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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ОПТИМІЗАЦІЯ АРХІТЕКТУРИ БАГАТОРІВНЕВИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ»

на здобуття освітнього ступеня магістра
зі спеціальності 172 Електронні комунікації та радіотехніка
(код, найменування спеціальності)
освітньо-професійної програми Телекомунікаційні системи та мережі
(назва)

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

_____ Владислав ЗАЙЦЕВ
(підпис) *(Ім'я ПРІЗВИЩЕ)*

Виконав: здобувач вищої освіти гр. ТСДМ-62
Владислав ЗАЙЦЕВ
Ім'я, ПРІЗВИЩЕ

Керівник: Юрій МЕЛЬНИК
д.т.н., професор *Ім'я, ПРІЗВИЩЕ*

Рецензент: _____
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Телекомунікацій

Кафедра ___Телекомунікаційних систем та мереж_____
Ступінь вищої освіти __магістр_____
Спеціальність ___172 Електронні комунікації та радіотехніка _____
Освітньо-професійна програма _____Телекомунікаційні системи та мережі_____

ЗАТВЕРДЖУЮ
Завідувач кафедри __ТСМ_____
_____ Віктор ЗАЇКА
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Зайцеву Владиславу Валентиновичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Оптимізація архітектури багаторівневих телекомунікаційних систем для підвищення якості обслуговування
керівник кваліфікаційної роботи Юрій МЕЛЬНИК, д.т.н., професор
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» 10. 2025 р. № 467

2. Строк подання кваліфікаційної роботи «___» _____ 2025 р.

3. Вихідні дані до кваліфікаційної роботи:
1. Телекомунікаційна архітектура.
2. Оптимізація архітектури багаторівневих телекомунікаційних систем.
3. Якість обслуговування.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Сучасний підхід до телекомунікаційної архітектури.
2. Використання архітектури паралельних обчислень.
3. Продуктивність програм в мережах масового обслуговування.
4. Сфера розробки прикладних телекомунікаційних систем.

5. Перелік ілюстративного матеріалу: ___презентація на ____слайдах_____

6. Дата видачі завдання «___» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	15.10.2025	Викон.
2	Розробка першого розділу	23.10.2025	Викон.
3	Розробка другого розділу	03.11.2025	Викон.
4	Розробка третього розділу	13.11.2025	Викон.
5	Розробка четвертого розділу	23.11.2025	Викон.
6	Висновки, вступ, реферат	03.12.2025	Викон.
7	Розробка презентації	13.12.2025	Викон.

Здобувач вищої освіти _____ Владислав ЗАЙЦЕВ
(підпис) *(Ім'я ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Юрій МЕЛЬНИК
(підпис) *(Ім'я ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут телекомунікацій
ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Зайцев В.В. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 172 Електронні комунікації та радіотехніка
(код, найменування спеціальності)
освітньо-професійної програми Телекомунікаційні системи та мережі
(назва)
на тему: «Оптимізація архітектури багаторівневих телекомунікаційних систем для підвищення якості обслуговування»

Кваліфікаційна робота і рецензія додаються.

Директор ННІТ _____ Владислав КРАВЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Зайцев Владислав Валентинович показав гарну теоретичну та інженерну підготовку, уміння володіти новими комп'ютерними технологіями, користуватися навчальною, довідковою і науково-технічною літературою. Працюючи над завданнями, які були поставлені керівником, проявив ініціативність, сумлінність та хист до наукової роботи.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Зайцева Владислава Валентиновича на оцінку «відмінно» та присвоїти йому кваліфікацію магістр з електронних комунікацій та радіотехніки.

Керівник кваліфікаційної роботи _____ Юрій МЕЛЬНИК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«____» _____ 20___ року

Висновок кафедри про кваліфікаційну роботу

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

Завідувач кафедри телекомунікаційних систем та мереж _____ Віктор ЗАЇКА
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

«____» _____ 20___ року

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу
на здобуття освітнього ступеня магістра

здобувача вищої освіти Зайцева Владислава Валентиновича
(прізвище, ім'я, по батькові)
на тему «Оптимізація архітектури багаторівневих телекомунікаційних систем для підвищення якості обслуговування»

Актуальність.
Сучасні телекомунікаційні системи працюють в умовах стрімкого зростання трафіку та вимог до швидкодії, надійності й адаптивності. Перехід до технологій 5G/6G, SDN, NFV та мікросервісних архітектур потребує оптимізації мережевих рішень для забезпечення високого рівня QoS. Традиційні монолітні системи вже не здатні ефективно масштабуватись і підтримувати роботу в режимі реального часу. Тому актуальним є пошук способів побудови гнучких, відмовостійких і продуктивних телекомунікаційних платформ із використанням модульності, паралельних обчислень та сервісних підходів. Це є ключем до стабільності, швидкого реагування та конкурентоспроможності мереж.
Позитивні сторони.
1. У роботі досліджена багаторівнева архітектура (n-tier), що є однією з найефективніших для сучасних телекомунікаційних систем.
2. Окрему увагу приділено модульності та супроводжуваності архітектури, які дають змогу локалізувати відмови, прискорити оновлення та тестування компонентів, забезпечуючи стабільність системи у режимі безперервної роботи.
Недоліки.
1. У роботі не враховано енергоспоживання та "Green IT" стандарти, що є важливими для сучасних ЦОД та телеком-операторів.
2. У роботі недостатньо уваги приділено QoE (Quality of Experience), як суб'єктивному показнику оцінки якості сервісів.

Відзначені зауваження не впливають на загальну позитивну оцінку магістерської кваліфікаційної роботи

Висновок: *кваліфікаційна робота на здобуття ступеня магістра заслуговує оцінку "відмінно", а здобувач Зайцев Владислав Валентинович заслуговує присвоєння кваліфікації: магістр з електронних комунікацій та радіотехніки.*

Рецензент _____ _____
науковий ступінь, вчене звання (підпис) *(Ім'я ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 83 стор., 21 рис., 9 табл., 27 джерел.

Мета роботи – розроблення підходів до оптимізації архітектури багаторівневих телекомунікаційних систем для підвищення якості обслуговування користувачів (QoS).

Об’єкт дослідження – архітектура багаторівневих телекомунікаційних систем нового покоління, що функціонують на основі сервісних, розподілених та програмно-керованих рішень (SDN, NFV, SOA, microservices).

Предмет дослідження – методи, моделі та принципи оптимізації архітектури телекомунікаційних систем з використанням модульності, паралельних обчислень, сервісно-орієнтованого підходу та технологій SDN/NFV для підвищення рівня QoS та ефективності роботи мережі.

Короткий зміст роботи: у роботі досліджено теоретичні та практичні аспекти оптимізації архітектури багаторівневих телекомунікаційних систем для підвищення їх продуктивності, надійності та рівня QoS. Проаналізовано сучасні архітектурні стилі (монолітний, SOA, мікросервісний, розподілений, паралельний) та визначено умови їх застосування залежно від типу системи й навантаження. Для кількісної оцінки ефективності застосовано моделі масового обслуговування M/M/1, що дозволило визначити оптимальне завантаження системи та вплив на затримки. Розглянуто роль паралельних обчислень (DLP, TLP, SIMT, MIMD) та гетерогенних архітектур CPU/GPU у прискоренні обробки даних у реальному часі. Показано, що модульність, балансування навантаження та сервісний підхід є ключем до створення адаптивних архітектур, здатних реалізувати концепції 5G/6G, SDN і NFV.

КЛЮЧОВІ СЛОВА: БАГАТОРІВНЕВА АРХІТЕКТУРА, QOS, SDN, NFV, 5G, 6G, SOA, МІКРОСЕРВІСНА АРХІТЕКТУРА, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, M/M/1, АДАПТИВНІСТЬ, ВІДМОВОСТІЙКІСТЬ, МОДУЛЬНІСТЬ, ОРКЕСТРАЦІЯ СЕРВІСІВ, ОПТИМІЗАЦІЯ МЕРЕЖІ

ABSTRACT

Text part of the master's qualification work: 83 pages, 21 figures, 9 table, 27 sources.

The purpose of the work is to develop approaches to optimizing the architecture of multi-tier telecommunication systems to improve the quality of user service (QoS).

Object of the study is the architecture of multi-tier telecommunication systems of the new generation, operating on the basis of service, distributed and software-driven solutions (SDN, NFV, SOA, microservices).

Subject of the study is methods, models and principles of optimizing the architecture of telecommunication systems using modularity, parallel computing, service-oriented approach and SDN/NFV technologies to increase the level of QoS and network efficiency.

Summary of the work: the work investigates theoretical and practical aspects of optimizing the architecture of multi-tier telecommunication systems to increase their productivity, reliability and QoS level. Modern architectural styles (monolithic, SOA, microservice, distributed, parallel) are analyzed and the conditions for their application are determined depending on the type of system and load. To quantify the efficiency, the M/M/1 queuing model was used, which allowed us to determine the optimal system load and the impact on delays. The role of parallel computing (DLP, TLP, SIMT, MIMD) and heterogeneous CPU/GPU architectures in accelerating real-time data processing is considered. It is shown that modularity, load balancing and a service approach are the keys to creating adaptive architectures capable of implementing the concepts of 5G/6G, SDN and NFV.

KEYWORDS: MULTI-TIER ARCHITECTURE, QOS, SDN, NFV, 5G, 6G, SOA, MICROSESERVICE ARCHITECTURE, PARALLEL COMPUTING, M/M/1, ADAPTABILITY, FAILURE TOLERANCE, MODULARITY, SERVICE ORCHESTRATION, NETWORK OPTIMIZATION

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1 СУЧАСНИЙ ПІДХІД ДО ТЕЛЕКОМУНІКАЦІЙНОЇ АРХІТЕКТУРИ 11
1.1 Архітектурна модульність 11
1.2 Супроводжуваність архітектури 15
1.3 Тестованість, розгортання та масштабованість 22
РОЗДІЛ 2 ВИКОРИСТАННЯ АРХІТЕКТУРИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ 35
2.1 Використання нейронних мереж 35
2.2 Рівні паралельних обчислень 38
2.3 Архітектури паралельних обчислень 49
РОЗДІЛ 3 ПРОДУКТИВНІСТЬ ПРОГРАМ В МЕРЕЖАХ МАСОВОГО ОБСЛУГОВУВАННЯ 55
3.1 Багаторівневі застосунки 55
3.2 Мережа масового обслуговування, що описує роботу трирівневої хмарної програми 57
3.3 Результати імітаційного моделювання 61
РОЗДІЛ 4 СФЕРА РОЗРОБКИ ПРИКЛАДНИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ 66
4.1 Структура та область розробки 66
4.2 Архітектурні стилі 69
4.3 Класифікація додатків 77
ВИСНОВКИ 82
ПЕРЕЛІК ПОСИЛАНЬ 84
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 87

ВСТУП

Сучасні телекомунікаційні системи розвиваються в умовах постійного зростання обсягів даних, підвищення вимог до швидкодії та стабільності зв’язку. Поява технологій 5G, 6G, SDN і NFV змінює підхід до побудови мережевих архітектур, надаючи операторам можливість створювати динамічні, масштабовані та керовані програмно інфраструктури. Разом із тим це вимагає нових рішень щодо архітектури систем, які б забезпечували не лише високу продуктивність, а й гарантовану якість обслуговування користувачів (QoS).

Одним із найважливіших напрямів підвищення ефективності телекомунікаційних систем є оптимізація архітектури багаторівневих мережевих рішень. Вона передбачає раціональний розподіл функцій між рівнями представлення, бізнес-логіки та зберігання даних, що дозволяє підвищити стабільність, спростити масштабування та полегшити модернізацію системи. Завдяки чіткій структуризації компонентів досягається стійкість до відмов і знижується ризик перевантаження мережі, що безпосередньо впливає на якість телекомунікаційних послуг.

З розвитком розподілених і мікросервісних підходів телекомунікаційна інфраструктура поступово переходить до децентралізованих архітектур, які дають змогу гнучко розподіляти навантаження, підвищувати відмовостійкість і спрощувати інтеграцію з хмарними сервісами. При цьому використання таких архітектурних патернів, як оркестрація та хореографія, забезпечує узгоджену взаємодію між сервісами, дозволяючи підтримувати безперервність процесів навіть у складних багатосервісних середовищах.

Одним із важливих напрямів оптимізації є впровадження паралельних обчислень. Моделі DLP, TLP, ILP, SIMT, MIMD дозволяють виконувати обробку великої кількості даних у реальному часі, рівномірно розподіляючи ресурси між обчислювальними вузлами. Такі підходи особливо ефективні у високонавантажених телекомунікаційних системах, де навіть незначні затримки можуть призвести до втрати пакетів або зниження пропускної здатності мережі.

Актуальність теми дипломної роботи зумовлена необхідністю створення архітектур, здатних одночасно забезпечувати продуктивність, масштабованість, гнучкість і надійність при мінімальних експлуатаційних витратах. В умовах стрімкого зростання кількості користувачів і трафіку відсутність оптимізованої архітектури призводить до зниження стабільності мережі, збільшення затримок і ризику збоїв, що безпосередньо впливає на якість послуг і конкурентоспроможність операторів зв'язку.

Метою роботи є розроблення підходів до оптимізації архітектури багаторівневих телекомунікаційних систем для підвищення якості обслуговування користувачів і ефективності використання ресурсів. Для досягнення цієї мети розглянуто різні архітектурні стилі, моделі систем масового обслуговування, методи балансування навантаження та принципи структуризації компонентів.

Практична цінність дослідження полягає у можливості використання його результатів для побудови телекомунікаційних платформ нового покоління, що відповідають концепціям 5G, 6G, SDN і NFV, забезпечують підвищення рівня QoS, ефективне керування трафіком і безперервність надання сервісів.

1 СУЧАСНИЙ ПІДХІД ДО ТЕЛЕКОМУНІКАЦІЙНОЇ АРХІТЕКТУРИ

1.1 Архітектурна модульність

Сучасні підприємства перебувають у стані постійної динаміки: ринок розвивається з безпрецедентною швидкістю, і це змушує бізнес постійно пристосовуватися до нових умов. Посилення конкуренції, глобалізація, зростання споживчих очікувань, технологічні інновації — зокрема автоматизація, машинне навчання та штучний інтелект — формують нові вимоги до інформаційних і телекомунікаційних систем. Такі зміни неминуче вимагають глибокої трансформації архітектури IT-інфраструктури, що лежить в основі діяльності будь-якої організації.

Однак змінюється не лише сам бізнес — змінюється і технічне середовище, у якому він функціонує. Поширення контейнеризації, перехід до хмарних середовищ, розвиток методологій DevOps і CI/CD, інтеграція віртуалізованих мережевих функцій (NFV) і програмно-визначених мереж (SDN) — усе це вимагає від архітектури гнучкості, адаптивності та здатності до швидкої еволюції. У телекомунікаційній сфері, де рівень навантаження, обсяг трафіку та вимоги до QoS (Quality of Service) постійно зростають, ці архітектурні принципи стають не просто перевагою, а стратегічною необхідністю.

Архітектура програмного забезпечення виступає «фундаментом» системи — її основною конструкцією, на якій будується вся функціональність. Проте, на відміну від архітектури фізичних споруд, програмна архітектура не може залишатися статичною. Вона повинна постійно змінюватися, щоб відповідати сучасним бізнес-моделям, новим технологіям і вимогам до швидкості обслуговування клієнтів. У телекомунікаціях це має безпосередній вплив на доступність сервісів, стабільність передачі даних та мінімізацію затримок (latency) — параметрів, що безпосередньо визначають якість користувацького досвіду.

У сучасному світі злиття й поглинання компаній стали звичним явищем, особливо у сфері телекомунікацій. Коли одна телеком-компанія придбає іншу, вона

успадковує не лише мережеву інфраструктуру, персонал та обчислювальні ресурси, а й нових абонентів. У таких умовах постає ключове питання: чи здатні наявні інформаційні системи масштабуватися, щоб підтримувати новий рівень навантаження, забезпечуючи при цьому незмінно високий рівень QoS? Саме масштабованість, гнучкість і розширюваність архітектури визначають, наскільки стабільно мережа функціонуватиме в умовах різкого зростання кількості користувачів або розширення сервісів.

Монолітні системи, які розгортаються як єдиний блок, зазвичай не здатні забезпечити потрібний рівень продуктивності й надійності. Коли навантаження зростає, обчислювальні ресурси (ядра процесора, оперативна пам’ять, потоки виконання) вичерпуються надто швидко, що призводить до зниження швидкодії сервісів, підвищення затримок та можливих збоїв. У телекомунікаціях подібні наслідки проявляються як погіршення пропускної здатності, збільшення часу відповіді систем керування мережею (OSS/BSS) і, зрештою, зниження задоволеності абонентів.

Для розуміння цього можна використати аналогію: сервер або віртуальна машина — це склянка, а застосунок — це вода. Якщо «вода» (тобто застосунок) продовжує збільшувати свій обсяг, вона заповнює склянку до країв, і система перестає бути стабільною. Додавання ще однієї «склянки» (серверу) не вирішує проблему — потрібно розподілити навантаження між кількома ємностями. Саме це і є принцип архітектурної модульності — розділення великих монолітних застосунків на незалежні, менші компоненти (сервіси або мікросервіси), які можуть масштабуватися окремо, залежно від реальних потреб.

Архітектурна модульність відкриває можливість швидкого масштабування, гнучкого керування ресурсами та оптимізації обчислювального навантаження. У контексті телекомунікацій це означає, що мережеві сервіси, системи білінгу, аутентифікації користувачів або моніторингу трафіку можуть оновлюватися та масштабуватися незалежно один від одного, не створюючи ризику відмови всієї системи. Такий підхід безпосередньо підвищує QoS, забезпечуючи стабільність, прогнозованість і мінімальні простої.

Крім того, модульна архітектура сприяє оперативному впровадженню інновацій — наприклад, нових методів оптимізації маршрутизації або алгоритмів керування трафіком у мережах 5G/6G. Це дає змогу швидко адаптуватися до змін попиту, ефективніше використовувати ресурси мережі та підтримувати безперервність надання послуг. Гнучкість і швидкість реагування, притаманні модульній архітектурі, дають змогу телеком-операторам забезпечувати високий рівень SLA (Service Level Agreement) навіть у пікові періоди навантаження.

Як зазначають Девід Бенджамін і Девід Комлос у журналі *Forbes* (2020), головна різниця між лідерами й тими, хто відстає, полягає у здатності швидко й ефективно змінювати курс. Для телекомунікацій ця теза має особливу вагу: ринок, технології та вимоги користувачів змінюються настільки швидко, що лише архітектурно гнучкі системи здатні гарантувати стабільну якість сервісів і мінімізувати втрати.

Отже, архітектурна модульність, масштабованість і гнучкість — це не лише питання ефективності розробки ПЗ, а ключові чинники, що визначають якість обслуговування користувачів у телекомунікаційних мережах. Вони забезпечують здатність компаній не лише витримувати високі навантаження, а й оперативно реагувати на зміни, підтримуючи високу надійність, стабільність і доступність сервісів — тобто ті параметри, які лежать в основі сучасних стандартів QoS та QoE.

Рушійні сили модульності

Архітектори не повинні розділяти систему на дрібні частини без наявності чітких бізнес-передумов. Основними причинами для поділу застосунків є швидкість виходу на ринок (її також називають *time-to-market*) та прагнення досягти певного рівня конкурентної переваги на цьому ринку.

Швидкість виходу на ринок досягається завдяки гнучкості архітектури — її здатності швидко реагувати на зміни зовнішнього середовища, вимоги користувачів чи бізнес-процеси. Гнучкість — це складна характеристика, що включає у себе інші важливі властивості, такі як супроводжуваність, тестованість і розгортання. Високий рівень цих характеристик дозволяє скоротити час розробки, впровадження

та модернізації системи, що є критично важливим для телекомунікаційних компаній, де ринок і технології розвиваються надзвичайно швидко.

Конкурентна перевага досягається завдяки поєднанню швидкого виходу на ринок із такими властивостями, як масштабованість, висока доступність та відмовостійкість застосунків. Чим ефективніше працює компанія, тим швидше вона розвивається, і тим вищою стає потреба в масштабованій архітектурі, здатній підтримувати зростаючу активність користувачів та обсяг сервісів.

Відмовостійкість — це здатність застосунку відновлюватися після збоїв і продовжувати роботу без значного впливу на користувачів. Вона дозволяє ізолювати помилки, запобігаючи їх розповсюдженню на інші частини системи, що особливо важливо для телекомунікаційних платформ реального часу, де будь-яка затримка або відмова може вплинути на якість обслуговування (QoS) та задоволеність абонентів.

Компанії мають залишатися гнучкими, щоб вижити у сучасному, швидкому та постійно змінюваному середовищі, а отже — і базові архітектури повинні мати відповідну гнучкість. П’ять ключових архітектурних характеристик, які забезпечують гнучкість, швидкість виходу на ринок і, зрештою, конкурентну перевагу, — це:

- доступність (відмовостійкість),
- масштабованість,
- розгортання,
- тестованість,
- супроводжуваність.

У сукупності вони визначають, наскільки швидко система може розвиватися, реагувати на зміну навантаження, а також підтримувати високу якість обслуговування користувачів, що є критичним фактором для телекомунікаційних операторів і провайдерів цифрових сервісів.

Варто зазначити, що архітектурна модульність не завжди передбачає повний перехід до розподіленої архітектури. Покращеної супроводжуваності, тестованості та розгортання можна досягти навіть у межах монолітних архітектур, зокрема таких

як модульний моноліт або мікроядерна архітектура (додаткову інформацію про ці стилі наведено в додатку Б).

Обидва підходи пропонують певний рівень модульності завдяки специфічному способу структуризації компонентів. Наприклад, у модульному моноліті компоненти групуються у логічно сформовані предметні області (домени), утворюючи так звану доменно-поділену архітектуру.

У мікроядерній архітектурі основна функціональність розподіляється між підключними компонентами, що дозволяє значно скоротити обсяг тестування, підвищити надійність оновлень і спростити процес розгортання. Такий підхід особливо ефективний для телекомунікаційних систем, які мають обслуговувати мільйони користувачів у режимі реального часу — адже він забезпечує стабільність, зменшує ризики збоїв і підтримує високу якість надання послуг.

1.2 Супроводжуваність архітектури

Під супроводжуваністю розуміють простоту внесення змін у систему — додавання, модифікації чи видалення функціональних можливостей, а також застосування внутрішніх технічних оновлень, таких як виправлення помилок, оновлення фреймворків або інтеграція сторонніх компонентів.

Як і більшість характеристик композитної архітектури, рівень супроводжуваності складно визначити об’єктивно, адже він залежить від багатьох чинників — структури системи, ступеня зв’язності між компонентами, якості коду та глибини модульності.

Попри це, у сучасній інженерній практиці існують різні спроби створення метрик, які дозволяють кількісно оцінювати супроводжуваність застосунку. Такі метрики зазвичай враховують складність архітектури, рівень залежностей, кількість точок взаємодії між модулями та загальну стабільність системи при внесенні змін.

$$ML = 100 \times \sum_{i=1}^{k} c_i \quad (1.1)$$

де ML — це оцінка супроводжуваності всієї системи (у відсотках від 0 до 100 %),
k — загальна кількість логічних компонентів у системі,
а c_i — рівень зв'язаності конкретного компонента, з особливим акцентом на вхідні зв'язки.

Згідно з цим рівнянням, чим вищий рівень вхідної залежності між компонентами, тим нижчою є загальна оцінка супроводжуваності програмного коду. Іншими словами, чим більше один компонент залежить від інших, тим складніше підтримувати, оновлювати й розвивати систему без ризику порушення її цілісності.

Якщо відкинути математичний аспект, то відносну супроводжуваність застосунку можна оцінювати за низкою архітектурних показників, які характеризують структуру та якість взаємодії між компонентами. До таких показників належать:

- Зв'язаність (coupling) — ступінь взаємозалежності між компонентами та спосіб, у який вони обмінюються інформацією.
- Зв'язність (cohesion) — рівень узгодженості й логічної цілісності операцій усередині кожного компонента.
- Цикломатична складність — показник рівня вкладеності, умовних переходів і розгалужень у коді, що впливає на простоту тестування й аналізу.
- Розмір компонента — кількість операторів або функціональних блоків, що входять до його складу.
- Технічне та предметне розділення — спосіб групування компонентів за технічним призначенням або за предметною областю системи.

Такі характеристики дозволяють оцінити, наскільки архітектура програмного забезпечення є структурованою, зрозумілою та придатною до масштабування й модернізації, що безпосередньо впливає на якість і стабільність телекомунікаційних систем.

У контексті архітектури компонент розглядається як архітектурний будівельний блок застосунку, який виконує певну інфраструктурну або бізнес-функцію. Зазвичай він має чітке представлення у вигляді пакета (package) у мовах типу *Java*, простору імен (namespace) у *C#*, або ж фізичної групи файлів класів, організованих у вигляді структури каталогів.

Наприклад, компонент OrderHistory (історія замовлень) може бути реалізований як набір класів, розташованих у просторі імен app.business.order.history. Така структуризація допомагає розмежовувати відповідальність між частинами системи, проте ефективність подібного підходу значною мірою залежить від загальної архітектурної організації та рівня зв’язності між компонентами.

У великих монолітних архітектурах супроводжуваність зазвичай є низькою. Це зумовлено тим, що функціональність розділяється технічно (за рівнями — інтерфейс, бізнес-логіка, база даних), а не предметно (за бізнес-сенсом). Компоненти в таких системах часто тісно пов’язані один з одним, що ускладнює внесення навіть незначних змін.

Уявімо ситуацію: у традиційній монолітній багаторівневій архітектурі потрібно додати дату завершення дії для товарів у списку бажань клієнта (тобто список товарів, які користувач планує потенційно придбати пізніше).

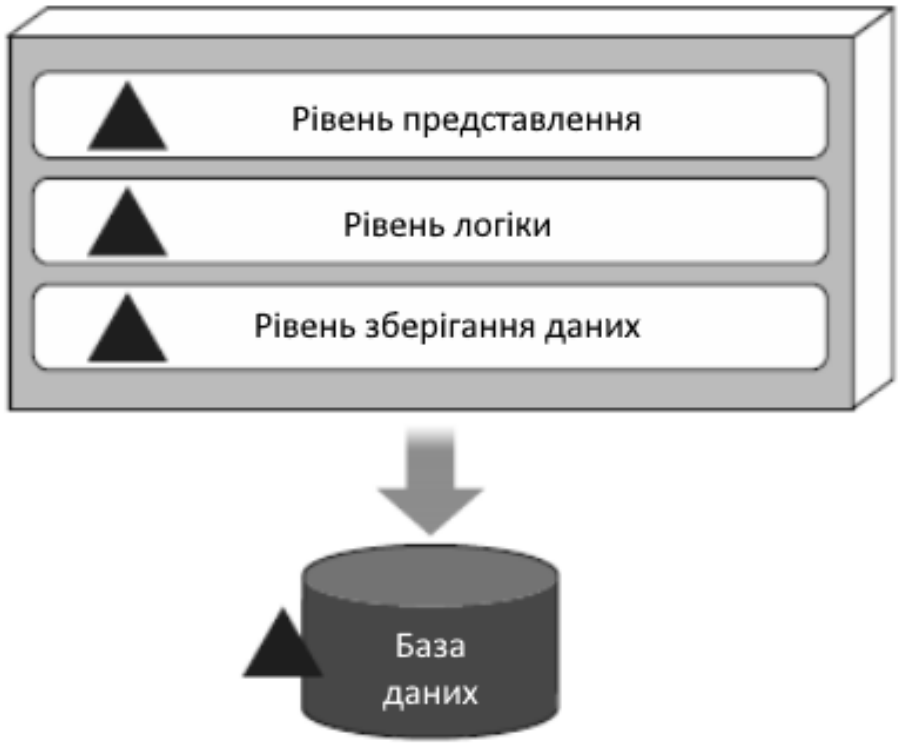


Рис. 1.1 Ілюстрація поширення змін між різними рівнями монолітної архітектури — інтерфейс, логіка, база даних

На цьому прикладі видно, що навіть проста зміна — додавання нового поля — може спричинити ланцюгову реакцію змін на всіх рівнях системи. Нова бізнес-вимога — «дата завершення дії елементів списку бажань» — має бути реалізована в декількох незалежних модулях: від інтерфейсу користувача до бази даних.

