МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
“ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кваліфікаційна наукова
праця на правах рукопису
УДК 621.391

Плесканка Мар’яна Вікторівна

ДИСЕРТАЦІЯ

ПІДВИЩЕННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ У МЕРЕЖІ ДОСТАВКИ КОНТЕНТУ

05.12.02 – телекомунікаційні системи та мережі
(шифр і назва спеціальності)

05 «Технічні науки»
(галузь знань)

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

_____
(підпис, ініціали та прізвище здобувача)

Подається на здобуття наукового ступеня
кандидата технічних наук

Науковий керівник–
Кирик Мар’ян Іванович,
д.т.н., професор

Ідентичність всіх примірників дисертації
ЗАСВІДЧУЮ:
Вчений секретар спеціалізованої
вченої ради /**М. І. Бешлей**/

Львів -2025

АНОТАЦІЯ

Плесканка М.В. Підвищення якості обслуговування у мережі доставки контенту. - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.12.02 - телекомунікаційні системи та мережі. - Національний університет «Львівська політехніка» Міністерства освіти і науки України, Львів, 2025.

У першому розділі **«Аналіз методів міграції сервісів від монолітної до мікросервісної архітектури»** розглянуто основні принципи та підходи, що стосуються використання монолітної та мікросервісної архітектури, наведено особливості роботи, переваги та недоліки кожної із них. Розглянуто сучасні методи щодо створення та управління інфраструктурою не шляхом фізичного налаштування обладнання чи використання інтерактивних інструментів налаштування, а за допомогою файлів конфігурації та підходів Інфраструктура як код IaC (Infrastructure as Code). Проведено грунтовний аналіз літератури на тему динамічного балансування навантаження, кешування даних, з метою визначення поточного стану стратегій, що використовуються, включаючи їх переваги, недоліки, труднощі впровадження, а також їхній вплив на ефективність роботи CDN мереж.

У другому розділі **«Методи та алгоритми балансування та розподіленої обробки запитів в CDN мережі»** досліджено основні архітектурні принципи роботи та маршрутизації запитів CDN мережі. Запропоновано інтегральний (комплексний) ключ кешування, що формується шляхом поєднання кількох параметрів, та може використовуватись для різних типів мережевого трафіку(перегляд веб-сторінок, інтерактивного цифрового телебачення, мультимедійні дані). Запропоновано метод обробки запитів, що потребують значних обчислювальних ресурсів, що який забезпечує можливість розподіленої

обробки даних в точках присутності CDN мережі та дає змогу ефективніше використовувати кешовані дані, зменшити навантаження на кореневий сервер, а також час відповіді на запити кінцевого користувача. Удосконалено метод балансування навантаження в граничних локаціях CDN мережі, який враховує значення інтегрального ключа кешування, наявність контенту на граничному сервері та стан функціонування доступних серверів.

У третьому розділі **«Модель оцінки ефективності роботи систем обробки мережевого трафіку у точках присутності CND мереж»** представлено комплексну модель масового обслуговування з трьома послідовно з'єднаними системами, яка використовується для характеристики процесу обслуговування запитів у хмарних сервісах із динамічним виділенням ресурсів. В якості математичної моделі першої підсистеми, а саме балансувальника навантаження, запропоновано систему масового обслуговування класу $M/M/1/k$ - система масового обслуговування з обмеженою довжиною черги k та 1 обслуговуючим пристроєм. Вузол обробки, та кореневий сервер можуть паралельно обробляти кілька завдань, саме тому розглядається як система масового обслуговування $M/M/$n. На основі математичного моделювання проведено оцінку параметрів якості обслуговування, а саме ймовірності миттєвого обслуговування, довжини черги обслуговування та часу відповіді в залежності від інтенсивності надходження запитів та продуктивності вузлів обслуговування. В результаті моделювання отримано графічні залежності, які показують, що системи обслуговування даних працюють досить добре при середніх навантаженнях вузлів обробки. Було встановлено, що система, яка складається з більшої кількості менш продуктивних вузлів обробки даних, працюватиме ефективніше, ніж система з одним вузлом вищої продуктивності, за умови, що особливості трафіку не вимагають складних обчислень і значних затрат процесорного часу.

У четвертому розділі **«Реалізація методу адаптивного розгортання мікросервісу в точці присутності CDN мережі»** запропоновано метод адаптивного розгортання мікросервісів для обробки динамічних даних у режимі реального часу в точках присутності CDN мереж. Даний метод забезпечує дублювання бізнес-логіки сервісу з кореневого сервера на граничні сервери, здійснює аналіз параметрів, що визначають якість послуги в режимі реального часу та адаптивно розподіляє запити на основі оцінювання рівня завантаженості граничних серверів. Подано графічне відображення результатів моделювання, які показують, що запропонований метод є досить ефективним, оскільки при збільшенні кількості запитів у два рази дає змогу забезпечити необхідні значення якості обслуговування QoS. Приведено приклад практичного використання методу адаптивного розгортання мікросервісів та результати його роботи на прикладі хмарного сервісу Google Cloud Run.

Ключові слова: CDN мережа, граничний сервер, мікросервісна архітектура, балансування навантаження, ефективність кешування даних, хмарні обчислення, кореневий сервер, маршрутизація запитів, хмарні сервіси, розподілена обробка даних.

Список публікацій здобувача:

<u>Наукові праці, в яких опубліковані основні результати дисертації</u>

1. M. Kyryk, N. Pleskanka, M. Pitsyk, V. Kyryk, “Infrastructure as code and microservices for intent-based cloud networking,” *Lecture Notes in Electrical Engineering: Future intent-based networking. On the QoS robust and energy efficient heterogeneous software defined networks*, vol. 831, pp. 51-68, 2022.

2. М.В. Плесканка, “Покращення параметрів якості обслуговування QoS в CDN мережі за рахунок використання модуля Edge Compute,” *Інфокомунікаційні технології та електронна інженерія,* Випуск.3, № 2, с. 64-73, 2023.

3. Н.М. Плесканка, М.В. Плесканка, Т.С. Слободзян,Б.М. Марко, “Аналіз ефективності використання мікросервісів при розробці web додатків,” *Комп'ютерні системи проектування. Теорія і практика.* Випуск 6, № 2, с. 146-157, 2024.

4. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, “Аналіз роботи методу оптимізованого кешування даних в мережі доставки,” *Проблеми телекомунікацій.* № 1 (22), с. 43-55, 2018.

5. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, “Дослідження ефективності використання мережі CDN,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* № 885, с. 31-40, 2017.

6. М.І. Кирик, В.Б. Янишин, М.В. Плесканка, “Оцінка ефективності методів спектральної мобільності у когнітивних радіомережах,”. *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* №849, с. 194 - 202, 2016.

7. Т.А. Максимюк,О.М. Яремко, М.В. Піцик, “Моделі конвергенції гетерогенних мереж мобільного зв’язку 5-го покоління на основі технології D2D,”. *Телекомунікаційні та інформаційні технології, Київ, ДУТ,* № 3, с. 91-102, 2016.

8. М.В. Кайдан, В.С. Андрущак, М.В. Піцик, В.З. Пашкевич, “Аналіз енергетичного балансу оптичної транспортної мережі з урахуванням технологічних і архітектурних підходів,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* №818, с. 120-129, 2015.

<u>Наукові праці, які засвідчують апробацію матеріалів дисертації</u>

9. M. Kyryk, O. Tymchenko, N. Pleskanka, and M. Pleskanka, “Methods and process of service migration from monolithic architecture to

microservices," in *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2022.

10. M. Kyryk, N. Pleskanka, M. Pleskanka, "Adaptive Edge Compute module in CDN networks," *Advanced Information and Communication Technologies: Proceedings of the 5th IEEE International Conference, Lviv, Ukraine,* 2023

11. M. Kyryk, N. Pleskanka, M. Pleskanka, "Dynamic Data Processing on Edge Locations of CDN Network," in *2024 IEEE 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET),* 2024.

12. M. Kyryk, N. Pleskanka, M. Pleskanka, and P. Nykonchuk, "Load balancing method in edge computing," in *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET),* 2020.

13. M. Kyryk, N. Pleskanka, and M. Pleskanka, "The analysis of the optimal data distribution method at the content delivery network," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2019.

14. M. Kyryk, N. Pleskanka, and M. Pleskanka, "Analysis of the technologies and methodologies of data transmission in distributed information systems," in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, 2018.

15. M. Kyryk, M. Pleskanka, and N. Pleskanka, "The efficiency and productivity of the CDNs," in *2017 2nd International Conference on Advanced Information and Communication Technologies (AICT)*, 2017.

16. M. Kyryk, N. Pleskanka, and M. Pitsyk, "QoS mechanism in content delivery network," in *2016 13th International Conference on Modern Problems*

of Radio Engineering, Telecommunications and Computer Science (TCSET), 2016.

17. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Content delivery network usage monitoring,” in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017.

18. M. Kaidan, V. Andrushchak, and M. Pitsyk, “Calculation model of energy efficiency in optical transport networks,” in *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, 2015.

19. Y. Pyrih, M. Kaidan, I. Tchaikovskyi, and M. Pleskanka, “Research of genetic algorithms for increasing the efficiency of data routing,” in *2019 3rd International Conference on Advanced Information and Communications Technologies (AICT)*, 2019.

ABSTRACT

Pleskanka M.V. Improving the quality of service in the content delivery network. - Qualifying scientific work on manuscript rights.

Dissertation for obtaining the scientific degree of Doctor of Philosophy in specialty 05.12.02 - telecommunication systems and networks. - Lviv Polytechnic National University of the Ministry of Education and Science of Ukraine, Lviv, 2025.

In the first chapter **"Analysis of services migration methods from monolithic to microservice architecture"** the main principles and approaches related to the use of monolithic and microservice architecture are considered, the specifics of operation, advantages and disadvantages of each of them are given. It explores modern methods of creating and managing infrastructure, not through physical hardware configuration or interactive setup tools, but using configuration files and Infrastructure as Code (IaC) approaches. An extensive review of literature on dynamic load balancing and data caching has been conducted to determine the current state of strategies employed, including their benefits, drawbacks, implementation challenges, and their impact on the efficiency of CDN networks.

In the second chapter **"Methods and algorithms of balancing and distributed requests processing in the CDN network"** the main architectural principles of operation and request routing in CDN networks are considered. It proposes an integral (comprehensive) caching key formed by combining multiple parameters, which can be used for different types of network traffic (e.g., web browsing, interactive digital television, multimedia data). A method for processing requests that require significant computational resources is suggested, enabling distributed data processing at CDN points of presence, improving the use of cached data, reducing the load on the root server, and decreasing response time for end user. Additionally, a load balancing method for edge locations of CDN networks is improved, taking into account the value of the integral caching

key, the availability of content on edge servers, and the operational status of available servers.

The third section **"A model for evaluating the performance of network traffic processing systems at the CND networks points of presence"** presents a comprehensive mass-service model with three sequentially connected systems, used to characterize the request handling process in cloud services with dynamic resource allocation. The first subsystem, the load balancer, is mathematically modeled as an $M/M/1/k$ queue system, which has a limited queue length k and a 1 servicing device. The processing node and origin server can handle multiple tasks in parallel and is thus modeled as an $M/M/$n system. Based on mathematical modeling, the quality of service parameters, such as the probability of immediate service, queue length, and response time, are evaluated depending on the intensity of incoming requests and the performance of processing nodes. Graphical dependencies obtained from the modeling show that data service systems perform well under moderate processing node loads. It was determined, that a system, consisting of a larger number of less productive processing nodes, is more efficient, than one with a single high-performance node, provided that the traffic characteristics do not require complex calculations or significant processor time.

The fourth chapter **"Development of the method of adaptive microservice deployment at CDN network point of presence"** chapter proposes a method for adaptively deploying microservices to process dynamic real-time data at CDN network points of presence. This method duplicates the business logic of the service from the root server to edge servers, analyzes parameters determining service quality in real-time, and adaptively distributes requests based on the load levels of edge servers. Graphical representations of the modeling results demonstrate that the proposed method is highly effective. For instance, doubling the number of requests still ensures the required quality of service (QoS) values. An example of the practical application of the adaptive

microservice deployment method and the results of its implementation are provided, using the Google Cloud Run cloud service as a case study.

Keywords: CDN network, edge server, microservice framework, load balancing, data caching efficiency, cloud computing, root server, request routing, cloud services, distributed data processing.

List of publications of the acquirer:

Proceedings where basic scientific results of thesis were published:

1. M. Kyryk, N. Pleskanka, M. Pitsyk, V. Kyryk, “Infrastructure as code and microservices for intent-based cloud networking,” *Lecture Notes in Electrical Engineering: Future intent-based networking. On the QoS robust and energy efficient heterogeneous software defined networks*, vol. 831, pp. 51–68, 2022.

2. M. Pleskanka, “The quality of service parameters, QoS improvement in CDN network with edge compute module”. *Information and communication technologies, electronic engineering*. Vol. 3, No.2, pp.64-73, 2023.

3. N. Pleskanka, M. Pleskanka, T. Slobodzian, B. Marko, “Analysis of the microservices efficiency using in the web applications development,” *Computer design system. Theory and practice.* Vol. 6, No. 2, pp. 146-157, 2024.

4. M. Kyryk., N. Pleskanka M. Pleskanka, “Analysis of the method of optimized data caching in the delivery network, *Problems of telecommunications*. № 1 (22), pp. 43–55, 2018.

5. M. Kyryk, N. Pleskanka. M. Pleskanka, “Study of the efficiency of using the CDN network,” *Herald of Lviv Polytechnic National University. Radioelectronics and Telecommunications,* № 885, pp. 31–40, 2017.

6. M. Kyryk, V. Yanishin, M. Pleskanka, “Evaluation of the effectiveness of spectral mobility methods in cognitive radio networks,” *Herald of Lviv Polytechnic National University. Radioelectronics and Telecommunications*, № 849, pp. 194–202, 2016.

7. T. Maksymyuk, O. Yaremko., M. Pytsik, “Convergence models of heterogeneous mobile communication networks of the 5th generation based on D2D technology,” *Telecommunications and Information Technologies, Kyiv, DUT*, № 3, pp. 91-102, 2016.

8. M. Kaidan, V. Andrushchak, M. Pitsyk, V. Pashkevich, “Analysis of the energy balance of the optical transport network taking into account technological and architectural approaches,” *Herald of Lviv Polytechnic National University. Radioelectronics and Telecommunications*, №818, pp.120-129, 2015.

Scientific works certifying the approval of the dissertation materials:

9. M. Kyryk, O. Tymchenko, N. Pleskanka, and M. Pleskanka, “Methods and process of service migration from monolithic architecture to microservices,” in *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2022.

10. M. Kyryk, N. Pleskanka, M. Pleskanka, “Adaptive Edge Compute module in CDN networks,” *Advanced Information and Communication Technologies: Proceedings of the 5th IEEE International Conference, Lviv, Ukraine,* 2023

11. M. Kyryk, N. Pleskanka, M. Pleskanka, “Dynamic Data Processing on Edge Locations of CDN Network,” in *2024 IEEE 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET),* 2024.

12. M. Kyryk, N. Pleskanka, M. Pleskanka, and P. Nykonchuk, “Load balancing method in edge computing,” in *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET),* 2020.

13. M. Kyryk, N. Pleskanka, and M. Pleskanka, “The analysis of the optimal data distribution method at the content delivery network,” in *2019 IEEE*

15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), 2019.

14. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Analysis of the technologies and methodologies of data transmission in distributed information systems,” in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, 2018.

15. M. Kyryk, M. Pleskanka, and N. Pleskanka, “The efficiency and productivity of the CDNs,” in *2017 2nd International Conference on Advanced Information and Communication Technologies (AICT)*, 2017.

16. M. Kyryk, N. Pleskanka, and M. Pitsyk, “QoS mechanism in content delivery network,” in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, 2016.

17. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Content delivery network usage monitoring,” in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017.

18. M. Kaidan, V. Andrushchak, and M. Pitsyk, “Calculation model of energy efficiency in optical transport networks,” in *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, 2015.

19. Y. Pyrih, M. Kaidan, I. Tchaikovskyi, and M. Pleskanka, “Research of genetic algorithms for increasing the efficiency of data routing,” in *2019 3rd International Conference on Advanced Information and Communications Technologies (AICT)*, 2019.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IaaS (скор. англ. Infrastructure as a Service) – інфраструктура як послуга.

PaaS (скор. англ. Platform as a Service) – платформа як послуга.

SaaS (скор. англ. Software as a Service) – програмне забезпечення як послуга.

GCP (скор. англ. Google Cloud Platform) – хмарна платформа Google.

AWS (скор. англ. Amazon Web Services) – веб сервіси Amazon.

API (скор. англ. Application Programming Interface) – інтерфейс програмування додатків.

CDN (скор. англ. Content Delivery Network) – мережа доставки контенту.

SOA - (скор. англ. Service-Oriented Architecture) – сервісно-орієнтована архітектура.

REST (скор. англ. Representational State Transfer) – передача репрезентативного стану.

HTTP (скор. англ. Hypertext Transfer Protocol) – протокол передачі гіпертексту. gRPC (скор. англ. Google Remote Procedure Call) – система віддаленого виклику процедур.

AMQP (скор. англ. Advanced Message Queuing Protocol) – розширений протокол черги повідомлень.

ETL (скор. англ. Extract, Transform, Load) – екстракція, трансформація, завантаження.

QoS- (скор. англ. Quality of Service) – якість обслуговування.

GCR-(скор. англ. Google Cloud Registry) – сховище для збереження образів.

LB-(скор. англ. Load Balancer) – балансувальник навантаження.

FBM-(Fractional Brown Motion) - фрактальний броунівських рух.

FGN - (Fraction Gaussian Noise) - Фрактальний гаусівський шум.

AR-(Autoregressive Models) - Авторегресійні моделі.

FPP-(Fractal Point Process) - Фрактальні точкові процеси.

APM-(скор. англ. Aplication Monitoring) – система моніторингу додатків та сервісів.

EC-(скор. англ. Edge Compute) – обробка даних на граничному сервері розподіленої мережі доставки контенту та послуг.

IoT-(скор. англ. Internet of Things) **-** інтернет речей.

CI/CD-(скор. англ. Continuous Integration and Continuous Delivery) – неперервна інтеграція та неперервна доставка сервісів.

DNS - (скор. англ. Domain Name System) **-** система доменних імен.

CA - (скор. англ. Cache Affinity) – ефективність кешування даних.

PoP - (скор. англ. Point of Present) – локація доступності мережі доставки даних

ES - (скор. англ. Edge Server) – граничний (кешуючий) сервер в мережі доставки даних

Cloud Run - керована платформа обчислення

STOMP - (англ. Simple (or Streaming) Text Oriented Messaging Protocol) — це протокол, заснований на фреймах, змодельованих на основі HTTP

Apache JMeter - інструмент для проведення навантажувального тестування

ВСТУП

Актуальність теми. Сучасний рівень розвитку інформаційних технологій характеризується інтенсивним впровадженням нових послуг та платформ. Досить поширеними в наш час стали Cloud-мережі та технологія CDN (Content Delivery Network). Все це призводить до підвищення вимог до каналів зв'язку та обслуговуючих пристроїв, які виконують обробку трафіку, щоб забезпечити необхідну якість послуг QoS (Quality of Service), оскільки відповідно до вимог часу, сучасні мережі передачі даних повинні будуватися як високонадійні системи, здатні забезпечити необхідні показники якості обслуговування.

Умови, які склались в сучасному світі, а це для прикладу COVID, війна та багато інших чинників диктують свої правила розвитку технологій та способів надання послуг. Для забезпечення різних вимог параметрів QoS мультисервісних послуг в системах передавання даних необхідно впроваджувати алгоритми та методи управління трафіком, які повинні враховувати особливості різних видів послуг, а також забезпечувати ефективне використання ресурсів мережі та вузлів обслуговування.

Окрему увагу слід приділити і новим архітектурним рішенням, які використовуються в наш час для розробки сучасних сервісів та додатків. Особливий акцент робиться на підходи, які дозволяють швидко наростити ресурси в моменти зростання навантаження, та так само швидко все згорнути при його відсутності. Не менш важливими є також методи, які дозволяють переносити обробку даних якнайближче до користувача. Якщо раніше такі підходи застосовувалися переважно для статичного контенту, такого як відео та аудіо трафік чи різного роду зображення, то зараз цього вже недостатньо. У сучасних CDN значну частку трафіку становить динамічний контент, вміст якого може змінюватися залежно від часу, місця, дій користувача та інших факторів.

Варто зазначити, що досить багато досліджень було зосереджено саме на покращення якості доставки як статичних так і динамічних даних в CDN мережах. Серед провідних наукових досліджень в даному напрямку, можна виділити авторів: Nakanishi K., Suzuki F., Dong Y., Gupta P., Goya M., Mijumbi R., Shitole A., Lugones D., Л. Глоба, О. Лемешко, С. Толюпа, В. Безрук, Б. Жураковський. У більшості розглянутих робіт автори досліджують методи балансування навантаження та ефективності кешування, проте недостатньо уваги приділено критеріям, за якими можна розподіляти типи запитів між граничними серверами, а також аналітичним даним, що прогнозують динаміку поведінки користувачів. Щодо обробки динамічних даних, не було враховано можливості перенесення самого процесу обробки та формування даних якомога ближче до локації користувача з метою ефективного використання обчислювальної інфраструктури.

Таким чином, зростання обсягів та різноманітності статичних і динамічних даних у мережах доставки контенту обумовлює необхідність розв’язання науково-практичного завдання підвищення якості обслуговування користувачів в умовах обмеженої обчислювальної інфраструктури шляхом розроблення методів і алгоритмів обробки запитів, що потребують значних обчислювальних ресурсів, кешування даних, реалізації балансування навантаження та адаптивного розгортання мікросервісів у точках присутності CDN мереж.

Зв’язок роботи з науковими програмами, планами, темами. Тематика дисертаційного дослідження виконувалась у відповідності до наукового напряму кафедри інформаційно-комунікаційних технологій (кафедри телекомунікацій) Національного університету «Львівська політехніка» - «Інфокомунікаційні системи та мережі», в межах низки держбюджетних науково-дослідних робіт: «Методи побудови та моделі інформаційно-телекомунікаційної інфраструктури на основі SDN-

технологій для систем електронного урядування» (ДБ/SDN), (№ держреєстрації 0115U000444, (2015-2016 рр.), «Методи побудови гетерогенних інформаційно-комунікаційних систем для розгортання програмно-конфігурованих мереж 5G подвійного використання» (№ держреєстрації 0117U004449, (2017–2018 рр.)).

Мета і завдання досліджень. Метою дисертаційної роботи є підвищення якості обслуговування у мережах доставки контенту шляхом розробки методів та алгоритмів ефективного кешування, балансування трафіку та адаптивного розгортання мікросервісів у точках присутності CDN мереж.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Провести аналіз існуючих підходів та архітектурних рішень, які застосовуються під час розробки та впровадження нових сервісів та послуг в CDN мережах.
2. Удосконалити метод обробки запитів, що потребують значних обчислювальних ресурсів у точках присутності CDN мережі.
3. Запропонувати інтегральний ключ кешування для ефективного кешування статичних даних в мережах доставки контенту.
4. Удосконалити метод балансування навантаження у мережах доставки контенту для забезпечення ефективного використання кеш-пам'яті та обчислювальних ресурсів граничних серверів та мережевих пристроїв.
5. Розробити метод адаптивного розгортання мікросервісів для обробки динамічних даних у режимі реального часу в точках присутності CDN-мереж.
6. Провести експериментальне дослідження для оцінювання ефективності запропонованих рішень на основі розробленого прототипу мережі доставки контенту.

Об’єкт дослідження – процеси обробки даних в точках присутності CDN мережі.

Предмет дослідження - методи та алгоритми обробки статичних та динамічних даних в мережах доставки контенту**.**

Методи дослідження. Дослідження виконано на основі використання методів об’єктно-орієнтованого проєктування, теорії систем масового обслуговування, паралельних та розподілених обчислень, математичного та комп’ютерного моделювання, а також на основі результатів експериментів.

Наукова новизна отриманих результатів.

1. Набув подальшого розвитку метод обробки запитів, що потребують значних обчислювальних ресурсів, який, на відміну від раніше відомих, для розподіленої обробки запитів враховує дані аналітики щодо популярності певних ресурсів (веб-сторінок, мультимедійного контенту) серед списку найбільш запитуваних, що дало змогу забезпечити ефективне використання кешованих даних, а також зменшити навантаження на кореневий сервер та час відповіді на запити кінцевого користувача.

2. Удосконалено метод балансування трафіку у точках присутності CDN мережі, який, на відміну від відомих, враховує значення інтегрального ключа кешування сформованого на основі розробленого методу обробки запитів, локацію клієнта, наявність контенту на граничному сервері та стан функціонування доступних серверів, що дало змогу підвищити якість обслуговування в мережах доставки контенту.

3. Вперше запропоновано метод адаптивного розгортання мікросервісів для обробки динамічних даних у режимі реального часу в точках присутності CDN-мереж, який, на відміну від відомих, частково дублює бізнес-логіку сервісу з кореневого сервера, що надається кінцевим користувачам, здійснює в режимі реального часу аналіз параметрів, які визначають якість послуги, та адаптивно розподіляє запити на основі оцінювання рівня завантаженості граничних серверів для забезпечення необхідної якості обслуговування.

Практичне значення одержаних результатів полягає у можливості їх безпосереднього застосування в існуючих мережах доставки контенту та провайдерів хмарних сервісів:

1. Удосконалений метод обробки запитів статичних даних у мережах доставки контенту дав змогу покращити якість обслуговування користувачів шляхом зменшення часу відповіді для кінцевого користувача до 9%, та на 10% завантаженість кореневого сервера.

2. Комплексне використання інтегрального ключа кешування, методу обробки запитів та балансування трафіку у точках присутності CDN мережі дало змогу підвищити коефіцієнт ефективності кешування на 30 %, а також зменшити час відповіді на запити кінцевого користувача на 40%.

3. Розроблений прототип мережі доставки контенту для передавання статичних та динамічних даних дав змогу підтвердити на практиці ефективність розробленого методу адаптивного розгортання мікросервісів для обробки динамічних даних у режимі реального часу в точках присутності CDN-мереж, а саме забезпечити необхідну якість обслуговування в умовах обмежених ресурсів кореневого сервера. Результати експериментального дослідження, проведеного для оцінки якості надання певного типу сервісу кінцевому користувачеві за умов високого навантаження системи, показали, що час відповіді на запити користувача становить 25 мс, що на 7% швидше порівняно з часом відповіді від кореневого сервера для клієнтів у тій самій локації.

Основні результати дисертаційної роботи використано і впроваджено в телекомунікаційних корпоративних мережах ТОВ “Телекомунікаційна компанія”, ТОВ ВТФ “Контех”, ТОВ “МаксіТех”, що підтверджено актами впровадження, а також у навчальному процесі кафедри інформаційно-комунікаційних технологій Національного університету «Львівська політехніка».

Особистий внесок здобувача. Усі результати наукових, теоретичних і практичних досліджень, викладені в дисертації, автор одержав особисто. У працях, опублікованих у співавторстві, дисертантові належать: у роботах [1,3,9] - представлено новий механізм розгортання інфраструктури, створення мікросервісів для майбутніх хмарних мереж, процес міграції монолітної програми, [2,10 - 12] - запропоновано метод обробки запитів, що вимагають значних обчислювальних ресурсів, що враховує дані аналітики та інтегральний ключ кешування для забезпечення високої ефективності кешування даних та використання ресурсів мережі доставки; розроблено метод адаптивного створення мікросервісу у граничній локації CDN мережі, який призначений для забезпечення необхідної якості обслуговування, [4,13,14] - запропоновано використання нового методу оптимізованого кешування даних як складової площини балансування навантаження, [5,15-17] - представлені основні методи та технології розповсюдження та доставки даних, виходячи з цільових функцій, [6] - розглянуто методи спектральної мобільності для когнітивного радіо, що надають змогу когнітивним користувачам перемикатися в частотні канали, [7] - запропоновано ряд моделей та методів для конвергенції гетерогенних мереж мобільного зв'язку п'ятого покоління з використанням технології D2D, [8,18] - представлено технічні та архітектурні підходи щодо підвищення енергоефективності телекомунікаційних мереж, [19] - представлено генетичний алгоритм для визначення маршруту передачі даних у мережах із змінною структурою**.**

Апробація результатів дисертації. Основні результати наукових досліджень доповідалися та обговорювались на всеукраїнських та міжнародних науково-технічних конференціях: Міжнародних науково-технічних конференціях «Сучасні проблеми радіоелектроніки, телекомунікацій, комп'ютерної інженерії» (м. Львів-Славське 2016, 2020, 2022 рр.); Problems of infocommunications science and technology, PIC S and T

2018: Proceedings of 5th International scientific-practical conference, Kharkiv, Ukraine, 9 - 12 October 2018.; Proceedings of the Second International Conference on Advanced Information and Communication Technologies (AICT'2017), Lviv, Ukraine:, Міжнародних науково-технічних конференціях «Досвід розробки та застосування приладо-технологічних САПР в мікроелектроніці» (Поляна-Свалява, 2017, 2019 рр.); 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET, Львів 2024).

Публікації. За результатами досліджень, які викладені у дисертаційній роботі, опубліковано 19 наукових праць, серед них 1 стаття у закордонному виданні, що входить до наукометричних баз даних Scopus [1], 7 статей у наукових фахових виданнях згідно з переліком МОН України [2-8] та 11 публікацій у збірниках праць міжнародних і всеукраїнських конференцій [9-19].

Структура та обсяг роботи. Робота складається з переліку умовних скорочень, вступу 4 розділів, висновків, списку використаних джерел і 3 додатків. Загальний обсяг роботи складає 199 сторінок друкарського тексту, із них 7 сторінок вступу, 160 сторінок основного тексту, 82 рисунки, 3 таблиці, список використаних джерел із 114 найменувань, 3 додатки на 13 сторінках**.**

.

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ МІГРАЦІЇ СЕРВІСІВ ВІД МОНОЛІТНОЇ ДО МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Під час розробки будь якого сервісу чи додатку багато уваги приділяється бізнес моделі, функціональним можливостям а також вибору технології та архітектури згідно якої він буде працювати. Вдала бізнес модель та підбір функціоналу дасть змогу успішно продавати сервіс та створювати конкуренцію на ринку. З іншої сторони не менш важливою є технологія та архітектура, адже саме вони будуть визначати наскільки якісно зроблений сервіс, наскільки стабільно він працює, та наскільки зрозумілий він для користувача і не тільки. І ці складові часто можна називати “внутрішня якість” сервісу.

Одна з основних особливостей внутрішньої якості коду полегшує розробнику зрозуміти як саме працює додаток. Якщо програмне забезпечення добре поділено на окремі складові модулі, розробнику не потрібно читати всі “100 - 500” рядків коду, він може швидко знайти кілька сотень рядків у кількох модулях. Якщо було докладено зусиль до чіткого іменування, також можна швидко зрозуміти, що роблять різні частини коду, не роздумуючи над деталями. Якщо код написаний структуровано та правильно, то в майбутньому це значно пришвидшить його оптимізацію та розширення. Для майбутніх розробників це значно спростить процес розробки та розширення функціональних можливостей.

