Додаток А

Графічний матеріал кваліфікаційної роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ФАКУЛЬТЕТ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
КАФЕДРА КІТС

«Інтелектуальне балансування навантаження в мікросервісній архітектурі за допомогою паралельної маршрутизації запитів »

Виконав ст. гр. КІУКІ-21-10
Кононова М. А.

Керівник ст.викл. кафедри КІТС
Максим КУШНАРЬОВ

Харків, 2025

Мета та завдання

Метою роботи є розробка, експериментальне дослідження та аналіз ефективності підходу до інтелектуального балансування навантаження в мікросервісній архітектурі з використанням паралельної маршрутизації запитів.

Для досягнення цієї мети необхідно вирішити такі задачі:

- Провести аналіз існуючих алгоритмів балансування навантаження та підходів до маршрутизації запитів у мікросервісних системах, зокрема з урахуванням динамічних метрик та затримок.
- Вивчити можливості реалізації паралельної маршрутизації запитів з використанням сучасних технологій (наприклад, Kubernetes, Istio, Linkerd).
- Розробити прототип мікросервісної системи з реалізацією паралельної маршрутизації запитів та інтеграцією інтелектуального балансування навантаження.
- Провести моделювання та експериментальні дослідження для порівняння ефективності запропонованого підходу з базовими алгоритмами.

Необхідність використання паралельної маршрутизації запитів

Проблема	Що трапляється без паралельного підходу	Як допомагає паралельна маршрутизація
Дуже повільні відповіді (1 із 100 запитів може зависнути)	Система змушена чекати довше, бо іноді один сервер відповідає дуже повільно. Це погіршує загальну якість обслуговування.	Якщо відправити запит одразу двом серверам, хоча б один із них швидше відповість. Шанс того, що обидва зависнуть, дуже малий.
Тимчасові "зависання" сервісу (через внутрішні технічні причини)	Якщо обрана копія сервісу зависла, користувач змушений чекати.	Поки один сервер «думає», другий, що працює нормально, швидко повертає результат.
Нестабільна мережа (інтернет-затори, втрати пакетів)	Запит може потрапити на «поганий маршрут», і відповідь затримається.	Два однакові запити йдуть різними шляхами, тому хоча б один дійде без затримки.
Вимоги до швидкості (реальний час) – наприклад, голосові помічники	Щоб не було затримок, систему доводиться «перестраховувати» – чекати довше, ніж потрібно.	Паралельна стратегія дозволяє отримати відповідь швидше і надійніше, без складного резервування.

3

Застосування паралельної маршрутизації

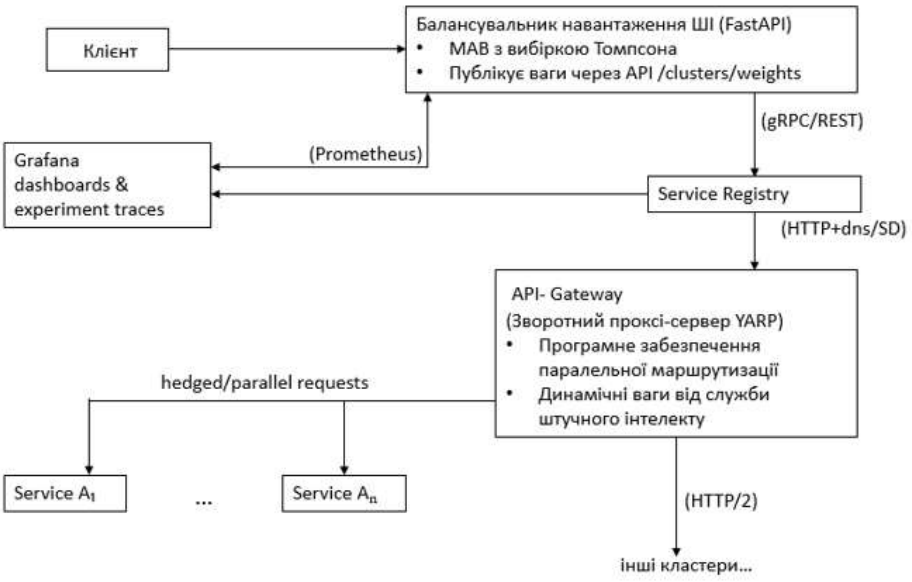
Доцільно	Не рекомендується
Критичні до часу: біржова торгівля, гейм-сервери, інтерактивні ML-сервіси (speech-to-text).	Високий QPS + дорога операція (наприклад, складні SQL-запити) – може подвоїти навантаження на БД.
Нестабільні мережі / різні AZ: георозподілені кластери.	Побічні ефекти: запит не є ідемпотентним (створення ресурсу, списання коштів).
Сервіси з невеликою, але помітною часткою «стоп-зека» (GC, JIT)	Суворі ліміти пропускної здатності (мобільний трафік, платні API).