Впровадження такого, на перший погляд, незначного оновлення потребує координації роботи щонайменше трьох команд розробників:

- Команда користувацького інтерфейсу (UI) має додати нове поле введення для дати завершення дії в інтерфейсі застосунку.
- Команда бізнес-логіки (Business Logic) повинна реалізувати відповідні бізнес-правила — наприклад, автоматичне видалення або сповіщення про завершення терміну дії, а також змінити контракти обміну даними між UI та серверною частиною.
- Команда бази даних (Database Team) мусить оновити схему БД — додати новий стовпець ExpirationDate у таблицю Wishlist, що зберігає елементи списку бажань.

Таким чином, у монолітних архітектурах навіть мінімальні зміни потребують багаторівневої координації та тестування всієї системи в комплексі. Це значно ускладнює підтримку, збільшує час розробки й підвищує ризики виникнення помилок у суміжних модулях.

Для телекомунікаційних систем, які характеризуються великим обсягом взаємозалежних сервісів, така архітектура створює суттєві бар’єри для оперативного оновлення функцій, впровадження нових протоколів або інтеграції з іншими системами. Саме тому сучасні підходи (такі як модульна або мікросервісна архітектура) стають необхідною умовою забезпечення гнучкості, масштабованості та високої якості обслуговування (QoS) у складних корпоративних і телекомунікаційних середовищах.

У межах монолітної архітектури предметна область, наприклад Wishlist (список бажань), охоплює всю систему загалом. Це означає, що навіть локальні зміни в межах цієї області впливають на інші компоненти, розташовані на різних рівнях — від користувацького інтерфейсу до бази даних. Через таку всеохопність підтримка окремої області або підобласті стає складною, трудомісткою та ризикованою, оскільки будь-яка модифікація потребує ретельної координації між кількома командами розробників і тестування всієї системи в цілому.

На відміну від цього, модульні архітектури розділяють систему на менші, незалежні домени та піддомени, кожен з яких реалізований у вигляді окремого компонента або мікросервісу, що може бути розгорнутий автономно. Така структуризація дозволяє виконувати зміни локально, не зачіпаючи інші частини системи, що істотно зменшує ризики та скорочує час оновлення.

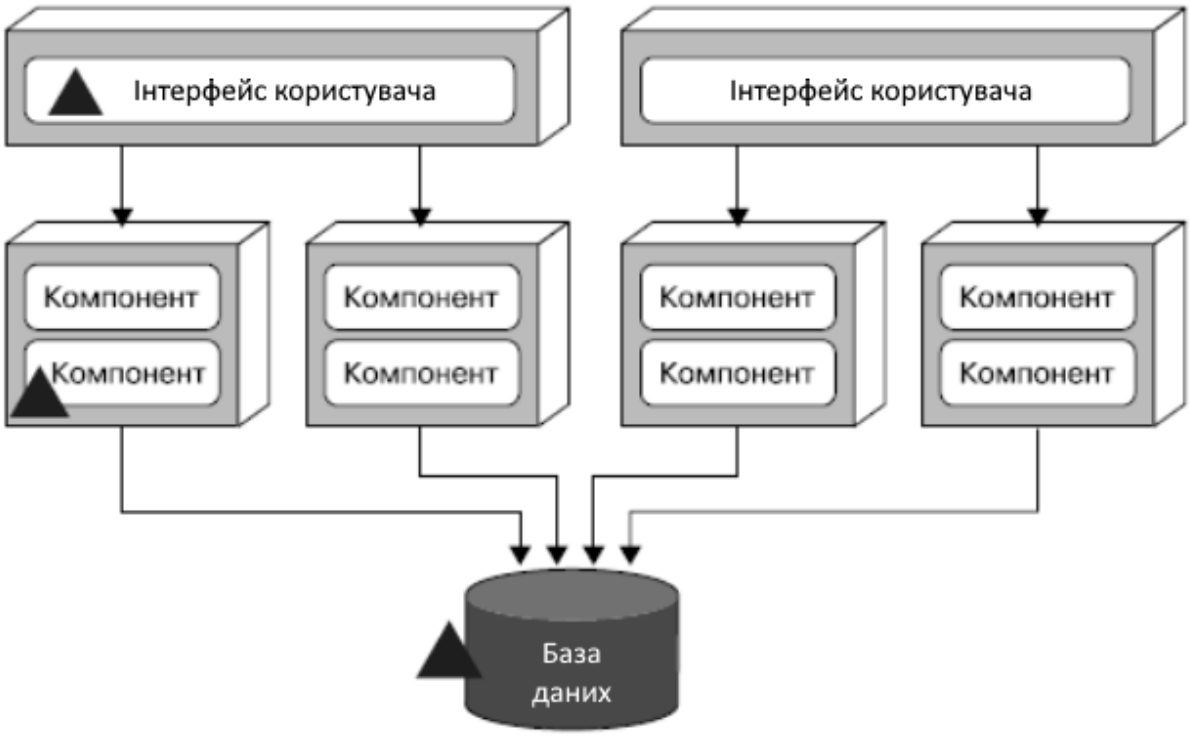


Рис. 1.2 Ілюстрація різниці між монолітною та модульною архітектурою — зокрема, локалізація змін у межах окремого сервісу

У розподіленій архітектурі на основі сервісів область зміни, пов’язана з новою вимогою, розташовується в межах конкретного домену або сервісу, а не поширюється на всю систему. Наприклад, якщо потрібно додати нове поле або логіку, це робиться лише в одному мікросервісі, відповідальному за цю функцію, без необхідності змін у суміжних підсистемах.

Такий підхід значно спрощує ідентифікацію компонентів, що потребують оновлення, і мінімізує вплив змін на інші частини системи. Крім того, це підвищує масштабованість, надійність та якість обслуговування (QoS) — критичні показники для телекомунікаційних платформ, де будь-які зміни мають відбуватися швидко, безперервно й без ризику порушення роботи мережевих сервісів.

Подальший розвиток архітектури в напрямку ще більшої модульності — наприклад, мікросервісної архітектури, як показано на рисунку нижче, — дозволяє локалізувати зміни в межах конкретного функціонального рівня. Це означає, що нові вимоги або модифікації обмежуються лише тим сервісом, який безпосередньо відповідає за певну функціональність — у нашому прикладі це сервіс, який реалізує роботу списку бажань (Wishlist).

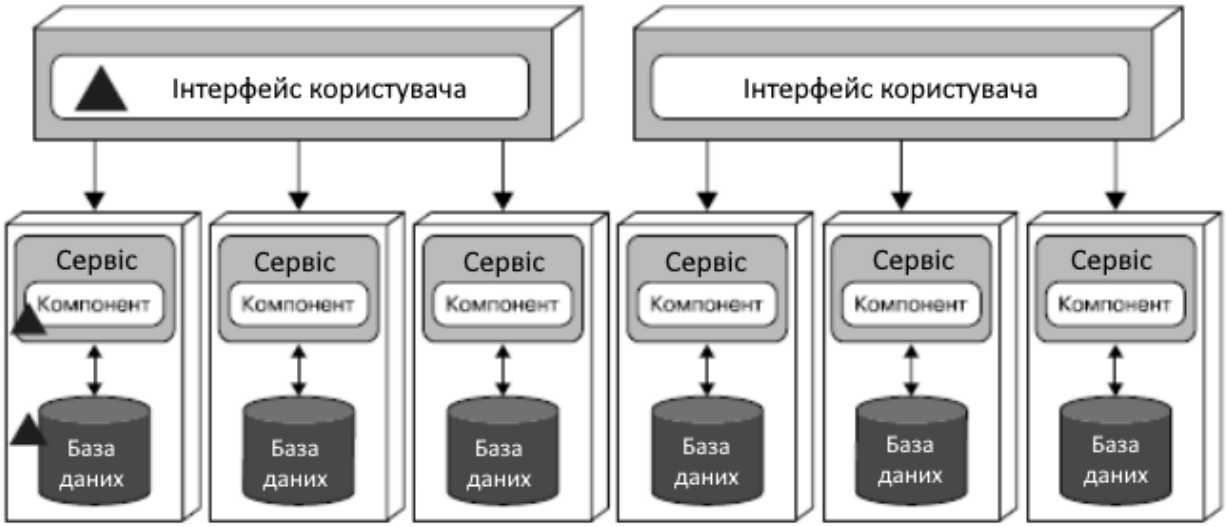


Рис. 1.3 Ілюстрація ізоляції змін у межах конкретного мікросервісу, відповідального за окрему бізнес-функцію

На відміну від монолітних систем, де будь-яка зміна поширюється на кілька рівнів і вимагає координації між різними командами, мікросервісна архітектура забезпечує функціональну незалежність модулів. Кожен сервіс має власний життєвий цикл, базу даних і логіку, тому внесення змін у нього не впливає на інші частини системи. Це не лише підвищує стабільність, а й дозволяє розробникам швидше впроваджувати нові можливості, оновлення чи виправлення помилок.

Таким чином, перехід до мікросервісної моделі демонструє, як зростання рівня архітектурної модульності безпосередньо призводить до спрощення супроводження системи. Чим вищий ступінь ізоляції компонентів, тим легше виконувати додавання, зміну або видалення функціональних елементів, не порушуючи загальної роботи застосунку.

Для телекомунікаційних систем, які часто складаються з великої кількості взаємопов’язаних сервісів (аутентифікація користувачів, білінг, маршрутизація трафіку, моніторинг тощо), така архітектура є особливо ефективною. Вона забезпечує гнучкість, масштабованість і високу якість обслуговування (QoS), дозволяючи системі еволюціонувати поступово, без ризику глобальних збоїв або втрати працездатності.

1.3 Тестованість, розгортання та масштабованість

Тестованість — це не просто здатність системи до перевірки правильності своєї роботи. Вона включає дві взаємопов’язані складові:

1. Легкість проведення тестування, зокрема можливість автоматизованої перевірки функціональності.
2. Повноту охоплення тестами, тобто наскільки всебічно перевіряються всі аспекти поведінки системи.

Це одна з найважливіших характеристик архітектури, адже саме від рівня тестованості залежить швидкість випуску нових версій, стабільність роботи продукту та здатність оперативно реагувати на зміни бізнес-вимог. Вона є фундаментом архітектурної гнучкості, що визначає, наскільки безпечно система може еволюціонувати без ризику порушень у її роботі.

У великих монолітних архітектурах, наприклад у класичних багаторівневих структурах, тестування перетворюється на складний і громіздкий процес. Навіть незначна зміна — скажімо, корекція бізнес-правила чи оновлення однієї функції — може потребувати запуску повного регресійного тестування всієї системи. Це означає перевірку сотень або навіть тисяч модулів, багато з яких узагалі не пов’язані з оновленим кодом.

Подібний підхід має низку недоліків:

- Велика тривалість виконання тестів — автоматизовані системи тестування можуть працювати годинами або навіть добами.
- Висока вартість підтримки тестів — будь-які зміни в структурі коду потребують оновлення значної кількості сценаріїв.
- Психологічне навантаження на розробників — коли після невеликої зміни «падають» десятки тестів, складно швидко знайти справжню причину.
- Ризик накопичення технічного боргу — оскільки команди відкладають тестування або скорочують його обсяг через надмірну складність.

У результаті гнучкість архітектури знижується: будь-яке оновлення стає ризиковим і повільним, а підтримка системи — витратною.

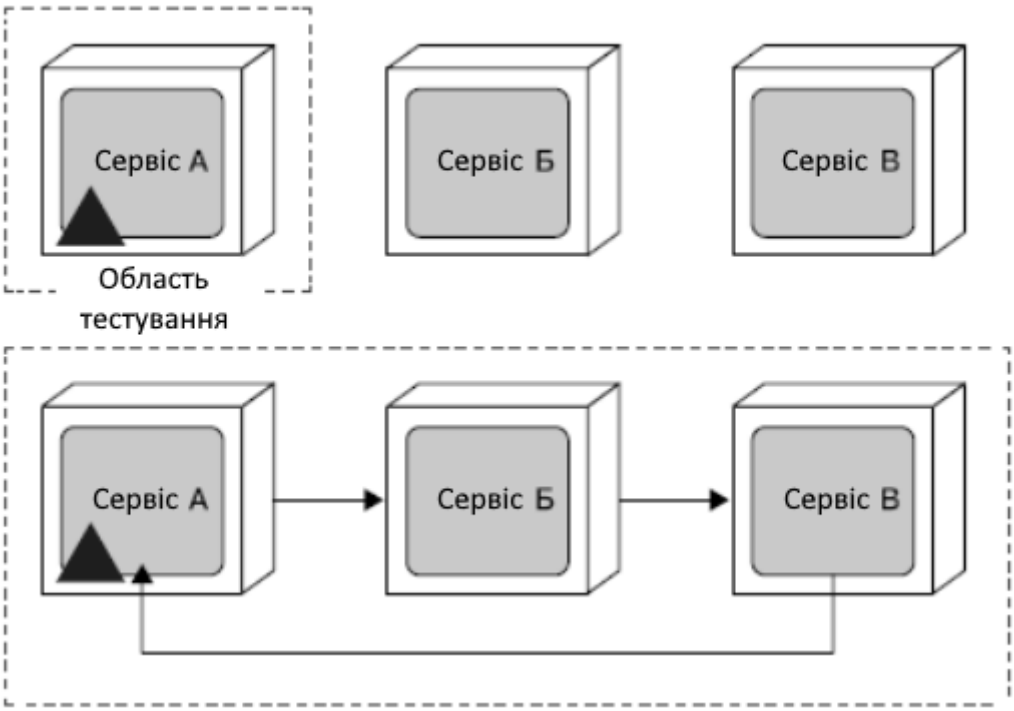


Рис. 1.4 Приклад розподілу системи на сервіси A, B, C і зміну області тестування залежно від рівня їхньої взаємодії

Вирішенням цієї проблеми стає архітектурна модульність — розділення застосунку на менші, незалежні одиниці розгортання, кожна з яких може тестуватися окремо. Такий підхід суттєво зменшує обсяг тестування після внесення змін, адже перевірці підлягає лише той модуль, де ці зміни були зроблені. Це підвищує повноту тестового охоплення і водночас спрощує сам процес.

Модульна архітектура дозволяє:

- створювати менші, спеціалізовані набори тестів, орієнтовані на окремі функції або сервіси;
- прискорити CI/CD-процеси, оскільки тести виконуються паралельно для різних компонентів;
- підвищити надійність релізів, адже кожна частина системи перевіряється незалежно;
- зменшити вплив людського чинника, оскільки автоматизовані тести легше масштабуються й оновлюються.

Візьмімо приклад застосунку, розділеного на три незалежні сервіси — A, B і C. Якщо зміни відбулися лише в сервісі A, то область тестування обмежується ним

самим. Сервіси B і C не залежать від цього оновлення, тому не потребують перевірки. Однак, якщо між сервісами встановлюється взаємодія (наприклад, A звертається до API B або отримує дані з C), то при внесенні змін у A потрібно перевіряти й інтеграцію з рештою компонентів. Таким чином, з ростом кількості зв’язків між сервісами зменшується рівень тестованості, оскільки зростає складність тестових сценаріїв і час на перевірку.

Попри це, навіть у складних взаємопов’язаних системах архітектурна модульність забезпечує значно вищу передбачуваність і стабільність, ніж традиційний моноліт. Ключовим завданням архітектора є правильне визначення меж між сервісами, щоб вони залишалися максимально незалежними та не створювали надмірних інтеграційних залежностей.

Для телекомунікаційних систем, які обслуговують мільйони користувачів у режимі реального часу, така властивість має стратегічне значення. Вона дозволяє проводити оновлення без простоїв, підтримувати високу доступність мережевих сервісів, забезпечувати якість обслуговування (QoS) на постійному рівні та зменшувати ризики під час масштабування чи розширення функціональності.

Розгортаність — це не лише простота самого процесу розгортання програмного забезпечення, а й частота таких оновлень та ризик виникнення помилок, що можуть супроводжувати цей процес. Високий рівень розгортаності означає, що система може швидко та безпечно оновлюватися, підтримуючи стабільність роботи і водночас оперативно реагуючи на зміни вимог користувачів або бізнесу.

Щоб забезпечити гнучкість архітектури і швидке впровадження інновацій, застосунки мають відповідати трьом ключовим критеріям:

1. Простота розгортання — мінімум ручних дій і ризиків;
2. Висока частота оновлень — можливість регулярного й безпечного випуску нових версій;
3. Низький рівень помилок під час розгортання — стабільність навіть у середовищах з великою кількістю сервісів.

Якщо ж оновлення здійснюється раз на два тижні або рідше, то ризик виникнення збоїв суттєво зростає. Причина полягає в тому, що одночасно розгортається велика кількість змін, які можуть конфліктувати між собою. Це також призводить до затримки випуску нових функцій чи виправлень, які вже готові, але чекають на чергове «вікно оновлення».

В ідеалі, архітектура має дозволяти регулярні та безпечні розгортання, при цьому не порушуючи роботу користувачів. Проте частота оновлень має бути збалансована з можливостями клієнта або кінцевого користувача — тобто система не повинна оновлюватися швидше, ніж користувачі здатні адаптуватися до змін.

Процес розгортання зазвичай супроводжується низкою церемоній, таких як заморожування коду, тестові релізи, перевірка стабільності середовища, ручні перевірки або схвалення оновлень. Для монолітних архітектур це особливо характерно — будь-яка зміна потребує ретельної координації між командами, тривалого тестування та великої кількості перевірок.

У результаті моноліти мають низьку розгортаність:

- частота оновлень знижується (від кількох тижнів до кількох місяців);
- ризик збоїв зростає, адже зміни впливають на багато частин системи одночасно;
- збільшується кількість ручних дій, що уповільнює процес і створює додаткові ризики.

На противагу цьому, модульні або мікросервісні архітектури, у яких застосунок складається з незалежно розгортуваних компонентів, дозволяють зменшити кількість церемоній, прискорити доставку оновлень і знизити ризики. Кожен сервіс може бути оновлений автономно, без потреби перевіряти весь продукт. Це робить систему більш гнучкою, надійною й готовою до безперервного вдосконалення.

Однак існує і зворотна сторона модульності. Якщо розмір сервісів стає надто малим, а взаємодій між ними — надто багато, це починає негативно впливати не лише на тестованість, але й на саму розгортаність. Коли між сервісами формується щільна залежність, будь-яке оновлення може спричинити ланцюгові помилки.

З’являється страх, що навіть невелика зміна в одному компоненті може порушити роботу інших.

Як зауважив Мэтт Стайн (Matt Stine) у своїй праці про оркестрацію мікросервісів,

«Якщо мікросервіси потрібно розгортати як повний набір у певній послідовності — краще зібрати їх назад у моноліт і не створювати собі зайвих проблем».

Це висловлювання дуже влучно описує ситуацію, коли архітектура втрачає свою модульність і перетворюється на те, що часто називають «великим клубком розподіленого безладу». У такій системі немає ані гнучкості, ані незалежності сервісів — лише складна, нестабільна структура, де кожне оновлення стає ризиком.

Отже, справжня архітектурна модульність полягає не лише у поділі застосунку на сервіси, а у здатності цих сервісів розгортатися автономно, без порушення загальної стабільності. Для телекомунікаційних систем, які працюють безперервно 24/7, цей принцип має особливе значення. Він гарантує високу доступність сервісів, скорочує час відновлення після збоїв (MTTR) та забезпечує постійно високу якість обслуговування (QoS) — навіть у процесі впровадження оновлень у реальному часі.

Масштабованість — це здатність системи залишатися стабільною, ефективною та чутливою до запитів користувачів у процесі поступового збільшення навантаження з часом. Іншими словами, масштабована система здатна розширювати свої ресурси — обчислювальні, мережеві або зберігання даних — у міру зростання кількості користувачів або обсягів операцій.

Тісно пов’язана з нею характеристика — адаптивність. Вона описує здатність системи швидко реагувати на раптові, короткочасні або нерівномірні сплески навантаження, зберігаючи при цьому стабільну швидкість обробки запитів. Якщо масштабованість забезпечує стійкість до поступового зростання попиту, то адаптивність гарантує миттєву реакцію на пікові ситуації.

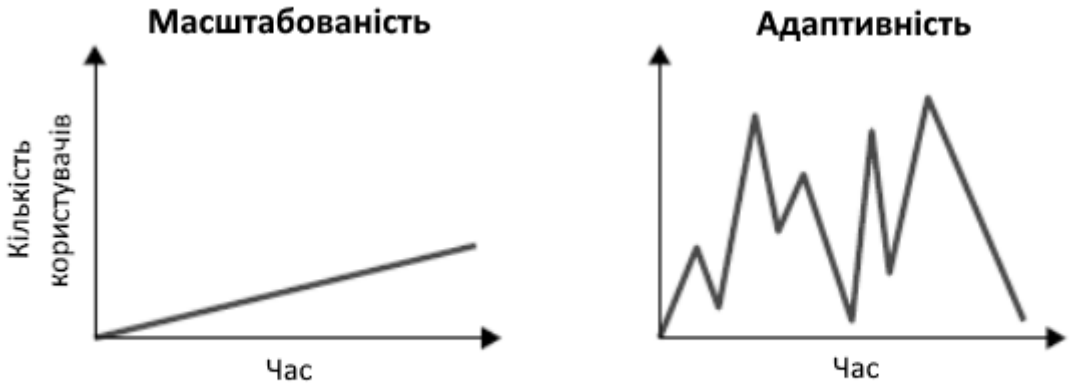


Рис. 1.5 Різниця між масштабованістю — плавним наростанням навантаження, та адаптивністю — різким короткочасним сплеском активності користувачів

З точки зору архітектури обидві ці властивості пов’язані з швидкістю відгуку системи як функцією від кількості одночасно оброблюваних запитів або активних користувачів.

Однак реалізуються вони по-різному:

- Масштабованість передбачає довготривалий розвиток системи, її здатність стабільно зростати разом із бізнесом. Вона розглядається як стратегічний показник, що відображає ефективність архітектури в умовах планомірного збільшення клієнтської бази, обсягів даних або кількості транзакцій.
- Адаптивність, навпаки, — це оперативна здатність системи реагувати на раптові стрибки навантаження. Вона забезпечує негайну стабільність і відсутність простоїв у моменти пікової активності користувачів.

Для ілюстрації цієї різниці можна навести приклад системи продажу квитків. У звичайні дні навантаження на неї незначне — запити надходять рівномірно, і система працює у стабільному режимі. Однак у момент, коли стартує продаж квитків на концерт популярного виконавця, кількість одночасних користувачів може зрости у сотні разів — наприклад, з 20 до 3000 за кілька секунд.

Щоб система залишалася працездатною, вона має автоматично масштабуватися: розподіляти навантаження між кількома серверами, піднімати нові екземпляри сервісів і балансувати запити в реальному часі. У цей момент

вступає в дію адаптивність — здатність архітектури швидко запускати додаткові ресурси, щоб поглинути різкий сплеск трафіку.

Ключовим показником адаптивності є середній час запуску сервісу (Mean Time To Startup, MTTS). Чим швидше система може активувати додаткові модулі, тим вища її адаптивність.

Для досягнення низького MTTS застосовуються такі архітектурні рішення, як:

- поділ системи на дрібні, автономні сервіси, які запускаються незалежно один від одного;
- використання легковагових платформ і середовищ виконання, що мінімізують час ініціалізації;
- попереднє кешування критичних компонентів і зберігання стану (stateful data) у швидкодоступних контейнерах.

Крім того, під час розробки системи важливо забезпечити гнучку оркестрацію сервісів — щоб нові процеси могли автоматично масштабуватися без ручного втручання операторів.

У контексті телекомунікаційних систем масштабованість і адаптивність мають особливе значення, адже навантаження на мережу постійно коливається. Наприклад, у пікові години збільшується кількість з’єднань, активується потокове передавання даних, а в певні моменти (наприклад, під час аварій чи масових подій) трафік може зрости в рази.

Система, що має високий рівень масштабованості та адаптивності, здатна:

- забезпечити стабільну якість обслуговування (QoS) навіть за пікових навантажень;
- зменшити затримки передачі даних і запобігти перевантаженню каналів;
- автоматично балансувати ресурси мережі, підвищуючи ефективність використання обладнання;
- знижувати ризики відмов і гарантувати безперервність надання послуг.

Масштабованість і адаптивність системи безпосередньо залежать від структури архітектури та розміру одиниць розгортання.

Зі зменшенням розміру сервісів обидві ці характеристики, як правило, покращуються, проте кожна з них має власні домінуючі чинники:

- Адаптивність найбільше залежить від деталізації — тобто від того, наскільки дрібними та автономними є окремі елементи системи. Чим менші сервіси, тим швидше вони запускаються і тим ефективніше реагують на раптові сплески навантаження.
- Масштабованість, своєю чергою, пов’язана з рівнем модульності — поділом застосунку на незалежні компоненти, які можна розгортати, оновлювати й масштабувати окремо.

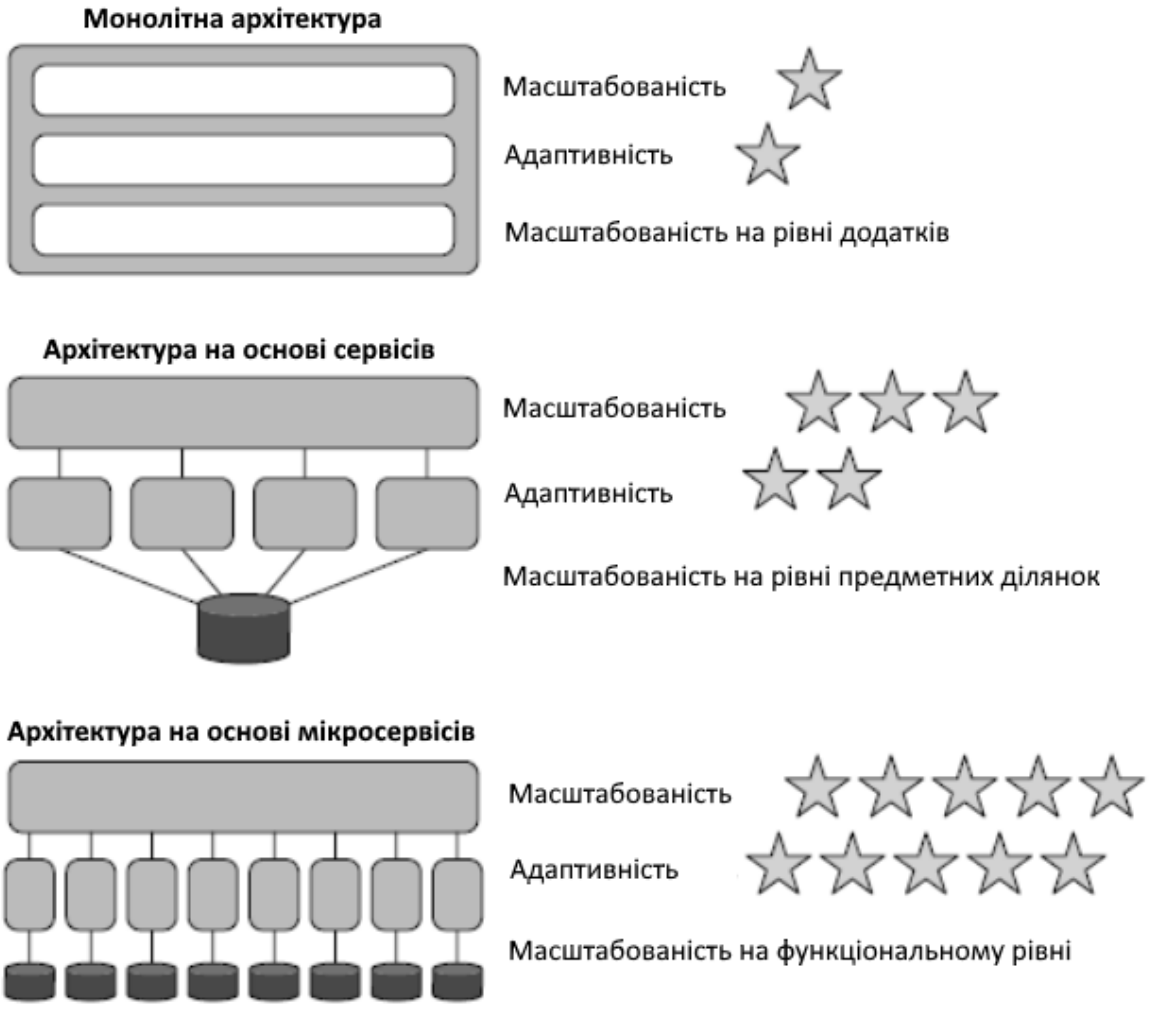


Рис. 1.6 Порівняння трьох архітектурних підходів — монолітну, сервісну та мікросервісну архітектури — за рівнем масштабованості та адаптивності

Щоб краще зрозуміти різницю, розглянемо три поширені архітектурні стилі:

1. Монолітна багаторівнева архітектура,

2. Архітектура на основі сервісів (SOA),
3. Мікросервісна архітектура.