На даний час питання щодо покращення внутрішньої якості сервісу та його архітектури здебільшого зосереджено на міграції монолітного типу архітектури на мікросервісну. В даній роботі детально розглянуто обидва типи архітектури, їхні відмінності та обгрунтування необіхдності переходу на мікросервіси.

1.1. Особливості організації монолітної архітектури

Термін "Моноліт" відноситься до традиційного стилю архітектури програм та сервісів, який не акцентує увагу на модульності як на принципі розбробки[1]. Спочатку "моноліт" застосовувався для опису великих програм, що функціонували автономно і ставали все складнішими в обслуговуванні з плином часу через постійне збільшення функціоналу протягом тривалого часу розробки.

Серед найбільш поширених підходів, які використовуються для монолітних програмних систем можна виділити однорівневий та модульний моноліт.

Однорівневий моноліт

Цей тип архітектури характеризується тим, що всі ключові компоненти – користувацький інтерфейс, бізнес-логіка та логіка доступу до даних – реалізуються в межах одного цілісного додатку. В результаті, навіть незначна зміна в будь-якій частині програми вимагає повного циклу тестування та повторного розгортання всієї системи. Така структура ускладнює масштабування, гальмує впровадження нових функцій і знижує гнучкість розробки. Однорівневі моноліти зазвичай не мають чітко окреслених внутрішніх модулів, які відповідали б конкретним бізнес-доменам чи функціональним зонам.

Модульний моноліт

На відміну від однорівневого, модульний моноліт зберігає цілісність єдиного додатку, проте логічно розділений на окремі модулі, кожен з яких виконує конкретну функцію або відповідає певному бізнес-домену. Незважаючи на логічну сегментацію, усі модулі модульного моноліту зазвичай працюють в одному процесі, розгортаються як єдине ціле, і можуть напряму взаємодіяти один з одним завдяки спільному простору і доступності коду. Це надає певну перевагу у швидкості взаємодії між

модулями, порівняно з мікросервісною архітектурою, однак все ще залишає деякі обмеження щодо незалежного масштабування.

Щодо доступу до даних – база даних у модульному моноліті зазвичай спільна для всіх модулів, і її структура часто нагадує ту, що використовується в одноріневому моноліті. Це означає, що хоча модулі можуть бути логічно ізольованими, зберігається можливість безпосереднього доступу до загальних таблиць, що іноді суперечить принципам повної модульності.

Загальна характеристика усіх форм монолітних застосувань полягає у наявності трьох архітектурних шарів: рівень інтерфейсу користувача, рівень бізнес-логіки та рівень бази даних. Проста ілюстрація трьох таких шарів наведена на діаграмі нижче (див. Рис. 1.1).

Рис.1.1. Модель монолітної архітектури

У цій системі перший шар відповідає за доступ до даних, забезпечуючи інтерфейс для отримання даних для наступного рівня нашої програми.

Другий шар відповідає за бізнес-логіку, що включає алгоритми, що визначають предметну область сервісу, який розробляється. Ця частина

системи є доволі гнучкою, оскільки ймовірно буде змінюватися в майбутньому. Вона використовує API доступу до даних для виконання своїх функцій.

Останній рівень - це рівень представлення, який забезпечує інтерфейс взаємодії з користувачем і виступає точкою входу даних у аплікацію.

На перших етапах розробки перших версій сервісу монолітна архітектура може здатися досить простою і зрозумілою, а також мати свої переваги, зокрема:

- розробка програми є доволі простою.
- Легкість внесення значних змін у програму.
- Простота тестування - розробники писали наскрізні тести, які запускали програму, викликали REST API та тестували інтерфейс користувача.
- Просте розгортання - все, що потрібно зробити розробнику, це скопіювати файл WAR на сервер, на якому встановлено Tomcat.
- Просте та зрозуміле тестування - розробники можуть створювати наскрізні методи тестування, які запускають програму, викликають REST API та перевіряють інтерфейс користувача.
- Прості методи деплойменту та достаки на виробничі середовища- для розробника достатньо скопіювати файли на сервер аплікації із встановленим веб сервером.
- Легкість та простота у масштабуванні.

Однак з часом розробка, тестування, розгортання та масштабування аплікації стають значно складнішими. В міру зростання функціональних можливостей монолітного сервісу, збільшується кількість коду. З часом система стає громіздкою, впроваджувати будь які зміни чи поправки стає важче. Проблеми із масштабуванням виникають через обмежені можливості масштабування монолітної системи, яка масштабується виключно горизонтально за допомогою запуску кількох окремих серверів, кожен із

яких містить свій власний моноліт. Іншою не менш важливою проблемою є оновлення змін на виробничому середовищі. Завжди існує ймовірність зламати частину функціоналу та вийти за рамки дозволених, чи наперед встановлених та погоджених часових інтервалів.

Все це безумовно впливає і на якість роботи самого сервісу, його швидкодію, надійність, стабільність та відмовостійкість.

Хоча монолітні програми і можуть бути успішними і навіть досить часто зустрічаються навіть в наш час, все більше людей починає розчаровуватись у них. Внесення змін до окремої частини програми вимагає перевірки та розгортання всього моноліту. З часом стає важче зберігати структуру модульності, що ускладнює процес впровадження змін, що мають впливати лише на окремий модуль у цілій монолітній системі . Масштабування вимагає розширення всієї аплікації, а не лише окремих частин, що потребують додаткових ресурсів.

1.2. Особливості організації мікросервісної архітектури

Мікросервісна архітектура - це методологія розробки програмного забезпечення, що полягає у створенні єдиної програми як набору невеликих підпрограм[2]. Кожна з цих програм працює у власному процесі та вся комунікація між програмами відбувається по протоколу HTTP. Основною особливістю цих програм є їх побудова на базі функціональності для бізнесу та їх незалежний деплоймент за допомогою засобів та процесів безперервного розгортання. Ці програми мають мінімальний рівень централізованого управління і можуть бути написані на різних мовах програмування та використовувати різні методи збереження даних.

1.2.1. Основні характеристики мікросервісної архітектури

Однією з ключових переваг використання сервісів у вигляді простих компонентів є їх незалежність в розгортанні. У випадку, коли програма

складається з кількох бібліотек, об'єднаних в один процес, навіть найменша зміна в одному з компонентів вимагає повторного розгортання всієї програми. Проте, якщо програма розбита на кілька сервісів, зміни у одному сервісі часто потребують лише оновлення цього конкретного сервісу [3-4]. Хоча деякі зміни можуть вплинути також і на інші сервіси в системі, правильне архітектурне планування мікросервісів ставить за мету мінімізувати ці взаємозалежності шляхом чіткої організації взаємодії між сервісами.

Коли відбувається поділ великого додатку на маленькі складові частини, це досить часто призводить до того, що задіюються різні команди в процесі розробки. Для прикладу це може бути команда яка відповідає за користувацький інтерфейс, команда розрозбників серверної частини, а також команда, яка спеціалізується на розробці баз даних (рис. 1.2).

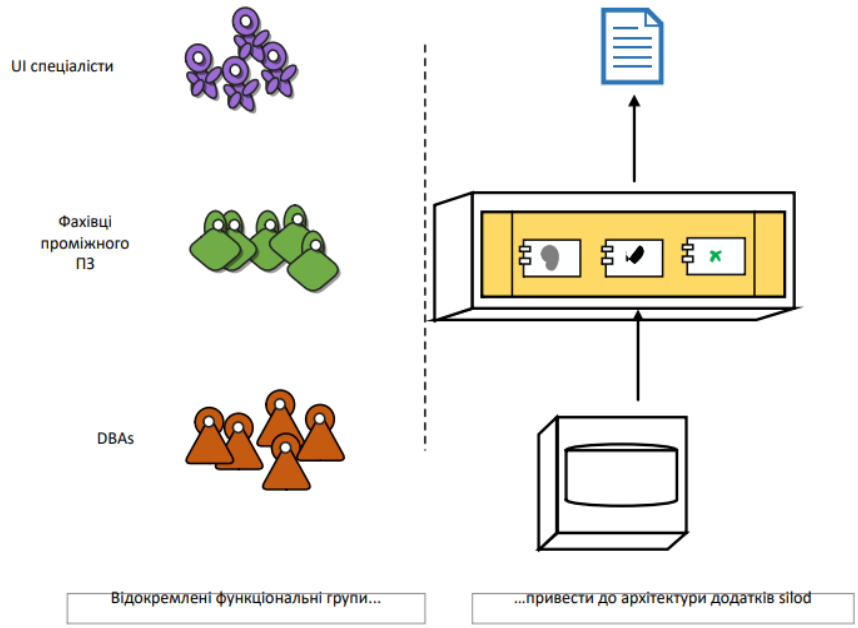


Рис.1.2. Схема поділу команд на технологічному рівні

Підхід, заснований на мікросервісній архітектурі, полягає у розбитті системи на окремі сервіси, кожен з яких спеціалізується на певній функціональності бізнесу. Ці сервіси включають широкий набір програмних компонентів для забезпечення цієї функціональності,

включаючи інтерфейс користувача, постійне сховище та будь-яку інтеграцію із зовнішніми сервісами. У зв'язку з цим, команди, які розвивають такі сервіс володіють всіма необхідними навичками для розробки та впровадження основних потреб та вимог бізнесу (рис. 1.3).

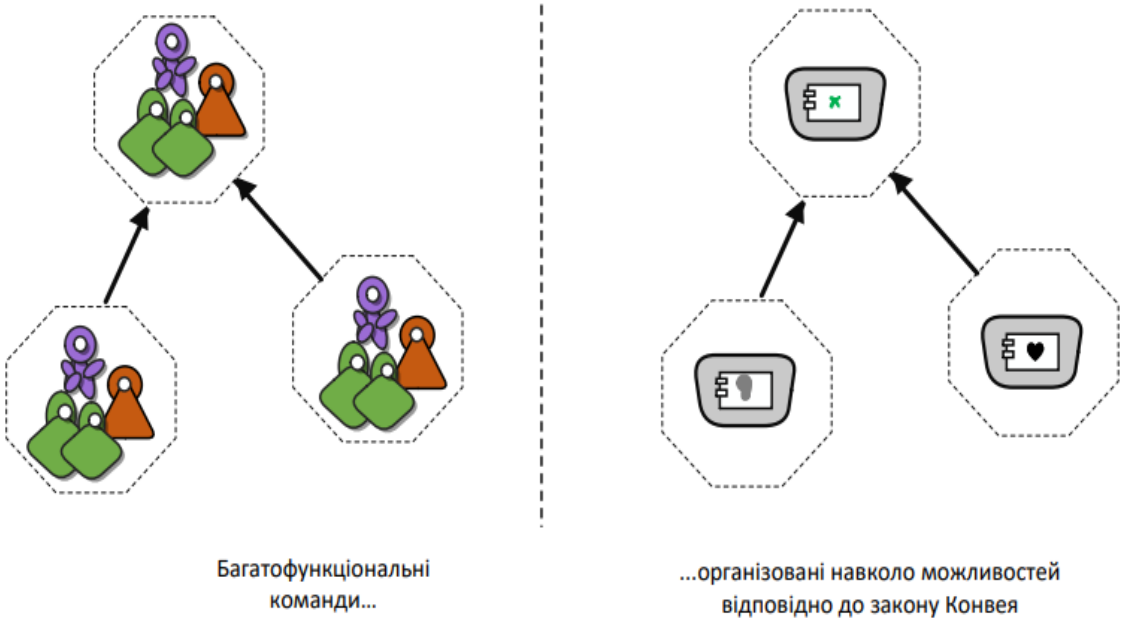


Рис.1.3. Схема поділу команд на основі бізнес функцій

Більшість підходів, які використовуються під час розробки додатків, ґрунтуються на моделі проекту: головною метою є створення певної частини програмного забезпечення, яка в кінці буде вважатись завершеною. Після завершення проекту весь контроль над ним, моніторинг його роботи передається команді, яка відповідає за технічне обслуговування, а команда, що працювала над розробкою проекту, розпускається.

Прихильники мікросервісної архітектури часто намагаються уникати такого підходу. В їхньому підході, команда людей, яка працювала над розробкою проекту, повинна супроводжувати його, підтримувати та розвивати протягом всього його життєвого циклу. Цією ідеєю часто надихається підхід Amazon "ти будуєш, ти керуєш", де розробники несуть повну відповідальність та здійснюють контроль за виробництвом

програмного продукту. Це надає їм щоденний контакт з клієнтами, які користуються програмним продуктом, оскільки вони беруть на себе частину відповідальності за технічну підтримку та супровід на виробництві.

Децентралізоване керування даними може мати різні прояви. На найвищому рівні це означає відмінність концептуальної моделі даних в різних системах. Існуючі моделі можуть мати різні атрибути, а також, що гірше, загальні атрибути з нюансованою семантикою. Крім того, мікросервіси децентралізують рішення щодо зберігання даних. На відміну від монолітних додатків, які часто використовують єдину базу даних для всіх даних, мікросервіси частіше віддають перевагу можливості кожному сервісу управляти своєю власною базою даних. Це може бути окремий екземпляр однієї технології баз даних або навіть зовсім різні системи бази даних (рис. 1.4).

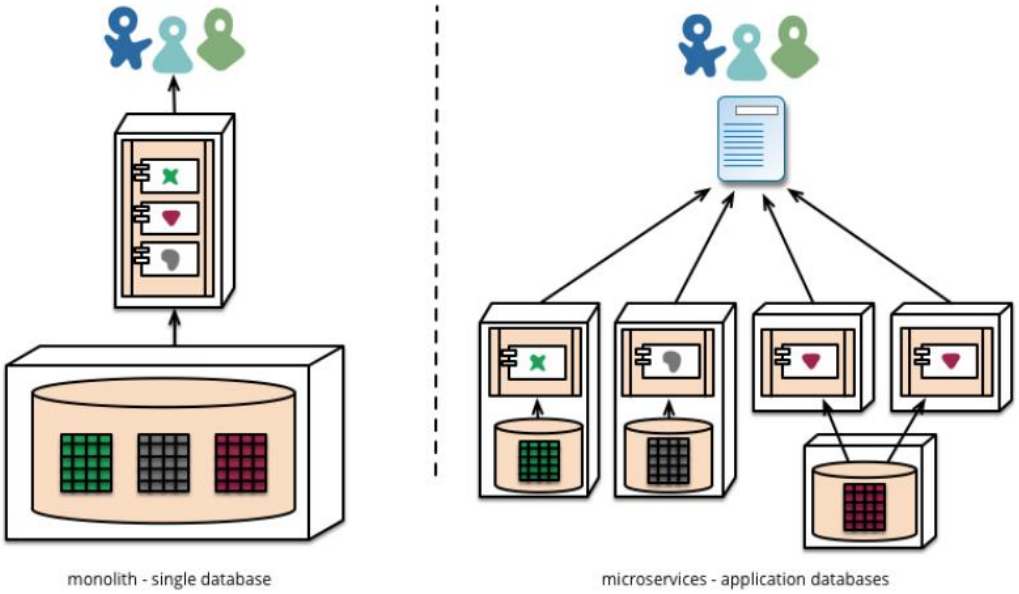


Рис.1.4. Архітектура баз даних при використанні мікросервісів

За останні роки, методи автоматизованого розгортання інфраструктури, використовуючи підходи IaC(Infrastructure as Code) значно посилилися. Розвиток хмарних технологій значно зменшив складність проектування, створення, розгортання та експлуатації мікросервісів. Багато

команд, які розробляють мікросервісні системи, мають значний досвід у безперервній інтеграції. Вони широко використовують технології інфраструктури як код. На рисунку 1.5, представлено як виглядає процес розгортання мікросервісу на виробниче середовище.

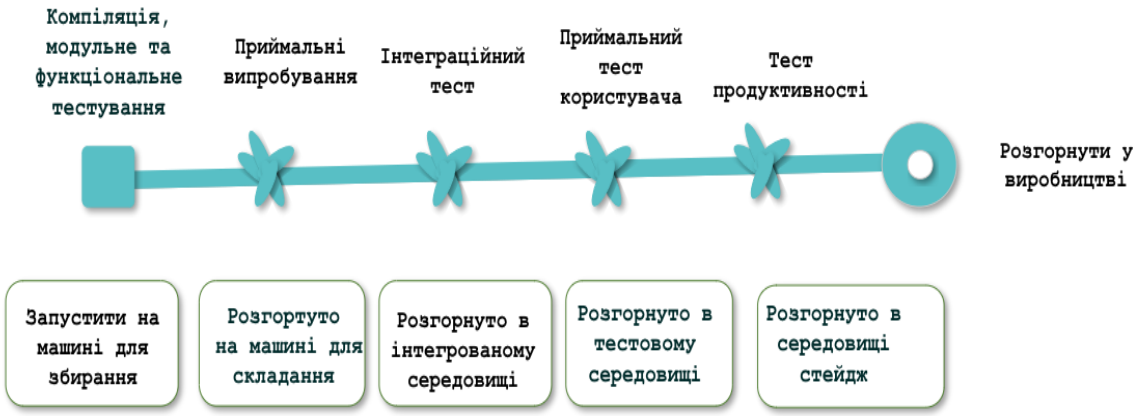


Рис.1.5. Процес автоматичного розгортання мікросервісу

1.2.2. Переваги та недоліки мікросервісної архітектури

Переваги архітектури мікросервісів.

- Дає можливість безперервної доставки та розгортання великих складних програм.
- Сервіси є компактними, що спрощує їх обслуговування.
- Кожен сервіс може бути розгорнутий незалежно від інших.
- Сервіси можуть масштабуватися незалежно один від одного.
- Мікросервісна архітектура дає певну автономію для різних команд розробників.
- Підхід мікросервісів є більш зручним для експериментів та досліджень.
- Така архітектура є кращою з точки зору локалізації та виявлення несправностей.

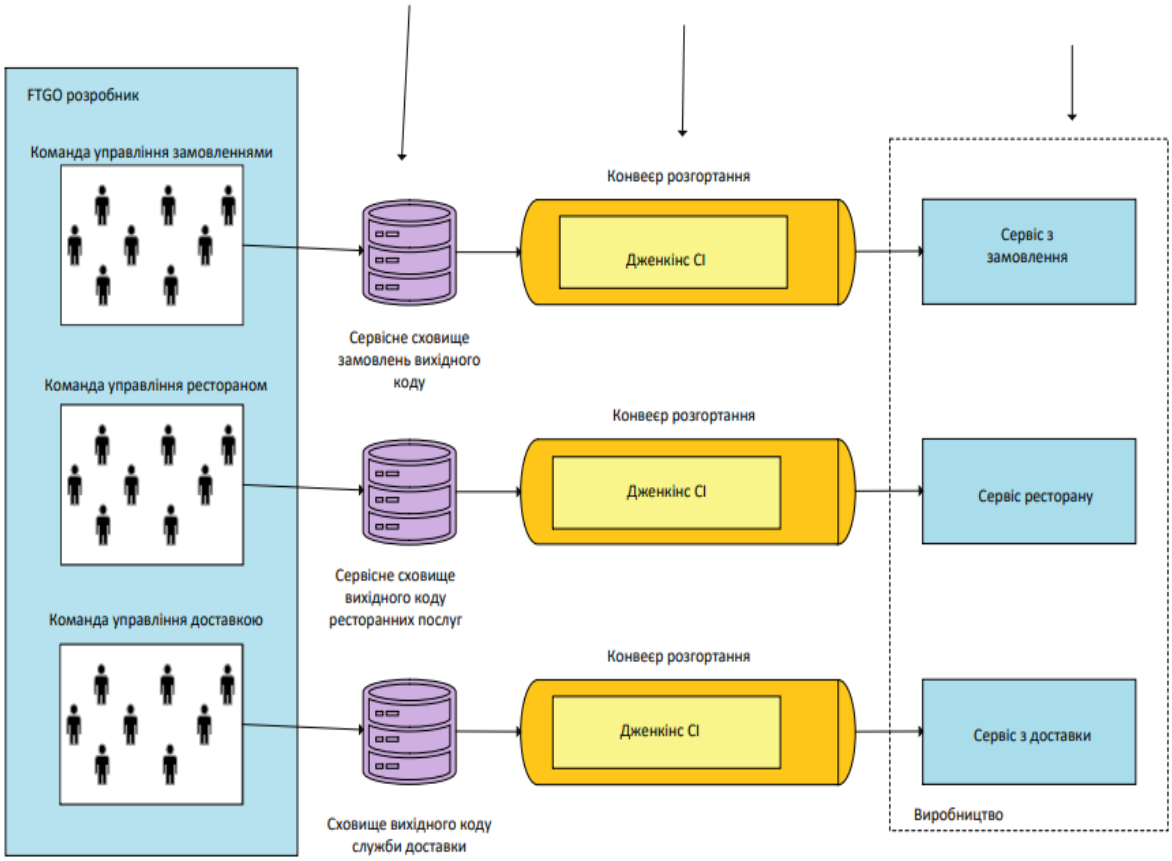


Рис.1.6. Переваги мікросервісної архітектури

Недоліки архітектури мікросервісів.

Звичайно, жодна технологія не є ідеальною, та крім переваг мікросервісна архітектура має певні недоліки та обмеження.

Серед основних проблем, можна виділити наступні:

- Виявлення правильного набору сервісів представляє собою значну складність.
- Розподілені системи, як правило є більш складні, а це в свою чергу ускладнює процеси розробки, тестування та розгортання.
- Вирішення питання про те, коли варто використовувати архітектуру мікросервісів, є важким завданням.

При вивченні переваг та недоліків архітектури мікросервісів виникає важливе питання: чи варто обирати цей підхід для вашої системи?

Відповідь на це запитання неоднозначна і залежить від кількох факторів. Спочатку слід звернути увагу на складність самої системи. Хоча підхід мікросервісів допомагає спростити складні системи, він також вносить свої власні виклики. Робота з мікросервісами вимагає автоматизації процесів розгортання, моніторингу, усунення помилок та можливої координації між сервісами [5]. І хоча існують ефективні методи вирішення цих проблем, вони вимагають додаткового часу та зусиль. Таким чином, якщо ваша система не настільки складна, як моноліт, можливо, варто знову переосмислити вибір архітектури мікросервісів.

1.3. Процес міграції від моноліту до мікросервісів

Представимо собі, що у нас є система "Каталог Продуктів", яка спочатку була монолітом (див. рис. 1.7). Ця система містить в собі інформацію про товари та всі пов'язані дані. З часом монолітна архітектура системи "Продукти" виросла до такого рівня, що включає у себе не лише базову інформацію про товари, таку як назва, категорія, а також інформацію про ціни, та логіку яка визначає процес їх формування. Тобто в системі втрачається чітка межа між основною частиною, яка відповідає за товари, та частиною, яка керує ціноутворенням.

Особливо слід звернути увагу на те, що темп змін у частині системи, яка відповідає за ціноутворення, виявився вищим, ніж у головній частині, яка відповідає за сам продукт та його складові. Цінова політика товарів змінюється набагато швидше, ніж базові атрибути товарів. Отже, стає очевидним той факт, щоб виділити частину системи, яка відповідає за ціноутворення, у окремий сервіс, та мати можливість розвивати його незалежно від інших підсистем [6].

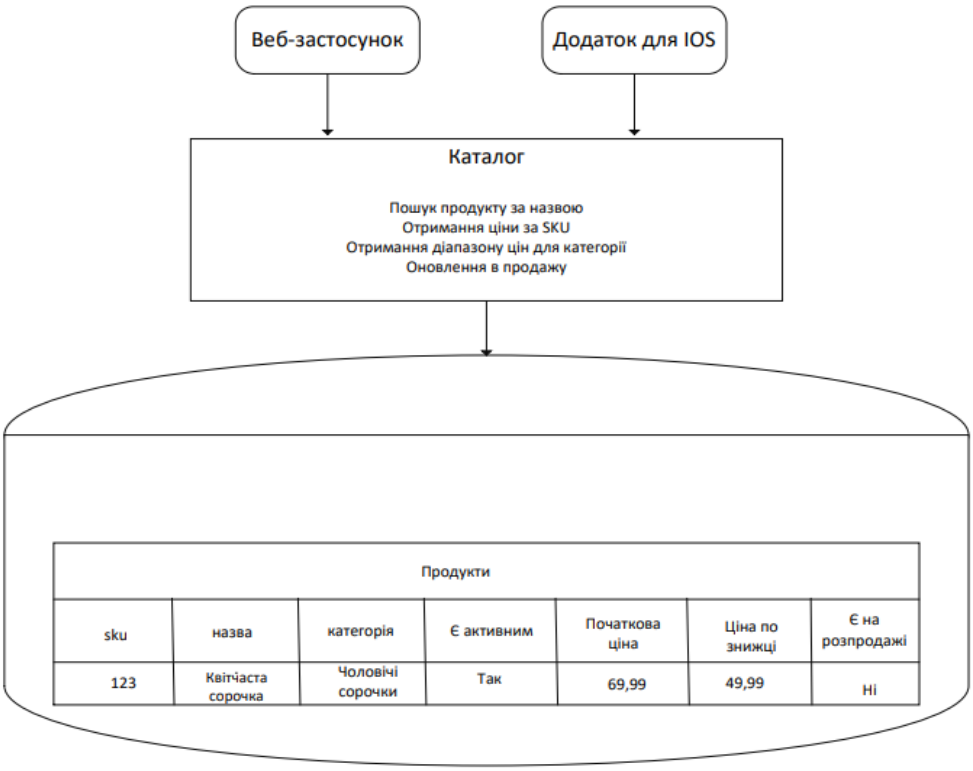


Рис.1.7. Схема монолітної системи каталогу продуктів

Процес міграції від монолітної аплікації до мікросервісів представлено на рисунку 1.8.

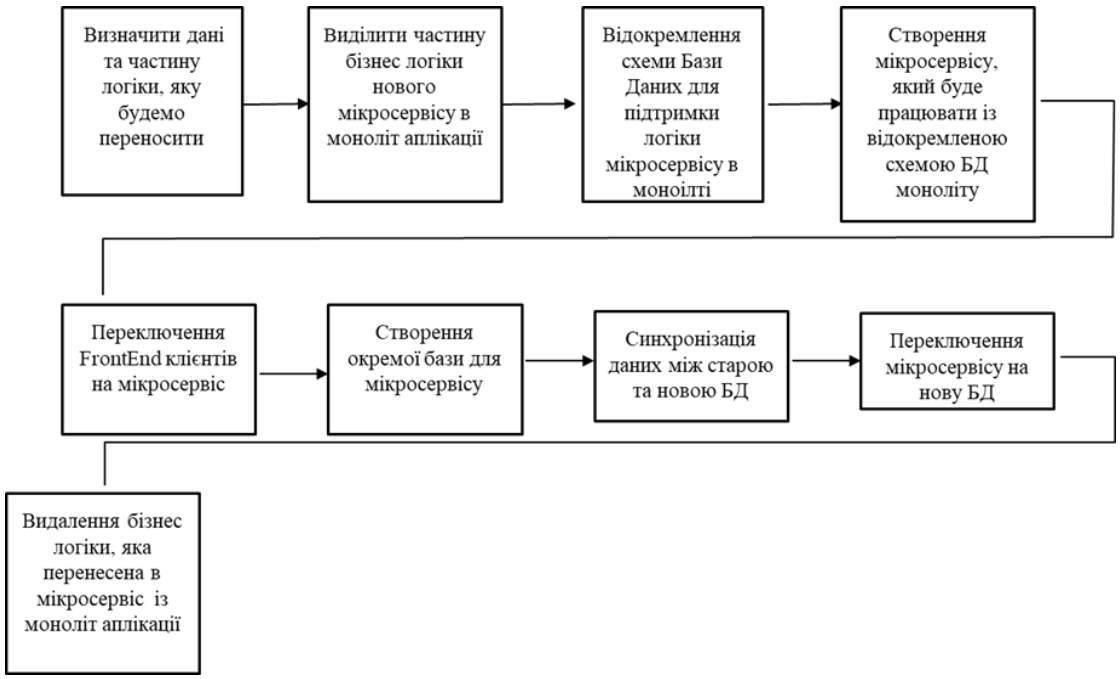


Рис.1.8. Процес міграції від моноліту до мікросервісів

Перший крок - полягає в ідентифікації даних та логіки, що стосується сервісу ціноутворення для товарів, який функціонує в монолітній системі. Програма "Каталог продуктів" містить таблицю "Products", яка включає основні атрибути продукту, а саме назва товару, ідентифікатор SKU та статус наявності на складі. У цій таблиці також містяться поля, що стосуються цін. В моноліті реалізована логіка, пов'язана з формуванням цін на товари та оновленням інформації про кількість товарів на складі та їх наявність.

Виділення логіки мікросервісу в моноліті - на цьому етапі, сам функціонал ще присутній в монолітній програмі, але логіка, яка стосується формування ціни, виділяється окремими складовими / функціональними блоками (рис. 1.9).

Крок третій - являє собою створення окремої схеми даних, яка все ще буде в БД моноліту, але сам моноліт використовувати її не буде. Це буде нова структура БД, яка буде використовуватись новим мікросервісом. В нашому випадку це буде вся схема даних, пов'язана із ціноутворенням та формування ціни яку бачить користувач.

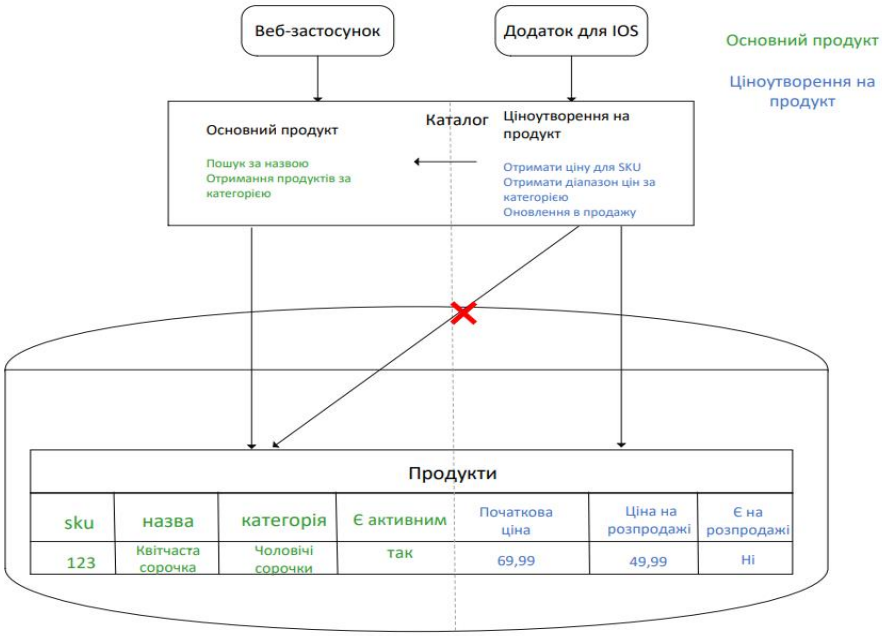


Рис.1.9. Логічний розподіл між ціноутворенням і основним продуктом

Створення нового мікросервісу - на цьому етапі ми створюємо новий мікросервіс, який буде відповідати за утворення цін на товари, але працюватиме він все ще з даними, які знаходять в БД моноліту. Після цього ми перемикаємо фронтенд частину аплікації на цей мікросервіс. В найпростішому випадку це WEB аплікація та мобільні аплікації для смартфонів.

