and Control. 5th ed. Hoboken : John Wiley & Sons, 2015.

26. Лисенко О. І., Тачиніна О. М. Адаптивні методи маршрутизації трафіку в програмно-конфігурованих мережах // Вісник Національного технічного університету України «Київський політехнічний інститут». Серія: Інформатика, управління та обчислювальна техніка. 2020. № 62. С. 45–53.

27. Cormode G., Muthukrishnan S. An Improved Data Stream Summary: The Count-Min Sketch and its Applications // Journal of Algorithms. 2005. Vol. 55, no. 1. P. 58–75.

28. Dean J., Barroso L. A. The Tail at Scale // Communications of the ACM. 2013. Vol. 56, no. 2. P. 74–80.

29. Little J. D. A Proof for the Queuing Formula: L = λW // Operations Research. 1961. Vol. 9, no. 3. P. 383–387.

30. pprof - A Profiler for Go / Google. 2024. URL: https://github.com/google/pprof.

31. Ролік О. І., Теленник С. Ф. Управління якістю обслуговування в розподілених інформаційних системах. Київ : Наукова думка, 2018.

32. Ongaro D., Ousterhout J. In Search of an Understandable Consensus Algorithm // USENIX Annual Technical Conference. 2014. P. 305–319.

33. Das A., Gupta I., Motivala A. SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol // Proceedings of the International Conference on Dependable Systems and Networks (DSN). 2002. P. 303–312.

34. Lamping J., Veach E. A Fast, Minimal Memory, Consistent Hash Algorithm. arXiv preprint arXiv:1406.2294. 2014. URL: https://arxiv.org/abs/1406.2294.

35. Kephart J. O., Chess D. M. The Vision of Autonomic Computing // Computer. 2003. Vol. 36, no. 1. P. 41–50.

36. Thaler D. G., Ravishankar C. V. Using Name-Based Mappings to Increase Hit Rates // IEEE/ACM Transactions on Networking. 1998. Vol. 6, no. 1. P. 1–14.

37. Beyer B., Jones C., Petoff J., Murphy N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol : O'Reilly Media, 2016.

38. Miao R., Zeng H., Kim C., Lee J., Yu M. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap // Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). 2017. P. 15–28.

39. Дорошенко А. Ю., Жереб К. А. Оптимізація розподілу навантаження у хмарних обчисленнях на основі прогнозування запитів // Проблеми програмування. 2021. № 2. С. 3–14.

40. Глибовець М. М., Гулаєва Н. М. Моделі та алгоритми балансування навантаження в розподілених системах обробки даних // Кібернетика і системний аналіз. 2020. Т. 56, № 4. С. 165–178.