Кожен із цих підходів має свої обмеження й переваги з точки зору масштабованості (Scalability) та адаптивності (Adaptability).

У монолітній багаторівневій архітектурі обидві характеристики залишаються на низькому рівні. Такі системи є великими, важкими у підтримці та дорогими в масштабуванні, оскільки всі функціональні частини застосунку масштабуються одночасно, незалежно від того, яка саме частина потребує додаткових ресурсів. Це створює масштабування на рівні всього застосунку, а не окремих сервісів, що робить процес неефективним.

Крім того, середній час запуску (Mean Time To Startup, MTTS) у таких архітектурах зазвичай високий — система повільно реагує на зміни навантаження, а хмарне масштабування в реальному часі може бути економічно невигідним.

У сервісно-орієнтованій архітектурі (SOA) ситуація покращується — з’являється можливість масштабування на рівні предметних областей (наприклад, управління замовленнями, робота зі складом, білінг тощо). Однак адаптивність залишається нижчою, ніж масштабованість. Причина полягає у тому, що предметні сервіси мають велику внутрішню структуру: вони охоплюють цілі бізнес-домени, тому їх запуск займає більше часу (MTTS залишається середнім). Унаслідок цього такі сервіси повільно реагують на пікові навантаження і не здатні оперативно адаптуватися до стрибків трафіку.

Масштабування в цьому випадку можливе, але воно відбувається на рівні доменів, а не окремих функцій, що обмежує загальну гнучкість системи. Мікросервісна архітектура, навпаки, пропонує найвищий рівень масштабованості та адаптивності.

Кожен мікросервіс — це маленька, спеціалізована одиниця, що виконує лише одну конкретну функцію (наприклад, обробку платежів, аутентифікацію користувача, оновлення інвентарю тощо). Завдяки цьому масштабування здійснюється на рівні окремих функцій, а не всієї системи, що значно підвищує ефективність використання ресурсів.

Мікросервіси характеризуються низьким MTTS, тобто швидким часом запуску, тому вони можуть миттєво розгортатися або дублюватися у випадку зростання навантаження. Це забезпечує високу адаптивність і стійкість навіть до різких сплесків активності користувачів.

Проте, як і у випадку з тестованістю чи розгортаністю, занадто велика кількість сервісів може створювати нові труднощі. Якщо для виконання однієї бізнес-транзакції потрібно взаємодіяти з десятками сервісів, це ускладнює систему й знижує її ефективність. Кожна додаткова інтеграція збільшує затримки, ризики відмов і складність керування залежностями. Тому важливо підтримувати баланс між деталізацією та зв'язністю: сервіси мають бути достатньо дрібними, щоб бути автономними, але не настільки, щоб перетворитися на «архітектурну мікромозаїку».

Ключовим принципом тут є мінімізація синхронних взаємодій між сервісами. Сервіси повинні спілкуватися через асинхронні механізми (черги повідомлень, подієві шини, брокери), що дозволяє уникати блокувань і забезпечувати стабільну продуктивність навіть при великій кількості одночасних запитів.

Таким чином, масштабованість і адаптивність системи зростають із підвищенням архітектурної модульності, однак ефективність цього зростання залежить від грамотного балансу між розміром сервісів, їх взаємодією та рівнем автономності.

Для телекомунікаційних систем, які працюють у режимі 24/7 і обслуговують тисячі користувачів одночасно, оптимальна комбінація цих властивостей дозволяє:

- забезпечувати високий рівень QoS (якості обслуговування) навіть за різких пікових навантажень;
- зменшувати затримки під час обробки запитів;
- підвищувати енергоефективність і рентабельність масштабування;
- гарантувати стабільність та безперервність роботи мережевих сервісів.

Отже, перехід до мікросервісної архітектури не є самоціллю, а є логічним етапом еволюції модульності, який дозволяє досягти максимального рівня

масштабованості, адаптивності та стійкості в умовах постійного зростання вимог до сучасних телекомунікаційних платформ.

Доступність та відмовостійкість як фундаментальні властивості архітектури системи.

Відмовостійкість (fault tolerance) — це здатність системи залишатися працездатною та доступною, навіть якщо окремі її компоненти виходять з ладу. У межах архітектурної модульності ця характеристика визначає, наскільки добре система може ізолювати збої й підтримувати стабільну роботу інших модулів, які не постраждали.

Інакше кажучи, відмовостійкість означає, що несправність одного елемента не призводить до зупинки всієї системи. Наприклад, якщо у застосунку для роздрібної торгівлі трапляється критична помилка в модулі, що відповідає за обробку платежів (наприклад, через нестачу пам’яті або збій стороннього API), користувачі все одно мають мати змогу переглядати товари, додавати їх до кошика або оформляти замовлення, навіть якщо платіжна функція тимчасово недоступна.

У монолітних системах рівень відмовостійкості зазвичай низький. Через щільне з’єднання компонентів усередині одного застосунку збій в одній частині (наприклад, помилка у функції бази даних або в бізнес-логіці) часто призводить до зупинки всього процесу. Хоча деяке підвищення стійкості можна досягти за рахунок балансування навантаження між кількома екземплярами застосунку, цей підхід має суттєві обмеження. Якщо помилка закладена у вихідний код, то вона повторюється в усіх копіях системи, і тому збої виникатимуть одночасно. Такий метод також є дорогим та неефективним, адже вимагає значних апаратних ресурсів і не вирішує саму проблему стабільності.

На відміну від цього, архітектурна модульність дозволяє реалізувати відмовостійкість на рівні предметних областей і окремих функцій. Коли система розділена на незалежні одиниці розгортання — сервіси або модулі, — будь-який катастрофічний збій локалізується в межах одного компонента, не поширюючись на інші частини системи. Таким чином, навіть при збої одного сервісу інші залишаються працездатними та доступними для користувачів.

Наприклад, у телекомунікаційній платформі модуль авторизації може тимчасово вийти з ладу, але це не має блокувати роботу модулів білінгу, маршрутизації викликів або моніторингу мережі. Таке ізолювання помилок — це один із головних принципів стійкої до відмов архітектури (resilient architecture).

Проте існує важливе застереження: відмовостійкість неможлива у випадку жорсткої синхронної залежності між сервісами.

Якщо один модуль очікує відповіді від іншого для продовження своєї роботи, то збій у залежному сервісі автоматично призводить до зависання або зупинки першого. Саме тому у розподілених системах надзвичайно важливо впроваджувати асинхронну взаємодію — наприклад, через черги повідомлень, подієві шини або брокери даних. Це дозволяє іншим сервісам продовжувати роботу, навіть якщо один із них тимчасово недоступний, і відновити нормальну взаємодію після відновлення з’єднання.

У сучасних телекомунікаційних системах, що працюють у режимі 24/7, вимоги до відмовостійкості особливо високі.

Система має залишатися функціональною навіть у випадку часткових відмов, оскільки будь-яка зупинка впливає на якість обслуговування (QoS) і може призвести до значних фінансових втрат.

Для досягнення цього впроваджуються такі архітектурні підходи:

- Розподіл функціональності між незалежними сервісами (доменна ізоляція);
- Дублювання критичних компонентів і автоматичне перемикання (failover);
- Системи моніторингу та самовідновлення, що виявляють і перезапускають збої;
- Асинхронна комунікація, яка дозволяє сервісам працювати автономно.

Отже, висока доступність (availability) і відмовостійкість (fault tolerance) — це не просто додаткові опції архітектури, а життєво необхідні умови для стабільного функціонування будь-якої сучасної системи, особливо у сфері телекомунікацій. Архітектурна модульність створює основу для того, щоб система могла зберігати стабільність, навіть коли окремі частини виходять з ладу,

забезпечуючи тим самим безперервність сервісів і високу якість обслуговування (QoS) у будь-яких умовах.

2 ВИКОРИСТАННЯ АРХІТЕКТУРИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

2.1 Використання нейронних мереж

Нейронні мережі за своєю природою є паралельними алгоритмами, тобто такими, що здатні виконувати велику кількість однотипних операцій одночасно. Ця властивість робить їх ідеально придатними для реалізації на апаратному рівні — з використанням багатоядерних процесорів (CPU), графічних процесорів (GPU) або кластерів серверів, які поєднують у собі декілька CPU і GPU. Завдяки цьому досягається суттєве прискорення обробки даних, що особливо важливо для глибинного навчання (Deep Learning, DL) та інтелектуальних систем обробки інформації, зокрема в сфері телекомунікацій.

Традиційний центральний процесор (CPU) виконує завдання послідовно — тобто одну операцію за іншою. Хоча сучасні моделі підтримують багатопоточність, кожен потік усе одно обробляється лінійно, у певній черзі. Такий підхід добре підходить для детермінованих і логічних обчислень, але втрачає ефективність у задачах, які передбачають масові паралельні операції — наприклад, при розпізнаванні зображень, обробці сигналів або аналізі трафіку в реальному часі.

Графічні процесори (GPU), навпаки, були спроєктовані саме для масово-паралельних обчислень. Вони здатні розподіляти завдання між сотнями або навіть тисячами обчислювальних ядер, виконуючи операції над великими масивами даних одночасно. Це означає, що ті самі операції, які CPU виконують за хвилини, GPU можуть завершити за декілька секунд. З кожним новим поколінням відеокарт зростає не лише кількість ядер, а й пропускна здатність пам’яті, енергоефективність і швидкість обчислень, що робить їх критично важливими для сучасних систем штучного інтелекту.

Основою глибинного навчання є лінійна алгебра — тобто набір фундаментальних операцій над матрицями та векторами: множення, додавання, транспонування, нормалізація тощо. Саме ці операції складають «математичне серце» будь-якої нейронної мережі, адже під час навчання або прогнозування вона

обчислює мільярди параметрів, кожен з яких впливає на результат. Наприклад, коли мережа аналізує зображення або аудіопотік, вона перетворює дані у тензори — багатовимірні масиви чисел, — і поступово виявляє закономірності: форму, колір, текстуру чи об’єкт.

Якщо об’єкт виявлено — система класифікує його як «істинний» результат, якщо ні — як «хибний». Але, щоб дійти до такого висновку, мережа виконує мільярди повторюваних і незалежних операцій. Ці ітерації можна виконувати паралельно, і саме тут розкривається повна потужність GPU: вони здатні обробляти тисячі потоків даних одночасно, тоді як CPU діє послідовно.

У телекомунікаційних архітектурах такий підхід має стратегічне значення. Сучасні мережі — мобільні, оптичні чи супутникові — генерують величезні масиви даних у реальному часі: журнали трафіку, сигнальні повідомлення, аналітику QoS, метадані користувацьких сесій тощо. Аналіз таких потоків традиційними методами є повільним і не дозволяє вчасно виявляти аномалії чи загрози. Натомість, інтеграція нейронних мереж на основі GPU у структуру мережевої інфраструктури (наприклад, у системи моніторингу або мережеві контролери SDN/NFV) дає змогу виявляти порушення QoS, DDoS-атаки чи деградацію каналів у реальному часі.

Завдяки паралельній обробці терабайтів телекомунікаційних даних GPU дозволяють підвищити якість обслуговування (QoS) без помітного збільшення затримок. Це виглядає ніби невелика деталь у загальній архітектурі, але саме вона визначає здатність мережі автоматично оптимізувати трафік, адаптувати маршрути й реагувати на перевантаження без участі оператора.

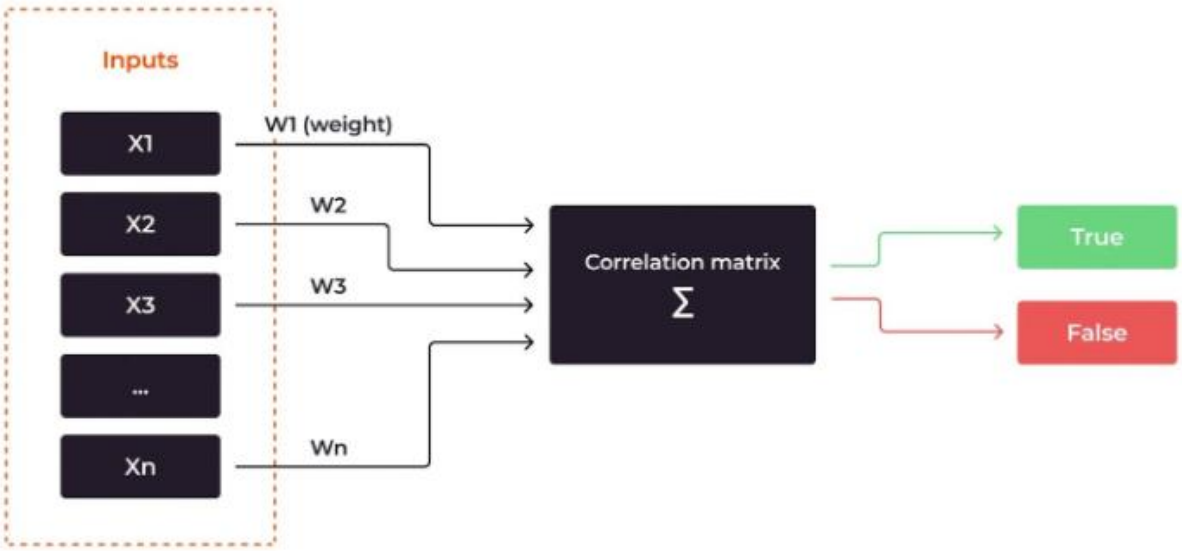


Рис. 2.1 Процес навчання нейронних мереж

Ще одним важливим аспектом є те, що глибинні нейронні моделі (DNN) можуть містити мільярди параметрів, і кожен із них впливає на кінцевий результат. В процесі навчання всі ці параметри проходять через ітераційний цикл оптимізації, який складається з повторюваних матричних операцій. GPU в цьому контексті виступає як «енергетичний центр» — завдяки своїй архітектурі з тисячами обчислювальних блоків він дозволяє здійснювати мільярди паралельних обчислень із мінімальною затримкою.

Це особливо важливо в задачах телекомунікаційного прогнозування:

- передбачення пікових навантажень у мережі,
- динамічне керування ресурсами радіочастотного спектра,
- оптимізація трафіку в мережах 5G/6G,
- аналіз аномалій та передбачення відмов обладнання.

Таким чином, паралельна архітектура нейронних мереж у поєднанні з обчислювальними можливостями GPU створює новий рівень ефективності для телекомунікаційних систем. Вона забезпечує реактивність, самонавчання та прогнозованість у середовищах, де мільйони пакетів проходять через мережу щосекунди. І хоча цей паралелізм виглядає лише технічним аспектом, саме він стає основою високої якості обслуговування (QoS) — тієї непомітної, але критично

важливої характеристики, яка дозволяє сучасним мережам працювати швидко, стабільно й інтелектуально.

2.2 Рівні паралельних обчислень

1. Паралелізм даних (Data-Level Parallelism, DLP)

Паралелізм даних (DLP) — це підхід, за якого одна й та сама операція виконується одночасно над великою кількістю елементів даних. Такий принцип є основою архітектур типу SIMD (Single Instruction, Multiple Data) — *«одна інструкція, багато даних»*. У цьому випадку процесор або графічний прискорювач отримує єдину інструкцію, але застосовує її до множини елементів даних паралельно, що суттєво збільшує продуктивність.

Типовим прикладом DLP є паралельна обробка зображень: коли потрібно змінити яскравість або контрастність фотографії, кожен піксель зображення може бути оброблений незалежно. Таким чином, тисячі обчислювальних ядер GPU одночасно виконують одну й ту ж операцію — наприклад, множення інтенсивності кольору на заданий коефіцієнт. Завдяки цьому обробка навіть зображень високої роздільної здатності триває долі секунди, що неможливо досягти при послідовній обробці на CPU.

У контексті нейронних мереж DLP має ще ширше застосування. При навчанні або прогнозуванні штучна нейронна мережа оперує величезними наборами даних (матриці ваг, вектори ознак, тензори вхідних даних), і всі ці елементи можуть бути оброблені паралельно, оскільки обчислення для одного елемента не залежать від іншого. Такий підхід забезпечує масштабованість і високу пропускну здатність системи, особливо у випадках, коли потрібно швидко аналізувати великі обсяги інформації — наприклад, трафік у телекомунікаційних мережах або сигнали з сенсорних систем.

Саме телекомунікаційна галузь є одним із головних бенефіціарів паралелізму даних. Обробка потоків сигналів, класифікація користувацьких з’єднань, виявлення шумів або сплесків у каналах передачі — усе це завдання, які чудово піддаються

SIMD-паралелізації. У системах моніторингу QoS, наприклад, GPU одночасно аналізують тисячі потоків трафіку, застосовуючи однакові алгоритми оцінки затримок, втрат пакетів або пропускної здатності. Таким чином, паралельна архітектура безпосередньо підвищує якість обслуговування (QoS), зменшуючи затримки аналізу і дозволяючи системі оперативно реагувати на деградацію каналів чи збої в мережі.

Важливо розрізняти паралелізм даних і паралелізм моделей (Model-Level Parallelism).

Якщо у випадку DLP дані діляться між обчислювальними ядрами, і кожне з них виконує однакові дії над своєю частиною, то у паралелізмі моделей усе навпаки — сама модель розподіляється між кількома GPU. У цьому підході різні частини нейронної мережі (наприклад, окремі шари або групи нейронів) обробляються на різних пристроях, і дані передаються між ними поступово.

Це означає, що обчислення мають послідовний характер, адже вихід одного шару (активації) стає входом для іншого, розташованого на іншому GPU. Така схема застосовується у випадках, коли модель є настільки великою, що не може поміститися в пам’яті одного пристрою, наприклад у мовних моделях або глибоких мережах для відеоаналітики. Хоча цей підхід не забезпечує повного паралелізму, він дозволяє розподіляти навантаження й працювати з моделями екстремальних масштабів.

Ще однією важливою концепцією є розподілений паралелізм даних (Distributed Data Parallelism, DDP) — розширена форма DLP, яка активно використовується під час навчання глибоких моделей (Deep Learning, DL). У цьому випадку обчислення відбуваються одночасно на кількох процесорах або вузлах обчислювального кластера. Кожен GPU або CPU отримує підмножину даних і обробляє її незалежно, виконуючи власний цикл прямого проходу (forward pass) і обчислення градієнтів (backward pass). Потім усі вузли синхронізуються між собою, обмінюючись обчисленими градієнтами, після чого оновлюються спільні параметри моделі.

Цей підхід дозволяє одночасно опрацьовувати петабайти навчальних даних, суттєво зменшуючи час навчання моделей. Наприклад, нейронна мережа, яку б на одному GPU довелося тренувати тиждень, у режимі DDP може бути навчена за кілька годин.

У телекомунікаційних мережах це має величезне значення для реального часу навчання систем прогнозування трафіку, виявлення аномалій, оптимізації розподілу навантаження чи навіть автоматичного налаштування параметрів антенних решіток у мережах 5G/6G.

2. Паралелізм завдань (Task-Level Parallelism, TLP)

Паралелізм завдань (Task-Level Parallelism, TLP) — це концепція, що передбачає одночасне виконання кількох незалежних потоків або процесів, кожен з яких вирішує власну частину обчислювального завдання. На відміну від паралелізму даних, де всі ядра виконують однакову операцію над різними частинами масиву, у TLP кожне ядро виконує окреме, логічно незалежне завдання. Це дозволяє ефективно розподіляти обчислювальні ресурси між різними типами операцій, оптимізуючи роботу системи.

У цьому підході одна велика обчислювальна задача поділяється на кілька підзадач, які можуть суттєво відрізнятися між собою — наприклад, одна з них виконує обчислення, інша — обробляє вхідні дані, ще одна — записує результати. Головна перевага полягає в тому, що всі ці підзадачі не залежать одна від одної, тому можуть виконуватися паралельно на різних ядрах або навіть на різних процесорах.

Такий тип паралелізму широко використовується у багатопоточних системах (multithreading) і багатоядерних архітектурах, де кожне ядро або процесор виконує власну «робочу нитку» (thread), не блокуючи інші. У результаті досягається максимальне завантаження апаратних ресурсів, а система стає більш продуктивною навіть при обмеженій кількості ядер.

Застосування паралелізму завдань має надзвичайно велике значення в телекомунікаційних системах, де різні типи процесів мають виконуватися

одночасно. Наприклад, у типовому мережевому контролері або сервері операторського рівня (core network node) одночасно відбуваються такі процеси, як:

- обробка сигналізації (control plane);
- маршрутизація пакетів даних (data plane);
- аутентифікація користувачів;
- моніторинг трафіку;
- керування політиками доступу або QoS.

Кожен із цих процесів може виконуватися в окремому потоці або ядрі, завдяки чому система залишається оперативною, гнучкою та стабільною навіть під великим навантаженням. Це забезпечує високий рівень якості обслуговування (QoS), адже навіть якщо одна з підсистем тимчасово перевантажена, інші продовжують функціонувати без затримок.

Типовим прикладом TLP у телекомунікаціях є паралельна обробка різних типів трафіку:

одне ядро процесора може обслуговувати відеопотік, інше — обробляти голосові дзвінки, третє — забезпечувати передачу службових повідомлень. Кожен із потоків має власні вимоги до пропускної здатності, затримки та стабільності, тому розподіл завдань між окремими ядрами дозволяє балансувати навантаження й підтримувати стабільність усієї системи.

Ще один приклад — мережеві системи управління (Network Management Systems, NMS) або системи аналітики телеметрії (Telemetry Analytics), які збирають і аналізують дані в режимі реального часу.

Тут паралелізм завдань дозволяє одночасно:

- отримувати потоки даних з різних мережевих пристроїв;
- проводити статистичну обробку;
- виконувати машинне навчання для виявлення аномалій;
- формувати звіти для операторів.

Усе це відбувається паралельно, забезпечуючи швидкість реакції системи, а також зниження середнього часу обробки запитів (latency).

З технічної точки зору, TLP реалізується через розподіл процесів у системі операційного рівня (OS-level multithreading) або через спеціалізовані системи оркестрації — наприклад, у контейнеризованих середовищах, де різні контейнери виконують незалежні мікросервіси. У великих телекомунікаційних платформах TLP поєднується з іншими типами паралелізму, такими як паралелізм даних (DLP) чи паралелізм моделей (MLP), створюючи багаторівневу архітектуру високої продуктивності.

Отже, паралелізм завдань — це ключовий підхід до побудови масштабованих, стійких і продуктивних систем, у тому числі в телекомунікаційній сфері. Його застосування дозволяє:

- оптимізувати використання апаратних ресурсів;
- забезпечити реактивність системи навіть за великих навантажень;
- зменшити затримки при обробці запитів;
- підвищити рівень QoS та надійність мережевих сервісів.

Завдяки TLP телекомунікаційна архітектура набуває справжньої багатопоточності, коли кожен процес системи діє незалежно, але узгоджено з іншими — формуючи єдиний, стабільний і самоналагоджуваний механізм, який гарантує безперервність зв’язку та якість обслуговування користувачів навіть у пікові моменти роботи мережі.

3. Паралелізм на рівні інструкцій (Instruction-Level Parallelism, ILP)

Паралелізм на рівні інструкцій (ILP) — це тип паралельних обчислень, який передбачає одночасне виконання кількох інструкцій у межах одного процесу або потоку. Такий підхід дозволяє підвищити продуктивність системи без збільшення кількості ядер, адже процесор або GPU може виконувати декілька команд у межах одного тактового циклу. Це дає змогу максимально ефективно використовувати наявні ресурси й суттєво скорочує час обробки даних.

ILP можна вважати “внутрішнім паралелізмом” процесора, адже він стосується архітектури виконання команд. Замість того щоб виконувати інструкції строго послідовно, система розділяє їх на незалежні частини та виконує одночасно.

Наприклад, одна інструкція може виконувати арифметичну операцію, тоді як інша — завантажувати дані в пам’ять або записувати результат.

У сучасних серверних GPU і процесорах ILP досягається завдяки поєднанню кількох технологій та архітектурних підходів:

- Багатозадачність (multitasking) — дозволяє розподіляти різні інструкції між апаратними блоками виконання.
- Конвеєризація (pipelining) — розділяє виконання інструкцій на етапи (завантаження, декодування, виконання, запис результату), що дозволяє одночасно обробляти кілька інструкцій на різних стадіях.
- Виконання інструкцій у довільному порядку (out-of-order execution) — процесор самостійно визначає оптимальну послідовність виконання, щоб уникати простоїв.
- SIMD (Single Instruction, Multiple Data) — одна інструкція виконується одразу над кількома елементами даних.
- Векторні та матричні інструкції — дозволяють виконувати операції над великими наборами даних (наприклад, над цілими масивами або матрицями) за один такт, що особливо корисно у нейромережах і графічних обчисленнях.

У телекомунікаційних системах ILP має велике значення для низькорівневих процесів, які вимагають максимальної швидкодії — наприклад, для обробки сигналів (DSP), маршрутизації пакетів, шифрування даних або аналізу потоків у реальному часі. Завдяки паралельному виконанню інструкцій на рівні процесора такі системи можуть миттєво реагувати на зміни трафіку, зменшувати затримки та покращувати якість обслуговування (QoS).

Таким чином, паралелізм на рівні інструкцій — це невидимий, але критично важливий механізм оптимізації, який дозволяє збільшити швидкість обчислень без розширення апаратної інфраструктури, роблячи сучасні телекомунікаційні архітектури швидшими, ефективнішими й надійнішими.

4. Паралелізм на рівні потоків (Thread-Level Parallelism, TLP)

Паралелізм на рівні потоків (Thread-Level Parallelism, TLP) — це форма організації обчислень, у якій завдання розділяються на незалежні потоки, що

виконуються одночасно на різних обчислювальних ядрах або пристроях. Якщо паралелізм на рівні інструкцій (ILP) зосереджується на прискоренні виконання команд усередині одного потоку, то TLP використовує перевагу одночасного виконання кількох потоків, кожен з яких вирішує власну частину задачі. Такий підхід забезпечує значне підвищення продуктивності та ефективне використання обчислювальних ресурсів.