Створення окремої бази даних - на даному етапі міграції, створюється окрема база даних із стуктурою, яка була задана раніше. Після цього дані із БД моноліту синхронізуються із новою базою даних. Після завершення синхронізації даних, новий мікросервіс перемикається на нову БД.

Цей етап є досить простим і полягає в створенні окремої бази даних для цін, яка відображає схему бази даних у моноліті (рис. 1.10). Під час розробки нового сервісу може виникнути ідея створити нову схему бази даних ціноутворення. Проте не слід забувати про процес міграції даних в нову базу. Якщо схема бази зміниться, то і процес суттєво ускладниться. Також слід зазначити, що при зміні схеми даних, новий сервіс ціноутворення має підтримувати дві різні схеми - одну з моноліту та іншу з мікросервісу. Після виділення бази даних цін в окрему ізольовану систему, будь-які зміни в ній відбуваються на тому ж рівні, що й зміни в коді сервісу, оскільки жоден з клієнтів не може напряму взаємодіяти з базою даних цін.

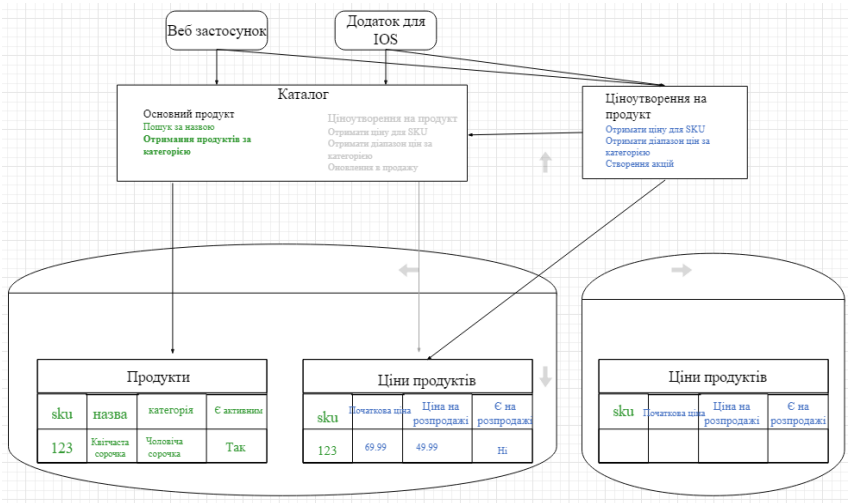


Рис.1.10 Схема створення окремої бази даних цін

Після створення нової бази даних, яка відповідає схемі бази даних монолітної програми, необхідно провести синхронізацію даних між двома базами, а саме базою моноліту та новою базою мікросервісу. Синхронізація є досить простою, якщо схеми даних у двох базах однакові.

Тепер все готове для переключення сервісу ціноутворення на нову базу даних в якій зберігаються ціни. Перед тим як розпочати цей етап, важливо, щоб усі клієнти моноліту, які потребують інформації про ціни, перейшли на новий сервіс. Після переведення усіх клієнтів на новий сервіс, можна повністю перемикати його на нову базу даних цін. Це є зміна підключення до бази даних з монолітної бази на нову базу даних (рис. 1.11).

Однією з переваг цього дизайну є те, що в будь який момент можна легко переключити підключення до старої бази даних у випадку, якщо стали помітні якісь проблеми. Якщо відбулась зміна схеми даних, тоді можуть виникнути проблеми у відсутності тих чи інших полів в новій схемі бази. Це може статися також і в тому випадку, якщо не всі дані були ідентифіковані на першому етапі. Перед тим, як переходити до наступного етапу, слід ідентифікувати та вирішити всі подібні проблеми.

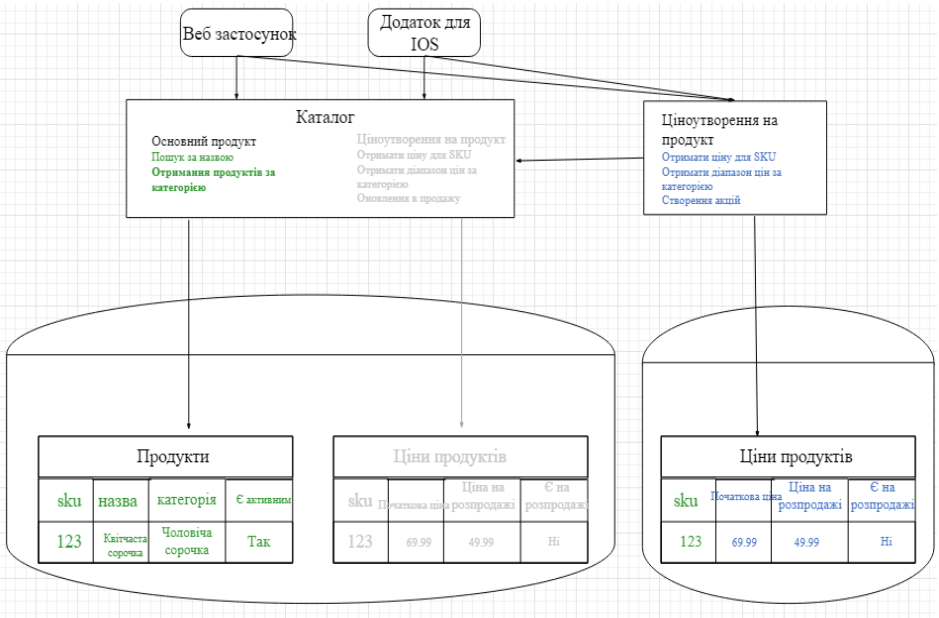


Рис.1.11 Схема перемикання сервісу ціноутворення на нову базу даних цін

Завершальним етапом - є видалення всієї логіки, яка пов'язана з утворенням цін та їх відображенням для клієнта на стороні моноліту.

В кінцевому результаті, система продуктів виглядатиме наступним чином

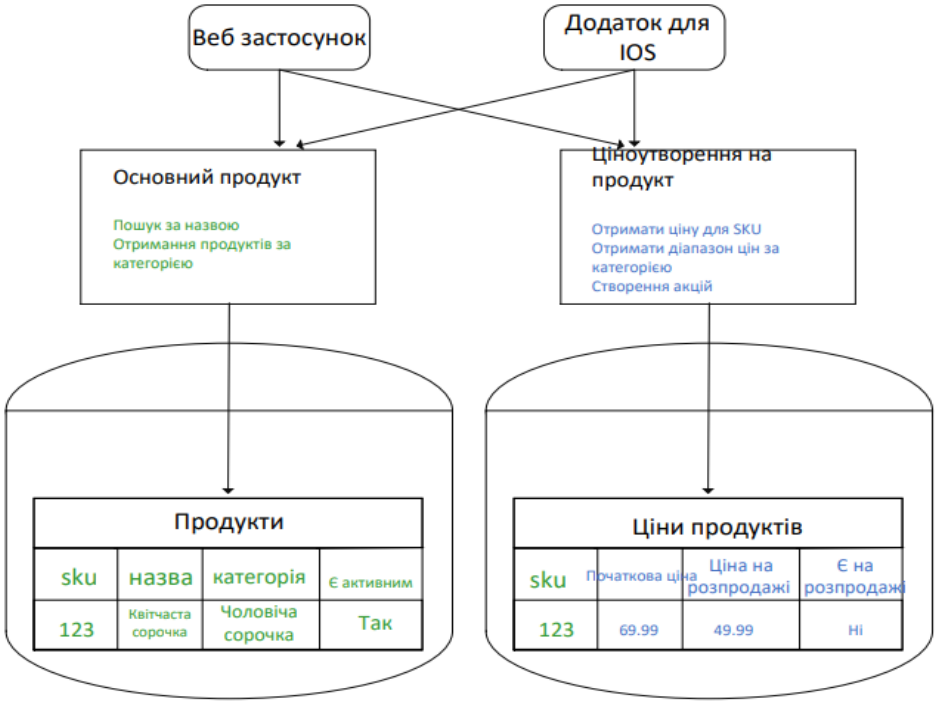


Рис.1.12 Схема системи продуктів із окремим мікросервісом ціноутворення

Основний (core) продукт все ще міститиме дані та частину логіки, що пов’язані з основними атрибутами продуктів, тоді як функціонал ціноутворення має дані та логіку, пов’язані з цінами. Взаємодія між ними відбувається тільки через логічний рівень. Згідно описаного вище алгоритму, можна виділити будь який функціонал моноліту в окремий мікросервіс.

1.4. Технології міжпроцесної взаємодії та обмін даними

Перенесення функціональності сервісу в окремий проект ускладнює процес взаємодії між іншими частинами моноліту та новоствореною

програмою, оскільки більше неможливо просто викликати певний метод сервісу на рівні коду. В даному варіанті реалізації потрібно імплементувати міжпроцесну взаємодію.

Існує багато різних підходів та технологій для організації міжпроцесної взаємодії. Можна використовувати методи які працюють по принципу запит/відповідь на основі HTTP, для прикладу REST або gRPC. Також можна використовувати асинхронні методи передачі даних на базі повідомлень, такі як AMQP або STOMP PubSub. Існує широкий спектр різних форматів повідомлень, доступних для використання. Сервіси можуть використовувати різні формати, такі як JSON або XML, які є зрозумілими для людини.

Існують різні способи взаємодії між клієнтом та сервісом, які можна класифікувати за двома основними типами.

Перший тип визначає, чи відбувається взаємодія один до одного або один до багатьох:

- Індивідуальний - один сервіс виділяється на обробку запитів клієнта .
- Один до багатьох -декілька сервісів можуть здійснювати запитів

Другий тип стосується того, як відбувається взаємодія синхронно чи асинхронно:

- Синхронний - клієнт очікує отримати відповідь від сервісу вчасно та може навіть блокувати його під час очікування.
- Асинхронний - клієнт не блокує сервіс, а відповідь може приходити не відразу.

Варто зазначити той факт, що синхронний стиль взаємодії запит-відповідь є досить поширеним у міжпроцесній взаємодії. Наприклад, сервіс може спілкуватися з іншим сервісом, використовуючи метод REST для обміну запитами та відповідями або ж механізми обміну повідомленнями. Цікавим є те, що навіть при умові взаємодії сервісів через брокера

повідомлень, може трапитись ситуація в якій клієнт може бути заблокованим у момент очікування відповіді від сервісу.

Типи взаємодій один до багатьох:

- Опублікувати/підписатися - клієнт публікує повідомлення, яке читають всі сервіси, що підписані на ці повідомлення.
- Опублікувати/асинхронні відповіді - клієнт надсилає повідомлення запиту, а потім очікує деякий час на відповідь від всіх, хто підписаний на ці повідомлення.

При застосуванні механізму міжпроцесної взаємодії IPC(Inter-Process Communication), який базується на віддаленому виклику процедури, клієнт ініціює запит до сервісу, який обробляє його та відправляє відповідь (рис.1.13). У деяких випадках клієнти можуть блокувати відповідь, однак, також можливі варіанти використання не блокуючої архітектури. Проте, в таких випадках, клієнт очікує, що відповідь буде відправлена вчасно.

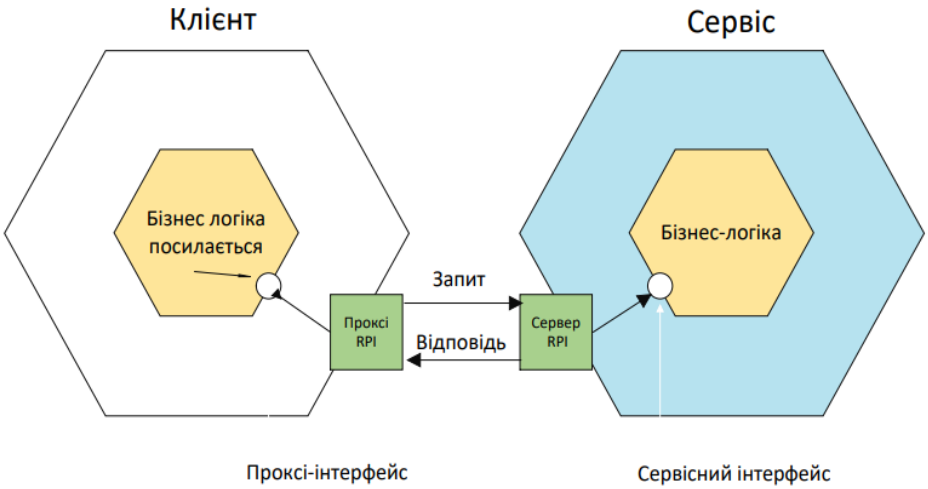


Рис.1.13 Взаємодія клієнт-сервер

В сучасних проектах та технологіях розробки, досить часто можна зустріти API в стилі RESTful. REST - це метод міжпроцесної взаємодії, що(майже завжди) використовує протокол HTTP.

У технології REST ключовим виступає ресурс, який зазвичай відображає один об'єкт, такий як користувач чи продукт, або набір таких

об'єктів. Для маніпулювання ресурсами, на які здійснюється посилання за допомогою URL-адрес, REST застосовує дієслова HTTP. GET запит, для прикладу, повертає значення ресурсу, яке може бути представлено у формі XML-документа або JSON об'єкта. Але слід зазначити, що бувають випадки використання і інших форматів. POST запити дають можливість створювати новий ресурс, а запит PUT робити зміни у вже існуючих ресурсах.. Наприклад, інтернет-магазин може мати POST запити /product для створення нового продукта, та запит GET /products/{productId} для отримання повної інформації про товар.

Звичайно, як і будь які інші технології, в REST є певні переваги та недоліки, серед основних можна зазначити наступні:

- Простота та зручність.
- Що стосується тестування, то будь який метод можна без проблем перевірити в будь якому браузері, чи командній стрічці, використовуючи досить поширені клієнти curl та wget.
- Підтримка безпосередньої взаємодії у форматі запиту/відповіді.
- Архітектура системи є доволі проста, оскільки не використовуються жодні брокери.
- Підтримка взаємодії тільки у форматі запит/відповідь.

Знижена доступність за рахунок відсутності буферизації повідомлень, що вимагає одночасної роботи клієнта та сервера.

- Клієнти повинні знати адреси (URL-адреси) всіх сервісів які вони використовують.

Однак, підсумовуючи всі аспекти та особливості, REST, все ще залишається найбільш використовуваною технологією для API, хоча і існують звичайно цікаві альтернативи.

1.5. Підходи та методи розгортання мікросервісів

Створити новий сервіс, описати його структуру та взаємодію із іншими сервісами є основними етапами створення нових мікросервісів. Однак, невід'ємною складовою цього процесу є розгортання та доставка його на робоче середовище. Тут можна використовувати багато методів та підходів. І весь цей процес може бути автоматизованим повністю чи частково [7]. Розглянемо основні методи які найчастіше використовуються.

Загальний метод розгортання мікросервісів представлено на рис.1.14. Першим із основних компонентів в нас безумовно виступає контроль версії. Найбільш часто тут використовують Bitbucket, Git, Svn. Простими словами - це репозиторій, куди розробник звантажує весь код, який стосується мікросервіса. Наступним етапом, після того як розробник завантажив нові зміни, є процес збірки аплікації. В залежності від того яка мова програмування використовується, аплікація може збиратись різними утилітами. Після цього, як правило, створюється docker образ в якому існує сам мікросервіс. Цей процес виконує як правило Jenkins, Teamcity, Bamboo.

Після цього, коли ми маємо готовий образ аплікації, його потрібно кудись зберегти. Тому використовується одне з багатьох можливих сховищ, серед яких може бути Docker Hub, Jfrog та багато інших [8]. І останнім кроком є оновлення аплікації із останнього образу на робочому середовищі. Це середовище може бути розгорнутим на локальній інфраструктурі, в хмарі, може використовуватись k8s, docker-swarm, та інші технології.

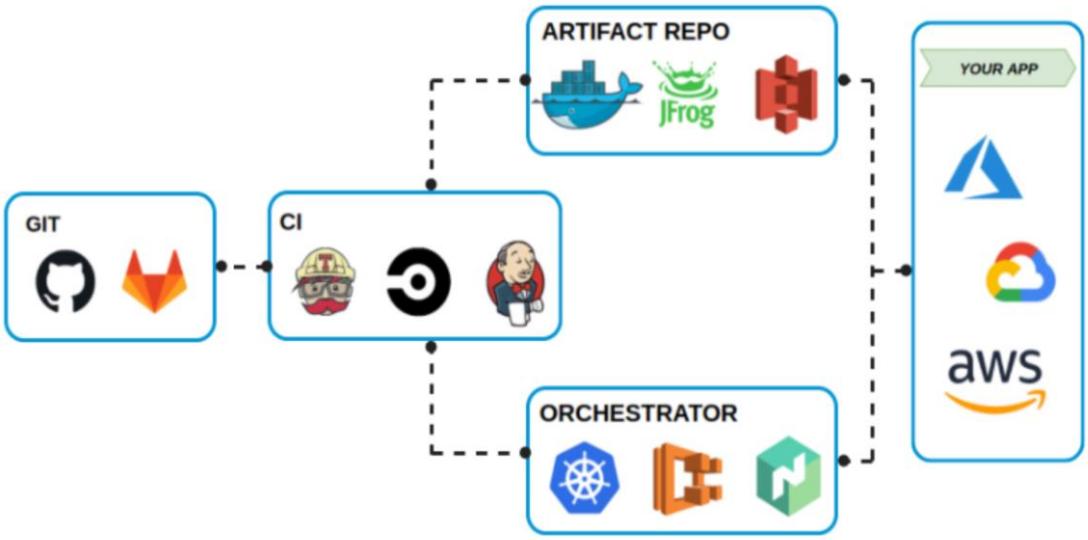


Рис.1.14. Загальна модель розгортання мікросервісів

Описаний вище метод є досить загальним і може мати багато модифікацій, розширень та додаткових кроків. Одним із таких розширень є метод із використанням terraform для побудови інфраструктури під час розгортання та argoCD для доставки цієї аплікації в Kubernetes кластер (рис.1.15).

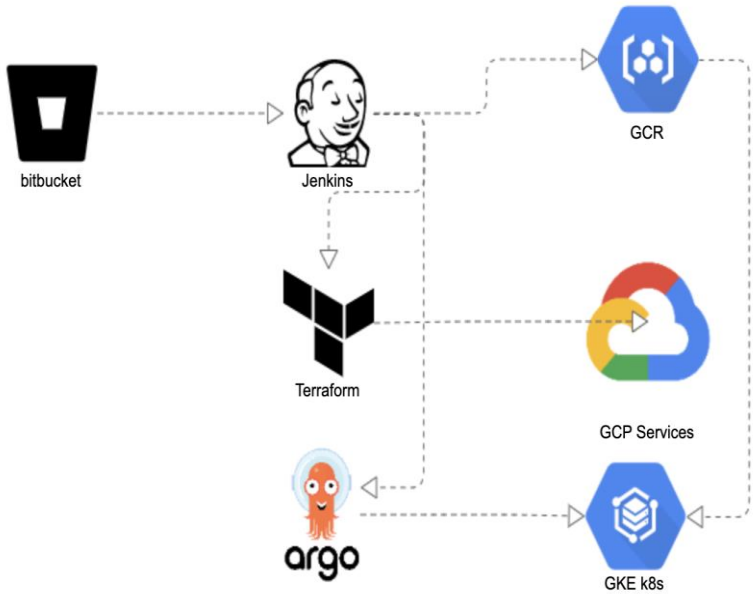


Рис.1.15. Метод розгортання мікросервісів із використанням terraform та argo

Відмінність чи перевага є в тому, що в залежності яких змін потребує інфраструктура для нормальної роботи нашого сервісу, процес розгортання та доставки аплікації зробить всі ці зміни автоматично. Крім того, у випадку неуспішного розгортання, завжди буде змога повернутись на попередню версію робочої аплікації [9].

Крім того принципом хорошого тону є звичайно використання окремих середовищ для розробки, тестування, відлагодження. Слідуючи принципам такого підходу з'являється ще один метод. (рис.1.16).

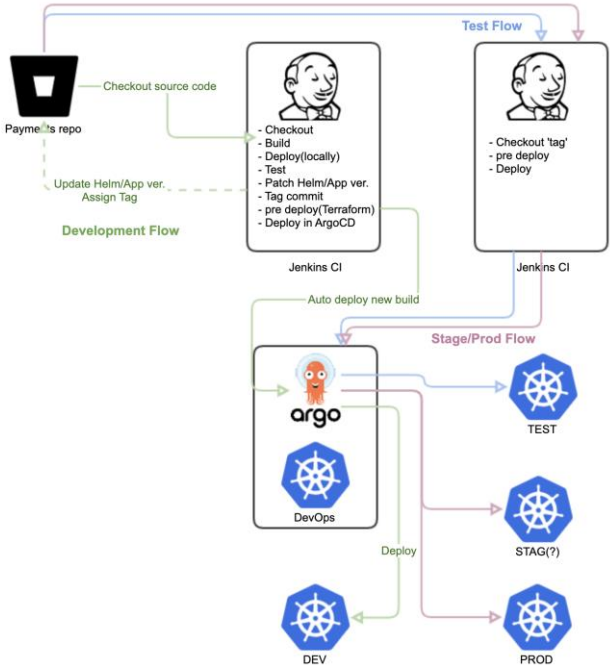


Рис.1.16. Модель розгортання мікросервісів із використанням різних середовищ

В порівнянні з попередніми, даний метод є дещо безпечнішим. Перевага полягає в тому, що новий сервіс із новим функціоналом доставляється не відразу на робоче середовище. Перед тим як він буде розгорнутий на робочому середовищі, весь функціонал та цілісність буде перевірено на тестовому, а також середовищі відлагодження. В більшості

випадків цей метод є не абсолютно автоматизованим, а включає в себе ручні етапи.

1.6. Інфраструктура як код – IaC, як метод створення та керування робочих середовищ

Донедавна управління IT-інфраструктурою було трудомістким завданням. Системні адміністратори повинні були вручну налаштувати та контролювати все апаратне та програмне забезпечення для організації якісної роботи різних сервісів. Проте нові тенденції, зокрема хмарні обчислення, покращили підходи до організації, розвитку та підтримки IT-інфраструктури. Найважливішою складовою цього тренду є «Інфраструктура як код - IaC».

IaC - це модель, у якій процес налаштування інфраструктури подібний до процесу розробки програмного забезпечення. По суті, програми можуть містити сценарії, які створюють власні віртуальні машини з наперед заданими параметрами та керують ними. IaC використовує мову кодування високого рівня для опису всієї існуючої інфраструктури у вигляді коду, а також інструменти, які автоматизують процес створення та оновлення IT-інфраструктури.

Інфраструктура як код - це процес керування центрами обробки даних не шляхом фізичного налаштування обладнання чи використання інтерактивних інструментів налаштування, а за допомогою файлів конфігурації. Системним адміністраторам і технікам слід зосередитися на розвитку послуг, а не на обслуговуванні IT-інфраструктури. Ручний процес часто викликає багато проблем, які необхідно вирішувати.

Висока вартість. Вам потрібно найняти багато професіоналів, щоб вручну налаштувати необхідне IT-середовище. Крім того, що всім цим людям потрібно платити, ними ще потрібно керувати. Це, у свою чергу, призводить до вищих накладних витрат і ускладнює спілкування.

Масштабованість і доступність (повільне встановлення). Щоб вручну налаштувати інфраструктуру, інженерам потрібно розташувати сервери, а потім налаштувати апаратне забезпечення та мережу відповідно до параметрів. Ці процеси займають багато часу і часто на цьому етапі виникають помилки.

Моніторинг і видимість продуктивності. Якщо виникає проблема з вашою інфраструктурою, яким чином точно можна визначити джерело проблеми? Варіантів може бути багато, наприклад проблема мережі, сервера чи програми або сервісу який запущений? Очевидно, що потрібні інструменти, які дають повний огляд того, як працює вся IT-інфраструктура та процеси, які в ній відбуваються.

Невідповідності середовища. У міру розвитку інфраструктури, все більше і більше людей потрібно залучати для конфігурацій ручного розгортання, що спричиняє неузгодженості і з часом стає важко відтворити однакові середовища. Ці невідповідності призводять до критичних відмінностей між різними середовищами, де працюють певні сервіси, а саме середовищем розробки, тестування якості та виробничим середовищем і викликають проблеми з розгортанням.

Із впровадженням інструментів IaC підвищується продуктивність, покращується якість встановлення будь-якого програмного забезпечення, а життєвий цикл розробки програмного забезпечення стає ефективнішим. «Інфраструктура як код» стала можливою завдяки платформам «Інфраструктура як послуга» (IaaS), які дозволяють досить швидко створювати ресурси в хмарних сховищах при необхідності і так само швидко згортати їх , використовуючи механізми API.

Однак, навіть якщо використовується IaC, в будь якому випадку слід контролювати стан інфраструктури та її сервісів, використовувати певні механізми моніторингу та стану працездатності для збору показників і журналів подій. Ці показники та журнали дозволяють контролювати статус

нашої інфраструктури та якість роботи послуг, які вона забезпечує і застосовувати деякі механізми автоматичного масштабування/згортання за необхідності. Щоб забезпечити постійну інтеграцію та розгортання додатків та інфраструктури, підприємствам знадобляться інструменти DevOps, щоб гарантувати якість обслуговування (QoS)/(QoE) [10-11]. Ці інструменти дають змогу швидше (навіть завчасно) реагувати на деградацію продуктивності та збої, витрачаючи при цьому набагато менше зусиль та виконуючи це з більшою ефективністю.

1.6.1. Переваги використання інфраструктури як коду - IaC

Існує дві категорії інструментів для IaC:

Інструменти керування конфігурацією, які призначені для встановлення та керування програмним забезпеченням на існуючих серверах (наприклад, Ansible, Chef, Puppet);

Інструменти ініціалізації (наприклад, CloudFormation і Terraform), які призначені для самостійного створення цих серверів, залишаючи роботу з налаштування (та решти вашої інфраструктури, як-от балансувальники навантаження, бази даних, налаштування мережі тощо) іншим інструментам.

Насправді більшість інструментів керування конфігурацією можуть виконувати певний рівень ініціалізації чи створення, а більшість інструментів ініціалізації виконують певний рівень керування конфігурацією, єдина різниця полягає в тому, що той чи інший тип краще підходить для певних завдань.

IaC означає керування вашою IT-інфраструктурою за допомогою конфігураційних файлів. Отже, конфігурація інфраструктури має форму файлу в якому вся інфраструктура описана у вигляді коду. Оскільки це лише текст, його можна легко редагувати, копіювати та поширювати. Ви можете поставити його під керування системами контролю версій (git, cvs,

bitbucket тощо), як і будь які інші файли звичайного програмного коду. У свою чергу, це дозволяє швидко створити таку кількість екземплярів усієї вашої інфраструктури, скільки буде потрібно в будь який момент часу Цей метод дає змогу постійно створювати ресурси без помилок, скорочуючи час на керування та налаштування вручну. Схема для розгортання інфраструктури наведена нижче (рис.1.17):

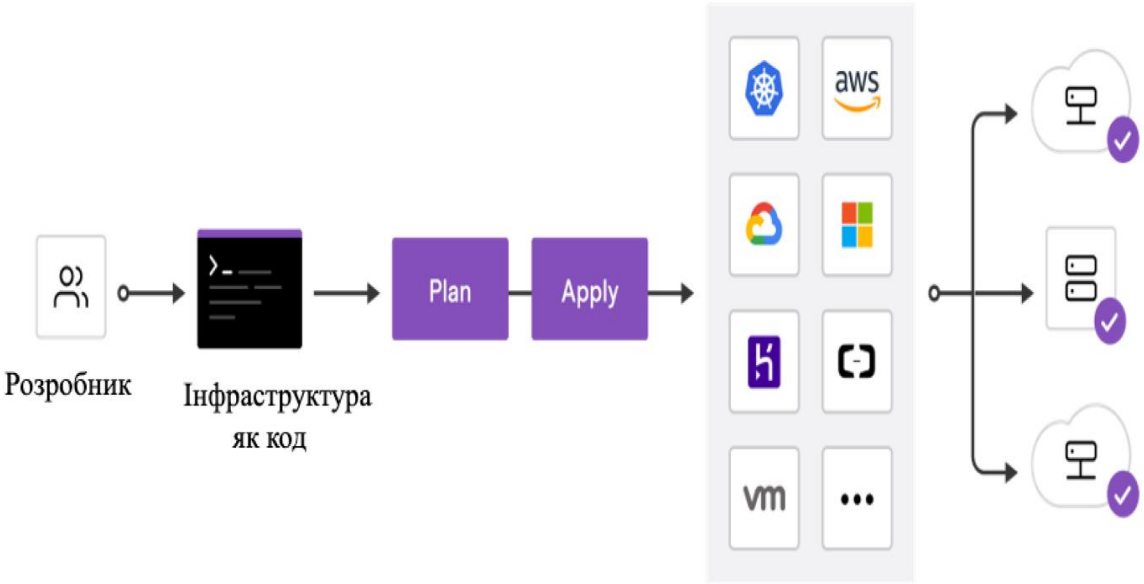


Рис.1.17. Схема розгортання інфраструктури в хмарному середовищі

Ключові компоненти цієї схеми:

- *Розробники / Автори* - особи, які будуть створювати дизайн інфраструктури та розробляти код;
- *Код у системі контролю версій* - усі компоненти вашої інфраструктури (бакети, IAM, мережа, віртуальні машини, K8 тощо);
- *Постачальник(провайдери) інфраструктури* - хмарний провайдер, який надає платформу, де будуть зберігатися всі компоненти інфраструктури (GCP, AWS, Azure тощо);
- *Робочі середовища* - окремі середовища для розробки, тестування та виробництва;

- *Служби / Сервіси* - список мікросервісів, що працюють у підготовлених середовищах.

У рамках робочого процесу IaC ви можете розгортати інфраструктуру кілька разів стандартизованим способом[12].

Коли інфраструктура, яку ви описуєте, відповідатиме всім вимогам, ви зможете правильно розгорнути її в хмарному середовищі. Коли з'являються нові вимоги, ви можете розглянути та врахувати їх і повторити процес стільки разів, скільки потрібно.

Інфраструктура як код приносить багато переваг і стає все більш популярною серед інженерів, саме в наш час. Основні переваги IaC (рис. 1.18):

- *Швидкість.* Можливість розгортати різні середовища від розробки до виробництва; створювати віртуальні сервери та попередньо налаштовані бази даних, балансувальники навантаження, системи зберігання даних, мережеву інфраструктуру або будь-який інший хмарний сервіс відповідно до ваших вимог. Організувати резервне копіювання та аварійне відновлення, розгорнувши свої інфраструктурні середовища в інших місцях.