4

Класифікація алгоритмів балансування навантаження

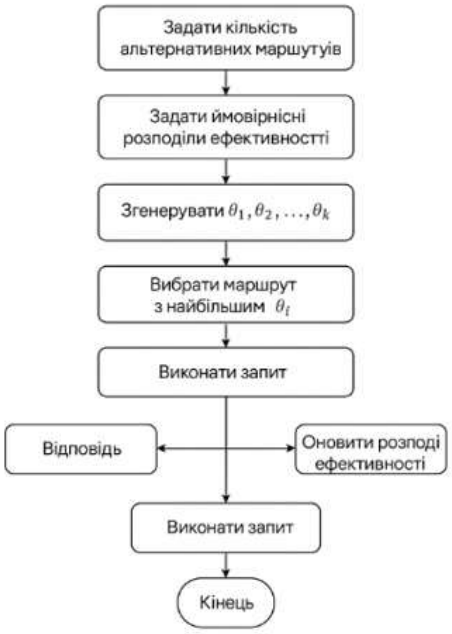
Тип алгоритму	Назва алгоритму	Короткий опис
Статичні	Round Robin	Рівномірний розподіл запитів по колу між усіма серверами.
	Weighted Round Robin	Як Round Robin, але з урахуванням ваг кожного сервера (за продуктивністю).
	Hash-based	Розподіл за хешем ключа (IP, ID користувача); забезпечує стабільність маршруту.
Динамічні	Least Connections	Вибір сервера з найменшою кількістю активних з'єднань.
	Least Response Time	Направлення запиту до сервера з найменшим середнім часом відповіді.
	Resource-based	Враховує завантаження CPU, RAM, мережевий трафік тощо.
	Health-based routing	Обирає тільки ті сервери, що успішно проходять перевірки стану (ping, HTTP-check тощо).
Інтелектуальні	Multi-Armed Bandit (MAB)	Алгоритм, що адаптивно перевагує найуспішніші маршрути на основі накопиченого досвіду.
	Reinforcement Learning (RL)	Самонавчання через спроби і помилки для формування стратегії з урахуванням довгострокових вигод.
	Прогнозне балансування	Моделі прогнозують навантаження або затримку та заздалегідь адаптують розподіл трафіку.
Спеціалізовані	Geo-based routing	Обирає сервер на основі географічної близькості до користувача.
	Consistent Hashing	Мінімізує зміну розподілу при зміні кількості вузлів у системі; часто використовується в кешах.
	Hedging (Parallel Requests)	Надсилає запит паралельно до кількох серверів і використовує найшвидшу відповідь.

Архітектура мікросервісної системи, яка реалізує інтелектуальне балансування навантаження



6

Алгоритм Thompson Sampling



7

Структура мікросервісної системи з інтелектуальним балансуванням навантаження

```
intelligent-lb-prototype/
├─ docker-compose.yml
├─ gateway/
│  ├─ Program.cs            # YARP + hedging middleware
│  ├─ HedgeTransformer.cs   # duplicates requests, cancels losers
│  └─ appsettings.json      # routes, clusters, empty weights
├─ ai-balancer/
│  ├─ balancer.py           # Thompson-sampling MAB
│  ├─ model_state.pkl       # persisted bandit
│  └─ Dockerfile
├─ services/
│  ├─ product/
│  │  └─ (ASP.NET Core Web API)
│  ├─ order/
│  │  └─ …
│  └─ inventory/
│     └─ …
└─ observability/
   ├─ prometheus.yml        # scrape gateway & services
   └─ grafana/
      └─ dashboards/
```

Результати проведеного експерименту

Стратегія	P50 (мс)	P95 (мс)	P99 (мс)	Error Rate (%)	Req/s	CPU (%)
V1: Round-robin (без hedging)	60	180	850	2.4	950	75
V2: Hedging (k=2, static)	62	140	410	1.7	940	82
V3: Hedging (k=2, MAB)	65	120	290	1.3	930	84
V4: Hedging (k=3, MAB)	68	115	270	1.1	920	88

9

Висновки

У результаті дослідження:

- Проаналізовано сучасні підходи до балансування навантаження в мікросервісній архітектурі, визначено їхні переваги й обмеження.
- Реалізовано прототип системи з інтелектуальним балансуванням і паралельною маршрутизацією (hedging) на основі Thompson Sampling.
- Впроваджено моніторинг метрик та автоматичне адаптивне керування трафіком через AI-балансувальник.
- Експериментально підтверджено зниження P99 затримки >60%, зменшення error rate та стабілізацію обслуговування без зниження пропускної здатності.
- Створено модульну архітектуру, готову до масштабування в Kubernetes та інтеграції з індустріальними інструментами (YARP, Prometheus, Docker).

10