У рамках TLP складна обчислювальна задача ділиться на підзадачі, які розподіляються між кількома потоками. Кожен потік виконується незалежно, що дозволяє досягати високого ступеня паралельності. Це особливо важливо для високопродуктивних обчислень (High-Performance Computing, HPC), машинного навчання (ML), комп'ютерної графіки, обробки сигналів та великомасштабного аналізу даних (Big Data), де потрібно одночасно виконувати тисячі або навіть мільйони операцій.

Одним із найяскравіших прикладів реалізації TLP є багатоядерні процесори (multicore processors). Кожне ядро в такому процесорі здатне виконувати власний потік, не заважаючи іншим, що дозволяє системі обробляти кілька програм або частин однієї задачі паралельно.

Однак у графічних процесорах (GPU) концепція TLP розвинена ще глибше. На відміну від CPU, де ядер зазвичай небагато (від 4 до 64), GPU мають тисячі обчислювальних ядер, організованих у спеціальні обчислювальні блоки.

Основою сучасної архітектури GPU є потокові мультипроцесори (Streaming Multiprocessors, SM) — групи ядер, які працюють спільно та здатні обробляти сотні потоків одночасно. У межах SM потоки об'єднуються в групи warp (у GPU NVIDIA — по 32 потоки в одній групі). Ці потоки виконують одну й ту саму інструкцію, але застосовують її до різних наборів даних — наприклад, до різних пікселів зображення або елементів вектора.

Такий принцип реалізується в моделі SIMT (Single Instruction, Multiple Threads) — «одна інструкція, багато потоків». На практиці це означає, що GPU здатен обробляти тисячі паралельних операцій за секунди, виконуючи

високошвидкісні операції над матрицями, фільтрацію зображень, лінійну алгебру чи обчислення в нейронних мережах.

5. Паралелізм на рівні вузлів (Node-Level Parallelism, NLP)

Паралелізм на рівні вузлів (Node-Level Parallelism, NLP) — це форма організації обчислень, яка передбачає розподіл навантаження між кількома фізичними обчислювальними вузлами, такими як сервери, GPU-кластери або цілі розподілені обчислювальні системи. На відміну від паралелізму на рівні інструкцій чи потоків, де обчислення виконуються в межах одного пристрою, NLP забезпечує взаємодію між кількома автономними системами, що працюють спільно над одним завданням.

У цьому підході завдання розподіляються не тільки на рівні окремих процесів чи потоків, а й між різними машинами, які обмінюються проміжними результатами через високошвидкісні мережі, такі як InfiniBand, Ethernet або NVLink. Це дозволяє масштабувати обчислення до сотень або навіть тисяч вузлів, забезпечуючи при цьому узгодженість і синхронізацію даних. Зазвичай NLP охоплює як обробку даних, так і виконання розподілених обчислень, які можуть бути фізично рознесені по різних серверах.

У сучасних високопродуктивних системах NLP реалізується через розподілені обчислювальні моделі та спеціалізовані бібліотеки, які дозволяють ефективно координувати роботу між вузлами. Серед найпоширеніших технологій — CUDA Multi-Node, TensorFlow та PyTorch із підтримкою розподіленого навчання. Вони дозволяють одночасно використовувати десятки або сотні GPU, розташованих у різних серверах, для тренування великих нейронних мереж чи виконання складних аналітичних розрахунків.

Для синхронізації обчислень і обміну даними між вузлами застосовуються спеціальні бібліотеки, такі як MPI (Message Passing Interface), NCCL (NVIDIA Collective Communications Library) або фреймворки Horovod і Ray, які мінімізують затримки під час передачі даних між обчислювальними вузлами. Завдяки цим технологіям можливо забезпечити стабільну комунікацію та низький рівень

латентності, що критично важливо при розподіленому навчанні моделей штучного інтелекту.

У сфері телекомунікаційних архітектур паралелізм на рівні вузлів відіграє ключову роль у побудові інтелектуальних розподілених систем керування мережею, які використовують кілька обчислювальних центрів для аналізу трафіку, управління навантаженням та виявлення аномалій. Такий підхід дозволяє обробляти великі обсяги даних у реальному часі та підтримувати високу якість обслуговування (QoS) навіть при різкому зростанні навантаження. NLP фактично стає основою розподілених телекомунікаційних платформ, що поєднують обчислювальну потужність, гнучкість і стійкість до збоїв у єдину інтелектуальну систему.

6. Гібридний паралелізм (Hybrid Parallelism, HP)

Гібридний паралелізм (Hybrid Parallelism, HP) — це підхід, що передбачає комбіноване використання кількох типів паралелізму, описаних раніше: від виконання інструкцій усередині одного GPU до масштабованих розподілених обчислень між кількома серверами або вузлами. Такий підхід поєднує гнучкість локального паралелізму (на рівні інструкцій, потоків і даних) із потужністю розподілених систем, забезпечуючи оптимальний баланс між швидкодією, масштабованістю та ефективністю використання ресурсів.

У межах одного сервера або обчислювального вузла можуть одночасно застосовуватись різні методи паралельних обчислень. Наприклад, розподіл навантаження між потоками та блоками потоків у межах GPU, що дозволяє задіяти всі доступні ядра пристрою. У цьому випадку часто використовується модель SIMT (Single Instruction, Multiple Threads) — «одна інструкція, багато потоків», за якою одна команда виконується паралельно в десятках або сотнях потоків, кожен із яких обробляє власний набір даних.

Додатково до цього, у межах одного GPU або вузла застосовується комбінація підходів SIMD і TLP:

- SIMD (Single Instruction, Multiple Data) виконує одну операцію над великими наборами даних, забезпечуючи високу ефективність при роботі з масивами, векторами або матрицями;

- TLP (Thread-Level Parallelism), своєю чергою, дозволяє розподіляти ці операції між численними потоками, які працюють незалежно один від одного.

Разом ці підходи створюють багаторівневу архітектуру паралельності, у якій кожен рівень доповнює інший: SIMD забезпечує швидкість на рівні обробки даних, а TLP — ефективне використання обчислювальних блоків GPU. Така гібридна модель дає змогу виконувати навіть найскладніші обчислення — від математичного моделювання до глибинного навчання — у реальному часі та з мінімальними затримками.

7. Паралелізм на рівні кооперації (Cooperative Parallelism, CP)

Паралелізм на рівні кооперації (Cooperative Parallelism, CP) — це підхід до організації обчислень, за якого потоки або завдання взаємодіють між собою, координуючи свою роботу для досягнення спільного результату. На відміну від класичного паралелізму, де кожен потік виконує незалежну задачу, у кооперативному паралелізмі всі обчислювальні одиниці — ядра GPU, кілька GPU або навіть різні обчислювальні вузли — діляться даними, синхронізують виконання та спільно вирішують одну складну задачу.

Такий тип паралелізму є надзвичайно важливим для глобально пов’язаних процесів, наприклад у розподіленому навчанні нейронних мереж, моделюванні фізичних процесів, обчислювальних симуляціях або аналітичних системах з елементами штучного інтелекту, де точність результату залежить від узгодженої роботи всіх компонентів системи. Наприклад, під час тренування великої моделі штучного інтелекту різні GPU можуть обчислювати окремі фрагменти градієнтів, після чого результати синхронізуються, щоб оновити загальні параметри моделі.

У серверних кластерах, що складаються з кількох вузлів (а кожен вузол має декілька GPU), кооперативний паралелізм реалізується так: кожен вузол обробляє свою частину загального завдання, але періодично обмінюється даними з іншими для підтримки єдиної узгодженої моделі або симуляції. Наприклад, один вузол може виконувати попередню обробку даних, другий — навчати модель, а третій — оцінювати результати. При цьому всі вони працюють як частини однієї системи, синхронно обмінюючись результатами проміжних розрахунків.

Для забезпечення ефективної взаємодії між вузлами широко застосовується MPI (Message Passing Interface) — стандарт, що дозволяє передавати повідомлення між процесами на різних машинах, забезпечуючи узгодженість і синхронізацію даних. Крім того, для зменшення затримок у комунікації використовуються високошвидкісні мережеві інтерфейси, такі як InfiniBand або NVLink, які забезпечують передачу даних із мінімальною латентністю навіть у масштабі великих розподілених систем.

8. Багаторівневий паралелізм (Multilevel Parallelism, MP)

Багаторівневий паралелізм (Multilevel Parallelism, MP) — це концепція, що передбачає поєднання кількох типів паралельних обчислень, виконуваних на різних рівнях системної архітектури — від базових операцій на рівні інструкцій і потоків до розподіленої обробки даних на рівні серверів, кластерів або навіть цілих дата-центрів. Такий підхід забезпечує максимальну ефективність використання обчислювальних ресурсів, дозволяючи системі працювати одночасно на кількох рівнях продуктивності.

У контексті серверних GPU багаторівневий паралелізм реалізується через комбінацію декількох моделей обчислень. Наприклад, усередині одного GPU одночасно застосовуються ILP (Instruction-Level Parallelism) для паралельного виконання інструкцій, TLP (Thread-Level Parallelism) для розподілу задач між потоками, а також DLP (Data-Level Parallelism) для одночасної обробки великих масивів даних. Далі, коли кілька GPU або вузлів працюють у спільному середовищі, додається NLP (Node-Level Parallelism) або навіть CP (Cooperative Parallelism) — тобто координація обчислень між пристроями. У результаті створюється багаторівнева ієрархія паралельності, де кожен рівень посилює інший.

Такий тип організації особливо ефективний у системах, які обробляють великі потоки даних у реальному часі, наприклад у телекомунікаційних мережах нового покоління (5G/6G). Тут паралелізм реалізується одночасно на кількох рівнях:

- на рівні інструкцій — при обробці сигналів і декодуванні;
- на рівні потоків — під час маршрутизації або шифрування трафіку;

- на рівні вузлів — при аналітиці трафіку у хмарних центрах обробки даних.

Це дозволяє системам не лише досягати високої швидкодії, але й підтримувати стабільну якість обслуговування (QoS) навіть при великих навантаженнях.

2.3 Архітектури паралельних обчислень

1. SIMD (Single Instruction, Multiple Data)

SIMD — це архітектура паралельних обчислень, у якій одна інструкція виконується одночасно над кількома наборами даних. Принцип її роботи полягає в тому, що всі операції виконуються паралельно, але над різними елементами даних із використанням однієї й тієї ж команди. Водночас, якщо потрібно виконувати різні операції над різними даними, підхід SIMD втрачає ефективність. Крім того, на відміну від інших типів паралельних архітектур, SIMD не підходить для задач, у яких дані змінюються під час обчислень.

SIMD дозволяє значно прискорити процес навчання моделей глибинного навчання, розпаралелюючи такі обчислення, як множення матриць або операції згортки. Цей підхід також активно застосовується у завданнях обробки великих масивів даних — наприклад, під час фільтрації, агрегації чи трансформації інформації. Популярні бібліотеки, такі як NumPy та TensorFlow, використовують SIMD-інструкції для оптимізації продуктивності на сучасних центральних (CPU) і графічних процесорах (GPU).

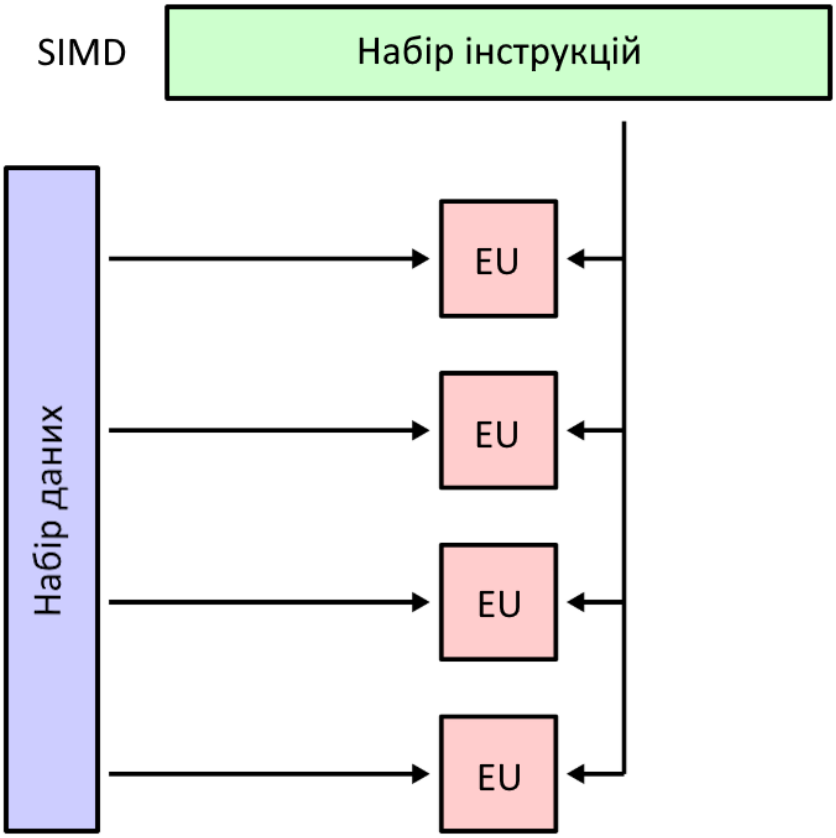


Рис. 2.2 Схематичне представлення архітектури SIMD: одна інструкція — кілька потоків даних

Де:

SIMD – Single Instruction, Multiple Data (Один комплект команд, множина даних)

EU – Execution Unit (Виконавчий пристрій)

2. SIMT (Single Instruction, Multiple Threads)

SIMT — це модель паралельних обчислень, яка використовується у сучасних графічних процесорах (зокрема, в архітектурі NVIDIA) для ефективної обробки великих обсягів даних. Вона дає змогу виконувати одну інструкцію одночасно у кількох потоках, що значно підвищує продуктивність системи при роботі з масовими обчисленнями. Іншими словами, усі потоки в групі виконують одну й ту саму операцію (наприклад, додавання, множення чи іншу арифметичну дію), але кожен потік працює зі своїми власними даними.

Модель SIMT особливо ефективна у задачах, де потрібна масштабна паралельна обробка — таких як глибинне навчання, рендеринг графіки, фінансове моделювання або наукові симуляції.

Усередині GPU потоки організовані в групи, які називаються warp. Зазвичай у графічних процесорах NVIDIA одна така група (warp) містить 32 потоки. Всі потоки в межах одного warp виконують одну інструкцію синхронно, тобто одночасно рухаються крок за кроком по одній послідовності команд. Це дає змогу максимально ефективно використовувати апаратні ресурси GPU для паралельної обробки.

Наприклад, якщо GPU виконує завдання з обробки зображення, то кожен потік у warp може обробляти один піксель, тоді як сам алгоритм (наприклад, застосування фільтра або корекції кольору) буде однаковим для всіх потоків. Таким чином, SIMT поєднує простоту керування з високим рівнем паралелізму, забезпечуючи баланс між продуктивністю та ефективністю при виконанні великомасштабних обчислень.

3. MIMD (Multiple Instruction, Multiple Data)

MIMD — це тип паралельної обчислювальної архітектури, який дає змогу кільком процесорам одночасно виконувати різні послідовності інструкцій над різними наборами даних. Такий підхід суттєво підвищує продуктивність систем, що виконують ресурсоємні або різнорідні обчислення. На відміну від архітектури SIMD, де одна інструкція одночасно застосовується до багатьох елементів даних, у MIMD кожен процесор може виконувати власну програму або набір команд, що робить систему гнучкішою та універсальнішою.

Цей підхід особливо ефективний у складних системах, де різні частини програми повинні виконувати різні типи операцій — наприклад, аналіз, обробку, синхронізацію чи управління потоками даних. Завдяки такій незалежності процесів архітектура MIMD добре підходить для наукових симуляцій, моделювання фізичних процесів, керування робототехнічними системами або розподілених обчислень.

Хоча MIMD не є основною архітектурою для GPU, які здебільшого працюють за моделлю SIMT (одночасне виконання однієї інструкції багатьма потоками), вона активно використовується у гібридних системах CPU+GPU. У таких середовищах CPU, що підтримують MIMD, можуть керувати високорівневими логічними процесами, тоді як GPU виконують великі обсяги однотипних обчислень. Це дозволяє оптимально розподіляти навантаження: різні типи обчислень виконуються паралельно, але з різним підходом — CPU обробляє різні інструкції, а GPU виконує масові паралельні операції над даними.

Таким чином, MIMD забезпечує гнучкість і масштабованість у багатокомпонентних системах, дозволяючи ефективно поєднувати різні типи обчислень і досягати максимальної продуктивності у складних багаторівневих середовищах.

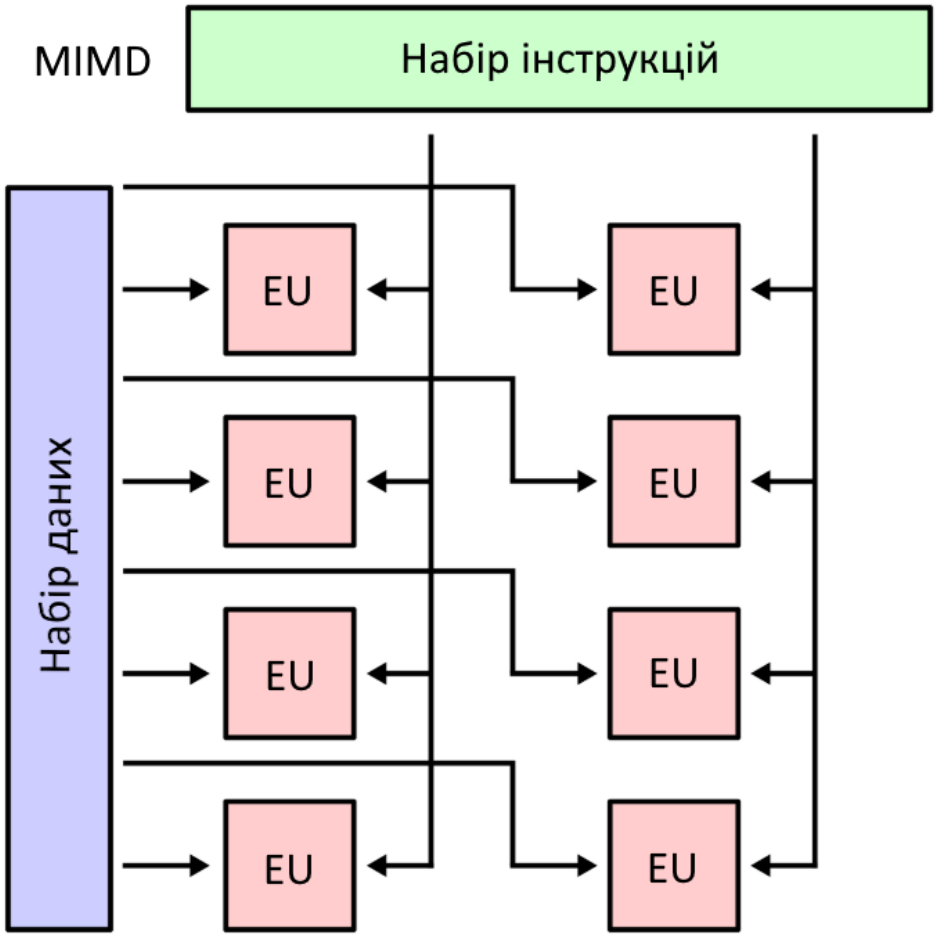


Рис. 2.3 Архітектура MIMD: паралельне виконання різних інструкцій над множиною даних

4. Гетерогенна архітектура

У сфері штучного інтелекту (ШІ) для використання в телекомунікаціх гетерогенні системи — це обчислювальні середовища, що поєднують різні типи апаратних компонентів або процесорів для ефективного виконання задач, пов’язаних із ШІ та машинним навчанням. Такі системи створені для того, щоб використовувати сильні сторони кожного типу апаратури, забезпечуючи максимальну продуктивність і енергоефективність під час обробки складних інтелектуальних навантажень.

У гетерогенних архітектурах зазвичай комбінуються центральні процесори (CPU), графічні процесори (GPU), програмовані вентильні матриці (FPGA), спеціалізовані інтегральні схеми (ASIC) та апаратні прискорювачі для ШІ. Кожен із цих компонентів виконує свою специфічну роль, що дозволяє системі одночасно обробляти різні типи обчислень — від керування потоками даних до глибокого навчання нейронних мереж.

Принцип роботи гетерогенних обчислювальних систем полягає в тому, що для кожного типу задач обирається найоптимальніше обчислювальне пристрій. Наприклад, CPU відповідає за загальну логіку роботи системи, управління процесами та обробку послідовних операцій, тоді як GPU виконує масово-паралельні обчислення, що потребують значної кількості операцій над великими обсягами даних.

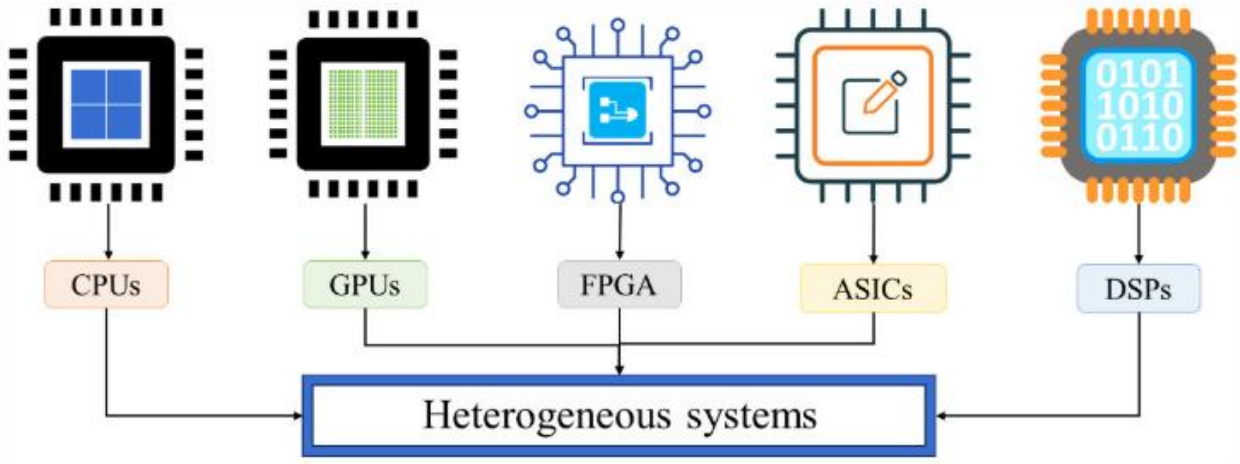


Рис. 2.4 Гетерогенні обчислювальні системи

У сучасних гібридних серверах така взаємодія забезпечує найвищу ефективність. CPU координує роботу системи, розподіляє ресурси, а GPU бере на себе інтенсивні обчислення — такі як аналіз великих даних, симуляції, навчання нейронних мереж, рендеринг графіки або візуалізація в реальному часі. Додаткові компоненти, як-от FPGA чи ASIC, можуть виконувати спеціалізовані завдання — наприклад, прискорення обробки сигналів або оптимізацію енергоспоживання при глибинному навчанні.

3 ПРОДУКТИВНІСТЬ ПРОГРАМ В МЕРЕЖАХ МАСОВОГО ОБСЛУГОВУВАННЯ

3.1 Багаторівневі застосунки

Розглянемо багаторівневу архітектуру хмарного застосунку, розгорнутого на кластері віртуальних машин. Така архітектура схематично зображена на рисунку 3.1.

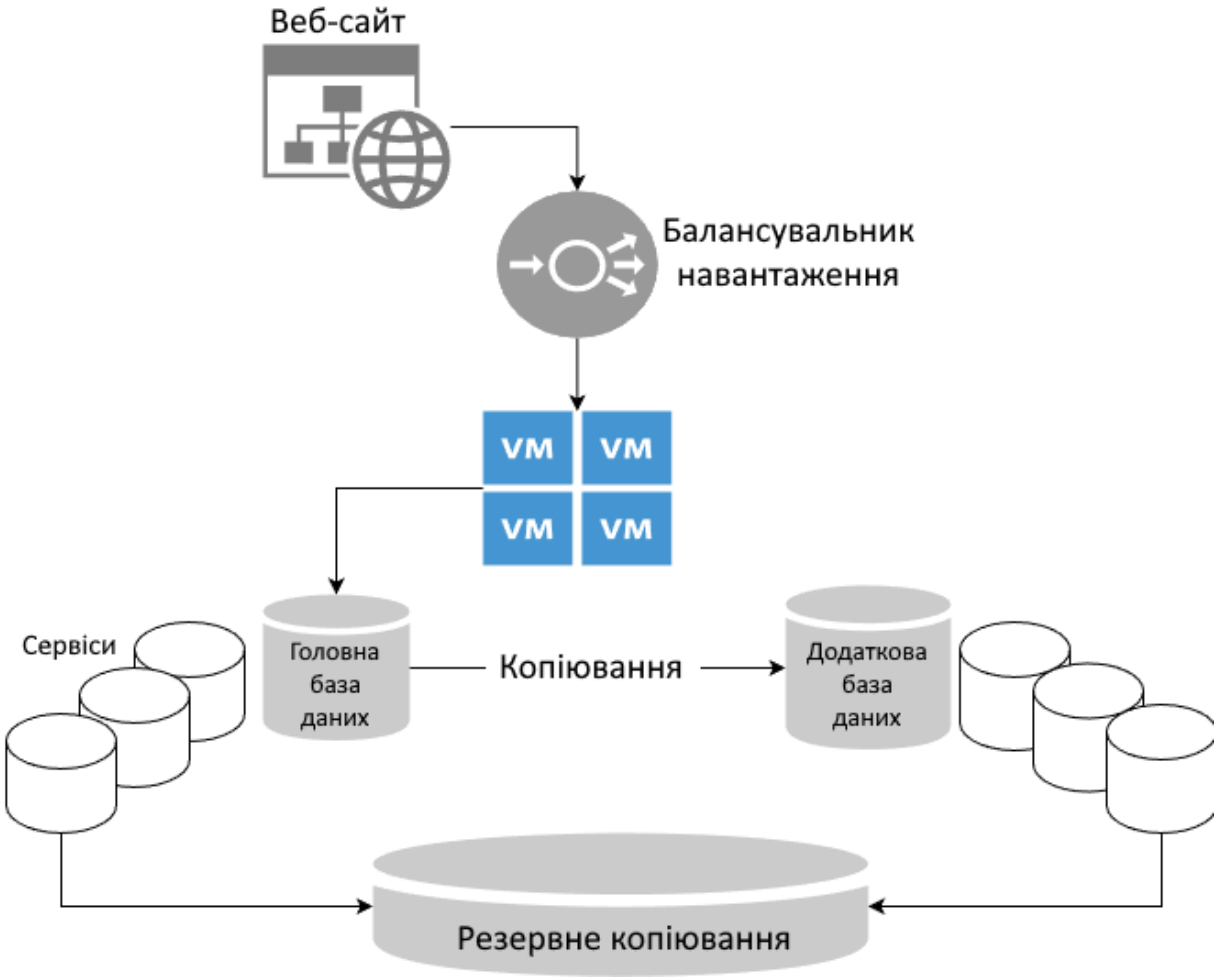


Рис. 3.1 Схема тришарової архітектури хмарного застосунку

Стандартна N-рівнева архітектура хмарного застосунку зазвичай складається з трьох основних рівнів: зовнішнього (presentation), логічного (application) та рівня зберігання даних (data storage). Насправді логічний рівень може включати кілька фізичних або логічних підрівнів — залежно від бізнес-задач і вимог до

масштабованості. Для кожного рівня вхідними даними є результати обробки з попереднього рівня, а передача інформації відбувається лише «вниз» по ієрархії.

Така архітектура відзначається гнучкістю та масштабованістю. Віртуальні машини, що утворюють логічний рівень, можуть відрізнятися інтенсивністю обробки запитів, яка вимірюється у кількості операцій за секунду. Ця інтенсивність залежить від технічних характеристик ВМ — продуктивності процесора, обсягу оперативної пам’яті та швидкості зберігання даних.