- *Послідовність*. Незважаючи на спроби підтримувати певну послідовність у процесі розгортання інфраструктури за допомогою стандартних операційних процедур, ручне керування інфраструктурою може призвести до неправильної конфігурації, оскільки люди завжди можуть помилятися. IaC вирішує цю проблему та зменшує ймовірність помилок завдяки стандартизації налаштування інфраструктури за допомогою конфігураційних файлів.

- *Відповідальність*. Оскільки IaC використовує систему контролю, завжди є можливість відстежувати, хто і коли робив певні зміни в компонент інфраструктури, спробувати зрозуміти з якою метою та за необхідності повернутись до попередньої версії.

- *Масштабованість*. Розгортання інфраструктури з IaC повторювані та стабільні. Написавши код один раз, можна швидко та надійно використовувати його багато разів, уникаючи залежностей ручного налаштування та неправильної конфігурації.

Рис.1.18. Основні переваги IaC

- *Ефективність*. Використання розгортання інфраструктури з коду робить увесь процес безпечнішим, а проблеми чи помилки можна виявити раніше, до запуску вашої інфраструктури у виробництво. Це дуже важливо для процесу безперервного розгортання (CD) [13].
- *Вартість.* Автоматизація конфігурації інфраструктури та зниження витрат на управління інфраструктурою є однією з головних переваг IaC. Використання засобів автоматизації розгортання IaC робить процес побудови та налаштування інфраструктури більш ефективним, зменшуючи витрати та зусилля, а також дозволяє IT-спеціалістам витрачати більше часу на пошук інноваційних рішень.

Розглянуті методи дозволяють керувати інфраструктурою за допомогою файлів конфігурації, а не через графічний інтерфейс користувача, що дає можливість створювати, змінювати та адмініструвати

свою інфраструктуру безпечним, узгодженим і повторюваним способом, визначаючи конфігурації ресурсів, та застосовуючи при цьому методи контролю версій.

1.7. Огляд наукових робіт в напрямку підвищення якості обслуговування у мережах доставки контенту.

Сучасний світ стрімко рухається до повної цифровізації. Онлайн-платформи стали невід'ємною частиною повсякденного життя - від стрімінгових сервісів, соціальних мереж та електронної комерції до хмарних додатків та онлайн-освіти. Усе більше сервісів розміщується на швидкому доступі до цифрового контенту, а це вимоги до якості роботи CDN-мережі та сервісів, які надаються користувачам. Затримки, повільні завантаження чи перебої можуть негативно вплинути на роботу бізнесів і загальне враження користувачів. Мережа CDN є основою для багатьох секторів економіки:

Електронна комерція: Швидкість завантаження сайтів впливає на конверсію та дохід.

Реклама: Онлайн-реклама залежить від якісної доставки відео, банерів та аналітики.

Освіта та робота: В умовах пандемії та переходу на дистанційну роботу CDN забезпечують доступ до навчальних і корпоративних платформ.

У цьому розділі наведено грунтовний аналіз літератури на тему динамічного балансування навантаження та використання мережі доставки контенту (CDN). Основною метою цього огляду є детальне обговорення поточного стану стратегій балансування навантаження, включаючи їх переваги, недоліки, труднощі впровадження, а також їхній вплив на ефективність роботи CDN мереж.

Динамічне балансування навантаження є ключовим компонентом, що дозволяє оптимізувати розподіл ресурсів, покращити якість обслуговування (QoS) та забезпечити мінімальні затримки при завантаженні контенту. З огляду на зростання обсягів трафіку та запитів до CDN, стає очевидною необхідність нових підходів, які забезпечать необхідну якість обслуговування при роботі із статичними та динамічними даними.

1.7.1. Огляд наукових досліджень в напрямку методів балансування навантаження в CDN мережах

Автори роботи [14] пропонують чутливий до помилок метод балансування навантаження для мереж доставки контенту (CDN) для покращення масштабованості та надійності. Використання балансування навантаження, маршрутизації запитів і реплікації ресурсів покращує продуктивність роботи CDN. В роботі представлено узагальнену стратегію балансування навантаження для подолання існуючих обмежень а також запропоновано метод балансування із врахуванням рівня помилок при обробці даних та формуванні відповіді користувачам.

Дослідження [15] фокусується на оптимізації доставки контенту в хмарі шляхом аналізу якості відеоконтенту та збільшення трафіку. Запропоновано ефективний метод доставки контенту, що враховує мережеву відстань і здійснює динамічне перенаправлення запитів на граничні сервери кешування, використовуючи для цього дані протоколу BGP. Запропонована в роботі стратегія дає можливість зменшити час відповіді при отриманні контенту.

В дослідженні [16] пропонується динамічне створення реплікації контенту в залежності від його популярності в певних локаціях. Реплікація буде використовуватись тільки для даних які користуються високою популярності і тільки у певних локаціях CDN мережі.

Розглянувши ряд робіт по балансуванню навантаження, можна зауважити що спільним недоліком є те, що пропонуються лише методи балансування між граничними локаціями CDN мереж. Не враховується можливість роботи балансувальника в межах однієї локації між різними кешуючими серверами. Також не враховується тип запитів та критерії по яких можна їх розподіляти між кешуючими серверами, щоб в подальшому здійснювати ефективне використання кешованих даних.

1.7.2. Огляд наукових досліджень в напрямку методів балансування навантаження із використанням технології Kubernetes.

Kubernetes (K8s) - це система з відкритим вихідним кодом для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Це дозволяє легко керувати великою кількістю мікросервісів, розгорнутих у контейнерах. Kubernetes допомагає ефективно використовувати обчислювальні ресурси, автоматично виділяючи CPU, пам’ять та інші ресурси для кожного контейнера. Як правило ця технологія використовується на стороні кореневого сервера, якщо використовується мікросервісна архітектура побудови сервісів.

У науковій праці [17] було запропоновано алгоритм балансування навантаження в кластері, який перед розподілом ресурсів оцінює завантаженість процесора, пам’яті та пропускної здатності мережі кожного вузла, на додаток до важливості та вагових коефіцієнтів вхідних запитів. Він також враховує взаємозалежності між мікросервісами для оптимізації доставки.

Автори роботи [18] досліджують методи горизонтального масштабування контейнерів в Kubernetes кластері. Проводилось порівняння метрик, які аналізує сам кластер, а саме Kubernetes Resource Metrics (KRM) та додаткових метрик від системи Prometheus, Prometheus Custom Metrics

(PCM), а також вплив метрик на варіанти масштабування контейнерів під час зростання навантаження.

Використовуючи специфічні для сервісів політики маршрутизації, у роботі [19] авторами був використаний Service Mesh Istio для динамічного розподілу трафіку між сервісами. Для кожного сервісу визначались індивідуальні політики, такі як обмеження пропускної здатності чи пріоритетність трафіку.

В роботі [20] автори досліджують методи поєднання роботи мереж доставки контенту та динамічного балансування на стороні кореневого сервера із використанням технології Kubernetes. Дослідження проводилось із використанням сервісу CloudFront, який являє собою мережу CDN розподілену по AWS локаціях. Основний акцент було зроблено на використання принципів роботи мережі CDN та балансування і масштабування на стороні тільки кореневого сервера, де власне знаходилась аплікація. Для організації ефективного балансування та масштабування використовувався Kubernetes кластер, який також знаходився в одній із локації AWS. Основним недоліком такого підходу було те, що під час зростання навантаження виконувалось масштабування сервісу на стороні кореневого сервера і балансування запитів між створеними інстансами.

В розглянутих роботах багато уваги було зосереджено на використанні технології Kubernetes саме на кореневих серверах, або ж кореневій локації. Висвітлено переваги роботи цієї технології, методи масштабування ресурсів. Однак не було розглянуто можливість перенесення процесу обробки даних на сторону локації клієнта, який запитує даний контент чи сервіс. Тобто не враховувався той факт, що процес обробки динамічних даних, при значному зростанні навантаження, динамічно можна перенести в локацію користувача. Це дало б змогу зменшити навантаження на кореневий сервер та зменшення часу відповіді

для кінцевого користувача і тим самим покращити якість обслуговування в мережах доставки контенту.

Короткий огляд наукових досліджень, підходів та основні результати представлено у вигляді табл.1.1.

Таблиця 1

Огляд наукових досліджень, підходів напрямку підвищення якості обслуговування у мережах доставки контенту

Автори	Назва дослідження	Підходи, які використовувались в роботі	Основні наукові результати
Gupta, P., Goya M. K.,	Reliability aware load balancing algorithm for content delivery network	Балансування навантаження, маршрутизація запитів та реплікація даних.	Запропоновано метод балансування навантаження із врахуванням рівня помилок при обробці даних та формуванні відповіді користувачам
Nakanishi, K., Suzuki, F., Ohzahata, S., Kato, T.	A container-based content delivery method for edge cloud over wide area network	Аналіз якості відеоконтенту, використання протоколу BGP для визначення мережевої відстані.	Запропоновано ефективний метод доставки контенту, що враховує мережеву відстань і динамічно здійснює перенаправлення запитів на граничні сервери кешування використовуючи дані протоколу BGP
Nguyen, T. T., Yeom, Y.	Horizontal pod autoscaling in	Kubernetes кластер,	Порівняння метрик, які

J., Kim, T., Park, D. H. and Kim, S	kubernetes for elastic container orchestration	горизонтальне масштабування ресурсів.	аналізує Kubernetes кластер, а саме Kubernetes Resource Metrics (KRM) та метрик від системи Prometheus, Prometheus Custom Metrics (PCM), та дослідження їхнього впливу на процес масштабування контейнерів під час зростання навантаження
Nitu Kumari та Diego Lugones	Analysis of dynamic application load balancing in kubernetes using CDN	AWS сервіс CloudFront, Kubernetes кластер	Досліджено методи поєднання роботи мереж доставки контенту CDN та динамічного балансування навантаження на стороні кореневого сервера із використанням технології Kubernetes

Огляд наукових досліджень демонструє широкий спектр стратегій балансування навантаження, спрямованих на підвищення ефективності хмарних обчислювальних середовищ. Запропоновані методи охоплюють різні сценарії використання, зокрема мережі доставки контенту (CDN), кластерів на основі технології Kubernetes та гібридні хмарні рішення з периферійними дата-центрами. Основна увага акцентується на важливості

балансування навантаження для оптимізації розподілу ресурсів, зменшення часу відповіді та покращення якості послуг для кінцевого користувача.

Попри те, що розглянуті підходи демонструють значний потенціал у підвищенні продуктивності та масштабованості, залишається потреба в подальших дослідженнях. Зокрема, необхідно усунути існуючі обмеження та розробити інтелектуальніші стратегії балансування навантаження на всіх рівнях роботи CDN мереж, здатні відповідати динамічним вимогам сучасних хмарних обчислювальних середовищ.

1.7. Висновок до розділу 1

В даному розділі проведено дослідження принципів та методів, переходу він монолітної архітектури до мікросервісної. Визначено основні фактори, які варто брати до уваги плануючи процес переходу до мікросервісного стилю. Було виділено декілька основних аспектів, що стосуються розглянутих архітектурних підходів. З точки зору монолітної архітектури, програма яку пише розробник являє собою єдиний цілісний блок, що описує певну бізнес логіку. Якщо потрібно зробити будь які зміни в моноліті, то це в свою чергу вимагає розгортання на робочому середовищі всього цілісного блоку програми. В підході мікросервісної архітектури, вся цілісна система поділена на маленькі логічні складові, які з точки зору впровадження, оновлення та розгортання на робочих середовищах є абсолютно незалежними. Варто зазначити, що кожен маленький сервіс буде містити також і власну базу даних.

Якщо мова йде про великий проект, то звичайно в таких випадках набагато краще та доцільніше використовувати підходи архітектури мікросервісів, оскільки це дає можливість пришвидшити процес розробки програмного забезпечення, за рахунок можливості паралельно працювати невеликим автономним командам.

Під час розгортання нового сервісу запропоновано декілька підходів, кожен з яких має свої особливості. Звичайно цих методів може бути набагато більше, однак основна увага акцентована на ключові компоненти та кроки. Використовуючи автоматичне розгортання, ми зводимо до мінімуму людський фактор. Цим самим робимо наш сервіс більш стабільним та надійним.

Розглянуто також сучасні методи створення та управління інфраструктурою не шляхом фізичного налаштування обладнання чи використання інтерактивних інструментів налаштування, а за допомогою файлів конфігурації та підходів Інфраструктура як код IaC. Такі методи створення та керування робочими середовищами мають ряд переваг, які дають можливість створювати, змінювати та адмініструвати свою інфраструктуру безпечним, узгодженим і повторюваним способом, визначаючи конфігурації ресурсів, та застосовуючи при цьому методи контролю версій.

В цьому розділі також проведено грунтовний аналіз наукової літератури на тему динамічного балансування навантаження, кешування даних, з метою визначення поточного стану стратегій, що використовуються, включаючи їх переваги, недоліки, труднощі впровадження, а також їхній вплив на ефективність роботи мереж доставки контенту та якість сервісів, які в них надаються. Попри те, що розглянуті підходи демонструють значний потенціал у підвищенні продуктивності та масштабованості, залишається потреба в розробці інтелектуальних методів балансування навантаження, кешування статичних даних, а також перенесення процесу обробки динамічних даних на граничні локації CDN мереж, які здатні відповідати динамічним вимогам сучасних хмарних обчислювальних середовищ.

РОЗДІЛ 2. МЕТОДИ І АЛГОРИТМИ БАЛАНСУВАННЯ ТА РОЗПОДІЛЕНОЇ ОБРОБКИ ЗАПИТІВ В CDN МЕРЕЖІ

2.1. Базові принципи роботи та організації CDN мережі

В наш час використання мереж передачі даних для обміну інформацією, є надзвичайно поширеним та активно застосовується у всіх сферах повсякденного життя. Кількість мережевих пристроїв, давачів, які використовують доступ до мережі передачі даних та Інтернет, є дуже великою, а обсяги даних, які вони генерують, суттєво збільшуються з часом. Все це призводить до того, що актуальними постають питання, які стосуються ефективності використання мережевих та інфраструктурних ресурсів, технологій ефективної передачі та обробки даних, розподілу та паралельної обробки навантаження та швидкості і гарантії доставки інформації кінцевим користувачам [21-24].

В ситуації яка складається, сервери, які є джерелом контенту та на яких виконується його обробка не завжди здатні витримувати навантаження в певні моменти часу та забезпечувати задовільну якість послуг кінцевим користувачам. Раніше досить часто можна було зустріти використання «серверних ферм», які представляли собою групу різних типів серверів, що знаходяться в одній географічній локації. Станом на сьогодні такі поняття зустрічаються вкрай рідко, та в цілому відійшли на другий план. Такі рішення були замінені технологією хмарних рішень і розподіленими мережами доставки контенту а також граничними розподіленими обчисленнями.

Основним завданням для провайдерів послуг та контенту, які виходять на перший план, є побудова масштабованої та розподіленої мережевої інфраструктури в умовах постійного зростання трафіку, а також організація розподілу контенту по різних географічно розподілених локаціях.

На сьогоднішній день існує багато технологій, які забезпечують надійність та контроль за якістю доставки даних, а також забезпечують можливість ефективно використовувати ресурси мережі та обслуговуючих пристроїв, які виконують обробку даних [25-29].

Одним із таких рішень є технологія доставки контенту CDN, яка працює по принципу кешування контенту та вибору найкращого, доступного для користувача граничного сервера. Що стосується кешування контенту та вибору сервера передачі, тут існує багато методів та способів організації [30]. В роботі запропоновано метод, який забезпечує покращення параметра ефективності кешування в межах CDN мережі.

Мережа доставки контенту - це географічно розподілена система передачі даних, що включає в себе багато серверів обробки та трансляції контенту, а також мережеві маршрути. Граничні сервери CDN мережі об'єднуються в розподілену систему з використанням оптичної транспортної мережі, що забезпечує високу швидкість передачі даних і мінімальні затримки [31-32]. Це дозволяє значно скоротити маршрут між користувачем і сервером трансляції контенту, що забезпечує більшу надійність доступу до ресурсів [33].

Основними перевагами використання CDN є:

- зростання швидкості передачі даних до кінцевого користувача. Користувачі із будь якої локації мають отримати дані, які вони запитують із збереженням максимальної якості та при цьому використовуватимуться оптимальні маршрути для транспортування;
- зниження навантаження на кореневий сервер. Всі статичні дані будуть зберігатись на кешуючих серверах і всі запити будуть опрацьовуватись як правило саме ними. На кореневому сервері в будь який момент часу буде залишатись тільки динамічний контент. Для прикладу, якщо мережа CDN використовується для трансляції відеоконтенту, то контент, який іде в реальному часі, транслюватиметься з основного сервера,

а весь інший, наперед записаний контент, віддаватиметься серверами кешування.

- використання сервісів обміну файлами для передачі даних стає все більш поширеним. Багато користувачів використовують онлайн сховища для обміну інформацією. Зазвичай, у таких сховищах зберігаються великі файли відео, аудіо та фотографії. Використовуючи мережі доставки контенту (CDN), дані такого формату можуть бути доставлені до користувача в будь-яку точку світу із збереженням якості самого контенту.

Базова схема організації мережі CDN представлена на рис. 2.1

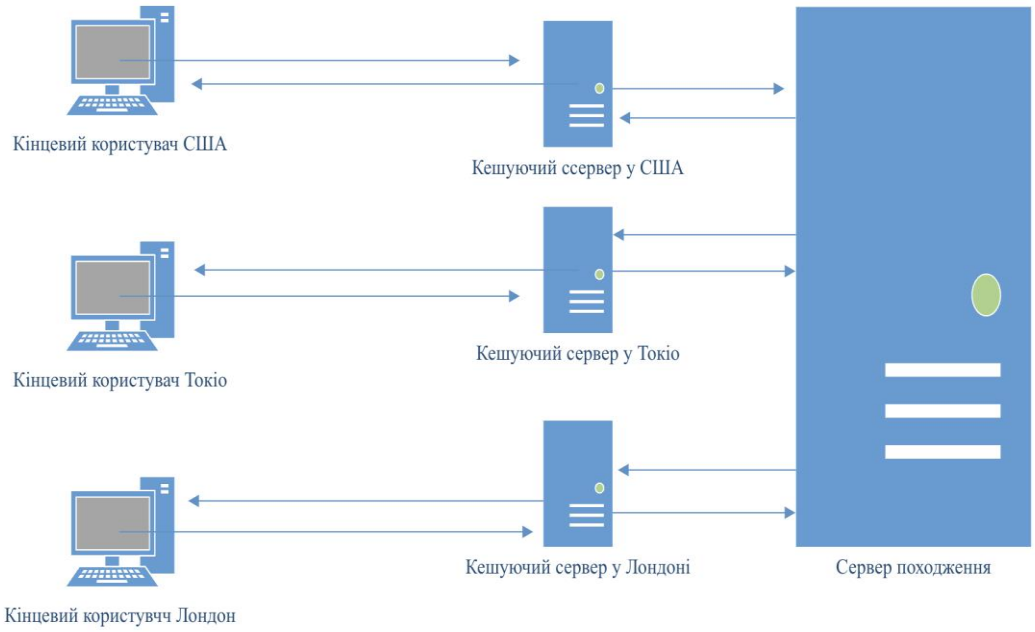


Рис.2.1. Базова схема організації CDN мережі

Кінцеві користувачі запитують дані від певного постачальника сервісу. Мережа CDN визначає місцезнаходження користувача, використовуючи при цьому IP адресу клієнта, та направляє запит до найближчого кешуючого сервера. Кеш-сервер перевіряє, чи є дані доступні в його локальному кеші. Якщо так, він відразу надає контент користувачу. Інакше запит надсилається до сусіднього кеш-сервера або до кореневого(Origin) сервера. Отримані від кореневого сервера дані

передаються користувачу, а також зберігаються в кеші для майбутніх запитів. Ця схема є досить простою, але має певні недоліки з точки зору безпеки. Один з основних недоліків полягає в тому, що доступ до кореневого сервера можливий з будь-якої локації. Крім того, така схема завжди призводить до більшого навантаження на кореневий сервер, оскільки кеш-сервери звертаються до нього у випадку відсутності даних у їхньому кеші [34-37].

Для того щоб виправити всі недоліки такої реалізації, в CDN мережах використовують технологію екранування(Shielding) CDN. Схема роботи із використанням цієї технології представлена на рисунку 2.2.

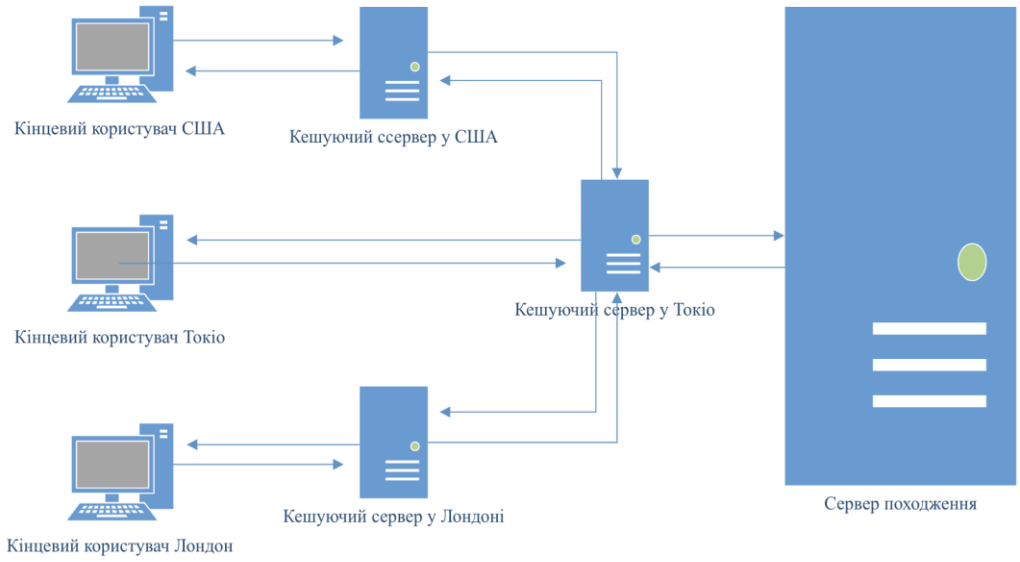


Рис.2.2. Схема роботи CDN мережі із використанням екранування сервера походження

У цій схемі особливість полягає в тому, що доступ до кореневого сервера дозволений лише з одного місця. Всі інші кеш-сервери з будь-яких інших місць будуть отримувати контент від проміжного кеш-сервера, який має свої власні правила кешування та звертається до сервера першоджерела лише тоді, коли не має необхідних даних у своєму кеші. На рисунку 2.2. цю

функцію виконує кеш-сервер у Токіо. За рахунок того, що кеш-сервери з будь-яких місць будуть звертатися до проміжного сервера вдається зменшити навантаження на сервер походження. Цей проміжний сервер, в більшості випадків, буде мати заздалегідь кешований контент. Ще однією перевагою даного методу організації CDN мережі є безпека, оскільки доступ до кореневого сервера доступний лише з одного місця, а у деяких випадках навіть однієї IP адреси [38-41].

Для отримання швидкої відповіді всі запити кінцевих користувачів, будуть направлені до найближчого або ж найменш завантаженого кешуючого сервера. Як правило основний механізм роботи CDN полягає у правильному виборі сервера віддачі контенту для кожного користувача, базуючись на його локації. Серед поширених методів вибору можна виділити наступні:

- вибір найближчого до користувача сервера, що базується на службі доменних імен DNS;
- визначення ступеня завантаженості кожного кешуючого сервера, та вибір сервера, який найменш завантажений та найближче до локації користувача;
- вибір сервера, що дасть змогу забезпечити мінімальне значення часу затримки при передачі даних на шляху до кінцевого користувача [42-43].

Всі ці методи, безумовно є важливими та потрібними для організації ефективного використання ресурсів вузлів доставки контенту та мережі передачі даних [44-45].

2.2. Схема маршрутизації запитів в мережі CDN

Використання CDN (мереж доставки контенту) контент-провайдерами сприяє підвищенню швидкості завантаження цифрового статичного контенту, такого як аудіо, відео, програмного забезпечення та

ігор, для користувачів Інтернету у точках, де доступна мережа CDN. Основна мета CDN - максимально наблизити контент до кінцевого користувача [46].

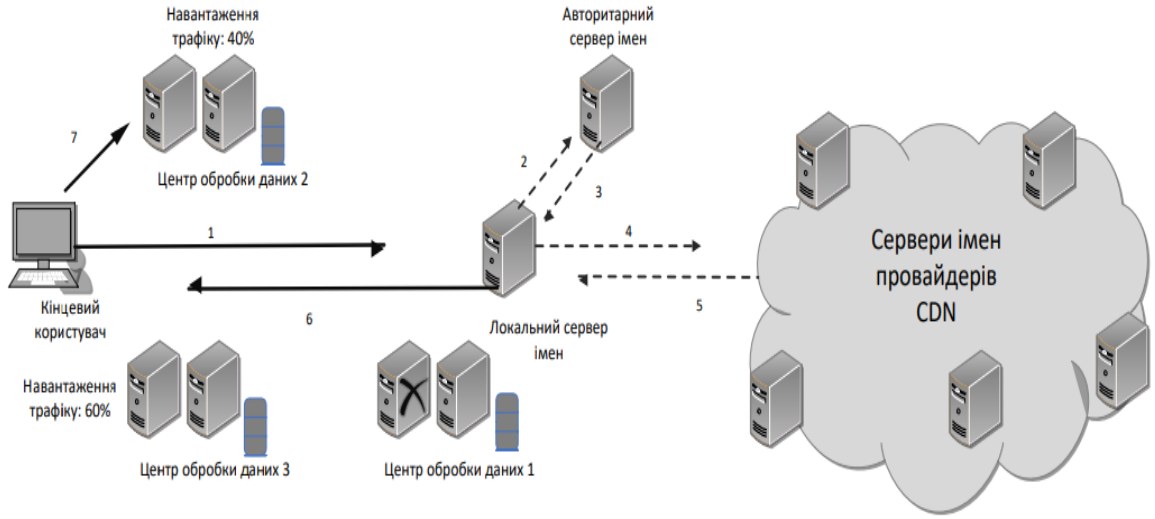


Рис.2.3. Схема маршрутизації запитів користувача у мережі доставки контенту CDN

Процес отримання статичних даних через CDN можна представити у кілька етапів (рис.2.3)

- Користувач ініціює процес отримання доступу до ресурсу (веб-сайт чи відеододаток) і його клієнт (в більшості випадків браузер) формує запит до DNS сервера для перетворення доменного імені в IP-адресу.
- Використовуючи стандартні механізми роботи служби DNS, клієнтський сервер надсилає запит на отримання IP-адреси до авторитетного (англ. *authoritative*) DNS.
- Авторитетний DNS сервер формує відповідь, включаючи в цю відповідь запис CNAME із посиланням (ip-alias) на ресурс CDN провайдера.
- Наступним кроком клієнтський DNS звертається до DNS сервера CDN провайдера, та очікує отримати адрес найближчого сервера для доставки контенту з його локації
- Формуючи відповідь, DNS сервери CDN провайдера надсилають список ip-адрес кешуючих серверів із локації користувача (або

найближчої до неї).

- DNS сервер клієнта віддає одну або декілька адрес клієнту(в найпростішому випадку це браузер)кінцевого користувача.
- Отримавши ip-адресу кешуючого сервера, клієнт, який використовується для перегляду контенту, звертається за цим контентом до конкретного кеш-сервера.

Варто також зазначити, що під час вибору сервера, до якого в подальшому будуть направлені запити користувачів із списку доступних до уваги береться, завантаженість та доступність серверів, які містять даний контент [47-48].

Основні сервіси, які використовують CDN:

Http-кешування. HTTP-контент автоматично зберігається на серверах розподілу і надсилається користувачам з найбільш підходящого сервера за їх запитом.

Відеострімінг за запитом (Video on Demand Streaming) - це можливість перегляду відеоконтенту у будь-який час, що зручно для кінцевого користувача. Він дозволяє переглядати відео з будь-якої точки без обов'язкового завантаження всього вмісту [49].

Трансляція відео в реальному часі (Live Streaming) - це можливість перегляду відеоконтенту в режимі реального часу. Це може бути трансляція з сервера постачальника контенту або безпосередньо з IP-камери. Іншим важливим аспектом Live Streaming є трансляція прямого ефіру телеканалів через Інтернет [50-52].

Кодування відео в режимі реального часу. Конвертація та перетворення файлів з одного формату у інші формати чи контейнери [53].

Перекодування відеопотоків. Перетворення типу контейнера, кодека (наприклад, MPEG2 в H.264), а також зміна бітрейту, структури відеокадрів чи розміру відео [54-58].

Варто зауважити, що, окрім значного зменшення навантаження на

кореневі сервери додатків, CDN мережа виконує інші важливі функції. Всі сучасні постачальники розміщують копії своїх додатків на всіх доступних для них кеш-серверах. Це призводить до помітного скорочення затримок у доставці даних кінцевому користувачеві та покращення якості надання послуг. Суттєве скорочення часу доставки контенту є однією із значущих переваг використання CDN мережі. Крім того, використання CDN дозволяє значно заощаджувати трафік. Передача файлів на інший континент один раз, їх зберігання на місці на локальному сервері та подальше розповсюдження через місцеві канали є більш вигідним, ніж передача того самого трафіку тисячі разів через міжконтинентальні канали передачі даних. Усі ці стратегії необхідні для оптимального використання ресурсів мережі та вузлів доставки контенту [59-61].

Для того, щоб забезпечити можливість швидко віддавати контент з кешуючого(Edge) сервера, без додаткових запитів за ним до кореневого(Origin) сервера, потрібно щоб цей контент був завантажений на кеш сервер (і залишався там). Існує багато схем організації ефективного кешування даних, найпоширенішими серед яких можна зазначити наступні:

- *Реплікація всього контенту.* Цей підхід має важливу перевагу у швидкості, включаючи першу відповідь. Однак його недоліком є високі витрати на реалізацію.
- *Реплікація за першим зверненням* (найбільш розповсюджена схема) характеризується повільним першим запитом, проте всі подальші запити відбуваються швидше.
- *Асинхронна реплікація* при перевищенні певного порога звернень. Більш економічний варіант, але обслуговування більшості клієнтів є повільним. Поряд з політикою кешування діє політика очищення кешу, що визначає, коли саме об'єкт видаляється з сервера на певній точці. Найпоширеніші підходи включають[62-65]:
- видалення після певного періоду "тайм-ауту", коли до ресурсу

не зверталися;

- видалення, якщо кількість звернень до ресурсу менша за певне значення;
- видалення через фіксований проміжок часу.

2.3. Балансування та розподіл навантаження у CDN

Шляхом постійного відслідковування доступності кеш-серверів в мережі, балансувальник навантаження забезпечує оптимальне розподілення запитів між ними. Варто відзначити, що прості методи балансування, такі як, наприклад, DNS, не здатні ефективно розподіляти трафік, оскільки не контролюють актуального навантаження на кеш-сервери. Для досягнення оптимального розподілу навантаження необхідно, щоб балансувальний пристрій мав інформацію про завантаженість кожного сервера.