Запити, що надходять на перший логічний рівень, обробляються балансувальником навантаження (load balancer). Вони можуть походити як від реального користувача, так і від іншого вебзастосунку. Час виконання запиту варіюється — від кількох мілісекунд до кількох хвилин, залежно від складності операції.

Балансувальник навантаження розподіляє вхідний трафік між віртуальними машинами логічного рівня, використовуючи різні алгоритми, що враховують поточну завантаженість вузлів, їхню продуктивність та можливі відмови. Якщо якась із машин виходить з ладу, балансувальник автоматично припиняє передавати їй трафік і перенаправляє запити на інші активні вузли. Як тільки несправна машина відновлює роботу, вона знову включається в процес розподілу навантаження.

У застосунку можуть використовуватися різні політики балансування — наприклад:

- випадкове розподілення (random distribution);
- найменше навантаження (least connections);
- циклічний перебір або так званий алгоритм «Карусель» (Round Robin).

Якщо кластер складається з віртуальних машин із різними рівнями потужності, політика балансування має враховувати коефіцієнт обчислювальної інтенсивності кожного вузла, щоб забезпечити оптимальний розподіл запитів і стабільну продуктивність усієї системи.

3.2 Мережа масового обслуговування, що описує роботу трирівневої хмарної програми

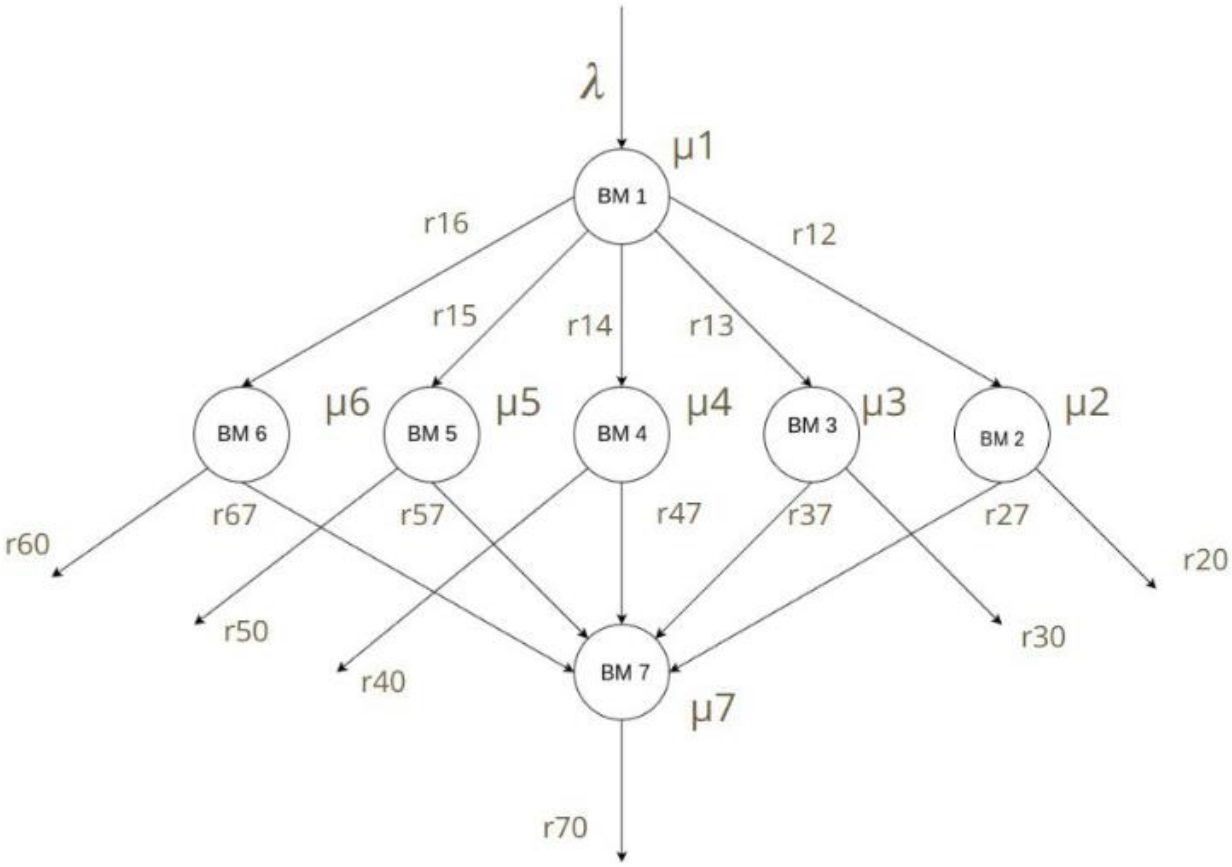


Рис. 3.2 Мережа масового обслуговування, що описує роботу трирівневої хмарної програми

Перший рівень складається з однієї віртуальної машини, яка виконує роль балансувальника навантаження. Саме він, використовуючи спеціальний алгоритм, розподіляє вхідний трафік між п’ятьма віртуальними машинами другого рівня. Балансувальник є ключовим елементом у забезпеченні стабільності, ефективності та рівномірного використання ресурсів усієї системи.

Другий рівень представлений п’ятьма віртуальними машинами, кожна з яких реалізує окремі бізнес-процеси застосунку, що відповідають за виконання логіки та функціональності системи. Використання кількох віртуальних машин на цьому рівні забезпечує відмовостійкість — у випадку виходу з ладу одного з вузлів робота

системи не припиняється, оскільки запити можуть бути оброблені іншими машинами.

Третій рівень — це рівень зберігання даних, який у даній моделі представлений у вигляді реплікованої бази даних. Такий підхід підвищує надійність і доступність інформації, адже копії даних зберігаються на кількох вузлах.

Кожен вузол моделює роботу віртуальної машини, розміщеної в хмарному кластері. Усі віртуальні машини з'єднані між собою в єдину систему. Обробка заявок може здійснюватися як на другому рівні, так і на третьому, але безпосередня взаємодія між першим і третім рівнями виключена — усі запити повинні пройти через рівень бізнес-логіки.

Кожен вузол представлено як систему масового обслуговування типу M/M/1 із потоком заявок, що підпорядковується розподілу Пуассона, та експоненційним законом розподілу часу обслуговування, при цьому черга очікування необмежена.

Ймовірність переходу системи зі стану i у стан $i - 1$ у процесі (t), який відображає кількість заявок, що перебувають у системі в момент часу t (для $i \geq 1$), визначається за такою формулою:

$$p_{i,i-1}(\Delta) = y_1(\Delta)x_0(\Delta) + \sum_{n=1}^{\infty} x_n(\Delta)y_{n+1}(\Delta) =$$
$$= \big(\mu\Delta + o(\Delta)\big)\big(1 - \lambda\Delta + o(\Delta)\big) + o(\Delta) = \mu\Delta + o(\Delta). \quad (3.1)$$

Перехід системи зі стану i у стан $i - 1$ означає, що кількість оброблених заявок за певний проміжок часу перевищує кількість нових, що надійшли, на одну. Іншими словами, за цей інтервал часу система встигла обслугувати більше запитів, ніж отримала.

Ймовірність переходу зі стану i у стан $i + 1$ визначається за такою формулою:

$$p_{i,i+1}(\Delta) = y_1(\Delta)x_0(\Delta) + \sum_{n=1}^{\infty} x_n(\Delta)y_{n-1}(\Delta) =$$
$$= \big(\lambda\Delta + o(\Delta)\big)\big(1 - \mu\Delta + o(\Delta)\big) + o(\Delta) = \lambda\Delta + o(\Delta). \quad (3.2)$$

Імовірність залишитися у стані i дорівнює:

$$p_{i,i}(\Delta) = \big(1 - \lambda\Delta + o(\Delta)\big)\big(1 - \mu\Delta + o(\Delta)\big) + o(\Delta) = \\ = 1 - (\lambda + \mu)\Delta + o(\Delta). \quad (3.3)$$

Геометричний розподіл кількості заявок, що знаходяться в системі, визначається завантаженням системи:

$$p_n = \rho^n(1 - \rho), \quad n = 0,1,2,3, \quad (3.4)$$

Знаючи функцію *P(z)*, що виробляє, можна визначити середню кількість заявок в системі, яка визначається як математичне очікування процесу:

$$M\xi = P'(z)|_{z=1} = \frac{\rho}{1 - \rho} \to \infty, \quad \text{коли } \rho \to 1 \quad (3.5)$$

За допомогою коефіцієнтів r_{ij} описуються ймовірності переходів заявки з вузла *i* у вузол *j*. Заявка, яка надходить у вузол *j* із вузла *i*, починає оброблятися негайно, без очікування. Після завершення обслуговування у вузлі j заявка миттєво та незалежно від інших може бути перенаправлена у наступний вузол *j+1* з певною ймовірністю $p_{j,j+1}$.

Для зручності опису поведінки заявок, що покидають систему, вводиться умовний нульовий вузол (0). Тоді ймовірність того, що заявка, оброблена у вузлі *j*, вийде із системи, тобто потрапить у нульовий вузол, визначається як p_{j0}.

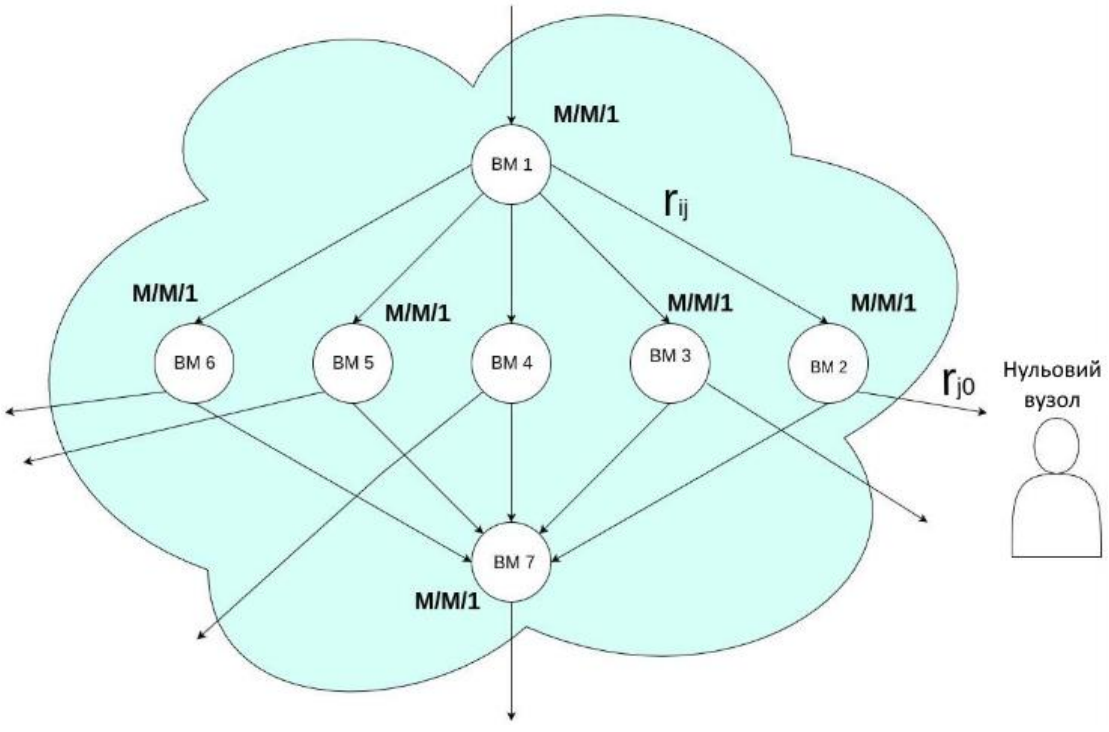


Рис. 3.3 СеМО хмарної програми

Для зручності опису ймовірностей переходів у системі вводяться стохастичні матриці розмірності (N + 1) × (N + 1), де N — це кількість вузлів у системі масового обслуговування з кінцевим числом елементів.

Стохастичну матрицю переходів обслугованих заявок можна записати у такому вигляді:

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} & p_{16} & p_{17} \\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} & p_{26} & p_{27} \\ p_{31} & p_{32} & p_{33} & p_{34} & p_{35} & p_{36} & p_{37} \\ p_{41} & p_{42} & p_{43} & p_{44} & p_{45} & p_{46} & p_{47} \\ p_{51} & p_{52} & p_{53} & p_{54} & p_{55} & p_{56} & p_{57} \\ p_{61} & p_{62} & p_{63} & p_{64} & p_{65} & p_{66} & p_{67} \\ p_{71} & p_{72} & p_{73} & p_{74} & p_{75} & p_{76} & p_{77} \end{pmatrix}$$

Рис. 3.4 Стохастична матриця переходів

Співвідношення для можливостей переходів:

$$p_{n_i,i^+} = \lambda_i, \tag{3.6}$$

$$p_{n_i,i^-} = \mu_i p_{i0}, \tag{3.7}$$

$$p_{n_i,i^+j^-} = \mu_j p_{ji}. \tag{3.8}$$

Середня кількість заявок, що перебувають у черзі i-го вузла, який моделюється як система масового обслуговування типу M/M/1 з потоком заявок, що підпорядковується розподілу Пуассона, та експоненційним часом обслуговування, обчислюється за такою формулою:

$$L_i = \frac{p_i}{1 - p_i}, \quad i = 1,2,3,4,5,6,7. \tag{3.9}$$

Загальна кількість запитів, що надійшли в систему обслуговування, але знаходяться в режимі очікування:

$$L = \sum_{i=1}^{7} L_i \tag{3.10}$$

Середній час обслуговування запиту в і черзі і віртуальної машини розраховується за формулою:

$$W_i = \frac{L_i}{\Lambda_i} = \frac{1}{\mu_i(1 - p_i)} \tag{3.11}$$

Загальний середній час обслуговування в системі не може дорівнювати сумі середніх значень часу обслуговування у вузла.

3.3 Результати імітаційного моделювання

Продуктивність процесора характеризується кількістю інструкцій, які він здатен обробити за одну секунду. Цей показник дозволяє оцінити реальну обчислювальну потужність системи та її здатність ефективно виконувати складні обчислювальні завдання. Для його позначення використовується абревіатура MIPS (*Million Instructions Per Second*), що перекладається як «мільйон інструкцій за секунду». Вона відображає швидкодію процесора, вимірюючи, скільки мільйонів інструкцій може бути виконано за одну тисячну частку секунди (10^{-3} с). Такий показник є важливим параметром при моделюванні продуктивності серверних систем, оскільки він безпосередньо впливає на здатність обробляти потоки запитів у реальному часі.

У розробленій моделі системи масового обслуговування кожен вузол (віртуальна машина) має власну інтенсивність обробки запитів, що відповідає його технічним характеристикам — таким як тактова частота, кількість ядер, тип архітектури та обсяг оперативної пам’яті. Це дозволяє більш точно враховувати різницю у швидкодії між різними компонентами системи. Висока інтенсивність обробки означає, що вузол здатен обслуговувати більшу кількість заявок за одиницю часу, тим самим знижуючи середній час очікування в черзі та підвищуючи загальну ефективність роботи системи.

У таблиці наведено значення інтенсивностей обслуговування для кожної *i-oї* віртуальної машини, що входить до складу моделі. Ці дані дозволяють провести порівняльний аналіз між вузлами, визначити вузькі місця в архітектурі та оптимізувати розподіл навантаження для досягнення максимальної продуктивності всієї системи.

Таблиця 3.1

Значення інтенсивностей обслуговування для кожної *i-oї* віртуальної машини

№ вузла (i)	Інтенсивність обробки запиту вузлом, MIPS
1	5
2	1
3	3
4	5
5	7
6	9
7	5

$$P = \begin{pmatrix} 0 & 0.15 & 0.15 & 0.2 & 0.2 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Рис. 3.5 Стохастична матриця переходів з *i*-го вузла в *j*-й

У таблиці 3.2 наведено один із можливих варіантів значень інтенсивності надходження запитів до кожного вузла, що розглядалися під час проведення експерименту.

Таблиця 3.2

Інтенсивність надходження запитів до кожного вузла

Λ_i	Розрахункова формула	Значення, 1/мс
Λ_1	λ	180
Λ_2	$\Lambda_1 \cdot r_{12}$	27
Λ_3	$\Lambda_1 \cdot r_{13}$	27
Λ_4	$\Lambda_1 \cdot r_{14}$	36
Λ_5	$\Lambda_1 \cdot r_{15}$	36
Λ_6	$\Lambda_1 \cdot r_{16}$	54
Λ_7	$\Lambda_2 \cdot r_{27} + \Lambda_3 \cdot r_{37} + \Lambda_4 \cdot r_{47} + \Lambda_5 \cdot r_{57} + \Lambda_6 \cdot r_{67}$	54

Використовуючи значення інтенсивності вхідного потоку до вузла та інтенсивності його обслуговування, можна визначити відсоток завантаженості вузла (таблиця 3.3), який відображає рівень використання обчислювальних ресурсів віртуальної машини. Це надзвичайно важливий показник, що застосовується під час планування та оптимізації розподілу ресурсів у системі.

Таблиця 3.3

Завантаженість вузла

Інтенсивність використання виділених ресурсів p_i	Розрахункова формула	Значення (у частках)
p_1	$\Lambda_1 / \mu_1 = \lambda / \mu_1$	0.81
p_2	Λ_2 / μ_2	0.72
p_3	Λ_3 / μ_3	0.74
p_4	Λ_4 / μ_4	0.76
p_5	Λ_5 / μ_5	0.78
p_6	Λ_6 / μ_6	0.80
p_7	Λ_7 / μ_7	0.83

Використовуючи всі отримані раніше обчислені значення, можна визначити один із найважливіших показників для аналізу продуктивності системи — час очікування запиту в системі. При цьому враховується можливість тимчасового виходу з ладу однієї з віртуальних машин логічного рівня. Розраховані значення для змінної інтенсивності надходження запитів у діапазоні від 160 до 250 одиниць за мілісекунду наведено в таблиці 3.4.

Таблиця 3.4

Час очікування заявки в *i*-ій черзі

Інтенсивність використання виділених ресурсів %	Середній час очікування в черзі, мс	Мінімальний час очікування в черзі, мс	Максимальний час очікування в черзі, мс
80	6.370	6.173	6.568
71	4115.760	3241.148	4990.493
73	803.147	701.255	905.242
75	434.460	401.364	467.566
77	277.480	249.982	304.969
79	236.522	219.649	253.417
81	69.620	69.678	73.330

На основі отриманих даних побудовано графік, який відображатиме розподіл часу очікування запиту в черзі для кожного вузла системи, що складається із семи вузлів із пуассонівським вхідним потоком та експоненційно розподіленим часом обслуговування.

Також розраховано загальний час перебування запиту в системі для різних значень інтенсивності вхідного потоку.

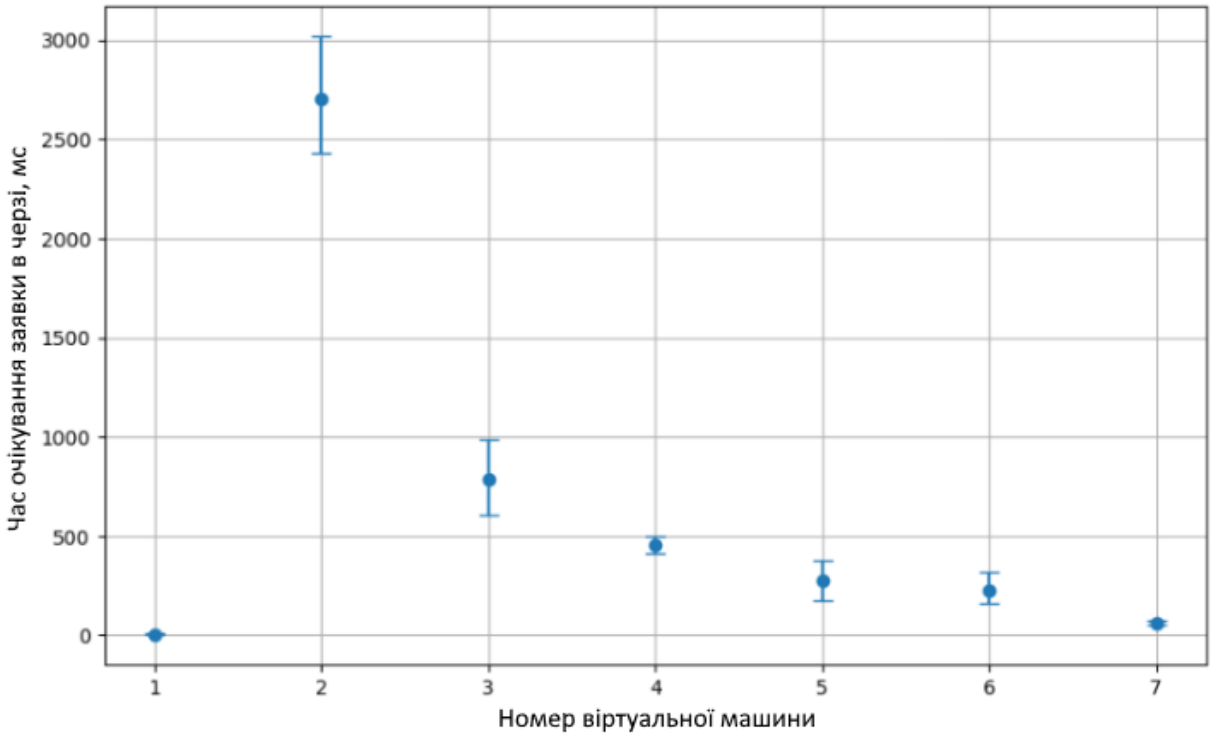


Рис. 3.6 Час очікування заявки в черзі в *i*-му вузлі

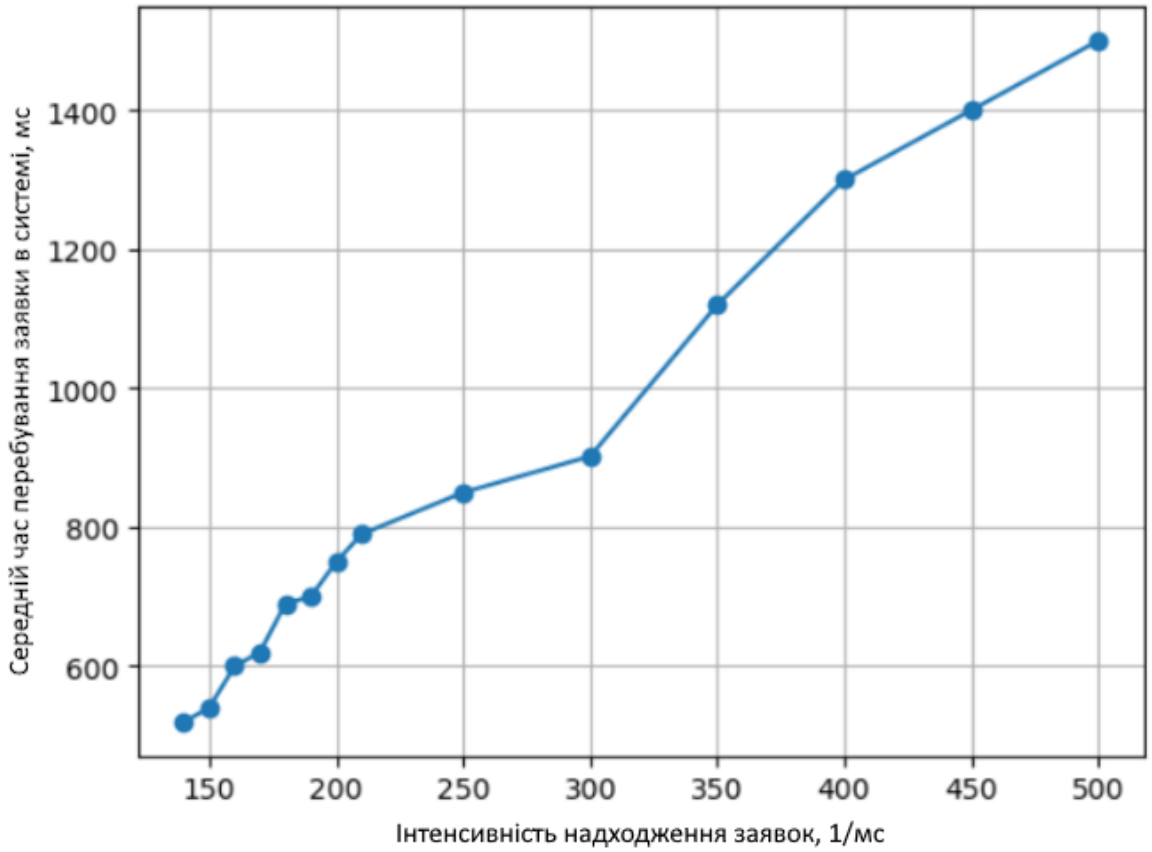


Рис. 3.7 Зміна часу перебування заявки у системі зі збільшенням інтенсивності надходження заявок

4 СФЕРА РОЗРОБКИ ПРИКЛАДНИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ

4.1 Структура та область розробки

Насамперед варто уточнити, що з точки зору архітектурного підходу суть технічного опису конкретних телекомунікаційних застосунків полягає не стільки у відповіді на запитання, *як саме* їх потрібно створювати, скільки у виборі оптимальних технологій для їх побудови та визначенні способів їх практичного застосування.

Ця область зосереджується переважно на організації процесу розробки, виборі інструментів і засобів проєктування, прийнятому на підприємстві життєвому циклі розробки телекомунікаційних систем, управлінні конфігураціями, контролі версій, використанні проміжного програмного забезпечення (middleware) та інших засобів, що забезпечують ефективну розробку та експлуатацію систем зв’язку.

Основна мета цього розділу архітектури — формування оптимального набору прикладних рішень, які реалізують бізнес-функції організації, підтримують цільові телекомунікаційні процеси та взаємодіють між собою стандартизованими інтерфейсами.

Ключові показники ефективності при цьому полягають у підвищенні якості та продуктивності процесу розробки завдяки:

- забезпеченню високої якості кінцевого ТК-продукту;
- підвищенню стабільності та керованості процесів його створення;
- зменшенню вартості розробки, впровадження та супроводу.

Розглянемо напрями організації проєктних заходів, які дозволяють досягти зазначених результатів.

1. Забезпечення якості телекомунікаційних рішень

Щоб гарантувати високу якість телекомунікаційного продукту, необхідно всебічно дослідити та описати предметну область, що автоматизується. Це вимагає поєднання різних архітектурних подань:

- поведінкового (динамічного) — опис процесів функціонування системи в реальному часі;
- структурного (статичного) — відображення компонентів і взаємозв’язків між ними;
- логічного — відповідність функціональним вимогам;
- фізичного — урахування географічного розподілу елементів мережі та взаємодії вузлів.

Зазвичай прикладний зріз телекомунікаційної архітектури представлений у вигляді таких моделей:

- статична структурна модель, яка відображає підсистеми та їх взаємозв’язки, розроблювані незалежно;
- динамічна модель виконання, що демонструє організацію процесів під час роботи системи;
- інтерфейсна модель, яка визначає сервіси, що надаються кожною підсистемою через уніфіковані API;
- модель взаємодій, яка ілюструє потоки даних, послідовність виконання операцій, реакції на події тощо.

2. Архітектура технологічних процесів у ТК-середовищі

Зріз організації технологічних процесів, який відображає архітектуру прикладних рішень, фокусується на виробництві телекомунікаційних систем і охоплює такі аспекти:

- набір технологій, обраних для створення програмних компонентів;
- інструменти проєктування, розробки та тестування;
- стандарти, що регламентують процес розробки та набір засобів;
- структура циклу розробки та контролю версій.