Для ефективнішого розподілу навантаження між серверами доставки контенту, розташованими в одному або кількох датацентрах, рекомендується використання хмарної інфраструктури (cloud). Такий підхід дозволяє рівномірно розподіляти трафік між усіма серверами доставки контенту та ефективно перерозподіляти навантаження у випадку недоступності будь-якого з цих серверів [66].

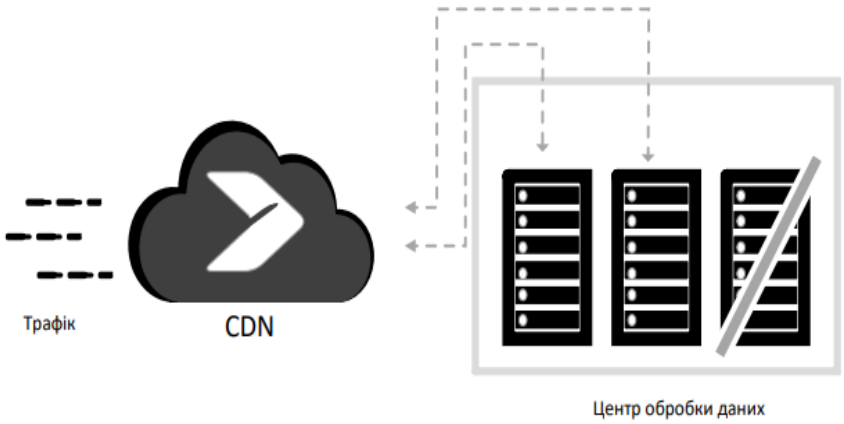


Рис.2.4. Балансування навантаження на рівні Cloud провайдера

Відповідно до рис. 2.4, процес балансування навантаження може бути виконано за двома критеріями. Один метод полягає у регулярному опитуванні серверів доставки контенту для визначення обсягу оброблюваних ними запитів у поточний момент, та направленні запитів на сервери з найменшим навантаженням.

Інший підхід полягає в моніторингу доступних з'єднань на кожному сервері та маршрутизації запитів до сервера з найбільшою кількістю доступних з'єднань.

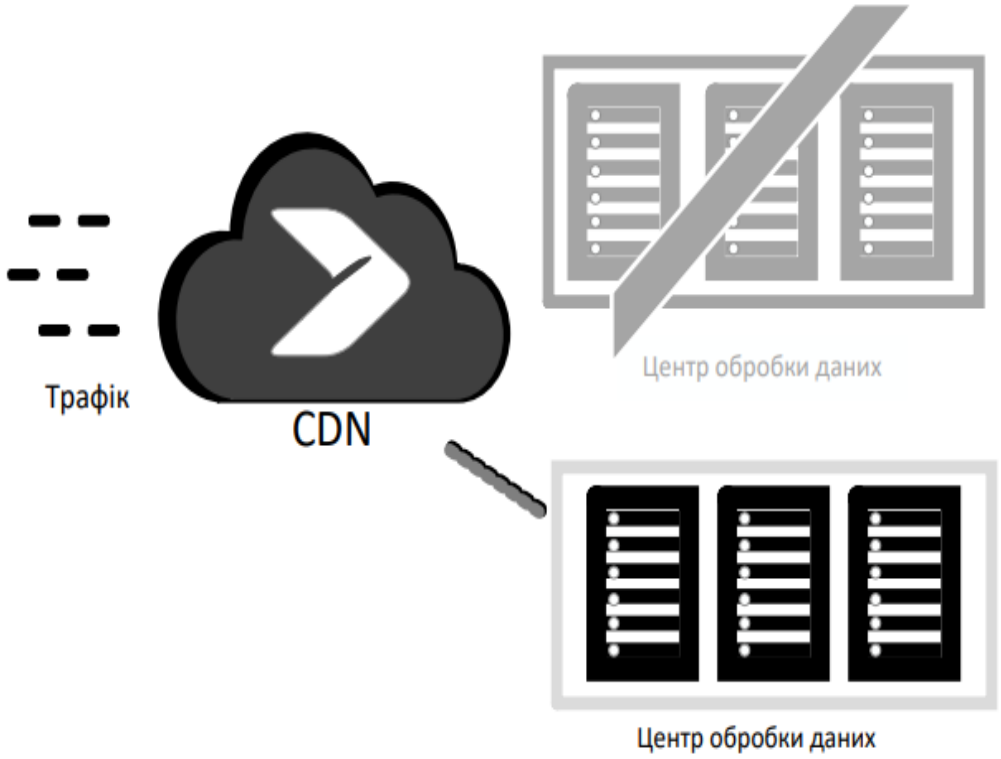


Рис.2.5. Забезпечення відмовостійкості CDN ресурсу

На рис.2.5 показано схему, яка передбачає резервування ресурсів для передачі через CDN. Це дозволяє негайно переключитися на альтернативний датацентр у разі проблем з основним. Очевидно, що забезпечення такого резервування призводить до збільшення фінансових витрат на побудову та обслуговування мережі CDN.

2.4. Метод розподіленої обробки запитів, що потребують значних обчислювальних ресурсів в точках присутності CDN мережі

Доволі часто постають питання про оцінку ефективності використання CDN мережі та її ресурсів. У багатьох випадках, коли користувач запитує один статичний файл, запит перенаправляється до граничного сервера, який, у разі наявності файлу в кеші, відразу надішле його клієнту. Такі типи запитів доволі просто кешувати і ефективність кешування в такому випадку буде досить високою. Однак, часто бувають випадки, у яких користувач запитує не один набір даних, а багато, використовуючи при цьому один запит. Найпростішим прикладом такого запиту може бути сторінка інтернет-магазину із списком товарів в певній категорії. В такому варіанті, в запиті від клієнта буде міститись велика кількість товарів на які потрібно отримати інформацію про їхній опис, характеристики, ціни, наявність на складі, рейтинг, відгуки і інша інформація. Якщо такий запит перенаправити на один граничний сервер, то ймовірність того, що інформація про всі ці товари буде міститися в кеші даного сервер дуже мала. Це означає, що весь цей запит буде перенаправлено на кореневий сервер, який повинен обробити запит та сформувати відповідь для граничного сервера, яка потім направиться клієнту та збережеться в кеші.

Схема проходження такого типу запиту представлена на рисунку 2.6. Як бачимо із схеми, користувач запитує інформацію про список продуктів та їхні деталі, а саме ціна, опис, характеристики, зображення та інше. Чим більше кількість товарів буде в такому запиті, тим меншою буде ймовірність того, що всі такі товари будуть знаходитись в кеші граничного сервера.

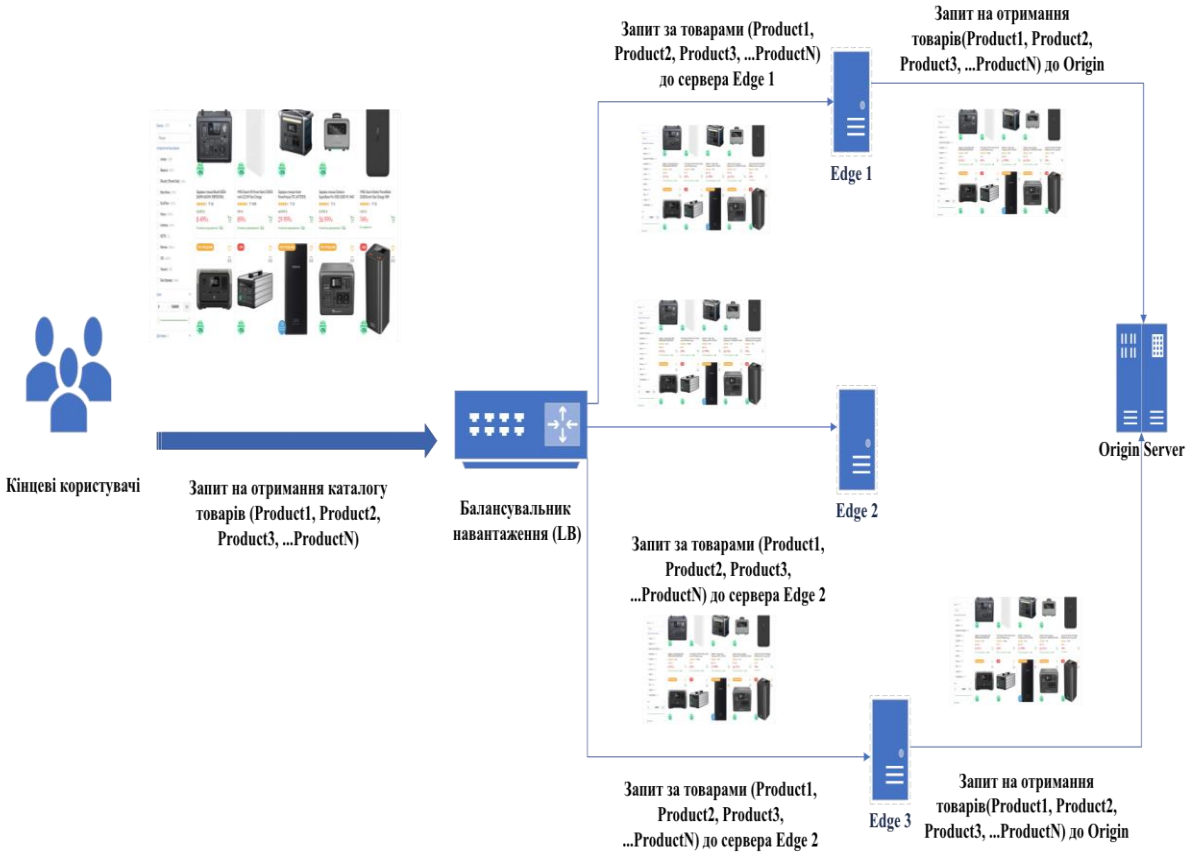


Рис.2.6. Схема проходження запиту на отримання списку товарів через CDN мережу

На рисунку 2.6, кількість товарів становить 10, що є зовсім мало. В такому випадку Load Balancer надсилає запити на кешуючі сервери. В даному представлені, Edge 1 та Edge 3 не мають даних про всі ці 10 товарів в кеші, тому весь запит перенаправляється на кореневий сервер. Сервер Edge 2 містить всі товари в кеші, тому відразу може сформувати відповідь клієнту. Зрозуміло, що чим більше буде таких запитів, та чим більшою буде кількість товарів, тим меншою буде ймовірність того що всі вони будуть оброблені на стороні граничного сервера без участі кореневого сервера. Також слід додати той факт, що чим більше інформації потрібно буде отримати від кореневого сервера, тим складніше йому буде її обробити та надіслати відповідь, оскільки все це вимагає значних обчислювальних ресурсів.

Для того, щоб покращити ефективність кешування та обробки запитів такого типу, а також зменшити навантаження на кореневий сервер, запропоновано схему із розпаралеленням одного запиту, що потребує значних обчислювальних ресурсів, на велику кількість простих. Для прикладу, якщо користувач запитує інформацію про 10 товарів, Product1, Product2, …. Product10, то Load Balancer розділить цей запит із 10 товарів на 10 простих запитів по одному товару. Load Balancer при направленні запиту на граничний сервер буде використовувати алгоритм роботи, який також враховує завантаженість самого граничного сервера та наявність контенту в його пам'яті. Відповідно на граничні сервери будуть надходити запити за окремим товаром. І в такому випадку к-сть запитів до граничних серверів від LB зросте, але тривалість їх обробки знизиться та ефективність кешування даних покращиться. Схема такого розділення запитів представлена на рисунку 2.7.

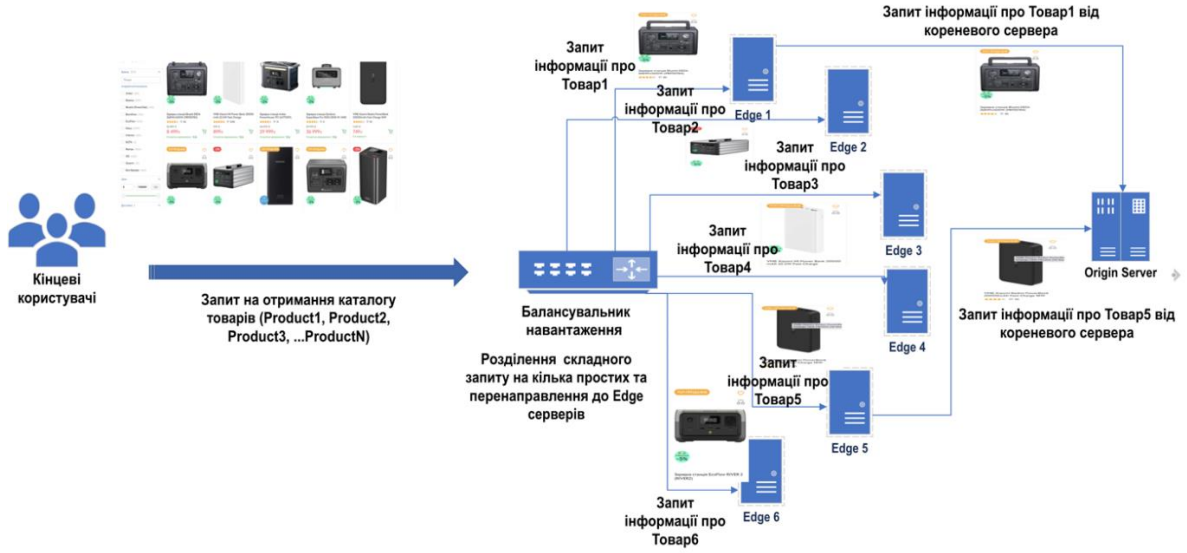


Рис. 2.7. Схема проходження запиту на отримання списку товарів через CDN мережу із розділенням запитів на рівні балансувальника навантаження

На рисунку 2.7 представлено запит користувачем списку 10 товарів. Цей запит надходить на балансувальник навантаження який працює на

рівні граничної локації. Балансувальник навантаження надсилає запити по кожному товару на один з 6 Edge серверів. В даному випадку, тільки два сервери Edge 1 та Edge 5 не знаходять даних про товар в кеш пам'яті та надсилають запит на кореневий сервер. Отже перевагою такого рішення є зменшення навантаження на кореневий сервер, зменшення часу відповіді клієнту навіть при умові збільшення кількості запитів на граничні кеш сервери.

Для перевірки роботи даного методу, було виконано програмне моделювання його роботи на рівні балансувальника навантаження. В процесі моделювання, запити, обробка яких вимагає значних обчислювальних ресурсів, та які містять в собі велику кількість інформації генерувались програмою Apache JMeter. В моделі було використано 1 віртуальна машина, на якій було запущено 10 контейнерів, що виконували роль граничних серверів. Також було додатково використано 1 віртуальна машина, яка виконувала функцію балансувальника навантаження, та 1 віртуальна машина, яка виконувала функцію кореневого сервера. Віртуальна машина граничних серверів та балансувальника навантаження знаходились в одній локації (us-east), а кореневий сервер в іншій, фізично віддаленій локації (us-west). Віртуальні машини створювались в Google Cloud із використанням сервісу Compute Engine. Тип віртуальних машин:

- граничний сервер - e2-standard-8 (8vCPU, 4 Core, 32GB RAM, 100GB SSD Storage)
- балансувальник навантаження - e2-standard-2 (2vCPU, 1 Core, 8GB RAM, 10GB SSD Storage)
- кореневний сервер - e2-standard-2 (2vCPU, 1 Core, 8GB RAM, 10GB SSD Storage)

Результати роботи CDN мережі із розділенням запитів на рівні балансувальника навантаження представлено на рисунках 2.8-2.9.

Рис.2.8. Кількість запитів на граничні сервери в межах однієї локації без використанням розподілених запитів та з розподіленими запитами

На рисунку 2.8 відображено кількість запитів які надходять на кешуючі Edge сервери в межах однієї локації, або одного балансувальника навантаження. Весь інтервал експерименту розділено на два проміжки, перший проміжок це до 6.45 на графіку. В цей проміжок балансувальник навантаження не виконував розпаралелення запитів, що потребують значних обчислювальних ресурсів, на кешуючі сервери, а просто перенаправляв їх в такому форматі в якому запит приходив від клієнта. Починаючи із 6.45 можна побачити різке зростання кількості запитів. В цей проміжок часу балансувальник навантаження почав виконувати розпаралелення запитів, що потребують значних обчислювальних ресурсів, на більшу кількість простих (запит на один товар). Запити перенаправлялись на кешуючі граничні сервери згідно логіки запропонованого методу (рис.2.7). Кількість запитів зросла досить суттєво, більше як у два рази. На рисунку 2.9 можна побачити як змінювався час затримки при обробці запитів клієнтів до моменту розділення запитів на більшу кількість простих, та після цього.

Рис.2.9. Графік залежності часу відповіді від навантаження в момент зростання кількості запитів

Час затримки при обробці запитів після застосування методу розподілу запитів на рівні балансувальника навантаження зменшився у порівнянні з тим, який був при обробці стандартних запитів. Без використання запропонованого методу, значення часу відповіді було на рівні 218мс. Після застосування методу розподілу запитів на рівні балансувальника, значення часу відповіді зменшилось до рівня 200мс, що є на 9% швидше. Це можна пояснити тим, що більша кількість даних була оброблена саме на кешуючих серверах, оскільки вже знаходилась в кеш пам’яті. Тобто запит не доходив до кореневого сервера, а відповідь формувалась прямо на стороні кешуючого Edge сервера. Такий підхід дає можливість покращити не тільки час відповіді а і суттєво знизити завантаженість кореневого сервера. З точки зору кореневого сервера, навіть при умові якщо кількість запитів до нього зросте на початку, то з часом ця кількість зменшиться, оскільки всі вони з часом появляться в кеші. Також важливо врахувати той факт, що з точки зору роботи аплікації, обробка запиту в якому міститься 1 товар буде суттєво швидшою та менш ресурсо затратною ніж запиту в якому міститься 10 товарів. Кількість запитів на кореневий сервер представлено на рис.2.10

Рис.2.10. Кількість запитів на кореневий сервер в межах однієї локації без використанням розподілених запитів та з розподіленими запитами

Рис.2.11. Завантаженість кореневого сервера без використанням розподілених запитів та з розподіленими запитами

Як можна побачити із рисунку 2.10 - 2.11, кількість запитів до кореневого зросла менше ніж на 10%, а завантаженість в перші години зменшилась із 45% до 40%, а потім знизилась до 36%. Незначне зменшення навантаження в перші години можна пояснити тим, що значна частина

даних була відсутня в кеш пам’яті граничних серверів, тому на початку кількість запитів була дещо вищою. Однак, навіть при більшій кількості запитів, навантаження зменшилось, оскільки самі запити потребували менше обчислювальних ресурсів для їхньої обробки. Можна стверджувати, що даний метод є ефективний, оскільки дає можливість розділяти запити, які потребують значних обчислювальних ресурсів, від користувачів на велику кількість простих запитів і при тому не створювати надлишкового навантаження на кореневі сервера та за забезпечувати зниження часу відповіді сервісу і тим самим покращувати якість сприйняття послуг кінцевим користувачем.

2.5. Оцінка ефективності кешування даних в мережі CDN

Окрім швидкої доставки даних до користувача та забезпечення надійності передачі даних, CDN технологія також повинна забезпечувати ефективність кешування даних та використання ресурсів мережі і кешуючих серверів.

2.5.1. Інтегральний ключ кешування.

Кешування даних в CDN, означає збереження копій статичного або динамічного контенту (наприклад, зображень, відео, HTML-сторінок, файлів JavaScript і CSS) у пам’яті на розподілених граничних серверах (Edge-серверах), розташованих у різних точках по всьому світу. Це дозволяє користувачам отримувати дані з найближчого до них сервера замість завантаження із основного (кореневого) сервера, що суттєво знижує затримки доставки даних та зменшує навантаження на основний сервер. Для ідентифікації даних в кеш пам’яті використовується ключ кешування.

Ключ кешування - це унікальний ідентифікатор, який використовується для пошуку даних у кеші. Коли надходить запит, система шукає дані в кеші за цим ключем. Якщо ключ збігається із записом у кеші

(cache hit), дані одразу повертаються користувачу; якщо ж ні (cache miss), виконується запит до кореневого сервера для отримання даних. Ефективність кешування визначається відношенням кількості запитів, відповідь на які була сформована на граничному сервері(без перенаправлення до кореневого) до загальної кількості запитів які були надіслані. Значення цього показника може знаходитись в межах від 0 до 1, або 0 -100%.

$$E_{cache} = \frac{N_{edge}}{N_{total}}, \quad (2.1)$$

де E_{cache} - ефективність кешування, N_{edge}- кількість запитів, відповіді на які були сформовані на граничному сервері (без звернення до кореневого сервера), N_{total} – загальна кількість отриманих запитів.

У системах кешування, ключ використовується для ідентифікації певного набору даних. Інтегральний (комплексний) ключ формується шляхом поєднання кількох параметрів або компонентів, які унікально визначають цей набір даних. Ключ кешування зазвичай формується на основі даних які передаються в запиті, а також параметрів самого запиту. Для формування ключа, може використовується лише URL-адреса, яка ідентифікує ресурс, або ж для більш складних варіанті також можуть використовуватись додаткові параметри, такі як заголовки, файли cookie, токени аутентифікації та інші службові дані.

На рисунку 2.12 представлено приклад формування інтегрального ключа кешування для веб застосунку. В даному прикладі, ключ містить 6 компонентів, однак їх кількість може бути змінена, в залежності від типу сервісу для якого він буде використовуватись.

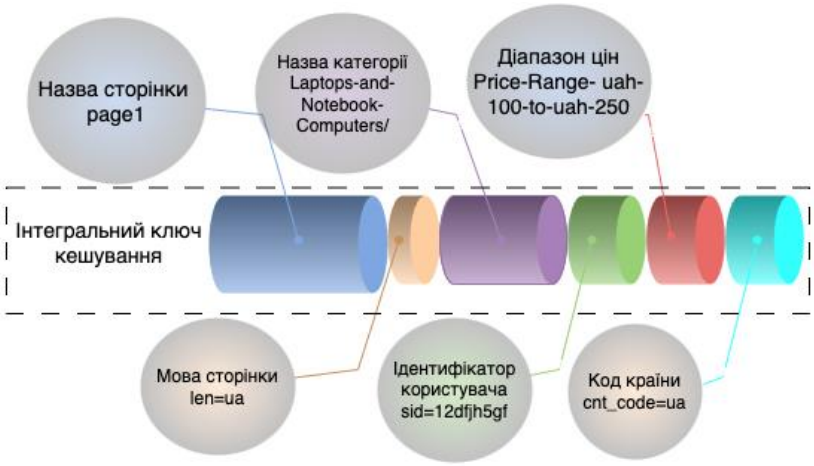


Рис.2.12. Приклад інтегрованого ключа кешування для веб застосунку

Варто зазначити, що ключ кешування використовується і для інших типів мережевого трафіку, для забезпечення ефективного використання закешованих даних. Ще одним прикладом формування інтегрованого ключа може бути сервіс реального часу - Інтерактивне цифрове телебачення. На рисунку 2.13 представлено компоненти які включатиме в себе такий ключ.

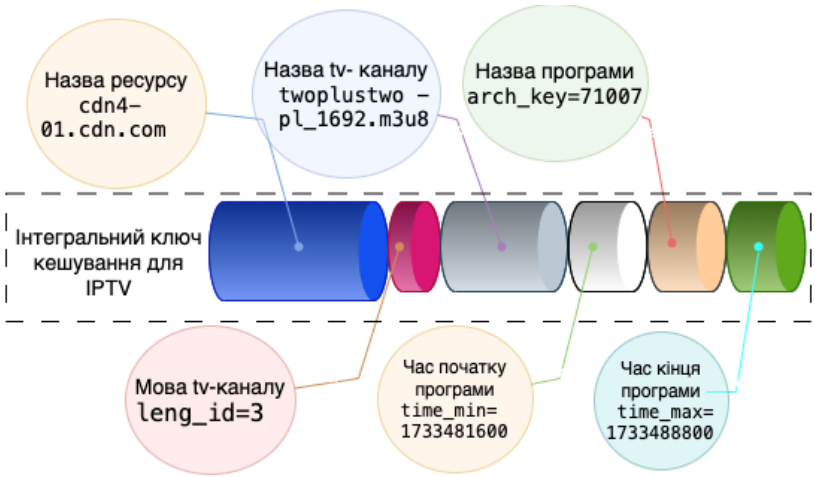


Рис.2.13. Приклад інтегрованого ключа кешування для сервісу iptv

Якщо зробити інтеграцію всіх цих параметрів, то ключ кешування матиме наступний вигляд:

cdn4-01.cdntvc.com/twoplustwo-pl_1692.m3u8?ticket=559225136&leng_id=3&arch_key=71007&time_min=17

33481600&time_max=1733488800

Інтегральний ключ кешування, можна представити наступним математичним виразом:

$$K = R_n + f(C_n, P_n, L_i, T_s, T_e) \quad (2.2)$$

$$f(C_n, P_n, L_i, T_s, T_e) = C_n + P_n + L_i + T_s + T_e \quad (2.3)$$

де:

R_n - назва ресурсу, до якого звертається користувач,

C_n - назва каналу, який переглядає користувач,

P_n - назва програми передач,

L_i - ідентифікатор мови трансляції програми,

T_s - часова мітка початку трансляції програми,

T_e -часова мітка завершення трансляції програми

Значення $f(C_n, P_n, L_i, T_s, T_e)$, може змінюватись та залежить від кількості складових які враховуються при визначенні інтегрального ключа кешування.

Дані, які відповідатимуть цьому ключу будуть виглядати наступним чином:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Cache-Control: no-cache
Content-Type: audio/mpegurl
Date: Sat, 07 Dec 2024 14:06:47 GMT
Expires: Sat, 07 Dec 2024 14:06:46 GMT
Server: nginx
Transfer-Encoding: chunked

#EXTM3U
#EXT-X-VERSION:3
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:19574328
#EXTINF:10.000,
file/twoplustwo-pl/1692/fZFsNxSAuYk0A3YKkNNcUA/1733594609/1733481000/seq-19574328.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/_n25y9D27h8RQN_X4suxGw/1733594609/1733481000/seq-19574329.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/syBo53V1lsXUdYIHk5RW4w/1733594609/1733481000/seq-19574330.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/okA0mYYS36BFJEat5nWl6g/1733594609/1733481000/seq-19574331.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/8ORLiWTRvyEZKbo8vHnPTg/1733594609/1733481000/seq-19574332.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/5OhTkocDjbthONQ1_GMx6Q/1733594609/1733481000/seq-19574333.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/8PrRqw9XWTn8C6pd57-Z-Q/1733594609/1733481000/seq-19574334.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/YCjjRXHkYGAxYpD4vrHFGA/1733594609/1733481000/seq-19574335.ts
#EXTINF:10.000,
file/twoplustwo-pl/1692/7vGCbu7RwLgMnNDbcBp4iw/1733594609/1733481000/seq-19574336.ts
```

Рис.2.14. Приклад даних, які відповідають інтегральному ключу кешування для сервісу iptv

Приклад роботи сервісу інтерактивного цифрового телебачення із можливістю перегляду програм із запису, де може бути застосовано даний інтегральний ключ представлено на рисунку 2.15:

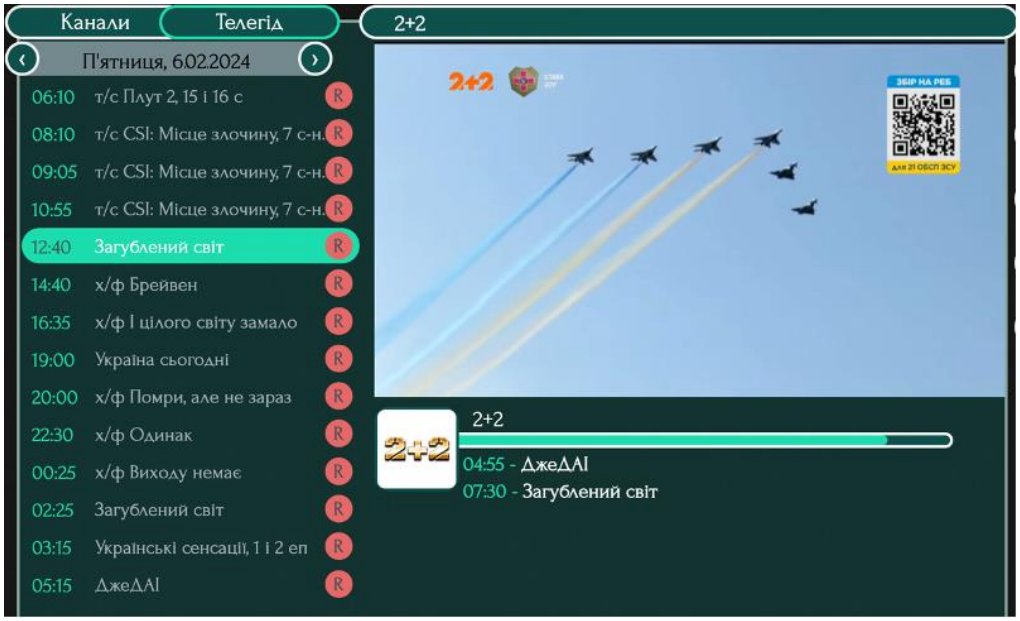


Рис.2.15. Приклад роботи сервісу IPTV із використанням інтегрованого ключа кешування

В даному випадку, можна побачити, що користувач має змогу вибрати окремий канал із списку та окрему програму із доступних в записі для цього каналу. Під час роботи даного сервісу збираються також і статистичні дані, для визначення найбільш популярних каналів серед користувачів певного регіону а також конкретних передач, або ж годин перегляду в які активність користувачів є найвищою. Ці дані в подальшому можуть використовуватись для аналітики, а також для формування інтегральних ключів кешування для різних регіонів.

Схему формування ключа кешування, в залежності від того що переглядають користувачі, та збережених в аналітиці даних представлено на рисунку 2.16.

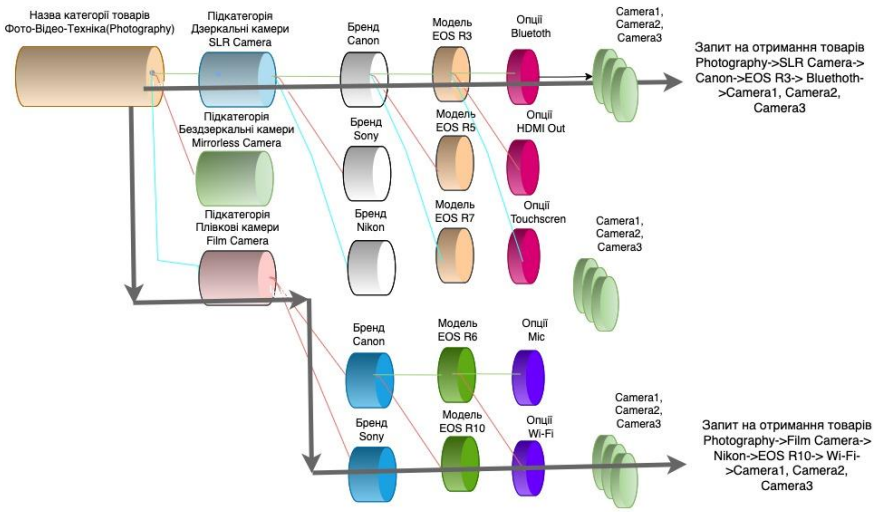


Рис.2.16. Схема формування інтегрованого ключа кешування для веб застосунків

На рисунку 2.16 представлено варіанти, як може формуватись запит користувача на різні категорії товарів, та як це впливатиме на ключ кешування. Кінцевий користувач має змогу вибрати як окрему категорію товарів (Фото-Відео техніка), так і підкатегорію в цій категорії (Дзеркальні камери, Плівкові камери,) а також окремі бренди, моделі та їхні опції і модифікації. В залежності від того, який запит сформує користувач, він надсилається на граничні сервери CDN мережі, які в свою чергу шукають по заданому ключу кешування відповідь в кеші, або ж переадресовують його на кореневий сервер. Також очевидним є факт, що чим більше параметрів (критеріїв) пошуку задасть користувач, тим менше товарів відповідатиме цим параметрам. Відповідно, збільшується імовірність того, що ці товари знаходитимуться в кеш пам’яті на граничних серверах CDN мережі. Отже, можна передбачити, що ефективність кешування сторінок великих категорій товарів(Фото-Відео техніка), які містять сотні або і тисячі товарів буде нижчою, аніж сторінок із підкатегоріями, або ж додатковими фільтрами та критеріями пошуку.

Для підтвердження цього факту, було проведено аналіз ефективності кешування даних для трьох різних сторінок діючого інтернет-магазину. На

кожній сторінці була різна кількість товарів. Перша сторінка - це основна категорія /Photography/Camera , друга сторінка - це підкатегорія першої сторінки /Photography/Camera/Camera-Accessories, третя - це також підкатегорія першої із застосуванням фільтра по бренду виробника /Photography/Camera/Mirrorless-Cameras(Canon). Збір та аналіз даних проводився на протязі 5 днів. Результати аналізу представлено на рисунку 2.17.

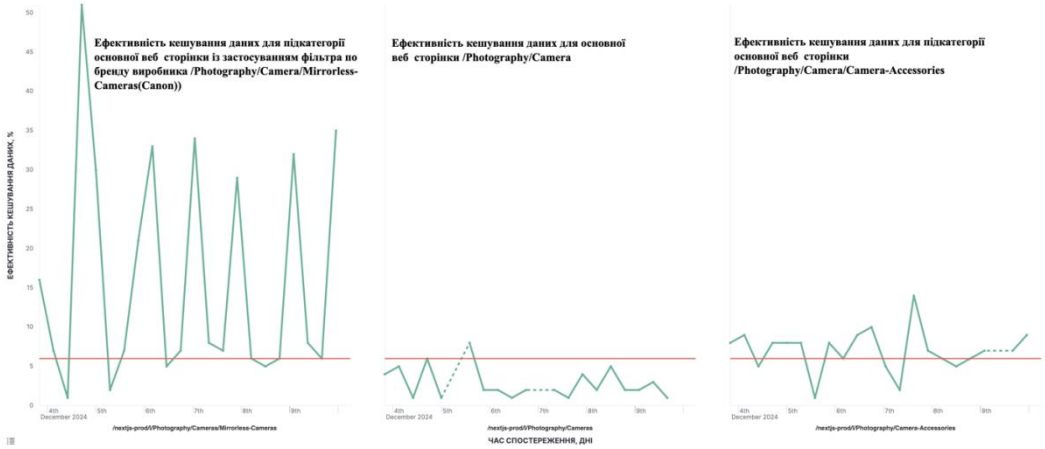


Рис.2.17. Ефективність кешування даних на різних сторінках інтернет-магазину

Результати, представлені на рисунку 2.17, показують, що ефективність кешування даних для сторінок з великою кількістю товарів (Основна категорія /Photography/Camera) є досить низькою і знаходиться в межах 5%, для сторінок підкатегорії - в межах 8%, а для сторінок підкатегорії із додатковим фільтром(назва бренду) - в межах 15%. Це можна пояснити тим, що чим більше товарів відповідає даному критерію пошуку, тим меншою буде імовірність того, що всі вони будуть в кеш пам’яті граничних серверів, або тим більша імовірність, що хоча б один товар буде зміненим і, як результат, видалений з кеш пам’яті. Відповідно, це підтверджує доцільність використання інтегрованих ключів кешування, які складаються із багатьох параметрів, чим дають можливість деталізувати критерії вибору даних, які кешуватимуться як одне ціле.

2.5.2. Використання методу розподіленої обробки запитів на рівні балансувальника навантаження в граничній локації CDN.

Для забезпечення ефективної обробки запитів та досягнення максимального рівня ефективності кешування в схемі мережі CDN пропонується використовувати локальний балансувальник навантаження (Load Balancer) у кожній із локацій (PoP - точка присутності). Схема представлена на рисунку 2.18.

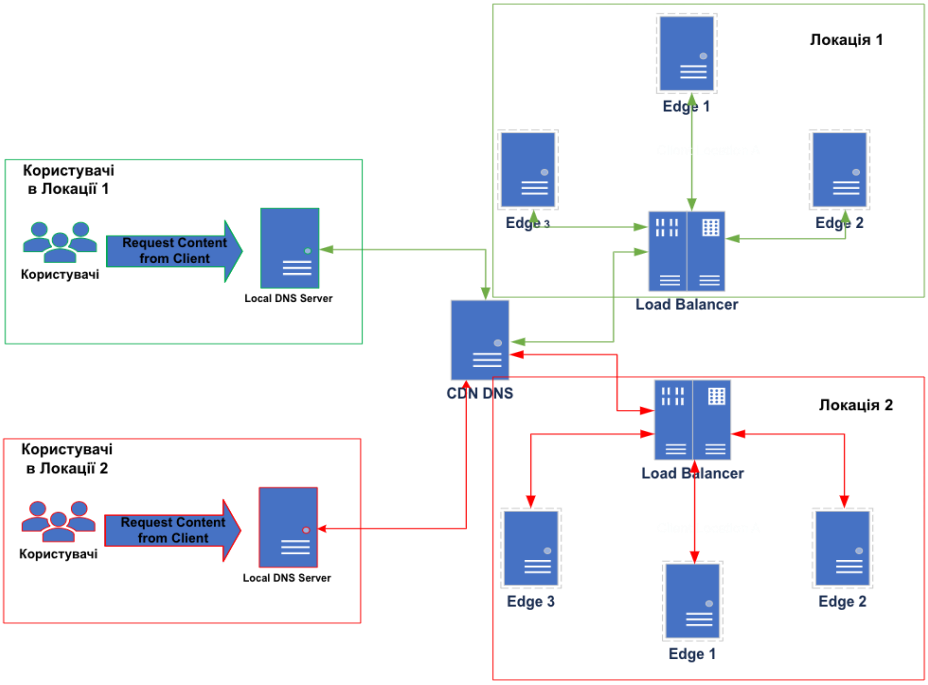


Рис.2.18. Схема CDN мережі із використанням балансувальника навантаження в кожній точці присутності (PoP)

Згідно даної схеми, є дві локації користувачів з локальними DNS серверами та дві локації присутності кешуючих серверів PoP. В кожній із локацій, окрім кешуючих серверів присутній балансувальник навантаження. Всі запити від користувачів будуть направлені саме на балансувальник. Це означає, що DNS сервер CDN провайдера буде завжди оперувати адресами балансувальників навантаження, а не окремих кеш серверів. LB буде перенаправляти запити на окремі кеш сервери (Edge1-3), які вже будуть

віддавати контент напряму користувачу. Сесія передачі даних буде створена між клієнтом та кешуючим сервером. Основна ціль такої схеми із використанням LB - це забезпечити максимально ефективне кешування даних. Яким чином це буде організовано:

- LB буде виконувати ф-ю моніторингу стану кожного кешуючого сервера який є доступним в його локації. Тобто збір метрик про стан роботи сервісів, завантаженість та інші параметри, будуть виконуватись саме на LB сервері.
- LB також буде взаємодіяти із системою розподілу запитів, що вимагають значних обчислювальних ресурсів, а також системою аналітики та маркетингових компаній. Дані цих систем будуть використовуватись у процесі формування інтегрованого ключа кешування, на основі якого буде здійснюватися розподіл запитів між граничними серверами.
- Оскільки LB буде перенаправляти запити користувачів на кеш сервери, то він буде мати інформацію який контент знаходиться на якому сервері. Ця інформація потрібна для того, щоб всі наступні запити, які надходитимуть від інших клієнтів з цієї ж локації за тим же контентом перенаправити саме на той кешуючий сервер, на який був направлений перший запит.
- При такій схемі, контент буде знаходитись вже в кеші Edge сервера і відповідно клієнт отримує його швидше, а також запит до кореневого сервера не буде виконуватись.
- Load Balancer звичайно повинен контролювати завантаженість кеш сервера, оскільки може виникнути ситуація, коли кількість запитів до одного сервера буде досить великою і його завантаженість стане критичною. В такому випадку, LB повинен попередньо закешувати найбільш популярний контент на сусідньому Edge сервері, завантаживши його з основного Edge, та почати перенаправляти запити нових клієнтів на сусідній Edge.

Завдяки такому підходу можна забезпечити максимальну ефективність кешування даних із контролем використання ресурсів.

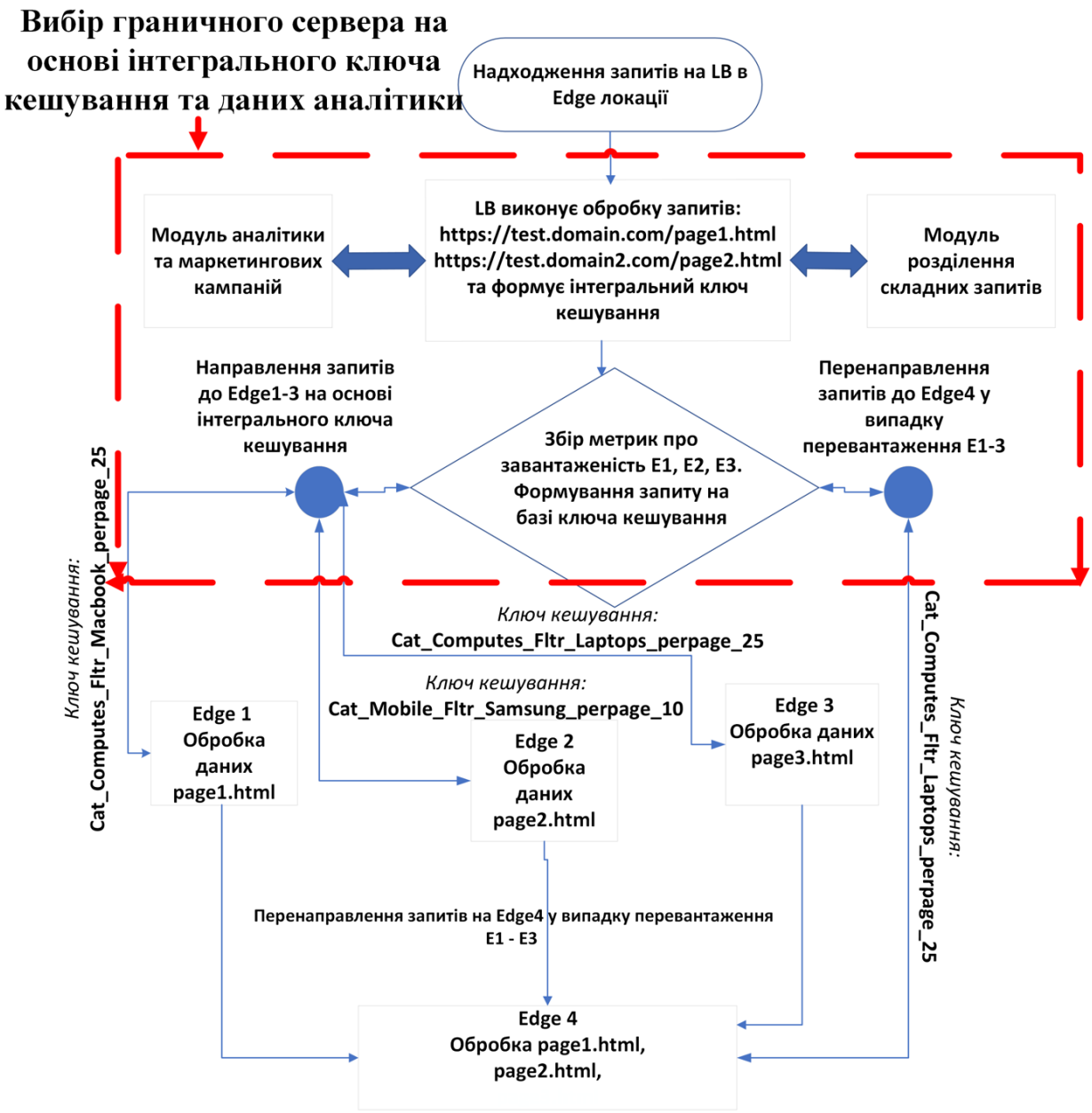


Рис.2.19. Блок схема алгоритму роботи балансувальника навантаження в граничній локації, який забезпечує максимальну ефективність кешування та контроль завантаженості кешуючих серверів

Згідно запропонованого алгоритму було проведено імітаційне моделювання, яке дає змогу порівняти ефективність кешування із

використанням Load Balancer в граничній локації CDN мережі та без нього. При відсутності балансувальника навантаження, запити розподілялись рівномірно між всіма граничними серверами, тобто не бралось до уваги значення ключа кешування. В процесі моделювання, всі запити генерувались програмою Apache JMeter. В моделі було використано 10 віртуальних машин, на яких було запущено по 15 контейнерів, що виконували роль граничних серверів. Також було додатково використано 1 віртуальна машина, яка виконувала функцію кореневого сервера. Віртуальні машини граничних серверів знаходились в одній локації (us-east), а кореневий сервер в іншій, фізично віддаленій локації (us-west). Віртуальні машини створювались в Google Cloud із використанням сервісу Compute Engine. Тип віртуальних машин:

- кореневий сервер – e2-standard-2 (2vCPU, 1 Core, 8GB RAM, 10GB SSD Storage)
- граничний сервер – e2-standard-8 (8vCPU, 4 Core, 32GB RAM, 100GB SSD Storage)

Параметри моделі були наступними:

- тривалість експерименту - 2 дні
- кількість запитів від користувачів - 1000 запитів в годину
- кількість кешуючих серверів -150
- всі користувачі запитували один файл.
- користувачі були унікальні, єдиним спільним параметром була їхня локація. При генерації запитів постійно змінювались User-agent, параметри запиту, додаються додаткові заголовки до запитів.
- періодично виконувався purge контенту, що означає видалення його із кеша на всіх Edge серверах.

Загальна кількість запитів на проміжку експерименту представлена на рис. 2.20. Результати моделювання представлено на рис. 2.21-2.22.

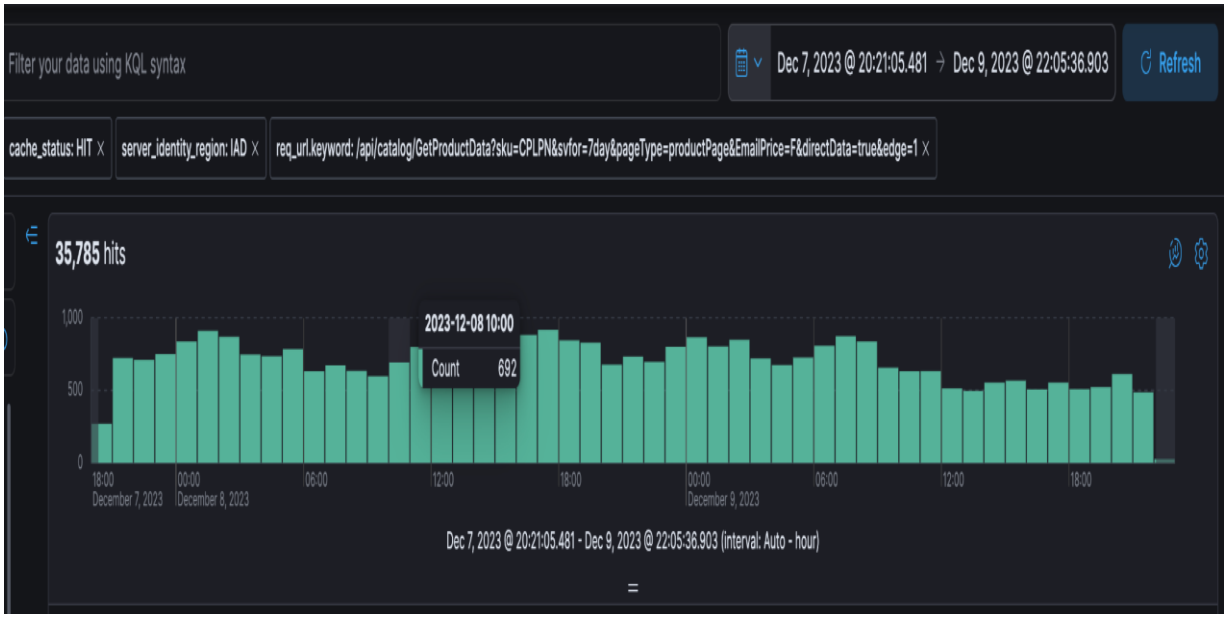


Рис.2.20. Загальна кількість запитів користувачів в граничній локації на проміжку часу моделювання

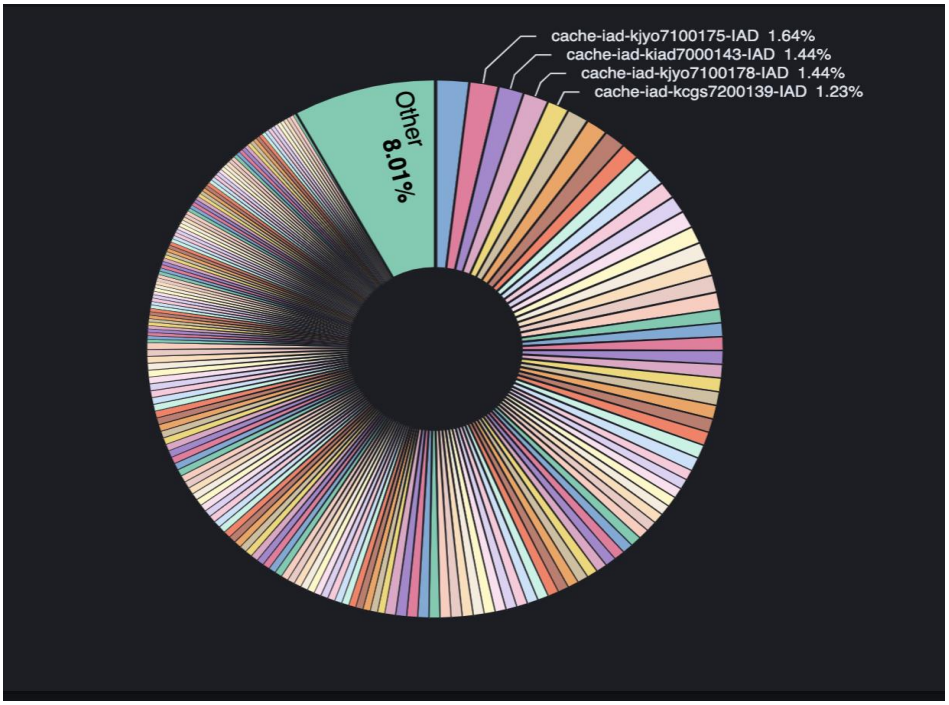


Рис.2.21. Ефективність кешування даних в граничній локації без використанням балансувальника навантаження

На рисунку 2.21 можна побачити яким чином запити від різних користувачів з однієї Edge локації розподіляються між серверами. В даному

варіанті запити рівномірно розподілені між усіма серверами. Як бачимо ефективність кешування становить 1.5% та є приблизно однаковою на всіх граничних серверах. В чому перевага такого підходу - всі кешуючі серери в межах локації, будуть рівномірно завантажені, тобто дуже рідко може траплятись ситуація коли один чи декілька з них будуть перевантажені. Однак серед недоліків такого підходу будуть наступні:

- зростання числа запитів до кореневого сервера, оскільки кожен граничний сервер буде перший раз запитувати контент з кореневого сервера,
- ефективність використання кешованих даних буде низькою. Отримавши контент з кореневого сервера в межах локації, не зовсім правильно є звертатись за цим же контентом ще раз з іншого граничного сервера, в межах цієї ж локації.
- користувачі, які запитують один і той же файл, попавши на одну і ту ж граничну локацію отримуватимуть його не з кеша, а це означає, що час відповіді буде довшим.

Наступний крок моделювання – використання в граничній локації балансувальника навантаження в поєднанні із методом розподіленої обробки запитів та контролем завантаженості кешуючих серверів. Основні функції які виконує балансувальник - це контроль стану та завантаженості кешуючого сервера а також перенаправлення запитів клієнтів, які запитують певний контент на Edge сервер, який вже містить цей контент в кеші. Також, у випадку перевантаженості кешуючого сервера, перенаправити запити на сусідній та ініціювати міграцію контенту з перевантаженого граничного сервера. Варто також зазначити, що у даному випадку не використовується інтегральний ключ кешування. На даному етапі моделювання, у якості ключа кешування використовувалась назва товару, яку формує метод розподілення запитів, що потребують значних обчислювальних ресурсів.

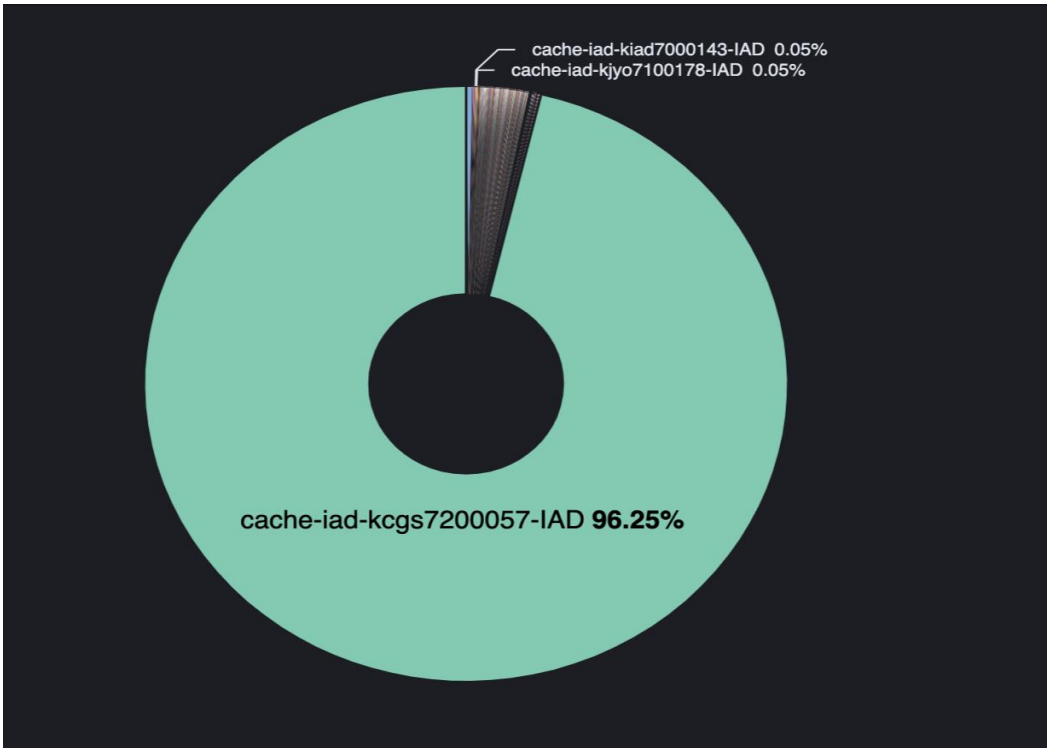


Рис.2.22. Ефективність кешування даних в граничній локації з використанням балансувальника навантаження та контролем завантаженості кешуючих серверів

Як бачимо із рисунку 2.22, ефективність кешування (Cache affinity) при використанні такого методу є досить великою. 96% запитів користувачів із однієї локації, які запитують один і той же контент будуть перенаправлені на один Edge сервер, в нашому випадку це id7200057. До переваг данго методу можна віднести високе значення ефективності кешування, зменшення навантаження на кореневий сервер, а також зменшення часу відповіді для кінцевого користувача. Недоліками такого методу є те, що в межах однієї локації є велика кількість серверів, то розподіл навантаження не буде рівномірним, а також, якщо користувачі запитують різні дані, то ефективність кешування різко знизиться, оскільки не враховується поведінка користувачів та дані про те, які ресурси є найбільш популярними. Що стосується кількості запитів, які перенаправлялись на кореневий сервер від граничних серверів, результати

можна побачити на рисунку 2.23-2.24.

Рис.2.23. Кількість запитів на кореневий сервер під час експерименту без використання балансувальника навантаження

Кількість запитів, яка надходила на кореневий сервер під час моделювання без використання балансувальника навантаження представлена на рисунку 2.23. В середньому число запитів було в межах 200 запитів на годину.

Рис.2.24. Кількість запитів на кореневий сервер під час експерименту з використанням балансувальника навантаження

Під час експерименту із використанням балансувальника навантаження, кількість запитів до кореневого сервера зменшилась практично в 3 рази і становила близько 75 запитів на годину. Ці два графіки дають можливість зрозуміти та оцінити ефективність роботи методу із використання балансувальника навантаження у граничній локації CDN мережі. Варто також зазначити, що балансувальник також всі дані про метрики стану роботи кешуючих серверів може передавати системі моніторингу CDN мережі в цілому.

2.5.3. Використання інтегрального ключа кешування в методі розподіленої обробки запитів на рівні балансувальника навантаження.

Інтегральний ключ кешування - це комплексний показник, який включає в себе сукупність компонентів, кількість яких буде визначатись типом сервісу, для якого він застосовується. Як вже згадувалось в роботі, цей ключ може використовуватись як для даних веб сервісів, таких як статичні та динамічні Інтернет сторінки, так і для сервісів доставки відеоконтенту, таких як цифрове інтерактивне телебачення IPTV.

При застосуванні методу розподіленої обробки даних, один запит користувача може розділятись на декілька запитів на рівні балансувальника навантаження. Запропонований метод розподілення запитів можна розвинути ще більше, зробивши інтеграцію із даними аналітики. Зазвичай, на будь якій сторінці збираються метрики про те, що переглядає користувач, які саме категорії, фільтри товарів, та інші параметри вибору. Всі ці метрики в подальшому надсилаються для аналітики та прогнозування поведінки користувача, наприклад Google аналітика. Маючи такі дані, можна сформувати інтегральні ключі кешування, базуючись на найбільш популярних сторінках інтернет-магазину, які можуть включати в себе назву категорії, фільтри, к-сть товарів на сторінці та навіть рекомендації для

користувача. Інтегральний ключ кешування буде формуватись на основі даних які приходитимуть з аналітики та дасть можливість закешувати основні сторінки з товарами, які користуються найбільшою популярністю серед користувачів. В такому варіанті імплементації, складні запити будуть розділені не на окремі товари, а на групи товарів, які відповідають певним критеріям, які найбільше цікавлять користувачів. Те саме можна запропонувати і для мультимедійних даних, адже будь який провайдер мультимедійних послуг має аналітичні дані про перегляд тих чи інших каналів, програм передач на певному каналі, та часових інтервалів, які користуються найбільшою популярністю серед глядачів.

Ще одним варіантом інтеграції може стати використання маркетингових кампаній. Досить часто великі магазини влаштовують певні кампанії по розпродажі товарів, а також появу нових брендів чи категорій товарів. В таких випадках створюється емейл чи смс розсилки великій кількості користувачів. І очевидним стає той факт, що попит на ці набори товарів різко зростає, а відповідно і кількість запитів від кінцевих користувачів. Зробивши інтеграцію на рівні балансувальника навантаження із маркетинговими кампаніями, знову ж таки можна запити групувати та розділяти більш ефективно, що значною мірою покращить час відповіді, ефективність кешування даних та знизить навантаження на кореневі сервери. Схема такої інтеграції методу розподіленої обробки запитів, із використанням даних аналітики та маркетингових компаній представлена на рисунку 2.25

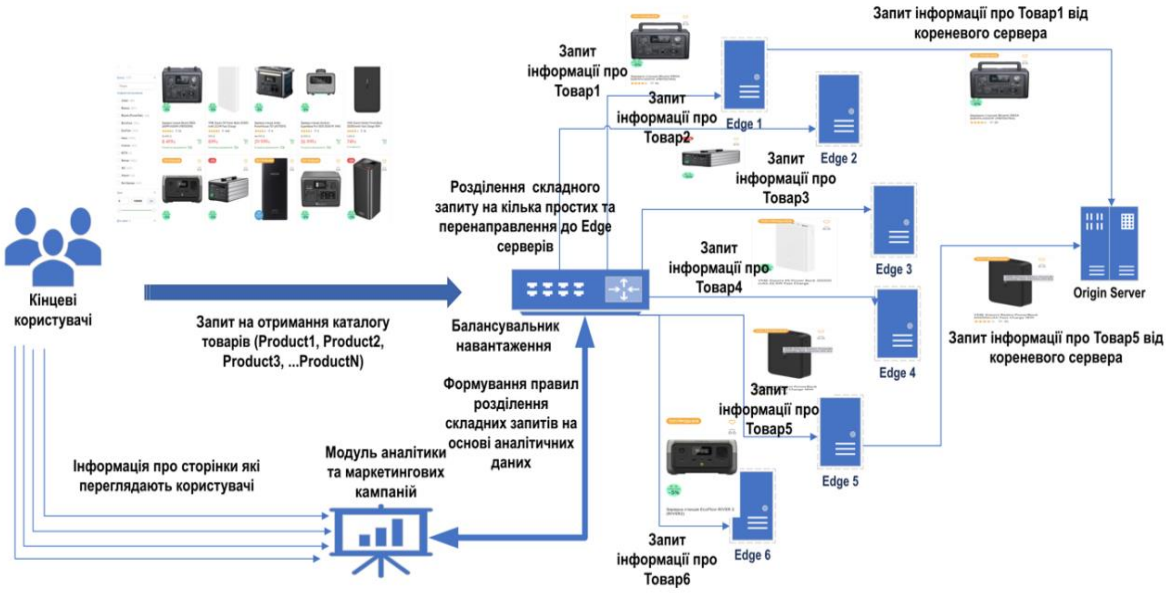


Рис.2.25. Інтеграції схеми розділення запитів, що потребуються значних обчислювальних ресурсів із модулем аналітики та маркетингових кампаній

Балансувальник навантаження отримує аналітичні дані про те, які сторінки (у випадку веб сайту) або ж канали чи програми телепередач (у випадку сервісу цифрового телебачення) є найбільш популярними серед користувачів. Приклад даних з системи аналітики представлено на рисунку 2.26

Шлях до сторінки й клас екрана за останні 30 хвилин

/I/Photography/ Рядків на сторінці: 10 Перейти до: 1 1–10 з 105

	Шлях до сторінки й клас екрана	↓ Активні користувачі	Перегляди
1	/specials/I/Photography/Flashpoint~ORLIT~Lighting-and-Studio	42	80
2	/I/Photography/Cameras	9	10
3	/I/Photography/Lenses/Mirrorless-Lenses	9	18
4	/I/Photography/Lenses	7	7
5	/I/Photography/Lighting-and-Studio	7	9
6	/I/Photography/Camera-Accessories	5	5

Рис 2.26. Дані аналітики про активність переглядів сторінок з певної категорії товарів

На рисунку представлено активність перегляду сторінок певної

категорії за певний інтервал часу. Використовуючи ці дані та метод розподіленої обробки запитів, можна розділяти запити, які надходять на основну категорію на більшу кількість запитів, що входять до неї. Відповідно, це можуть бути запити на підкатегорії, або на підкатегорії із додатковими фільтрами. Для цих розподілених запитів, можна сформувати інтегровані ключі кешування, на основі яких будуть зберігатись дані в кеш пам’яті на граничних серверах. Таким чином, використання аналітичних даних, дає можливість більш раціонально розділяти ресурсозатратні запити користувачів на більш прості з точки зору зберігання в кеш пам’яті та обробки їх на кореневому сервері.

В роботі проведено дослідження, як впливає розподілена обробка даних у поєднанні із інтегральним ключем кешування та даними аналітики на ефективність кешування та час відповіді для кінцевого користувача. Дані аналітики були приблизно спрогнозовані, оскільки для достовірності даних потрібен тривалий час спостереження. Для дослідження було використано сервіс нереального часу, а саме ефективність кешування даних на сторінках фіксованої категорії товарів інтернет-магазину. Результати ефективності кешування представлено на рисунку 2.27

Рис.2.27. Ефективність кешування даних із використанням інтегрального ключа кешування та методу розподіленої обробки запитів

Дослідження тривало 7 днів та було розділено на два часові інтервали. В першому інтервалі 2-5 December метод розподіленої обробки запитів та інтегральний ключ кешування не використовувався. Балансування навантаження в граничній локації здійснювалось на основі TCP-сесії користувача, що означає перенаправлення всіх запитів одного користувача на той самий граничний сервер. Значення ефективності кешування в такому випадку становить 50-60%. На другому часовому інтервалі 5-8 December було використано аналітичні дані за попередні дні та застосовано метод розподіленої обробки ресурсозатратних запитів та інтегральний ключ кешування на рівні балансувальника навантаження. Як бачимо, ефективність кешування зросла до рівня 90%. На рисунку 2.28 показано як при цьому змінювався час відповіді для користувача та кількість запитів до балансувальника навантаження.

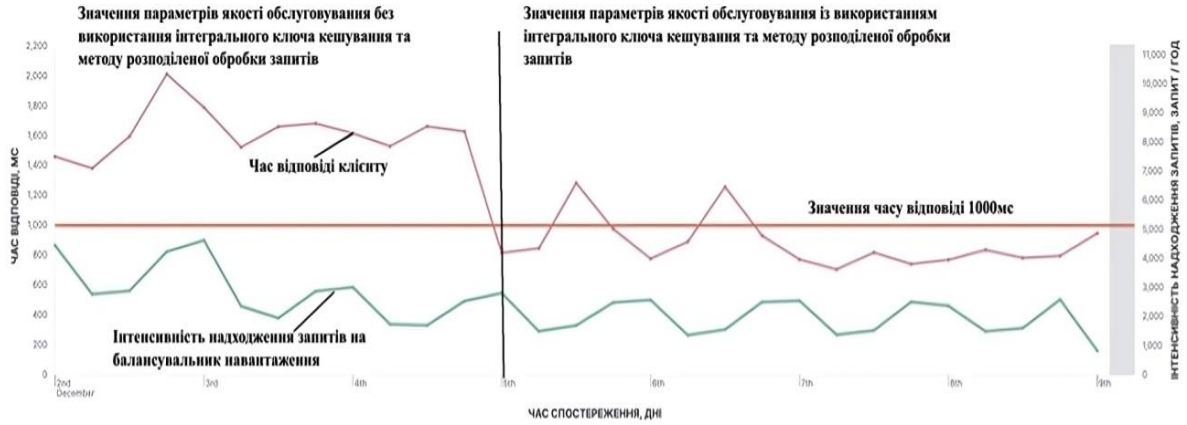


Рис.2.28. Час відповіді для кінцевого користувача із використанням інтегрального ключа кешування та методу розподіленої обробки запитів

На рисунку 2.28 показано, як змінювалися час завантаження сторінки та кількість запитів до балансувальника навантаження до і після використання інтегрального ключа кешування та методу розподіленої обробки запитів. Значення часу завантаження сторінки зменшилось із 1600мс до 800мс. На графіку також можна спостерігати значні сплески часу