Такий підхід дає змогу зменшити кількість варіантів технічних рішень, що використовуються при проєктуванні, конфігуруванні, інтеграції та експлуатації систем, а також мінімізує витрати на навчання персоналу.

3. Уніфікація архітектурних рішень

Очевидно, що основою будь-якого архітектурного підходу є уніфікація методів розробки. Важливу роль тут відіграє частина архітектури, пов’язана з розробкою прикладних телекомунікаційних систем — зокрема використання шаблонів (патернів) проєктування.

Патерни являють собою структуровані рішення архітектурних завдань, що описують, як повинні бути організовані компоненти та їхні взаємодії для забезпечення необхідної функціональності. Вони діють на вищому рівні абстракції, ніж шаблони програмування, але виконують ту ж місію — забезпечення якості, масштабованості та стійкості системи.

Завдяки своєму поширенню архітектурні патерни є зрозумілими для більшості інженерів і значно спрощують взаємодію в командах розробників. Далі у розділі буде наведено приклади застосування таких шаблонів у телекомунікаційних рішеннях.

4. Роль інформаційної системи в архітектурі ТК

У центрі будь-якої телекомунікаційної архітектури розташована інформаційна система (ІС), або прикладна програма — апаратно-програмний комплекс, що поєднує компоненти, бази даних, протоколи взаємодії та реалізує ключові бізнес-функції оператора зв’язку або підприємства.

До складу ІС зазвичай входять:

- програмні продукти (модулі управління трафіком, білінгові системи, VMS тощо);
- моделі даних (бази користувачів, сесій, журналів подій);
- інтерфейси (API) для взаємодії між підсистемами;
- користувацькі інтерфейси для операторів та адміністраторів.

Як і в інших архітектурних шарах, прикладні рішення поділяються за рівнями абстракції, утворюючи три основні домени:

- Рівень представлення — усе, що пов’язане з взаємодією користувача з системою.
- Бізнес-рівень — логіка, алгоритми, обробка даних та реакції на події.

- Рівень доступу до даних — збереження, вибірка, модифікація та видалення інформації.

Кожен із цих доменів несе окрему відповідальність, а разом вони формують єдиний каркас системи, який забезпечує масштабованість і стійкість до змін.

5. Масштабованість та еволюційність системи

Архітектура прикладних телекомунікаційних систем охоплює не лише застосунки, але й механізми та інструменти їх створення, тому ключовим критерієм «гарної архітектури» є можливість масштабування як самої системи, так і процесів її розробки.

Наявність добре спроєктованої компонентної архітектури дозволяє:

- масштабувати систему за рахунок додавання модулів і підсистем;
- забезпечити ремонтопридатність, коли зміни в одному модулі не потребують змін в інших;
- гарантувати заміщуваність компонентів;
- забезпечити тестованість і можливість ізольованої перевірки;
- підвищити повторне використання компонентів у різних контекстах;
- спростити супровід та оновлення рішень.

6. Архітектурні стилі та стандартизація в ТК

Щоб структурувати та уніфікувати підходи до побудови телекомунікаційних систем, що визначають структуру компонентів, правила їх взаємодії та гарантують досягнення необхідних якісних характеристик (гнучкості, масштабованості, безпеки, прогнозованості, зниження вартості тощо), активно застосовуються архітектурні стилі — такі як клієнт-серверна модель, SOA, мікросервісна архітектура, SDN/NFV-підходи та інші.

4.2 Архітектурні стилі

Архітектурний стиль — це сукупність принципів, правил і шаблонів, які визначають типову структуру компонентів системи та способи їхньої взаємодії. Він задає концептуальну модель побудови системи, формує набір корпоративних

технологій і операційних середовищ, а також описує вимоги до компонентів, їхніх зв’язків, механізмів обміну даними та поведінкових характеристик.

Метою застосування архітектурного стилю є забезпечення системи певними властивостями — стабільністю, масштабованістю, узгодженістю, безпечністю, — які орієнтовані на підтримку визначеного класу бізнес-процесів або телекомунікаційних сервісів.

Як випливає з цього визначення, вибір архітектурного стилю безпосередньо залежить від вимог до характеристик інформаційної (або телекомунікаційної) системи та способів її використання.
На практиці архітектурні стилі часто застосовуються у комбінованій формі, поєднуючи переваги кількох підходів для досягнення необхідного рівня гнучкості та ефективності системи.

Архітектурний стиль паралелізму (Concurrency)

Паралелізм — це властивість системи, за якої кілька незалежних завдань виконуються одночасно або з частковим перекриттям у часі. Такий підхід дозволяє підвищити ефективність обчислень та раціонально використовувати ресурси апаратного забезпечення.

Основні принципи:
окремі частини програми або процесу виконуються незалежно одна від одної, потенційно паралельно, що забезпечує більш ефективне використання процесорних ядер та обчислювальних блоків.

Переваги:
паралельне виконання завдань може суттєво підвищити продуктивність системи, особливо в середовищах із багатопроцесорною або багатоядерною архітектурою.

Недоліки:
проектування таких систем потребує ретельного контролю над потоками виконання, оскільки можуть виникати проблеми станів змагання (race conditions), взаємного блокування (deadlocks), або некоректної синхронізації доступу до спільних ресурсів. Вирішення цих проблем вимагає використання спеціалізованих архітектурних підходів та шаблонів.

Сфери застосування:

системи реального часу, високопродуктивні обчислювальні комплекси (HPC), сервери обробки запитів, веб-сервери, розподілені аналітичні платформи, а також телекомунікаційні системи, що одночасно обслуговують тисячі з’єднань або потоків даних.

Залежно від типу ресурсу, який використовується паралельно, застосовуються різні варіації цього стилю — наприклад, архітектура функціонального паралелізму (розподіл завдань за функціями) або архітектура паралелізму даних, де єдиний набір даних обробляється одночасно кількома обчислювальними елементами в різних процесах.

Використання таких підходів породжує низку складнощів — від конфліктів доступу до ресурсів до необхідності синхронізації даних між потоками. Для подолання цих проблем у практиці проєктування застосовуються відповідні архітектурні патерни паралелізму, що забезпечують стабільність, узгодженість та безпечне виконання процесів у багатопотоковому середовищі.

Наприклад, патерн «Оркестратор» керує взаємодією між окремими сервісами системи. Його основне призначення — організувати потік бізнес-логіки та забезпечити узгоджене виконання всіх дій у правильній послідовності.

Простими словами, один центральний сервіс (оркестратор) визначає, який сервіс, коли та в якій послідовності має виконати свою частину роботи. Після отримання запиту оркестратор самостійно викликає потрібні сервіси, передає їм дані, обробляє результати, приймає рішення щодо подальших кроків і контролює виконання процесу, включно з обробкою можливих помилок.

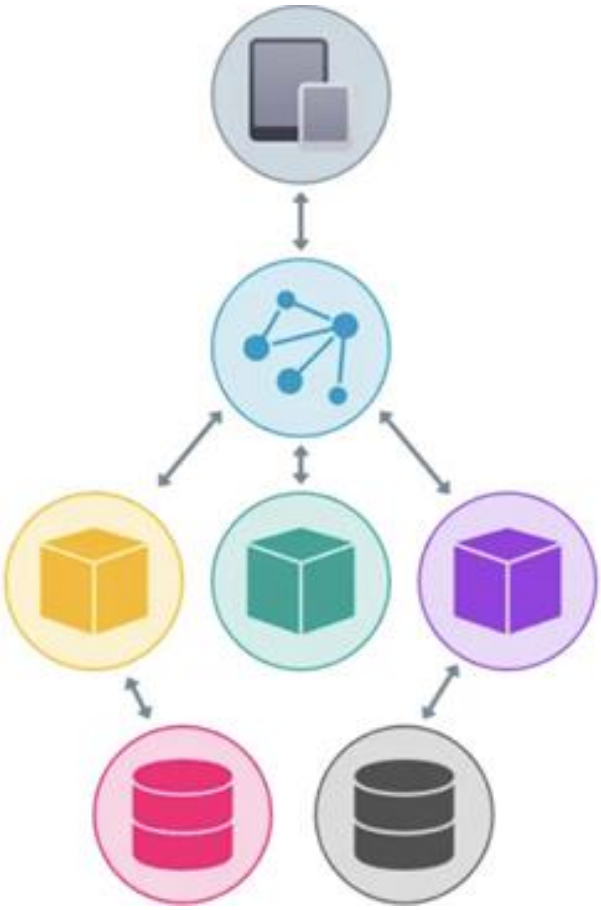


Рис. 4.1 Патерн «Оркестратор»

Такий підхід часто застосовується у складних телекомунікаційних або корпоративних бізнес-процесах, де потрібно узгодити роботу багатьох сервісів (наприклад, модулів білінгу, авторизації, моніторингу, маршрутизації тощо) та забезпечити централізоване управління потоком операцій.

Або, наприклад, патерн «Хореографія» — це підхід до побудови системи сервісів, у якій кожен сервіс працює автономно та взаємодіє з іншими через обмін подіями. Простими словами, кожен сервіс «знає свою партію» й виконує її тоді, коли відбувається певна подія, без наявності центрального керуючого елемента.

Взаємодія між сервісами у такій архітектурі ґрунтується на розподілених правилах: кожен сервіс публікує власні події та підписується на події інших сервісів. Таким чином формується мережа взаємозалежних компонентів, які координують роботу один одного без централізованого контролю.

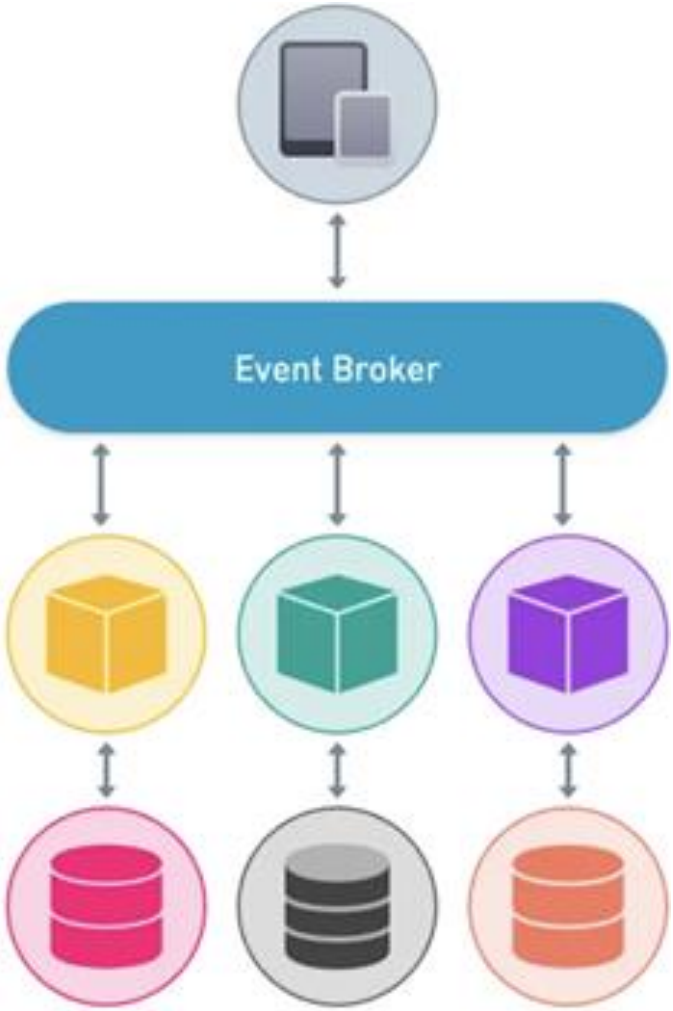


Рис. 4.2 Патерн «Хореографія»

Патерн «Хореографія» широко застосовується у децентралізованих телекомунікаційних або мікросервісних системах, де важливими є висока гнучкість, масштабованість і стійкість до збоїв, адже кожен компонент може незалежно функціонувати, оновлюватися чи замінюватися без впливу на роботу всієї системи.

Багатошаровий архітектурний стиль є одним із найпоширеніших архітектурних патернів, який активно використовується під час розробки традиційних веб-застосунків та корпоративних інформаційних систем.

Основні принципи:

цей архітектурний підхід передбачає розподіл функціональності системи на окремі шари, кожен з яких виконує свою роль. Типовим прикладом є тришарова архітектура, що включає:

- шар представлення (інтерфейс користувача);

- шар бізнес-логіки (обробка запитів, алгоритми, правила роботи системи);
- шар зберігання даних (бази даних, файлові сховища).

Переваги:

архітектура зрозуміла для реалізації, зручна в тестуванні та супроводі; окремі шари можуть розроблятися, розгортатися й оновлюватися незалежно, що підвищує гнучкість системи.

Недоліки:

може спричиняти збільшення накладних витрат на обмін даними між шарами; зміни, які зачіпають одразу кілька рівнів архітектури, часто вимагають додаткових узгоджень і переробки.

Сфери застосування:

веб-застосунки, корпоративні системи, а також телекомунікаційні програмні платформи, що мають чітко структуровану логіку обробки запитів, наприклад білінгові, CRM-системи або системи моніторингу мереж.

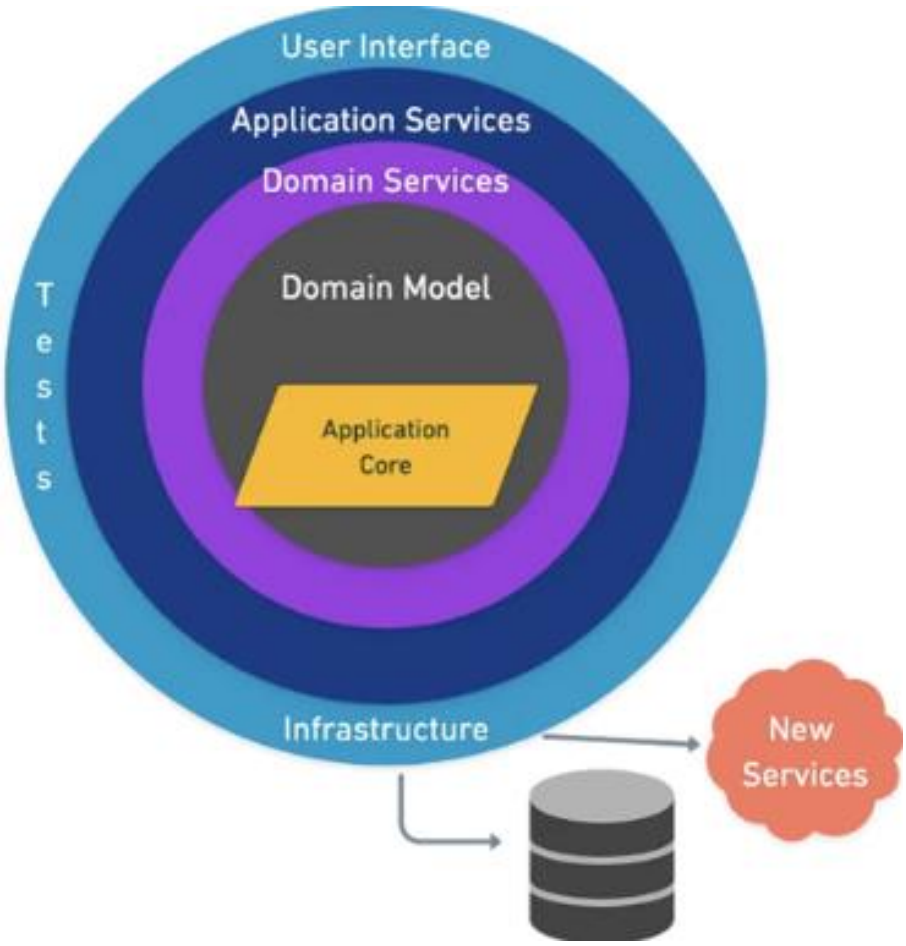


Рис. 4.3 Багатошаровий патерн

Багатошаровий (n-tier) патерн — це підхід, за якого система поділяється на N окремих шарів, кожен із яких виконує чітко визначену функцію та відповідає за певний аспект роботи системи.

Найпоширеніша модель передбачає поділ на три основні шари:

- шар представлення — забезпечує взаємодію користувача із системою;
- шар бізнес-логіки — обробляє дані, реалізує правила та алгоритми роботи;
- шар зберігання даних — відповідає за доступ, збереження й оновлення інформації в базах даних.

Архітектурний стиль розподілених систем — це підхід, за якого компоненти системи розміщуються на різних мережевих вузлах і взаємодіють між собою шляхом обміну повідомленнями для досягнення спільної мети. Така архітектура забезпечує узгоджену роботу багатьох обчислювальних елементів, які спільно функціонують як єдина система для кінцевого користувача.

Основні принципи:

розподілені компоненти об’єднуються в мережу та взаємодіють через стандартизовані протоколи, створюючи ілюзію єдиного цілісного середовища. Кожен вузол системи може виконувати власну функцію, обмінюватися даними з іншими та брати участь у спільному обчислювальному процесі.

Переваги:

- масштабованість — можливість розширення системи шляхом додавання нових вузлів;
- стійкість до відмов — збій окремого елемента не призводить до зупинки всієї системи;
- спільний доступ до ресурсів — ефективний розподіл обчислювальних потужностей і даних між серверами.

Недоліки:

- підвищена складність проєктування та налагодження;
- залежність від якості мережевого з’єднання;
- проблеми узгодженості даних у реальному часі між різними вузлами.

Сфери застосування:

розподілені бази даних, хмарні обчислення, мережеві платформи, телекомунікаційні системи управління трафіком, маршрутизацією або передачею сигналів, де необхідна висока доступність і надійність.

Одним із найвідоміших архітектурних патернів цього типу є «Простір даних» (Space-Based Pattern), також відомий як «Простір кортежів» або «Хмарна архітектура». Його мета — усунути єдину точку відмови та запобігти вузьким місцям продуктивності шляхом рівномірного розподілу сервісів і ресурсів між кількома серверами. Такий підхід забезпечує стійкість і динамічне балансування навантаження в масштабних телекомунікаційних середовищах.

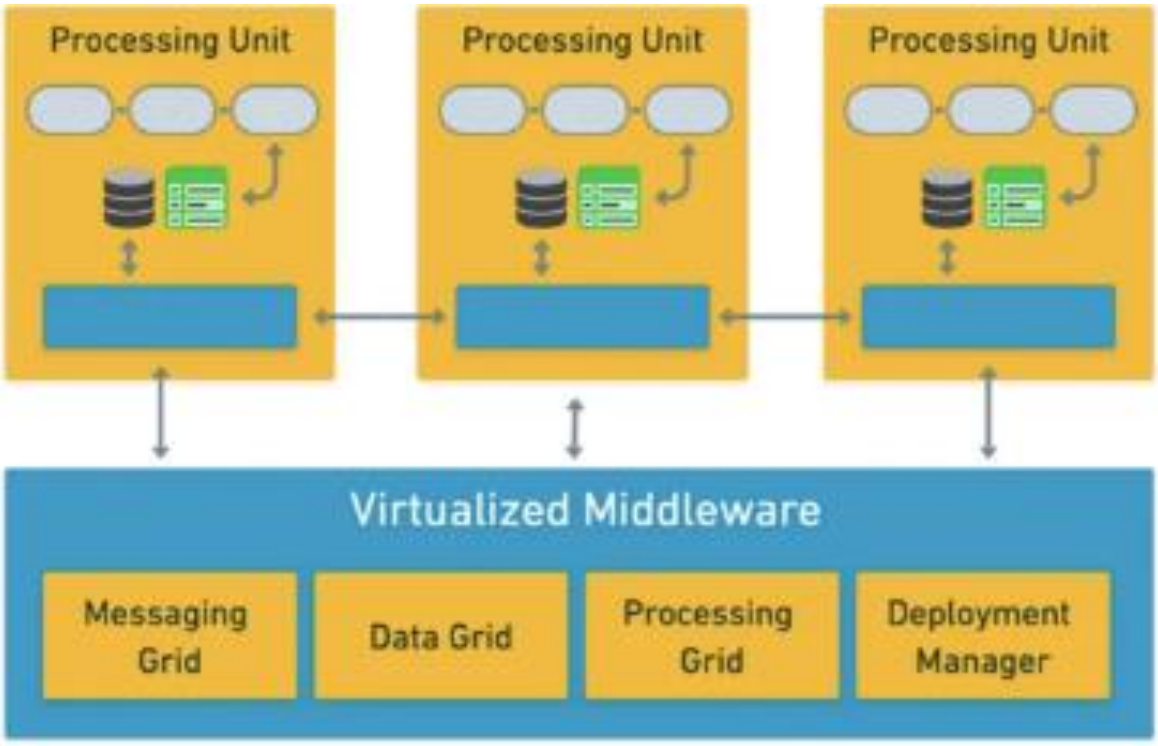


Рис. 4.4 Стиль розподілених систем

Він є ідеальним рішенням для високонавантажених застосунків, які потребують стовідсоткової безвідмовної роботи та можливості горизонтального масштабування. Такий підхід особливо ефективний у фінансових торговельних системах, онлайн-платформах для ігор, сервісах потокової передачі даних або телекомунікаційних вузлах, де критично важливо забезпечити безперервність обслуговування та стабільну продуктивність навіть під час пікових навантажень.

Слід зазначити, що існує багато архітектурних підходів, кожен з яких має власні переваги, обмеження та рекомендації щодо застосування. На практиці

архітектура програмної або телекомунікаційної системи майже ніколи не обмежується одним стилем — найчастіше вона являє собою поєднання кількох архітектурних стилів, які взаємодоповнюють один одного та формують надійний фундамент усієї системи.

4.3 Класифікація додатків

У попередньому було частково розглянуто питання вибору архітектурних рішень залежно від характеристик програмних продуктів. З огляду на їхнє різноманіття, тепер важливо визначити принциповий підхід до раціонального вибору архітектури для різних типів інформаційних систем, які автоматизують різні бізнес-процеси та потребують специфічного інформаційно-технологічного середовища — з різною продуктивністю, реактивністю, надійністю, рівнем безпеки тощо.

Для вирішення цього завдання була сформульована концепція “категорії застосунку”, яка визначає групу програмних рішень як певну сукупність корпоративних технологій та операційних середовищ, орієнтованих на обслуговування конкретного класу бізнес-процесів.
У цьому контексті архітектурний стиль виступає інструментом структуризації системи (сегментації) та сприяє повторному використанню перевірених архітектурних шаблонів, які допомагають розв’язувати типові проблеми під час розробки прикладних систем.

Крім того, категоризація застосунків є ефективним інструментом для планування інвестицій, управління ризиками, розвитку ТК-інфраструктури, оптимізації портфеля застосунків і підвищення керованості архітектури підприємства.

З цієї точки зору прикладні системи доцільно класифікувати за п’ятьма основними архітектурними категоріями:

1. Застосунки, що обслуговують велику кількість транзакцій

Це системи, які мають справу з інтенсивними потоками операцій, наприклад:

- білінгові системи телекомунікаційних операторів,
- системи бронювання авіаквитків,
- платіжні шлюзи або процесинги банківських карток.

Основні вимоги до таких систем — забезпечення цілісності даних, масштабованість, висока продуктивність, надійність та транзакційна узгодженість.

2. Операційні системи реального часу

Це застосунки, які повинні реагувати на події миттєво, з мінімальною затримкою. Приклади:

- системи управління транспортом в аеропортах,
- системи моніторингу стану пацієнтів у лікарнях,
- телекомунікаційні системи керування мережею або сигналізацією.

Такі рішення мають забезпечувати високу швидкість реакції, безперервну роботу 24/7, надійність і можливість пріоритизації запитів.

3. Аналітичні та експертні системи (Business Intelligence, Decision Support Systems)

Ці застосунки орієнтовані на аналітику великих масивів даних, пошук закономірностей, прогнозування та підтримку ухвалення управлінських рішень. Приклади:

- системи аналізу Big Data,
- платформи прогнозування попиту чи ризиків,
- системи оцінки кредитоспроможності клієнтів.

Основні вимоги — зрозумілість і наочність результатів, здатність працювати з великими обсягами даних, аналітична точність та інтерпретованість результатів.

4. Системи підтримки спільної роботи (Collaborative Applications)

Такі системи забезпечують взаємодію користувачів у режимі асинхронного або синхронного обміну даними. Приклади:

- електронна пошта, корпоративні месенджери, форуми, групові календарі,

- сервіси спільного редагування документів чи управління проєктами. Основні вимоги — висока швидкість поширення інформації, зручність використання, надійність та велика пропускна здатність мережевих каналів.

5. Корпоративні та сервісні застосунки (Enterprise Applications)

До цієї категорії належать стандартні системи управління підприємством, наприклад:

- ERP-системи (Enterprise Resource Planning),
- CRM-системи (Customer Relationship Management),
- HRM-системи (Human Resource Management),
- системи обліку та розрахунку заробітної плати. Для них характерні висока надійність, порівняно низька вартість обслуговування, складність інтеграції з іншими корпоративними платформами та потреба у централізованому управлінні.

Поділ прикладних систем за архітектурними категоріями дозволяє чітко визначити вимоги до їхньої структури, технологічного середовища, рівня надійності, продуктивності та безпеки, що є ключовим для ефективного проєктування телекомунікаційних та корпоративних рішень.

Для наочності класифікації доцільно використовувати матрицю категорій, у якій у рядках зазначаються вимоги до системи, а в стовпцях — категорії.

У комірках на перетині рядків і стовпців фіксуються значення або рівні відповідних характеристик, що дозволяє зручно порівнювати вимоги різних типів систем між собою.

Таблиця 3.5

Матриця категорій

Вимога	1. Процеси з великою кількістю транзакцій	2. Операції в реальному часі	3. Аналітичні процеси та бізнес-аналітика	4. Спільна робота (колаборація)	5. Корпоративні (сервісні)
Стратегічні потреби	Надання послуг	Час реакції системи	Здатність пояснювати результати	Поширення знань	Надійність, низька вартість з точки зору ТК
Бізнес-вимоги	Обслуговування клієнтів; Зменшення витрат; Робота 24/7; Цілісність даних	Економічність і безпека; Робота 24/7/365	Підвищення ефективності та продуктивності; Наглядність подання інформації	Швидкість оновлення знань; Повторне використання знань	Економічність; Покращення бізнес-процесів
Відмінні характеристики	Низька вартість (на одну транзакцію); Надійність; Масштабованість; Продуктивність; Резервування	Сканування та фільтрація потоків даних; Пріоритизація запитів; Надійність; Публікація та підписка на дані	Механізми аналітики; Потужність обробки даних; Сховища даних	Простота використання; Надійність; Висока пропускна здатність; Обмін даними «по горизонталі»	Стандартні процеси; Кандидати на аутсорсинг
Інтегруючі технології	Системи інтеграції корпоративних застосунків	Спеціально розроблений програмний код	Сховища даних	Спільно використовувані дані та обмін інформацією	Стандартні інтерфейси (API), XML

Існують й інші способи сегментації характеристик, які застосовуються для класифікації інформаційних систем із метою вибору найоптимальніших архітектурних стилів. Наприклад:

- за ступенем структурованості та формалізованості завдань;
- за типом представлення та логічною організацією збережених даних;
- за масштабом і рівнем інтеграції компонентів;
- за рівнями управління;
- за функціональною ознакою;
- та іншими критеріями.

Важливо розуміти, що не система «пристосовується» до архітектурного стилю, а навпаки — саме стиль підбирається під особливості застосунку, його бізнес-логіку та вимоги середовища.
Наприклад, для корпоративних телекомунікаційних застосунків, які потребують інтеграції з великою кількістю підсистем та зовнішніх сервісів, доцільно використовувати SOA-архітектуру (Service-Oriented Architecture), що забезпечує стандартизовану взаємодію між компонентами.
У випадку складних або багаторівневих бізнес-процесів ефективним рішенням може стати використання патерну оркестрації, який забезпечує централізоване керування потоками виконання.

А для високонавантажених телекомунікаційних платформ, де критичною є продуктивність і гнучке масштабування, найкраще підходить мікросервісна архітектура, яка дозволяє незалежно масштабувати окремі модулі системи та знижує ризик виникнення єдиної точки відмови.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досліджено теоретичні та практичні аспекти оптимізації архітектури багаторівневих телекомунікаційних систем для підвищення ефективності їх функціонування та якості обслуговування користувачів. Показано, що архітектурні рішення мають вирішальний вплив на продуктивність, надійність, відмовостійкість і масштабованість систем зв’язку, а отже — безпосередньо визначають рівень якості телекомунікаційних послуг.

Встановлено, що багаторівнева архітектура (n-tier) є однією з найефективніших для сучасних телекомунікаційних систем. Вона забезпечує логічне розділення компонентів на рівень представлення, бізнес-логіки та зберігання даних, що підвищує керованість і спрощує модернізацію системи. Така модель сприяє швидкому масштабуванню, зменшує взаємозалежність модулів і підвищує стабільність роботи при змінному навантаженні.

Досліджено низку архітектурних стилів і патернів, серед яких розподілений, сервісно-орієнтований (SOA), мікросервісний і паралельний підходи. Доведено, що вибір стилю має базуватися на характеристиках конкретної системи: для корпоративних платформ доцільно використовувати SOA, для високонавантажених — мікросервіси, а для складних багатопотокових процесів — архітектури з підтримкою паралелізму. Використання патернів «оркестрації» та «хореографії» дозволяє ефективно узгоджувати взаємодію між сервісами, забезпечуючи гнучкість і централізований контроль.

Для кількісного аналізу ефективності архітектури застосовано моделі систем масового обслуговування *M/M/1*. На основі розрахунків визначено залежність між інтенсивністю вхідного потоку, коефіцієнтом завантаження вузлів і середнім часом очікування заявок. Експериментальні результати підтвердили, що оптимальне завантаження вузлів становить близько 70–80 %, при якому досягається баланс між використанням ресурсів і мінімізацією затримок, що позитивно впливає на параметри QoS.

У роботі розглянуто роль паралельних обчислень на рівні даних і потоків (DLP, TLP, SIMT, MIMD), що забезпечують ефективну обробку великих масивів інформації у реальному часі. Використання гетерогенних архітектур із CPU та GPU підвищує швидкодію й стабільність систем, дозволяючи адаптувати їх до пікових навантажень без втрати продуктивності.

Окрему увагу приділено модульності та супроводжуваності архітектури, які дають змогу локалізувати відмови, прискорити оновлення та тестування компонентів, забезпечуючи стабільність системи у режимі безперервної роботи. Розроблено узагальнену матрицю категоризації телекомунікаційних інформаційних систем, що допомагає узгоджувати вимоги до архітектури з особливостями кожного класу — транзакційних, аналітичних, реального часу, корпоративних і колаборативних.

Узагальнюючи результати, можна стверджувати, що оптимізація архітектури багаторівневих телекомунікаційних систем є ключовим чинником підвищення ефективності, стабільності та якості обслуговування користувачів. Застосування модульного підходу, механізмів балансування навантаження, паралельних обчислень і гнучких архітектурних принципів забезпечує побудову адаптивних телекомунікаційних платформ, готових до реалізації концепцій 5G, 6G, SDN і NFV, а також до забезпечення гарантованого рівня QoS у складних багаторівневих мережах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Davidson J., Peters J., Gracely B. Voice over IP Fundamentals (2-ге або 3-тє вид.). – Cisco Press, 2007/2011. – 372 с.
2. Johnston A. B. SIP: Understanding the Session Initiation Protocol (4-те вид.). – Artech House, 2015. – 540 с.
3. Poikselkä M., Mayer G., Khartabil H., Niemi A. The IMS: IP Multimedia Concepts and Services (3-тє вид.). – Wiley, 2012. – 520 с.
4. Camarillo G., García-Martín M.-A. The 3G IP Multimedia Subsystem (IMS) (2-ге вид.). – Wiley, 2006. – 480 с.
5. Hattingh C., Sladden D., van der Ven A. SIP Trunking. – Cisco Press, 2010. – 280 с.
6. Van Meggelen J., Bryant L., Madsen J. Asterisk: The Definitive Guide (4-те вид.). – O’Reilly, 2013. – 500 с.
7. Gonçalves F. E. Building Telephony Systems with OpenSIPS (2-ге вид.). – Packt, 2016. – 320 с.
8. Kamailio Admin Book. – Asipto Publishing Group, 2018. – 400 с.
9. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach (8-ме вид.). – Pearson, 2021. – 864 с.
10. Stallings W. Foundations of Modern Networking: SDN, NFV, QoE, IoT and Cloud. – Addison-Wesley, 2016. – 600 с.
11. Голь В. Д. Телекомунікаційні та інформаційні мережі: навчальний посібник. – Київ: ІСЗЗІ КПІ ім. І. Сікорського, 2021. – 250 с.
12. Микитишин А. Г., Міщенко М. М. Телекомунікаційні системи та мережі: навчальний посібник. – Львів: Львівська політехніка, 2020. – 300 с.
13. Lemeshko O. Л. «Model of data traffic QoS fast rerouting in infocommunication networks». – Інформаційні технології та системи зв’язку, 2019. – Т. 9. – С. 127–134.
14. Serag R. H. «Software Defined Network Traffic Classification for QoS Optimization». – Journal of Network and Systems Management, 2025. – С. 98.

15. Palmieri F. «An MPLS-based architecture for scalable QoS and traffic engineering in converged networks». – Computer Networks, 2005. – С. 211–228.
16. Boostanimehr H., Bhargava V. K. «Unified and Distributed QoS-Driven Cell Association in Multi-Tier Heterogeneous Networks». – IEEE Transactions on Wireless Communications, 2014. – С. 415–427.
17. Wang J., Guan W., Huang Y., Schober R., You X. «Distributed Optimization of Hierarchical Small Cell Networks: A GNEP Framework». – IEEE Transactions on Communications, 2016. – С. 1435–1448.
18. Ordonez-Lucena J., Ameigeiras P., Lopez D. та ін. «Network Slicing for 5G with SDN/NFV: Concepts, Architectures and Challenges». – IEEE Communications Magazine, 2017. – С. 128–135.
19. Jamal E. T. «Comprehensive Analysis, Modeling and Optimization of IoT Networks». – Data & Knowledge Engineering, 2025. – С. 210–225.
20. Haryadi S. «Telecommunication Network Performance and Quality of Service: Network Architectures». – Telecommunication Review Series, 2018. – С. 55–72.
21. Ivanov D., Petrova N. «Optimisation of Multilayer Architectures in Telco Systems for Improved QoS». – International Journal of Telecommunication Systems, 2023. – С. 97–105.
22. Lee S., Kim H. «Design of Hierarchical Optical Network Layers for Enhanced Service Quality». – Optical Switching and Networking, 2022. – С. 64–71.
23. Zhang Y., Li X. «Multilevel Network Function Virtualisation and QoS in Next-Generation Networks». – Journal of Network and Computer Applications, 2021. – С. 190–205.
24. Nguyen T., Tran M. «Layered Architecture Optimization for Telecom Systems with Large-Scale Subscriber Base». – Telecommunication Systems, 2020. – С. 58–69.
25. Petrov A. «Evaluation of Service Quality in Multilayer Telecommunication Architectures». – Proceedings of the International Conference on Telecommunications, 2023. – С. 98–104.
26. Sokolov V., Kuznetsov D. «Adaptive Resource Allocation in Multi-Tier Telecom Systems for QoS Assurance». – Wireless Networks, 2024. – С. 75–84.

27. Fernández J., García M., Rodríguez C. «Hierarchical Network Engineering: Achieving High QoS in Tiered Telecommunication Systems». – IEEE Communications Magazine, 2021. – С. 98.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ

КВАЛІФІКАЦІЙНА РОБОТА

ОПТИМІЗАЦІЯ АРХІТЕКТУРИ БАГАТОРІВНЕВИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ

Виконав:
студент групи ТСДМ-62
Владислав ЗАЙЦЕВ

МЕТА РОБОТИ ТА АКТУАЛЬНІСТЬ

- Мета роботи – розроблення підходів до оптимізації архітектури багаторівневих телекомунікаційних систем для підвищення якості обслуговування користувачів (QoS).
- Об’єкт дослідження – архітектура багаторівневих телекомунікаційних систем нового покоління, що функціонують на основі сервісних, розподілених та програмно-керованих рішень (SDN, NFV, SOA, microservices).
- Предмет дослідження – методи, моделі та принципи оптимізації архітектури телекомунікаційних систем з використанням модульності, паралельних обчислень, сервісно-орієнтованого підходу та технологій SDN/NFV для підвищення рівня QoS та ефективності роботи мережі.

Актуальність.

- Сучасні телекомунікаційні системи працюють в умовах стрімкого зростання трафіку та вимог до швидкодії, надійності й адаптивності. Перехід до технологій 5G/6G, SDN, NFV та мікросервісних архітектур потребує оптимізації мережевих рішень для забезпечення високого рівня QoS. Традиційні монолітні системи вже не здатні ефективно масштабуватись і підтримувати роботу в режимі реального часу. Тому актуальним є пошук способів побудови гнучких, відмовостійких і продуктивних телекомунікаційних платформ із використанням модульності, паралельних обчислень та сервісних підходів. Це є ключем до стабільності, швидкого реагування та конкурентоспроможності мереж..

СУПРОВОДЖУВАНІСТЬ АРХІТЕКТУРИ

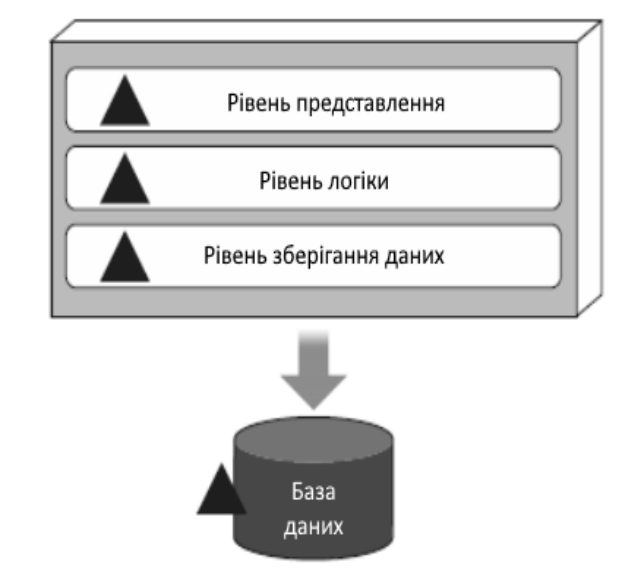
Супроводжуваність — це здатність системи легко змінюватися: додавати функції, виправляти помилки, оновлювати компоненти. Вона залежить від модульності, рівня зв'язності між компонентами, структури коду та якості архітектури.

Оцінку супроводжуваності можна здійснювати за метриками:

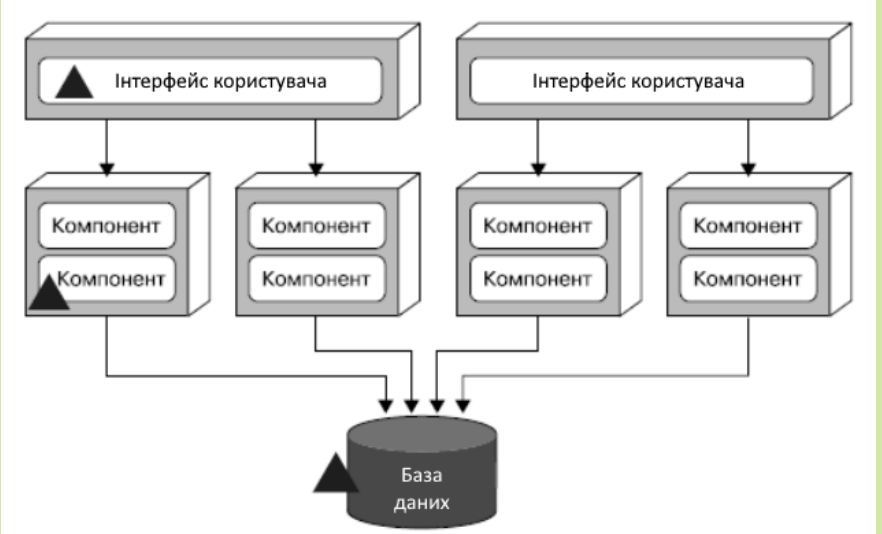
- coupling – ступінь залежності між модулями;
- cohesion – узгодженість логіки всередині компонента;
- цикломатична складність, розмір компонентів, предметне розділення.

Монолітна архітектура зазвичай має низьку супроводжуваність: навіть проста зміна потребує редагування кількох рівнів системи та координації між командами.

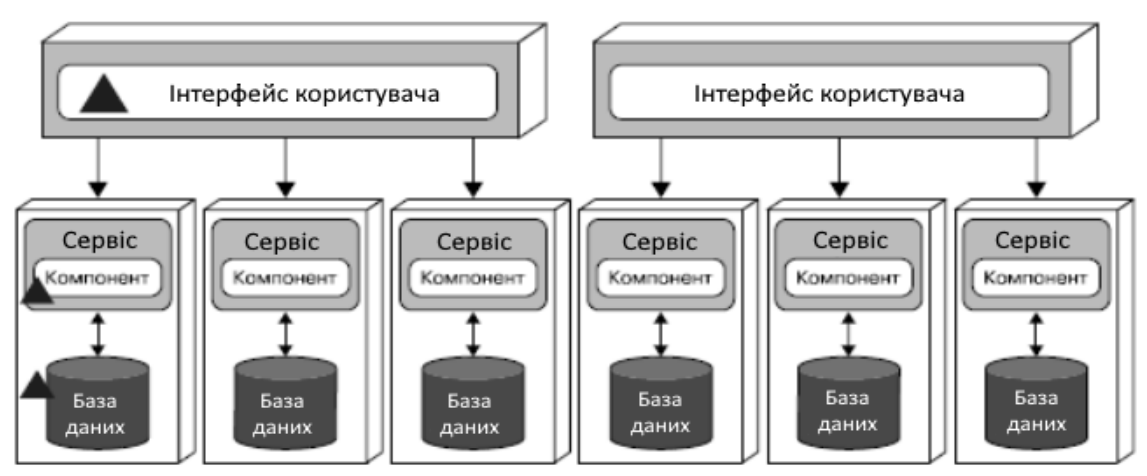
Модульна та мікросервісна архітектури дозволяють локалізувати зміни у межах одного сервісу, без впливу на решту системи. Це суттєво прискорює оновлення, знижує ризики та підвищує QoS — особливо в телекомунікаційних системах, де зміни повинні впроваджуватись швидко та безперервно.



Ілюстрація поширення змін між різними рівнями монолітної архітектури — інтерфейс, логіка, база даних



Ілюстрація різниці між монолітною та модульною архітектурою — зокрема, локалізація змін у межах окремого сервісу



Ілюстрація ізоляції змін у межах конкретного мікросервісу, відповідального за окрему бізнес-функцію

ТЕСТОВАНІСТЬ, РОЗГОРТАННЯ ТА МАСШТАБОВАНІСТЬ

Тестованість

- У монолітах тестування складне та повільне — зміна однієї функції потребує перевірки всієї системи. У модульній і мікросервісній архітектурі тестується лише змінений компонент → швидкі релізи та менше ризиків.

Розгортаність

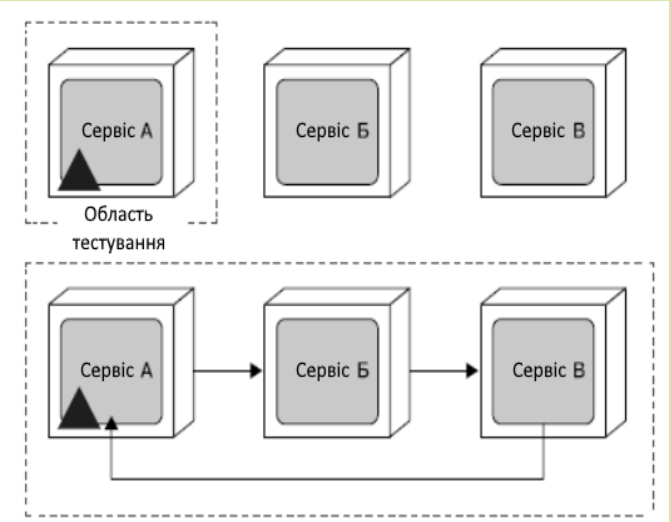
- Моноліти оновлюються рідко й ризиковано. Мікросервіси дозволяють часті автономні оновлення без простоїв і з мінімальними помилками.

Масштабованість і адаптивність

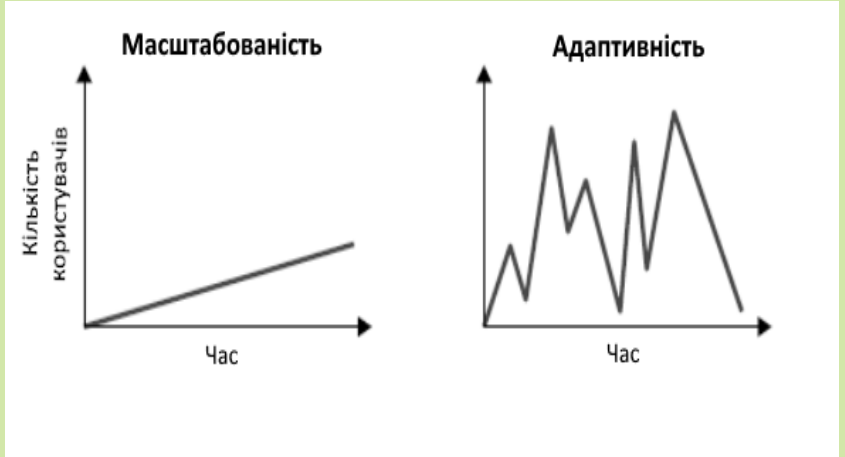
- Моноліт масштабується цілим блоком, що неефективно. SOA масштабує домени. Мікросервіси найкраще адаптуються до пікових навантажень (швидкий запуск сервісів).

Відмовостійкість

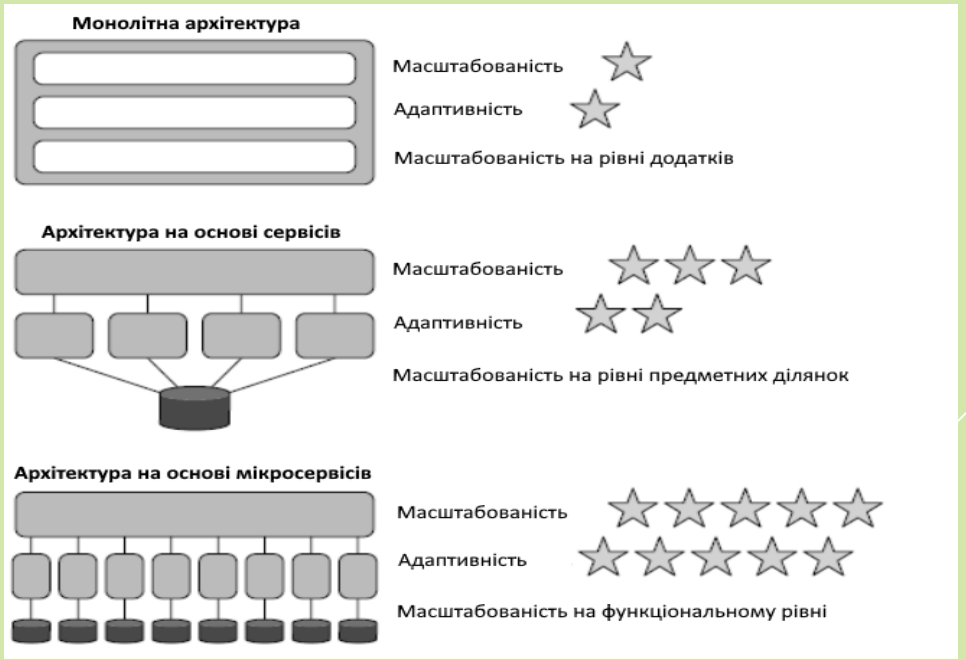
- Модульність дозволяє ізолювати збої — один сервіс може вийти з ладу без зупинки всієї системи. Це особливо важливо для телекомунікацій та підтримки QoS.



Приклад розподілу системи на сервіси A, B, C і зміну області тестування залежно від рівня їхньої взаємодії



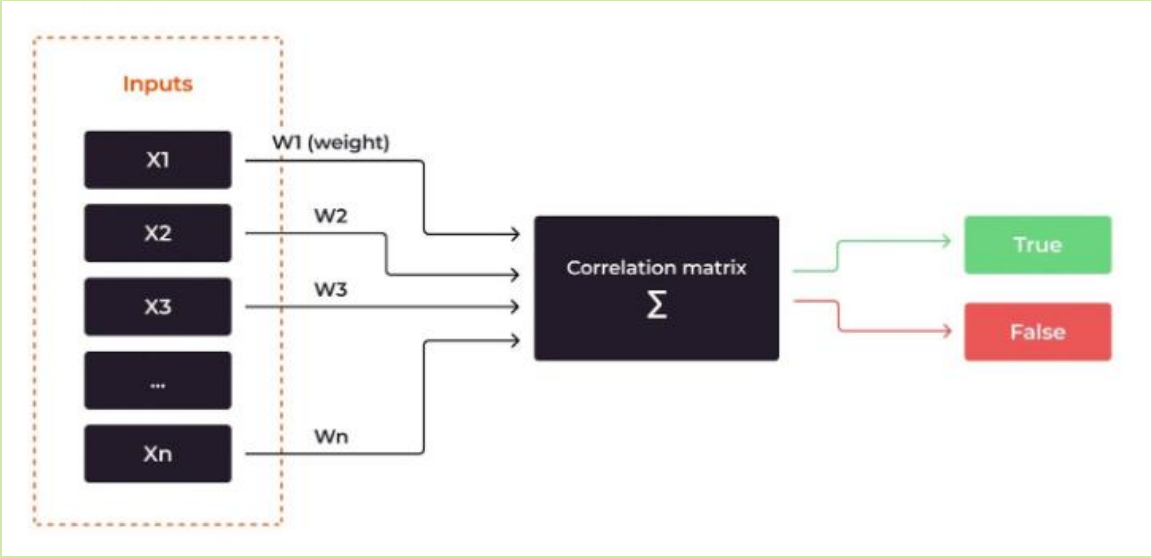
Різниця між масштабованістю — плавним наростанням навантаження, та адаптивністю — різким короткочасним сплеском активності користувачів



Порівняння трьох архітектурних підходів — монолітну, сервісну та мікросервісну архітектури — за рівнем масштабованості та адаптивності

ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ

- Нейронні мережі виконують багато однотипних операцій одночасно, тому ідеально реалізуються на GPU та кластерах серверів. Це забезпечує високу швидкість обробки й дозволяє аналізувати великі масиви телекомунікаційних даних у реальному часі — виявлення аномалій, QoS-контролю, прогнозування навантажень.
- Паралелізм — ключ до інтелектуальних телеком-систем. Саме GPU та розподілені кластерні архітектури забезпечують швидке навчання моделей, низькі затримки, автоматичну оптимізацію маршрутів і високу якість обслуговування (QoS).



Процес навчання нейронних мереж

Рівні паралельних обчислень

Рівень	Суть	Значення для телекомунікацій
DLP (Data-Level)	Одна операція над багатьма даними (SIMD)	Аналіз трафіку, сигналів, пакетів QoS
TLP (Task-Level)	Різні задачі виконуються одночасно	Контроль QoS, маршрутизація, моніторинг
ILP (Instruction-Level)	Кілька інструкцій в одному циклі	Зменшення затримок при обробці сигналів
Thread-Level	Кожен потік — окреме завдання	Реальна багатопоточність системи
Node-Level	Паралельна робота серверів/кластерів	SDN/NFV-контролери, розподілена аналітика
Hybrid / Multilevel	Поєднання кількох рівнів	5G/6G, великі AI-моделі, дата-центри

АРХІТЕКТУРИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

SIMD (Single Instruction, Multiple Data)

- Одна інструкція виконується над багатьма елементами даних одночасно. Ефективно для матричних обчислень, згорток, обробки сигналів та великих масивів даних. Використовується у NumPy, TensorFlow та сучасних CPU/GPU.

SIMT (Single Instruction, Multiple Threads)

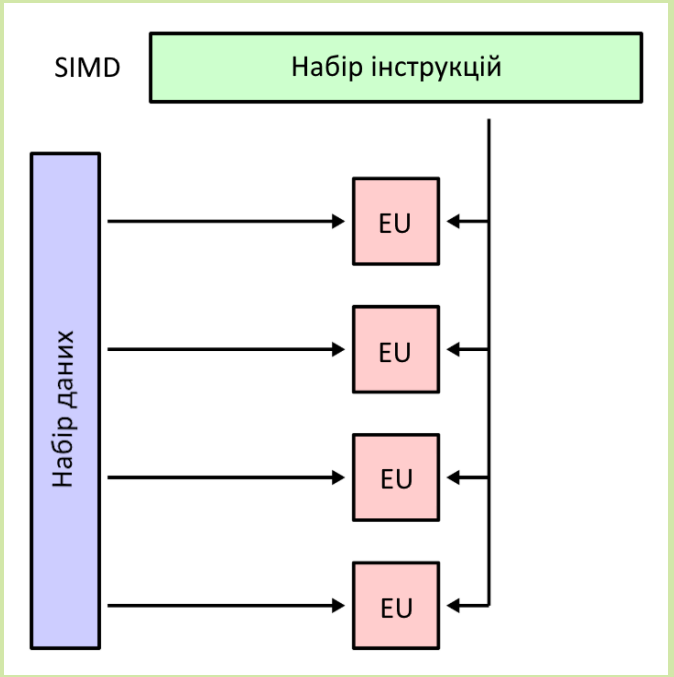
- Модель GPU (особливо NVIDIA): одна команда виконується паралельно у багатьох потоках (warp = 32 потоки). Підходить для DL, рендерингу, телеком-аналітики та симуляцій.

MIMD (Multiple Instruction, Multiple Data)

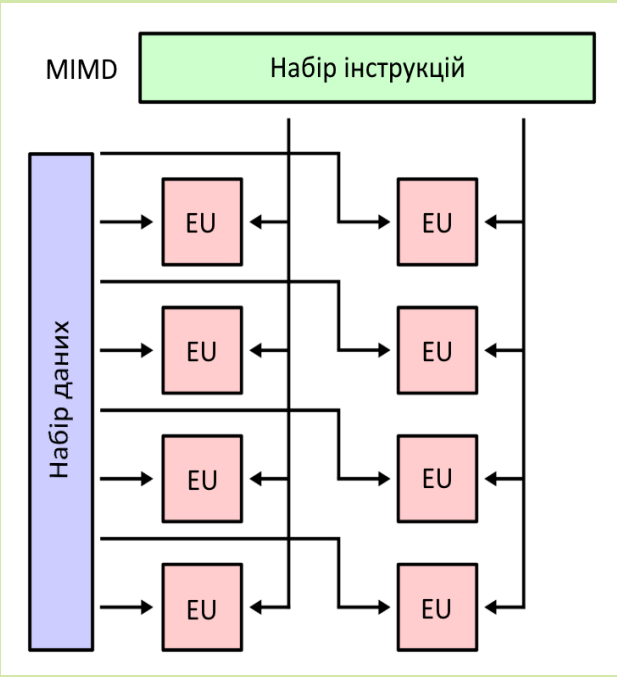
- Різні інструкції над різними даними виконуються одночасно. Дає гнучкість у складних системах: CPU обробляє логіку, GPU — масові обчислення. Використовується у гібридних CPU+GPU системах.