відповіді в певні моменти часу. Це свідчить про те, що в ці моменти відбувалось багато змін по товарах, які були присутні на цих сторінках, які використовувались в процесі дослідження та визначали інтегральний ключ кешування. Відповідно, при зміні будь якого із товарів, весь набір даних, який відповідав сформованому інтегральному ключу кешування, видалявся із кеш пам'яті граничних серверів. Також варто зауважити, що час завантаження сторінки досить суттєво впливає на рейтинг веб ресурсу в пошукових системах. Якщо на веб ресурсі буде присутня велика кількість сторінок, час завантаження яких буде значним, тоді пошукова система суттєво понизить рейтинг такого ресурсу. Для прикладу, як виглядає результат низької якості веб ресурсу в пошуковій системі Google:

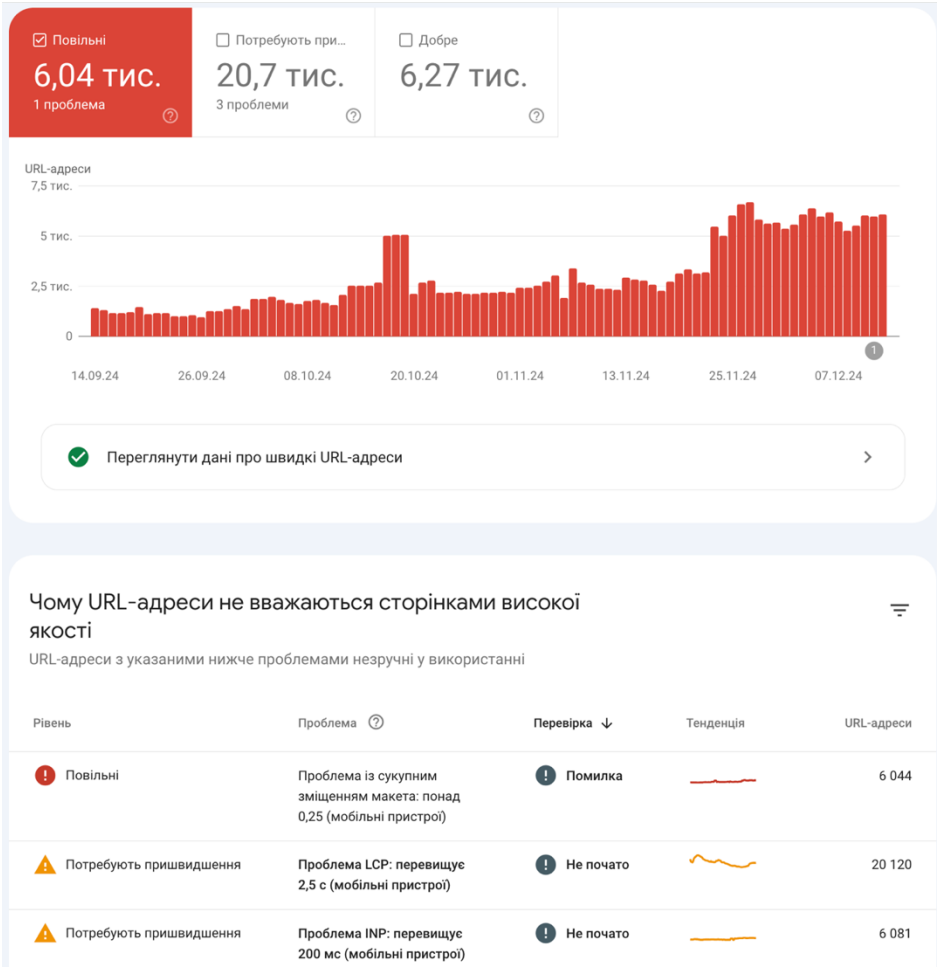


Рис.2.29. Приклад низької якості веб ресурсу у пошуковій системі Google

Як можна побачити основним критерієм, на який звертає увагу пошукова система є структура сторінки та час її завантаження. Саме тому, забезпечення мінімального часу завантаження сторінки веб ресурсів є важливим не тільки для кінцевого користувача, а безумовно і для збереження хорошого рейтингу ресурсу в пошукових системах, які використовуються в глобальних мережах.

2.6. Методи моделювання та аналізу мережевого трафіку

Властивості мультисервісного мережевого трафіку можуть призвести до неточностей та помилок при визначенні параметрів якості обслуговування, використовуючи при цьому стандартні методи розрахунків. У цьому розділі будуть описані основні підходи, які використовуються при моделюванні систем із самоподібним трафіком [67].

2.6.1. Підходи до моделювання систем з мережевим трафіком

Наукові дослідження, проведені протягом останніх років, підтверджують, що мережевий трафік має характер самоподібного (self-similar) або фрактального (fractal) процесу [68].

Термін "самоподібність" відноситься до властивості процесу зберігати свою поведінку та зовнішні ознаки при розгляді в будь-якому масштабі. Враховуючи цю властивість самоподібності мережевого трафіку, стає зрозумілим, що методи моделювання та розрахунку мережевих систем, які базуються на використанні пуассонівських потоків, не відображають повністю та точно того, що відбувається в мережі.

Окрім цього, варто зазначити, що самоподібний трафік характеризується особливою структурою, яка залишається незмінною при збільшенні масштабування. У практичних застосуваннях часто можна спостерігати невеликі сплески активності в мережі при порівняно стабільному середньому рівні трафіку.

Дане явище призводить до погіршення характеристик мережі, таких як збільшення втрат, затримок та джитеру пакетів під час проходження самоподібного трафіку через мережеві вузли. У практичних сценаріях це може спостерігатися через те, що пакети, переміщуючись з великою швидкістю по мережі, часто потрапляють на вузол не поодинці, а групами, що може призвести до втрати пакетів у випадку обмеженого буфера, розрахованого за традиційними методами [69]. Ця особливість мережевого трафіку спричинила значний ріст інтересу до досліджень та публікацій, що стосуються аналізу, моделювання та передбачення самоподібного трафіку [70].

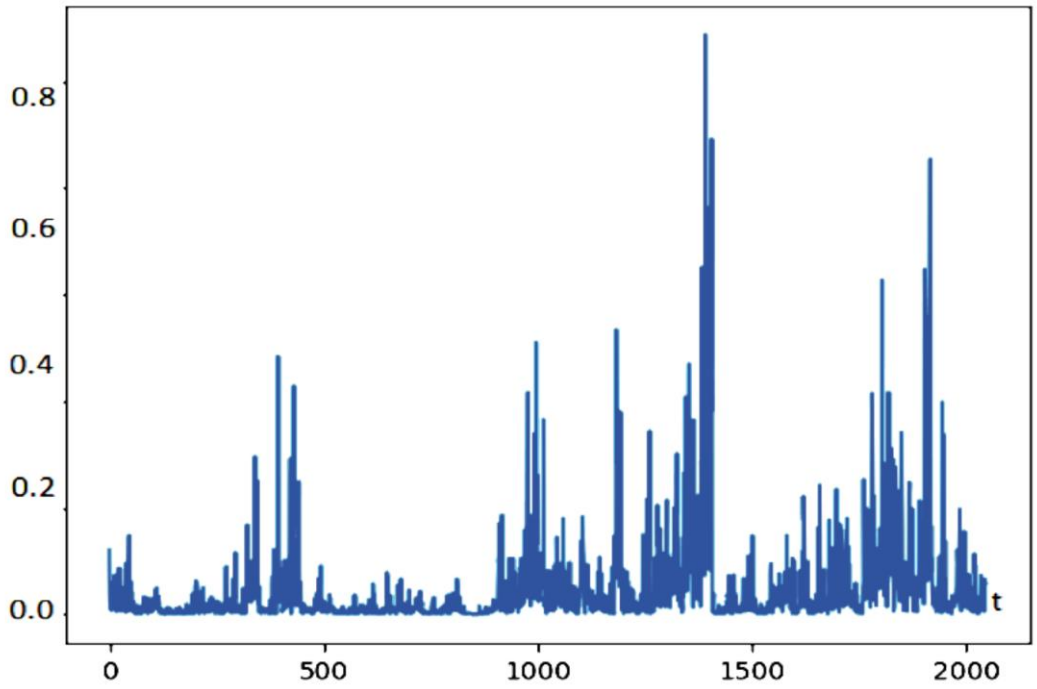


Рис.2.30. Приклад самоподібного трафіку

Незважаючи на широку популярність теми самоподібного трафіку та тривалий час його активного дослідження, слід визнати, що досі існують багато невирішених питань і завдань. Наприклад, досі не існують універсальних, достатньо точних та легко повторюваних методів моделювання та прогнозування поведінки самоподібного трафіку, а також

методів проектування мультисервісних мереж. Дані недоліки стимулюють подальше дослідження та розвиток відповідної області досліджень.

У всіх моделей існує певна структура, яка може бути статичною або динамічною, матеріальною або ідеальною, і вона має схожість до структури оригінального об'єкта. Під час функціонування модель виступає як відносно самостійний квазіоб'єкт. Якщо результати того чи іншого дослідження підтверджуються та можуть бути використані для прогнозування відповідних об'єктів, то говорять, що модель відповідає об'єкту. Варто зазначити, що власне адекватність самої моделі залежить від мети моделювання та критеріїв, які можна вважати прийнятними [71].

Сам процес проведення моделювання передбачає наявність наступних складових:

- об’єкт, над яким проводяться дослідження;
- особу дослідника, яка проводить дослідження та експерименти;
- моделі, що створюється з метою отримати інформації про сам об’єкт, який потрібний для вирішення задачі.

Одним із ключових аспектів у моделюванні систем є проблема визначення цілей. Кожну модель розробляють з урахуванням мети, яку ставить перед собою дослідник, тому однією з основних труднощів у моделюванні є визначення цільового призначення. У якості цілі повинно бути визначено завдання вивчення конкретного аспекту функціонування об'єкта.

Після побудови моделі, одним з основних завдань стає її ефективна реалізація та використання. Наша мета полягає в забезпеченні мінімального часу для отримання результатів і забезпеченні їхньої точності. Важливо враховувати, що модель повинна відображати лише ті закономірності, які є ключовими для дослідження, і ігнорувати ті властивості системи-оригіналу, які на даний момент не є критичними.

Основні вимоги, які ставлять до моделювання:

- Моделі повинні належним чином відтворювати відповідні системи або технологічні завдання. Це означає, що модель повинна відображати реальні умови та параметри, які характеризують об'єкт дослідження або технологічну задачу.
- До того ж, важливо забезпечити необхідну точність у відтворенні процесів та результатів. Це дозволить дослідникам отримати достовірні дані та здійснити точні прогнози або аналіз.
- Під час створення моделі також слід враховувати зручність для користувача - технологічного фахівця або аналітика. Це означає, що інтерфейс управління моделлю повинен бути інтуїтивно зрозумілим, щоб спростити процес роботи з нею.
- Потрібно також забезпечити достатню швидкість роботи програми, щоб користувач міг оперативно отримувати результати та взаємодіяти з моделлю.
- Крім того, важливо, щоб результати моделювання були наочними та зрозумілими, щоб користувач міг легко аналізувати та інтерпретувати отримані дані.
- Нарешті, не менш важливою є ефективність витрат на розробку та використання засобів моделювання, щоб забезпечити їх доступність та використання у широкому спектрі задач і досліджень.

Оскільки мультисервісні мережі володіють трафіком, який має характер самоподібності, використання відповідних методів моделювання стає важливим завданням. На сьогоднішній день існує значна кількість розроблених моделей, призначених для ефективного імітування цього фрактального трафіку [72-73].

Для успішного моделювання трафіку у мультисервісних мережах необхідно враховувати його самоподібність та фрактальні властивості. Розроблені моделі дозволяють здійснювати дослідження та аналіз трафіку з певною точністю та надійністю. Крім того, вони враховують різноманітні

сценарії використання мережі, що дозволяє адаптувати модель до конкретних умов та потреб користувачів.

Проводячи аналіз моделей для моделювання самоподібного трафіку, можна виокремити наступні підходи:

Фрактальний рух Броунівського типу (Fractional Brownian Motion - FBM): Ця модель ґрунтується на випадковому процесі, який починається в початковій точці з незалежними невеликими гаусівськими приростами. FBM може бути аналітично описаний. Крім того, для генерації FBM широко використовуються алгоритми, такі як алгоритм випадкового переміщення середньої точки (RMD) та алгоритм послідовного випадкового додавання (SLA).

Фрактальний гаусівський шум (Fractional Gaussian Noise – FGN) є стаціонарним стохастичним процесом з визначеними параметрами, такими як середнє значення, дисперсія і параметр Херста, а також має певний вид автокореляційної функції. Параметр Херста визначає ступінь фрактального масштабування FGN. Одним з основних викликів у роботі з FBM та FGN є вибір оптимальних значень параметрів для отримання синтетичного трафіку, який максимально точно відтворює експериментально отримані реалізації трафіку.

Моделі ON/OFF розглядають трафік як комбінацію джерел, що його генерують. Кожне джерело має свою структуру: протягом певного періоду часу воно може генерувати пакети інформації (ON-періоди), після чого настає OFF-період, коли генерація припиняється. Розмір ON- і OFF-періодів є випадковим, але має скінченне математичне очікування та безкінечну дисперсію. Зазначимо, що ON/OFF процес відносять до фрактальних процесів відновлення, які чергуються.

Моделі, побудовані на базі систем масового обслуговування.

Зазвичай такі моделі ефективно відображають трафік з пуассонівськими потоками. Однак, модель M/G/∞ може створювати

приблизно самоподібний трафік, регулюючи "хвіст" будь-якого розподілу обслуговування користувачів, що призводить до довготривалої залежності.

Крім того, варто згадати інші підходи, такі як моделі, що використовують нечітку логіку, нейромережеві моделі, фрактальний рух Леві (Fractional Leva Motion - FLM), мультифрактальні моделі (Multifractional – MF), вейвлет моделі (Wavelet Models) [74], а також моделі, засновані на техніці "динамічного моделювання Маркова".

2.6.2. Математичне представлення самоподібного (фрактального) процесу

Стохастичний процес X(t) вважається статистично самоподібним з параметром Херста (0.5≤H≤1), якщо для будь-якого додатного числа a відбувається таке, що процеси X(t) та a-HX(at) матимуть однакові розподіли.

$$\{X(t_1), X(t_2) \dots X(t_n)\} \sim \{a^{-H}X(at_1), a^{-H}X(at_2) \dots a^{-H}X(at_n)\} \quad (2.4)$$

в даному виразі відношення ~ означатиме асимптотичну рівність у відношенні розподілу. На практиці, самоподібність грунтується на тому, що повинні виконуватись наступні умови:

- усереднене значення

$$[X(t)] = \frac{E[X(at)]}{a^H} \quad (2.5)$$

- дисперсія

$$ar[X(t)] = \frac{Var[X(at)]}{a^{2H}} \quad (2.6)$$

- функція автокореляції

$$(t, \tau) = \frac{R(at, a\tau)}{a^{2H}} \quad (2.7)$$

де H – параметр Херста, який відображає ступінь самоподібності випадкового процесу. Якщо значення в межах H<=0.5 - це показує відсутність самоподібності у випадкового процесу, а значення H, які близькі до 1 показуть, що процес володіє високою ступінню самоподібності. Це

означає, що якщо стохастичний процес з довготривалими залежностями демонструє тенденцію до зростання (або зменшення) значень у минулому, то з високою ймовірністю ця тенденція також буде спостерігатися у майбутньому. Враховуючи це, можна припустити, що лінійні зміни в процесі відбуватимуться з аналогічною напрямленістю у майбутньому, підтверджуючи його стабільність або тенденцію до змін в тривалій перспективі.

2.6.3. Математичне представлення дискретного самоподібного процесу

Розглянемо часовий процес $X = \{X_n, n \in Z^+\}$ та визначимо інший часовий процес (m-aggregated) $X^{(m)} = \{X_n^{(m)},\ n \in Z^+\}$ за допомогою усереднення вихідного процесу на суміжні блоки, що мають довжину m, та не перетинаються:

$$X_n^{(m)} = \frac{1}{m}\sum_{i=nm-(m-1)}^{nm} x_i \tag{2.8}$$

Самоподібні стохастичні процеси можна розділити на наступні класи:

- абсолютно самоподібні процеси;
- асимптотично самоподібні процеси [75].

Процес X можна вважати точно самоподібним з певним параметром β (0<β<1) якщо для величини $m \in Z^*$ будуть виконуватись такі умови:

- дисперсія випадкової величини

$$ar\left[X^{(m)}\right]\frac{Var[X]}{m^{\beta}} \tag{2.9}$$

- функція автокореляції

$$\left(k, X^{(m)}\right) = R(k, X) \tag{2.10}$$

Існує взаємозв'язок між параметром β та параметром Херста H, який можна представити у вигляді співвідношенням (вираз 2.11):

$$\beta = 2(1 - H) \tag{2.11}$$

Що стосується іншого класу самоподібних процесів, то процес можна назвати асимптотично самоподібним якщо для великих значень k

- дисперсія випадкової величини

$$Var\left[X^{(m)}\right] = \frac{Var[X]}{m^{\beta}} \tag{2.12}$$

- функція автокореляції

$$R\left(k, X^{(m)}\right) = R(k, X) \tag{2.13}$$

при значеннях m → ∞

Є також результати певних досліджень, які показують , що для двох класів самоподібних процесів дисперсія випадкової величини $Var[X^{(m)}]$ зменшується суттєво повільніше аніж 1/m при умові, що m → ∞ в порівнянні із стохастичними процесами, для яких дисперсія випадкової величини зменшується прямо пропорційно до 1/m і наближається до 0 при m → ∞.

Математично, фрактальний об'єкт характеризується не лише своєю складною структурою, але й дрібною розмірністю. Ця розмірність може бути визначена за допомогою відповідної формули, яка відображає властивості самоподібності та інших характеристик об'єкта. (2.14):

$$d = \frac{logN}{log\left(\frac{1}{r}\right)} \tag{2.14}$$

де N - представляє собою кількість однакових частин, на які треба розділити об'єкт, при цьому кожна частина стане зменшеною копією цілого в 1/r разів.

2.6.4. Опис моделі самоподібного процесу

Існує кілька основних підходів та методів формування самоподібного потоку, і один з найвідоміших був запропонований Мандельбротом.Цей метод базується на суперпозиції декількох незалежних джерел ON/OFF з однаковим розподілом, які чергуються у строгій послідовності і для яких присутній ефект Ноа. Під строго черговими джерелами ON/OFF ми

розуміємо модель, де періоди активності (ON) та неактивності (OFF) чергуються, тривалості ON періодів незалежні та мають однаковий розподіл, а так само і OFF періоди, і послідовності тривалостей ON і OFF періодів не залежать один від одного. Проте тривалості ON та OFF періодів можуть мати різні розподіли. Такий підхід дозволяє отримувати потік, який має самоподібні властивості.

Ефект Ноа в розподілі тривалостей ON/OFF періодів стає ключовою точкою при моделюванні самоподібного трафіку, та має певні відмінності від моделей, що використовують стандартний експоненційний розподіл. Цей ефект, який також відомий як синдром нескінченної дисперсії, виникає на основі емпіричних спостережень, що багато природних явищ можуть бути описані розподілом з безкінечною дисперсією. З математичної точки зору, для досягнення ефекту Ноа можна використовувати розподіл Парето або логарифмічно-нормальний розподіл, відомий як розподіл з важким хвостом. Розподіл Парето є найбільш популярним в цьому випадку. Його перевагою є можливість визначення фрактальності трафіку за допомогою його параметрів. Однак недоліком є те, що цей розподіл має нескінченну дисперсію, що призводить до високої мінливості вхідного трафіку [76].

Функцію розподілу, яка описує розподіл Парето (формула 2.15)

$$F(x) = 1 - (\frac{\beta}{x})^{\alpha} \tag{2.15}$$

де α – виступає параметром, що характеризує чи буде заданий розподіл мати нескінченне або ж скінченне середнє значення та дисперсію випадкової величини, і β – це параметр граничної межі (мінімально можливе значення величини x).

Для розподілу Парето, густина визначається наступною функцією (формула 2.16):

$$f(x) = \frac{\alpha}{\beta}(\frac{\beta}{x})^{\alpha+1} \tag{2.16}$$

для значень $x > \beta$ та $\alpha > 0$

$$f(x) = F(X) = 0 \tag{2.17}$$

для значень $x \leq \beta$.

Використовуючи значення параметра α, можна вважати, що середнє значення та дисперсія випадкової величини x будуть наступними:

для значень $\alpha \leq 1$ нескінченне середнє значення для розподілу;

для значень $1 \leq \alpha \leq 2$ скінченне середнє та нескінченна дисперсія для розподілу;

для значень $\alpha \leq 2$ нескінченна дисперсія для розподілу;

Також існує наступна залежність між параметром Херста H та параметром α (формула 2.18):

$$H = \frac{3-\alpha}{2} \tag{2.18}$$

Паралельно з розподілом Парето, ще одним часто використовуваним розподілом при моделюванні фрактального трафіку є розподіл Вейбула. Закон розподілу імовірностей для розподілу Вейбула може бути представлений наступним чином: (2.19):

$$F(x) = 1 - e^{-(\frac{x}{\beta})^{\alpha}} \tag{2.19}$$

де α, β - параметр форми та масштабний параметр відповідно.

Для визначення параметра Херста часового ряду використовуються різноманітні методи, серед яких можна відзначити аналіз індекса дисперсії , вейвлет-аналіз, R/S статистику та інші [77].

Таким чином, одним із методів для визначення коефіцієнта H є аналіз R/S статистики (номінального розмаху). Цей метод виконується в кілька етапів:

- Визначення номінального розмаху (R) для ряду даних на різних масштабах;
- Обчислення середнього квадратичного відхилення (S) для кожного масштабу;
- Розрахунок відношення R/S для кожного масштабу;

➢ Побудова залежності R/S від логарифмів масштабів і аналіз отриманих результатів.

За формулою (2.20) визначається математичне очікування випадкового ряду X_k, (k =1..N):

$$M_N = \frac{1}{N}\sum_{k=1}^{N} X_k \quad (2.20)$$

a) Визначається дисперсія вибірки (формула 2.21):

$$S_N^2 = \frac{1}{N}\sum_{k=1}^{N}(X_k - M)^2 \quad (2.21)$$

b) Визначається інтегральне відхилення (формула 2.22):

$$D_j = \sum_{k=1}^{j} X_k - j * M, \;\; j \in [1, N] \quad (2.22)$$

c) Визначається рознесення випадкового процесу (формула 2.23):

$$R_N = maxD_j - minD_j \quad (2.23)$$

d) З встановленого Херстом співвідношення (формула 2.24):

$$\frac{R}{S} \approx (\frac{N}{2})^H \quad (2.24)$$

визначається параметр Херста H (за формулою 2.25):

$$H = \frac{log\,(\frac{R}{S})}{log\,(\frac{N}{2})} \quad (2.25)$$

Параметр Херста (H) виступає показником стійкості статистичного випадкового явища та тривалості довгострокової залежності. Значення H=0,5 свідчить про відсутність довгострокової залежності, тоді як значення, що наближається до 1, вказує на вищий рівень стійкості цієї залежності. Дослідження різних типів трафіку у сучасних телекомунікаційних мережах показали, що у більшості випадків, значення параметра Херста знаходяться в діапазоні 0,5 < H < 1. Таким чином, кожен тип трафіку мережі має свою унікальну степінь фрактальності. Для наглядності приведені певні значення параметра Херста для різних типів трафіку у таблиці 2.1 та кругової діаграми на рис. 2.31.

Таблиця 2.1.

Значення параметра Херста для різних категорій мережевого трафіку

Категорія мережевого трафіку	Ступінь фрактальності (H)
Протокол http / https	0.7-0.9
Трафік відео потоків	0.62-0.91
Аудіо стрімінг	0.61-0.91
Точка -Точка	0.55-0.85

Як можна побачити із представленої вище таблиці, для основних категорій мережевих додатків, значення параметра Херста становить більше 0,5.

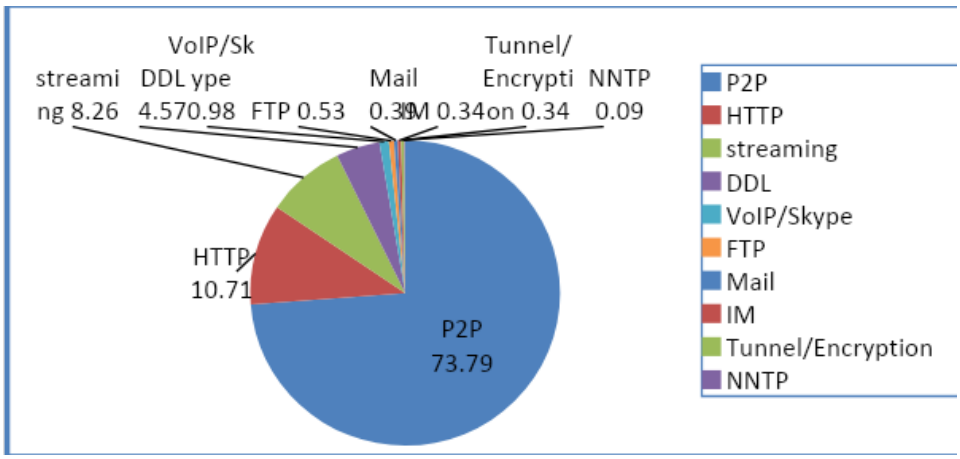


Рис.2.31. Степінь фрактальності різних типів трафіку

2.6.5. Аналіз часу затримки в CDN мережі

Оскільки, одним із основних завдань CDN -є зменшити час затримки, під час доставки даних від сервера походження до кінцевого отримувача, варто розглянути яким саме способом визначати значення цього параметру. У всіх варіантах впровадження мереж CDN, сервери кешування активно обробляють велику кількість запитів від кінцевих користувачів. Це важлива функція, яка забезпечує швидку та ефективну доставку контенту в мережі.

Будемо вважати, що $\lambda_i, i = 1...n$ це інтенсивність поступлення запитів на сервер *i* в момент часу *t*. Таким чином, сумарна інтенсивність поступлення запитів у мережі передачі даних буде розраховуватись наступним чином:

$$\lambda = \sum_{i=1}^{n} \lambda_i \quad (2.26)$$

Важливо відзначити, що кожен кеш-сервер, окрім обробки власних запитів, може також отримувати запити від інших серверів у мережі. Очевидно, що відсоток ресурсів сервера, які використовуються для обробки таких запитів від інших серверів, буде залежати від різних факторів. Отже, інтенсивність надходження запитів від сусідніх серверів у CDN мережі може бути обчислена за допомогою наступної формули:

$$\alpha_i = \sum_{j=1}^{n} \lambda_i \cdot \omega_{ji} \quad (2.27)$$

Де ω_{ji} - це та частина трафіку λ_i , яка перенаправляється від сервера *j* до сервера *i*. Як результат, середня інтенсивність навантаження, що поступає на вхід кешуючого сервер *i* в певний момент часу, розраховується за наступною формулою:

$$\alpha_i = \lambda_i - \sum_{j=1}^{n} \lambda_i \cdot \omega_{ji} \quad (2.28)$$

Перший доданок визначає частину навантаження, яке кешуючий сервер *i* переадресовує до інших серверів із своєї локації, наступний – частину запитів які були отримані кешуючим сервером i від інших серверів-сусідів із цієї ж локації. Інтенсивність обробки запитів μ_i сервером кешування *i* має бути більшою, ніж інтенсивність надходження запитів λ_i в режимі нормального функціонування.