Гетерогенна архітектура

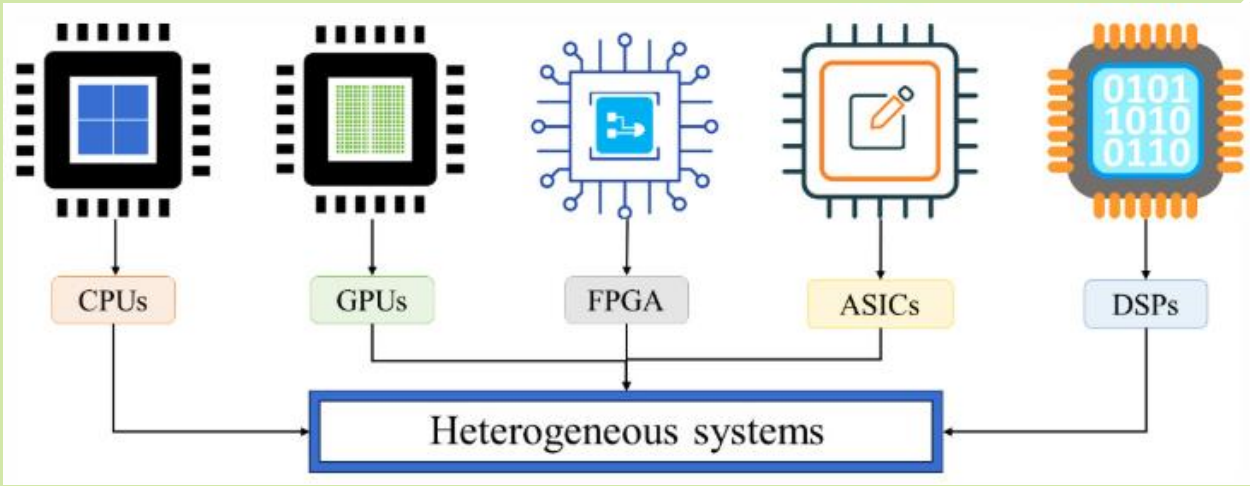
- Поєднання CPU, GPU, FPGA, ASIC. CPU керує логікою, GPU — масові обчислення, FPGA/ASIC — спеціалізовані задачі (обробка сигналів, енергоефективність). Забезпечує максимальну продуктивність для ШІ у телекомі.



Схематичне представлення архітектури SIMD: одна інструкція — кілька потоків даних



Архітектура MIMD: паралельне виконання різних інструкцій над множиною даних



Гетерогенні обчислювальні системи

БАГАТОРІВНЕВІ ЗАСТОСУНКИ

Багаторівнева архітектура хмарного застосунку реалізується на кластері віртуальних машин і зазвичай включає три основні рівні: **presentation, application та data storage**. Логічний рівень може мати кілька підрівнів залежно від бізнес-вимог та масштабованості системи.

Всі запити надходять на логічний рівень, де обробляються **балансувальником навантаження**. Він розподіляє трафік між віртуальними машинами згідно з їхньою потужністю, поточним навантаженням та станом. У разі відмови вузла трафік автоматично перенаправляється на інші машини.

- Використовуються різні алгоритми балансування:
- Round Robin;
- Least Connections;
- Random Distribution.

Якщо кластер має вузли з різною обчислювальною потужністю, політика розподілу повинна враховувати **інтенсивність обробки (MIPS)** для забезпечення оптимальної продуктивності системи.

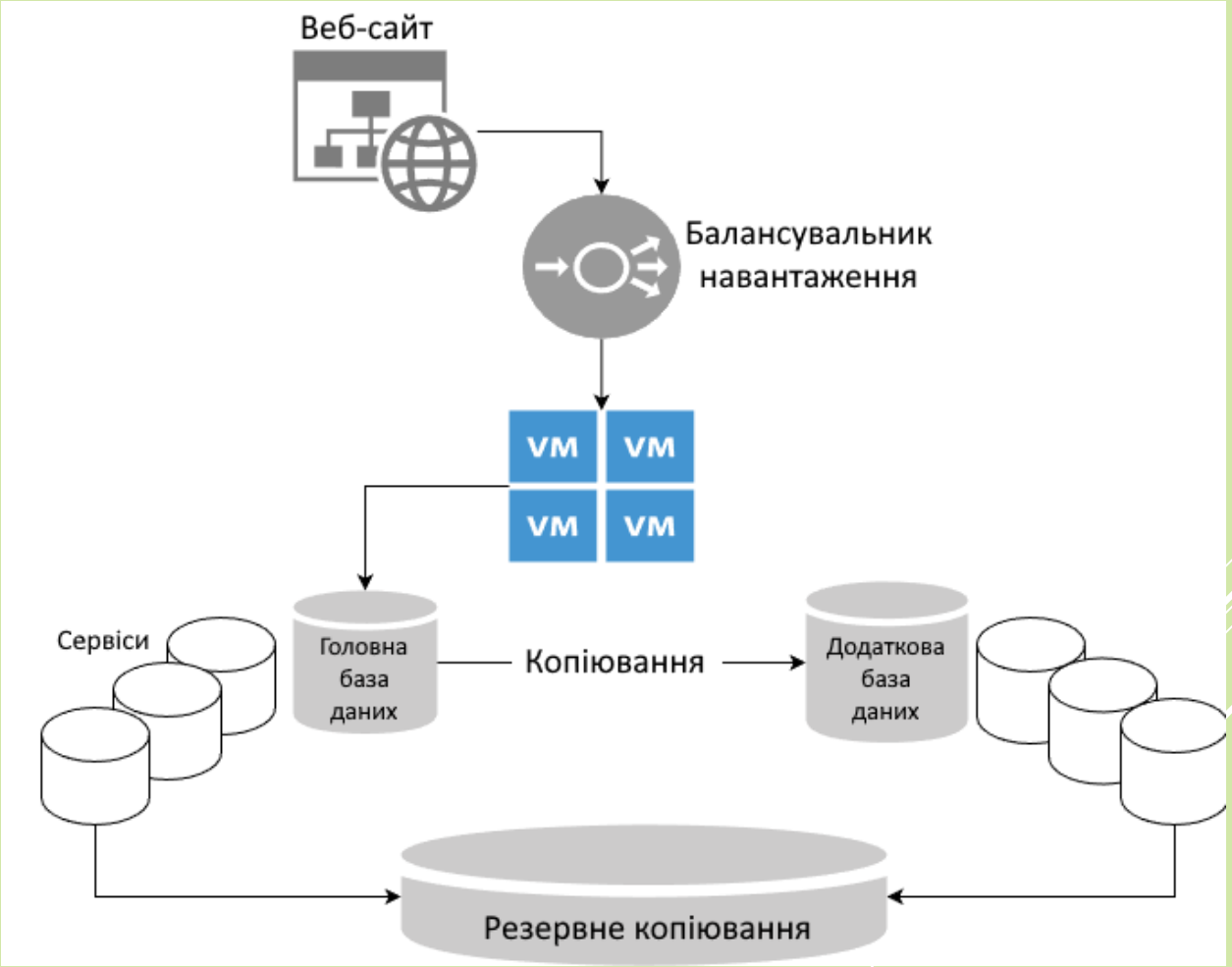


Схема тришарової архітектури хмарного застосунку

МЕРЕЖА МАСОВОГО ОБСЛУГОВУВАННЯ, ЩО ОПИСУЄ РОБОТУ ТРИРІВНЕВОЇ ХМАРНОЇ ПРОГРАМИ

У моделі запропоновано **трирівневу архітектуру**:

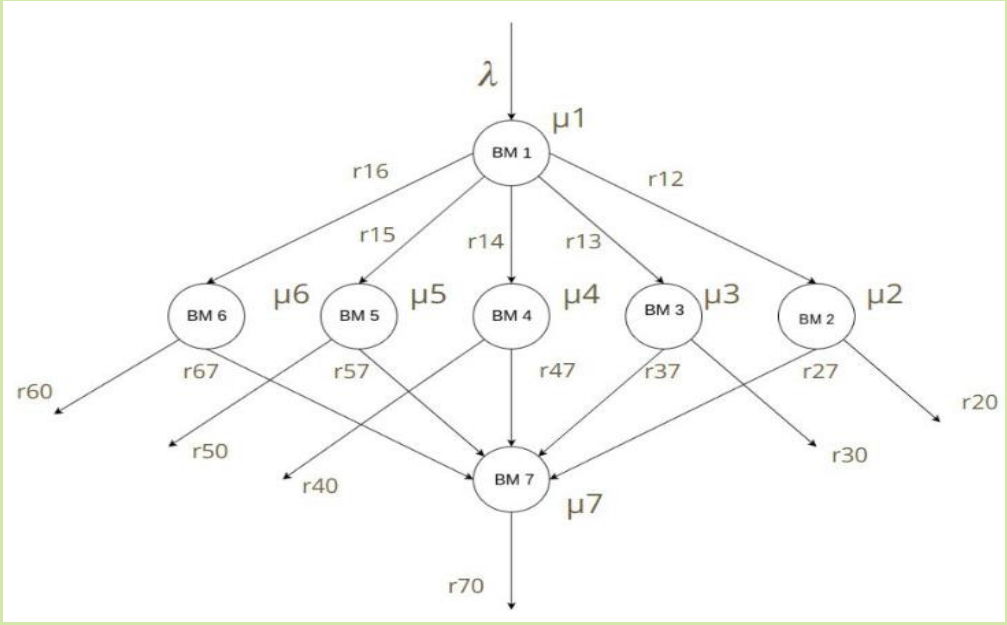
- **1 рівень** — балансувальник навантаження, який розподіляє потік заявок;
- **2 рівень** — п’ять бізнес-логічних вузлів, що забезпечують паралельне обслуговування та відмовостійкість;
- **3 рівень** — реплікована база даних, що гарантує збереження та доступність інформації.

Кожен вузол моделюється як **система масового обслуговування M/M/1** з нескінченною чергою та пуассонівським потоком заявок. Для опису переходів між вузлами застосовано **формули** та **геометричний розподіл**.

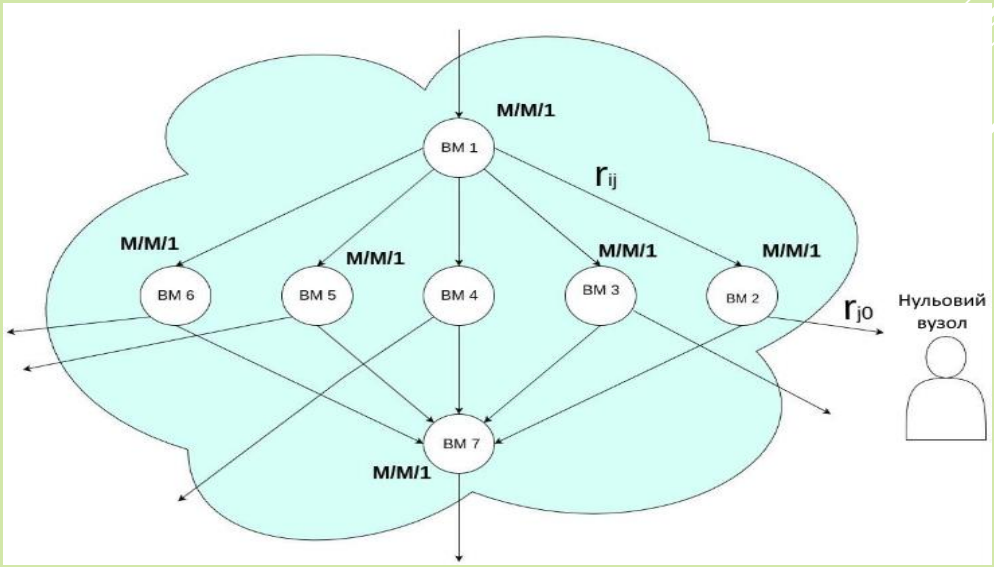
- З метою оцінки продуктивності використано **ймовірності переходів між станами, стохастичну матрицю** та **формули** для розрахунку кількості заявок у черзі та середнього часу обслуговування.
- Такий підхід дозволив оцінити завантаження вузлів, виявити вузькі місця та визначити оптимальні параметри балансування запитів у системі.

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} & p_{16} & p_{17} \\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} & p_{26} & p_{27} \\ p_{31} & p_{32} & p_{33} & p_{34} & p_{35} & p_{36} & p_{37} \\ p_{41} & p_{42} & p_{43} & p_{44} & p_{45} & p_{46} & p_{47} \\ p_{51} & p_{52} & p_{53} & p_{54} & p_{55} & p_{56} & p_{57} \\ p_{61} & p_{62} & p_{63} & p_{64} & p_{65} & p_{66} & p_{67} \\ p_{71} & p_{72} & p_{73} & p_{74} & p_{75} & p_{76} & p_{77} \end{pmatrix}$$

Стохастична матриця переходів



Мережа масового обслуговування, що описує роботу трирівневої хмарної програми



СеМО хмарної програми

РЕЗУЛЬТАТИ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ

Продуктивність процесорів оцінюється через показник **MIPS (Million Instructions Per Second)**, який відображає кількість інструкцій, що можуть бути оброблені за секунду. Цей параметр є базовим для моделювання продуктивності телекомунікаційних систем та визначає здатність вузлів обробляти запити в реальному часі.

У розробленій моделі кожен вузол (віртуальна машина) має власну інтенсивність обслуговування, що залежить від тактової частоти, кількості ядер і типу архітектури. Це дає змогу оцінити різницю у швидкодії та визначити **вузькі місця системи**.

На основі експериментальних даних розраховано:

- **інтенсивність надходження запитів (Λ_i)**;
- **пропускну здатність вузлів (μ_i)**;
- **завантаженість ресурсів (p_i)**;
- **час очікування в черзі**.

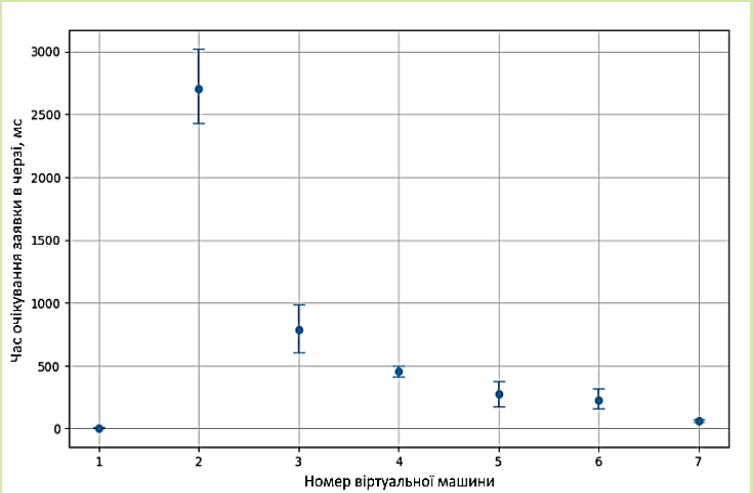
Результати представлені у таблицях та графіках, що демонструють вплив навантаження на продуктивність системи та дозволяють визначити оптимальний розподіл запитів між вузлами.

Інтенсивність надходження запитів до кожного вузла

Λ_i	Розрахункова формула	Значення, 1/мс
Λ_1	λ	180
Λ_2	$\Lambda_1 r_{12}$	27
Λ_3	$\Lambda_1 r_{13}$	27
Λ_4	$\Lambda_1 r_{14}$	36
Λ_5	$\Lambda_1 r_{15}$	36
Λ_6	$\Lambda_1 r_{16}$	54
Λ_7	$\Lambda_2 r_{27} + \Lambda_3 r_{37} + \Lambda_4 r_{47} + \Lambda_5 r_{57} + \Lambda_6 r_{67}$	54

Завантаженість вузла

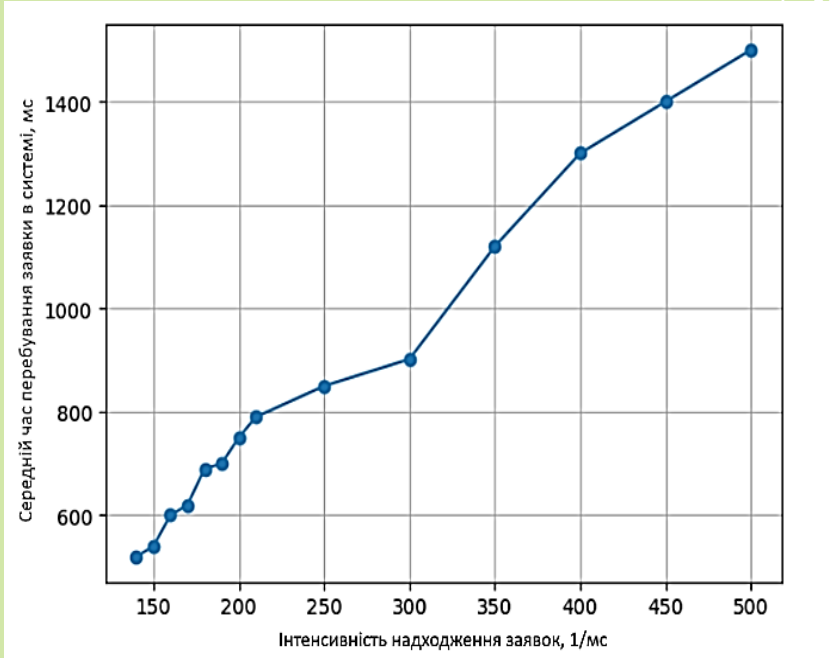
Інтенсивність використання виділених ресурсів p_i	Розрахункова формула	Значення (у частках)
p_1	$\Lambda_1 / \mu_1 = \lambda / \mu_1$	0.81
p_2	Λ_2 / μ_2	0.72
p_3	Λ_3 / μ_3	0.74
p_4	Λ_4 / μ_4	0.76
p_5	Λ_5 / μ_5	0.78
p_6	Λ_6 / μ_6	0.80
p_7	Λ_7 / μ_7	0.83



Час очікування заявки в черзі в *i*-му вузлі

Час очікування заявки в *i*-ій черзі

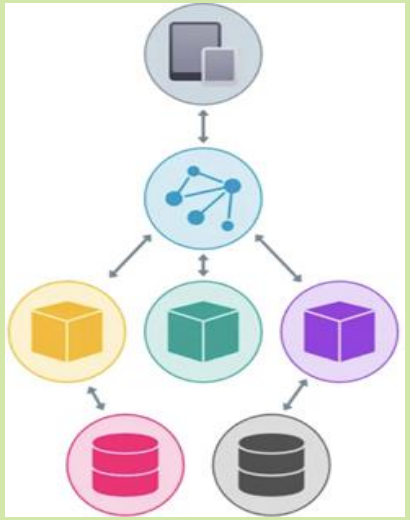
Інтенсивність використання виділених ресурсів %	Середній час очікування в черзі, мс	Мінімальний час очікування в черзі, мс	Максимальний час очікування в черзі, мс
80	6.370	6.173	6.568
71	4115.760	3241.148	4990.493
73	803.147	701.255	905.242
75	434.460	401.364	467.566
77	277.480	249.982	304.969
79	236.522	219.649	253.417
81	69.620	69.678	73.330



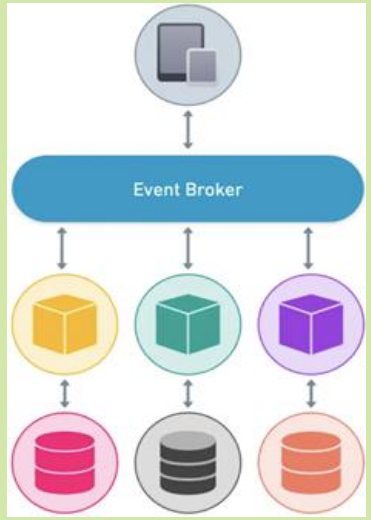
Зміна часу перебування заявки у системі зі збільшенням інтенсивності надходження заявок

АРХІТЕКТУРНІ СТИЛІ

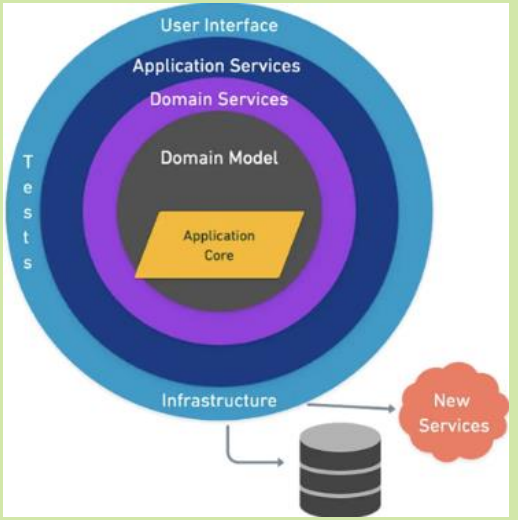
- Архітектурний стиль визначає структуру системи та спосіб взаємодії її компонентів. Вибір стилю залежить від вимог до продуктивності, надійності, масштабованості та типу бізнес-процесів. На практиці стилі поєднуються для забезпечення гнучкості та ефективності.
- **Паралельний стиль** дозволяє виконувати кілька завдань одночасно, що підвищує продуктивність у системах реального часу та високообчислювальних середовищах. Використовуються патерни «Оркестратор» (централізоване управління) і «Хореографія» (децентралізована взаємодія).
- **Багатошаровий стиль (n-tier)** поділяє систему на шари представлення, логіки й даних, спрощуючи розробку й масштабування.
- **Розподілені системи** розміщують компоненти на різних вузлах, забезпечуючи стійкість до збоїв та горизонтальне масштабування.



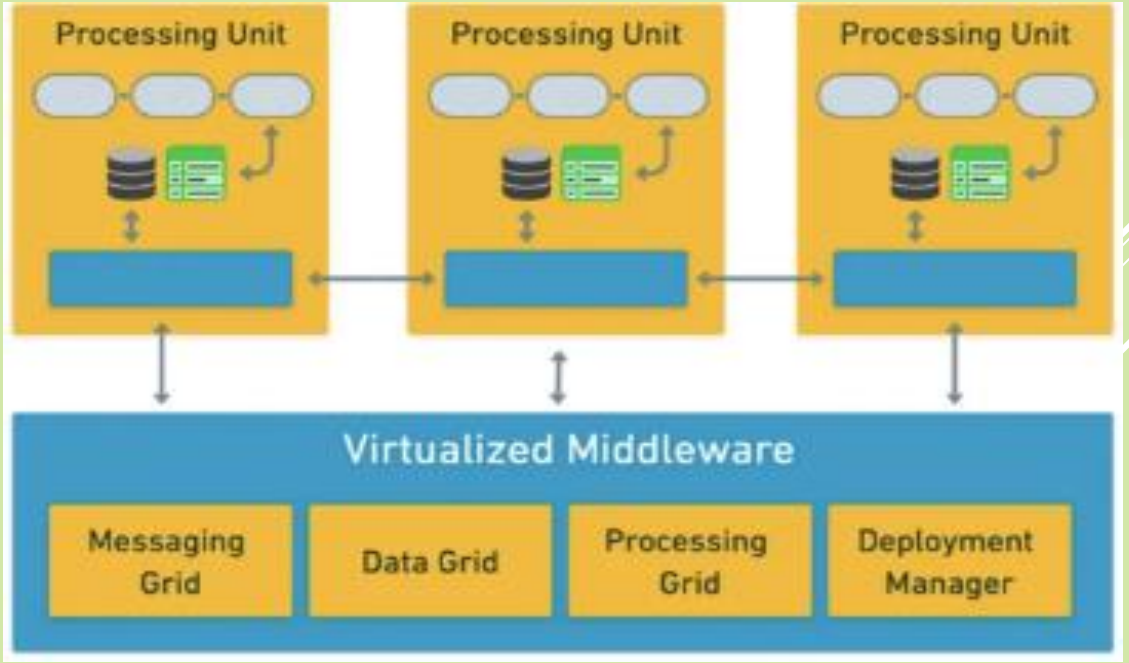
Патерн «Оркестратор»



Патерн «Хореографія»



Багатошаровий патерн



Стиль розподілених систем

КЛАСИФІКАЦІЯ ДОДАТКІВ

У роботі було розглянуто вибір архітектури залежно від типу програмного продукту. Для формалізації цього підходу введено поняття **«категорії застосунку»**, що дозволяє класифікувати інформаційні системи за їхніми вимогами до продуктивності, реактивності, надійності та безпеки.
Виділено **п’ять основних категорій**:
1.транзакційні системи (білінг, банківські процесинги);
2.системи реального часу (керування транспортом, телеком-мережами);
3.аналітичні системи (BI, прогнозування, Big Data);
4.колаборативні системи (месенджери, пошта, документообмін);
5.корпоративні системи (ERP, CRM, HRM).
Для порівняння вимог запропоновано **матрицю категорій**, що дозволяє визначити оптимальний архітектурний стиль (SOA, мікросервіси, моноліт, оркестрація).
Ключовий принцип: **не система пристосовується до архітектури — архітектура обирається під застосунок**.

Матриця категорій

Вимога	1. Процеси з великою кількістю транзакцій	2. Операції в реальному часі	3. Аналітичні процеси та бізнес-аналітика	4. Спільна робота (колаборація)	5. Корпоративні (сервісні)
Стратегічні потреби	Надання послуг	Час реакції системи	Здатність пояснювати результати	Поширення знань	Надійність, низька вартість з точки зору ТК
Бізнес-вимоги	Обслуговування клієнтів; Зменшення витрат; Робота 24/7; Цілісність даних	Економічність і безпека; Робота 24/7/365	Підвищення ефективності та продуктивності; Наглядність подання інформації	Швидкість оновлення знань; Повторне використання знань	Економічність; Покращення бізнес-процесів
Відмінні характеристики	Низька вартість (на одну транзакцію); Надійність; Масштабованість; Продуктивність; Резервування	Сканування та фільтрація потоків даних; Пріоритизація запитів; Надійність; Публікація та підписка на дані	Механізми аналітики; Потужність обробки даних; Сховища даних	Простота використання; Надійність; Висока пропускна здатність; Обмін даними «по горизонталі»	Стандартні процеси; Кандидати на аутсорсинг
Інтегруючі технології	Системи інтеграції корпоративних застосунків	Спеціально розроблений програмний код	Сховища даних	Спільно використовувані дані та обмін інформацією	Стандартні інтерфейси (API), XML

ВИСНОВКИ

- У роботі досліджено теоретичні та практичні аспекти оптимізації архітектури багаторівневих телекомунікаційних систем для підвищення їх продуктивності, надійності та рівня QoS.
- Проаналізовано сучасні архітектурні стилі (монолітний, SOA, мікросервісний, розподілений, паралельний) та визначено умови їх застосування залежно від типу системи й навантаження.
- Для кількісної оцінки ефективності застосовано моделі масового обслуговування M/M/1, що дозволило визначити оптимальне завантаження системи та вплив на затримки.
- Розглянуто роль паралельних обчислень (DLP, TLP, SIMT, MIMD) та гетерогенних архітектур CPU/GPU у прискоренні обробки даних у реальному часі.
- Показано, що модульність, балансування навантаження та сервісний підхід є ключем до створення адаптивних архітектур, здатних реалізувати концепції 5G/6G, SDN і NFV.

ДЯКУЮ ЗА УВАГУ!

Протокол аналізу звіту подібності науковим керівником

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

Автор: Владислав ЗАЙЦЕВ

Назва: ОПТИМІЗАЦІЯ АРХІТЕКТУРИ БАГАТОРІВНЕВИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ

Координатор: Юрій МЕЛЬНИК

Підрозділ: ННІТ

Коефіцієнт подібності 1:0.3

Коефіцієнт подібності 2:0

Тривога: 7

Після аналізу Звіту подібності констатую наступне:

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

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

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

…… ……

Дата *Підпис Наукового керівника*