Будемо вважати, що потік запитів, що надходять на кешуючі сервери є Пуассонівським. Кожен сервер можна представити у вигляді системи масового обслуговування типу M/M/1 [78-80].

Виходячи із цього, можна розрахувати середній час затримки за наступною формулою: [81]:

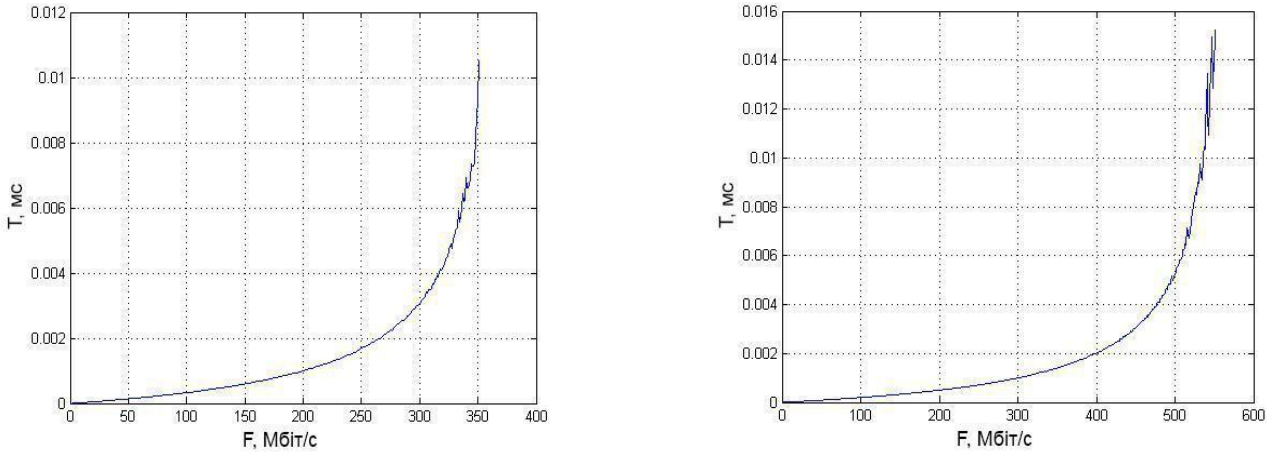
$$T = \frac{1}{\lambda} \cdot \sum_{i=1}^{n} a_i \cdot T_i = \frac{1}{\lambda} \cdot \sum_{i=1}^{n} \left(\frac{a_i}{\mu_i - a_i} \right) \quad (2.29)$$

Враховуючи затримку при передаванні даних між сусідніми серверами із одної локації, інтенсивність обробки запитів і інтенсивність надходження до кешуючого сервера, середня затримка визначається як:

$$T = \frac{1}{\lambda} \cdot \sum_{i=1}^{n} \cdot \left(\frac{F_i}{C_i - F_i} + T_{Fi} \right) \quad (2.30)$$

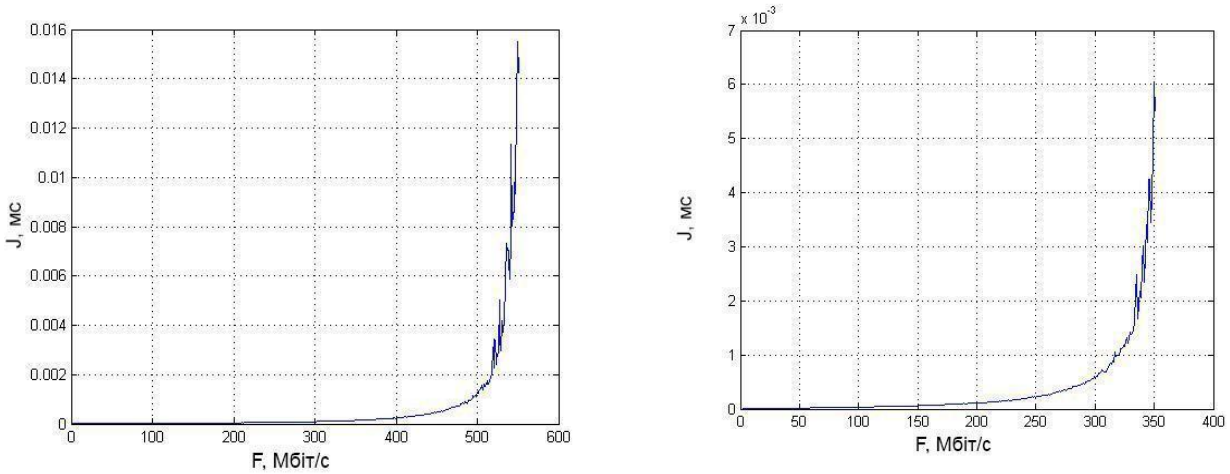
T_{Fi} — затримка передачі між кешуючими серверами з однієї локації, C_i - інтенсивніть обслуговування запитів граничним сервером кешування, Значення затримки буде визначатиметься пропускною здатністю та характеристиками самого обладнання, F_i - інтенсивність надходження запитів на сервер обслуговування [82]. Із формули очевидно, що затримка буде зростати коли сервер буде перевантажений обробкою запитів [83].

Графічні представлення, де відображені взаємозв'язки між часом очікування та кількістю кеш-серверів, а також інтенсивністю трафіку, що проходить через них представлено на наступних рисунках



а) б)

Рис.2.32. Залежність часу затримки обробки даних на сервері кешування від інтенсивності надходження заявок користувачів при різних значеннях інтенсивності обробки (а) C_i=600Мбіт/с, б) C_i=400Мбіт/с)



а) б)

Рис.2.33. Залежність джитера від інтенсивності надходження заявок користувачів при різних значеннях інтенсивності обслуговування (а) C_i =600Мбіт/с, б) C_i =400Мбіт/с)

З аналізу діаграм, які представлені на рисунках 2.32 - 2.33 випливає, що збільшення інтенсивності вхідного потоку призводить до зростання часу очікування та джитера. Для зменшення цих значень можна вжити кілька заходів. З одного боку, можна збільшити потужність вузлів обробки даних або зменшити затримки між граничними серверами кешування. Однак, розширення ресурсів пристроїв обробки даних не завжди ефективно. Тому, альтернативою може бути використання механізму балансування навантаження між серверами CDN мережі. Це дозволить раціональніше розподіляти запити між серверами кешування, що забезпечить високу якість послуг, зокрема зменшення часу очікування та джитера, що особливо важливо для послуг реального часу.

2.7. Дослідження ефективності використання ресурсів CDN мережі

Часто виникають питання щодо доцільності використання CDN мережі та необхідної кількості граничних серверів кешування для

забезпечення якісної передачі мережевого трафіку. У зв'язку з цим, було проведено імітаційне моделювання роботи CDN з використанням програмного середовища CDNsim. Це середовище призначене для тестування та оцінки ефективності CDN мереж та для побудови експериментальних моделей. Далі будуть наведені результати цього моделювання, які допоможуть відповісти на поставлені питання та проаналізувати ефективність використання CDN [84-85].

Було здійснено дослідження мережі, яка включає 100 граничних серверів кешуванн. Кожен сервер одночасно може обслуговувати 1000 одночасних клієнтських з'єднань. Розмір пам'яті, відведеної на кешування даних, буде визначатися відсотковим співвідношенням від загального обсягу даних ресурсу, що обслуговується.

Під час моделювання було також враховано наявність маршрутизаторів на шляху передачі даних від постачальника послуг до кінцевого споживача. Ці маршрутизатори формують мережеві магістралі, до яких підключаються інші елементи мережі [86].

Запити формуються симулятором, що враховує структуру та особливості ресурсів і створює їх у вигляді послідовності сторінок, які максимально схожі з тими, які можуть генерувати справжні користувачі. Ресурси CDN мережі характеризуються таким терміном як популярність. Популярність вимірюється кількістю запитів до певного контенту в порівнянні із загальною кількістю запитів, що мають відношення до того ж CDN ресурсу. Таке співвідношення може коливатися від -1 до 1. Позитивне значення свідчить про те, що великі об'єкти є більш популярними, аніж менші. Значення нуль означає відсутність взаємозв'язку між розміром та популярністю.

Hit ratio - це відношення запитів до CDN ресурсу, які були оброблені без потреби звернення до інших кешуючих серверів або сервера джерела, до загальної кількості запитів. Це значення може змінюватися від 0 до 1.

Високі показники коефіцієнта попадання є бажаними, оскільки вони значно скорочують час очікування відповіді.

Для визначення ефективності роботи CDN використовується поняття доцільності CDN мережі. Доцільністю вважається відношення обсягу відвантажених даних до загального обсягу завантажених. Це значення може знаходитися в діапазоні від 0 до 1. Очевидно, що для оптимально функціонуючої CDN мережі ця величина повинна перевищувати 0,5.

На рис. 2.34 показано залежність продуктивності використання CDN від розміру пам'яті кешування для різних мережевих топологій.

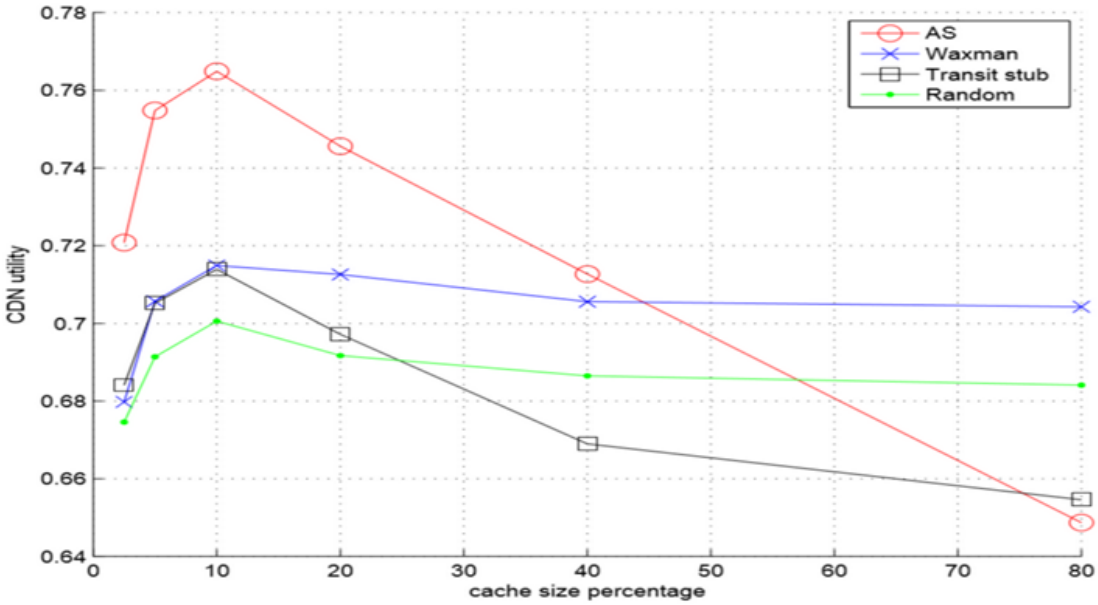


Рис.2.34. Залежність продуктивності використання CDN від розміру кеш пам'яті

На графіку чітко спостерігається пікове значення, коли розмір кеш пам'яті становить 10% від загального обсягу CDN ресурс що обслуговується. Також можна припустити, що максимальна продуктивність є приблизно однаковою для всіх досліджуваних топологій. Досягнення піку супроводжується тим, що розмір кешу на початкових етапах є надто малим, а в самій CDN мережі відносно невелика кількість контенту, який переглядають кінцеві користувачі. Як результат, практично всі запити

будуть направлені на кореневий сервер. У такому випадку ефективність використання мережі CDN є досить низькою. Проте зі збільшенням розміру кешу зростає відповідно і кількість реплікацій вмісту даних та обмін даними між граничними серверами. Якщо розмір пам'яті кешування стає надто великим, то в такому разі припиняється обмін даними між граничними серверами, оскільки весь вміст буде збережено в пам'яті кожного із них [87].

Необхідно пам'ятати, що повністю кешувати весь контент неможливо, оскільки частина даних постійно змінюється (динамічний контент). Динамічні дані потрібно завжди запитувати / оновлювати безпосередньо від кореневого сервера [88].

Проаналізувавши графічні залежності можна зробити висновок, що розширення розміру кеша не завжди гарантує ефективне використання CDN. Це слід брати до уваги при визначенні оптимального розміру кеш пам'яті для граничних Edge серверів. При невеликому обсязі контенту спостерігається зростання продуктивності і досягаються прийнятні результати, оскільки зменшується потік даних. Крім того, важливо зазначити, що збільшення пропускної здатності та обсягу пам'яті безпосередньо впливає на вартість послуг CDN.

На рисунку 2.35 наведено графік, який відображає вплив розміру кеша на коефіцієнт попадання. Коефіцієнт попадання визначається як співвідношення запитів, що були оброблені без звернення до інших Edge серверів або сервера походження, до загальної кількості запитів. Ця величина змінюється в діапазоні від 0 до 1. Високі значення цього коефіцієнта є бажаними, оскільки вони сприяють зменшенню часу відповіді.

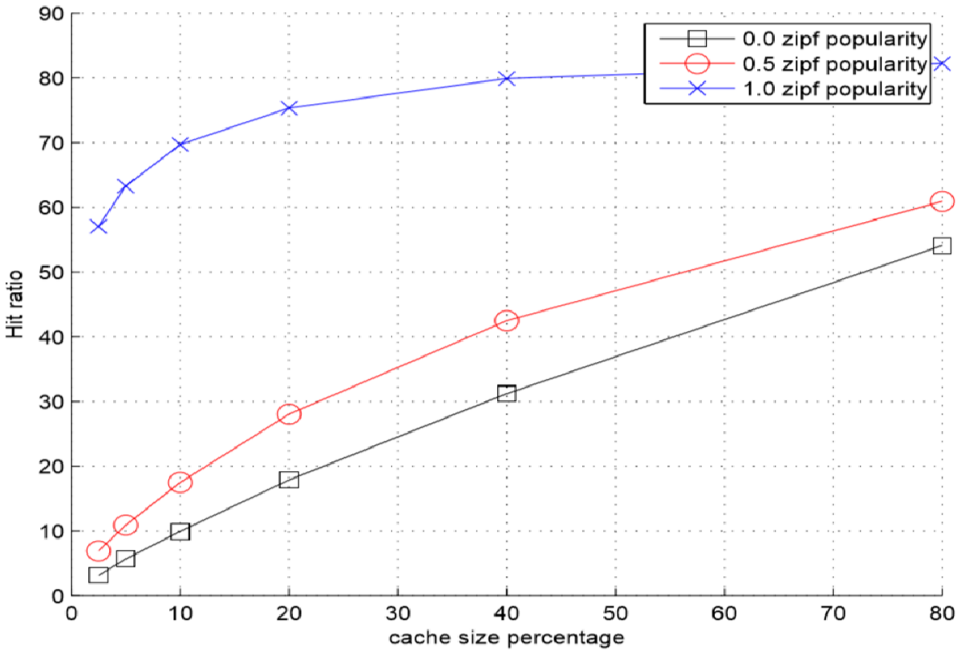


Рис.2.35. Залежність коефіцієнта попадання від розміру кеша

Як можна побачити із представленої залежності, збільшення розміру кеш пам’яті значення коефіцієнта попадання також пропорційно зростає.

Більші значення параметра k призводять до підвищення продуктивності CDN. Це можна підтвердити і з результатів які представлені на рисунку 2.35, де при k=0 об'єкти рівномірно запитувані, і коефіцієнт попадання залишається дуже низьким навіть для великих значень розміру кеша. У випадку k=1 коефіцієнт попадання стає дуже високим, навіть при дуже невеликих розмірах кешу. В такому випадку розмір кешованих даних є достатньо великий і зберігається в кеші тривалий час [89].

Один із ключових критеріїв для визначення ефективності роботи CDN - це час, необхідний для відповіді на запити користувачів. Цей показник визначає, як швидко користувач отримає дані, які запитує в CDN ресурсу. На рис. 2.36 показано як залежить час відгуку від розміру кешу для різних методів перенаправлення.

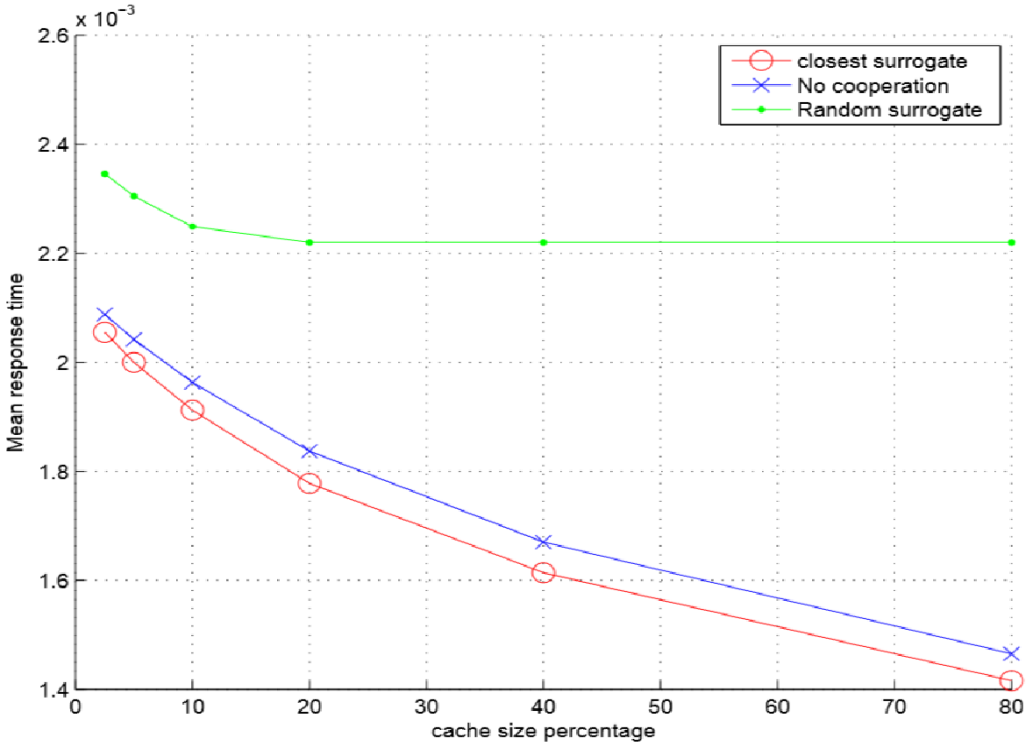


Рис.2.36. Залежність часу очікування від розміру кеша

Найкращої ефективності можна досягти, використовуючи граничні сервери , які взаємодіють між собою. Час відповіді може бути трохи більшим при використанні методу перенаправлення без взаємодії між граничними серверами, оскільки в такому випадку запити надсилаються безпосередньо на кореневий сервер. У випадку використання випадкового граничного сервера із взаємодією, середній час відповіді буде більшим, ніж у попередніх методах. Це спричинено випадковим розподілом запитів без врахування будь-яких умов близькості мережі.

Очевидним також є той факт, що із збільшенням розміру кешу час відповіді буде зменшуватись, так як більшість статичних даних буде розміщена в кеш пам'яті граничних серверів, які розташовані або в локації кінцевого користувача, або ж максимально близько до нього.

2.8. Висновок до розділу 2

У даному розділі розглянуто основні архітектурні принципи роботи CDN мережі, основні методи балансування навантаження які використовуються в сучасних мережах.

Запропоновано метод розподіленої обробки запитів, що потребують значних обчислювальних ресурсів, який враховує дані аналітики щодо популярності певних ресурсів (веб-сторінок, мультимедійного контенту) серед списку найбільш запитуваних, що дало змогу забезпечити ефективне використання кешованих даних, а також зменшити навантаження на кореневий сервер та час відповіді на запити кінцевого користувача. Як показують результати досліджень, час затримки при обробці запитів після застосування методу розподілу запитів на рівні балансувальника навантаження зменшився на 10% у порівнянні з тим, який був при обробці стандартних запитів. Це можна пояснити тим, що більша кількість даних була оброблена саме на кешуючих серверах, оскільки вже знаходилась в кеш пам’яті. Такий підхід дає можливість не створювати надлишкового навантаження на кореневі сервера, а також забезпечувати зниження часу відповіді і тим самим покращувати якість сприйняття послуг кінцевим користувачем.

Представлено інтегральний ключ кешування, що формується шляхом поєднання кількох параметрів або компонентів, які унікально визначають набір даних для збереження в пам’яті граничних серверів. Наведено приклад інтегрованого ключа для двох типів сервісів, а саме – веб сервіс, та сервіс інтерактивного цифрового телебачення.

Удосконалено метод балансування трафіку в граничній локації мережі доставки даних, який за рахунок інтеграції із методом розподіленої обробки запитів, використання інтегрального ключа кешування, взаємодії із даними аналітики, а також контролю завантаженості граничних серверів, дає змогу підвищити якість обслуговування в мережах доставки контенту. Проведено

програмно імітаційне моделювання роботи запропонованого методу. Приведено результати моделювання, які показують, що значення ефективності кешування збільшилось до 90%. Значення ефективності кешування даних без використання запропонованого методу становило 60-65%. Час завантаження сторінки зменшився із 1600мс до 800мс, що дає змогу значно покращити якість обслуговування для кінцевого користувача.

Оцінка ефективності використання ресурсів CDN мережі показала, що збільшення обчислювальних потужностей вузлів та розміру кеш пам'яті не завжди є раціональним з точки зору загальної оптимізації, оскільки ці параметри мають суттєвий вплив на собівартість послуг для провайдера. Більш ефективним та доцільним є використання механізмів балансування навантаження, які дають змогу більш раціонально перенаправляти запити до серверів, ефективніше використовувати їхні ресурси і тим самим забезпечити кращу якість сервісу для кінцевого користувача.

РОЗДІЛ 3. МОДЕЛЬ ОЦІНКИ ЕФЕКТИВНОСТІ РОБОТИ СИСТЕМ ОБРОБКИ МЕРЕЖЕВОГО ТРАФІКУ У ТОЧКАХ ПРИСУТНОСТІ CDN МЕРЕЖ

Хмарні обчислення (Cloud Computing) є досить поширеним рішенням, що забезпечує надання послуг користувачам та сприяє покращенню якості цих послуг за рахунок максимально швидкої та ефективної обробки даних [90-91]. Багато провайдерів хмарних обчислень пропонують обчислювальні ресурси користувача, використовуючи дві моделі організації цих ресурсів. Перша модель - це статичні ресурси, які виділяються на весь час їхньої оренди, друга модель - динамічно виділені ресурси, які виділяються та змінюються в залежності від потреб сервісу [92-93].

3.1. Загальна характеристика систем обробки мережевого трафіку

Здатність забезпечити бажану якість обслуговування (QoS), що є стандартним елементом SLA, який визначається між споживачами послуг та хмарними (Cloud) провайдерами, які надають такі послуги, є одним з найважливіших завдань для Cloud провайдерів. Зазвичай, QoS визначається такими параметрами як:

- середній час відповіді,
- середня довжина черги на обслуговування,
- середній час очікування,
- пропускна спроможність системи,
- ймовірність блокування обслуговування,

Всі ці показники можна аналізувати та описувати за допомогою теорії систем масового обслуговування [94-95].

Задачі, які обробляються з використанням технології хмарних обчислень можна розділити на кілька етапів:.

- Запит клієнта надходить на балансувальних навантаження та попадає в чергу на обслуговування.
- Балансувальних навантаження перевіряє наявність ресурсів на обслуговуючих пристроях та перенаправляє запит на обробку до сервера, який буде в найближчій із наявних локацій до користувача.
- Обслуговуючий пристрій (в нашому випадку це контейнер в якому запущена наша аплікація) бере запит із черги на обслуговування, виділяє частину ресурсів та починає обробляти даний запит.
- Формується відповідь для користувача та запит покидає систему обслуговування.
- У випадку якщо кількість місць у черзі на обслуговування на рівні балансувальника навантаження буде недостатньою, та всі обслуговуючі пристрої будуть зайняті - то запит буде заблокований.

Модель черги для обробки даних в мережі доставки контенту може аналізувати взаємозв’язок між трьома складовими, а саме: інтенсивність поступлення запитів, інтенсивність обслуговування та ймовірність того, що запит потрапить на вузол обслуговування. Постачальники хмарних (Cloud) сервісів можуть використовувати цю інформацію для аналізу управління своїми системами та оптимізації розподілу обчислювальних ресурсів [96-97].

Варто зазначити той момент, що хмарна система має щонайменше два рівні планування обробки запитів, а саме: головний планувальник функцію якого в нашому випадку виконуватиме балансувальник навантаження, який визначає завантаженість робочих інстансів та перенаправляє запити до них, та другий планувальник (інстанс/контейнер виконання), який виконує обробку запитів які прийшли від балансувальника навантаження. Швидкість та кількість запитів до основного планувальника балансувальника навантаження відрізняється від кількості, яка потрапляє на окремий інстанс обробки, і це потрібно враховувати в аналізі обробки черг.

У розділі використано комплексну модель обробки черг для дослідження продуктивності та ефективності обробки запитів на рівні хмарних сервісів.

В даному розділі було змодельовано роботу системи хмарних сервісів, як комплексна система масового обслуговування для характеристики процесу обслуговування. Дві системи масового обслуговування складають складну модель. Перша - це рівень входу в точку присутності сервісу в хмарі, друга – виконання обробки даних на прикладі системи масового обслуговування на рівні вузла обробки запитів, який є найближче до користувача та має для цього наявні ресурси.

На основі цієї складної комплексної моделі обслуговування черг та використання теорії масового обслуговування, в роботі аналізується інтенсивність обробки запитів головним планувальником, а саме балансувальником навантаження, інтенсивність надходження запитів на вузол обробки, середній час перебування запиту у системі, ймовірність блокування та інші показники ефективності.

Хоча багато наукових досліджень було проведено з аналізу продуктивності хмарного центру на основі моделей масового обслуговування [98-103], фактори, що включають, неоднорідність і динамічність інфраструктури хмарного сервісу не розглядалися. Однак жодна з наведеної вище літератури не розглядали випадок, що основний планувальник задач, а саме балансувальник навантаження та інстанси виконання мають свої власні черги обробки завдань, що сервер виконання з різною продуктивністю матимуть різний час обробки, та різні темпи надходження запитів на обслуговування.

3.2. Моделювання систем, які виконують обробку мережевого трафіку

Математичне моделювання полягає у створенні та використанні математичних конструкцій для аналізу поведінки систем або об'єктів у

різних ситуаціях. Його основна мета полягає в тому, щоб отримати різноманітні властивості або характеристики об'єкту без необхідності вимірювання або з обмеженою кількістю вимірів [104-105]. В рамках математичного моделювання склалися два підходи:

- аналітичний;
- імітаційний.

Аналітичний метод ґрунтується на розробці формульних виразів, які описують взаємозв'язки між різними параметрами та компонентами системи. Довгий час цей підхід вважався власне математичним. Проте, при дослідженні складних систем деякі математичні вирази стають дуже складними, і для їхнього отримання потрібно проводити значну кількість вимірів.

Вивчення властивостей процесів у складних системах за допомогою аналітичних методів часто стикається з великими труднощами. Ці труднощі можуть призвести до необхідності значного спрощення моделей під час їх побудови або в ході подальшої роботи з ними, що може позначитися на точності отриманих результатів. Таким чином, хоча аналітичні моделі дуже зручні для теоретичних досліджень, проте побудова адекватної аналітичної моделі може виявитися нетривіальною для більшості джерел.

Імітаційний, що базується на статистичних підходах, використовує ймовірнісне відображення параметрів системи. Під час попереднього аналізу системної моделі, встановлюються закони розподілу випадкових величин параметрів. У цьому методі використовуються аналітичні зв'язки між параметрами елементів системи, але вони мають спрощений характер порівняно з аналітичним моделюванням. Ці зв'язки значно простіші, ніж у аналітичному підході. У зрозумілій формі, імітаційна модель - це набір алгоритмів, що створює послідовність, що має схожі характеристики з реальною послідовністю, отриманою з діючого об'єкта. Ця послідовність може включати, наприклад, мережевий трафік. Використання імітаційної

моделі зручніше, але такий метод зазвичай обмежений у застосуванні і вимагає значної роботи для адаптації моделі до нових умов.

Також можливі комбіновані моделі, що об'єднують як аналітичні, так і алгоритмічні компоненти. Основні вимоги, які ставляться до процесу моделювання:

- модель має бути адекватною відповідним технологічним завданням або ж системам;
- має бути забезпечена необхідна точність;
- повинна бути забезпечена зручність для користувача - спеціаліста з обробки та аналізу інформації (управління);
- зрозумілий інтерфейс управління моделюванням;
- висока швидкість роботи;
- задовільна вартість розробки моделі та використання інструментів моделювання.

Станом на сьогодні, існує достатньо велика кількість моделей, які можуть використовуватись для імітування та моделювання фрактального мережевого трафіку [106-108].

3.3. Модель обробки черг для аналізу продуктивності системи хмарних сервісів

У цьому розділі представлено комплексну модель масового обслуговування з двома об’єднаними системами масового обслуговування для характеристики процесу обслуговування запитів у хмарних сервісах із динамічним виділенням ресурсів. Динамічне виділення ресурсів представляє собою механізм, який дає можливість створювати нові інстанси для обробки даних лише за необхідності, тобто при умові, що існуючі не встигають обробляти всі запити, що надходять в систему, або якість сервісу, які забезпечують існуючі інстанси обробки, стає недопустимою для забезпечення хорошого сервісу який отримує кінцевий користувач.

У середовищі хмарних обчислень - постачальник хмарних обчислень будує центр обробки даних, як інфраструктуру для надання послуг користувача. На рисунку. 3.1 зображено архітектуру неоднорідного центру обробки даних і модель його обслуговування.

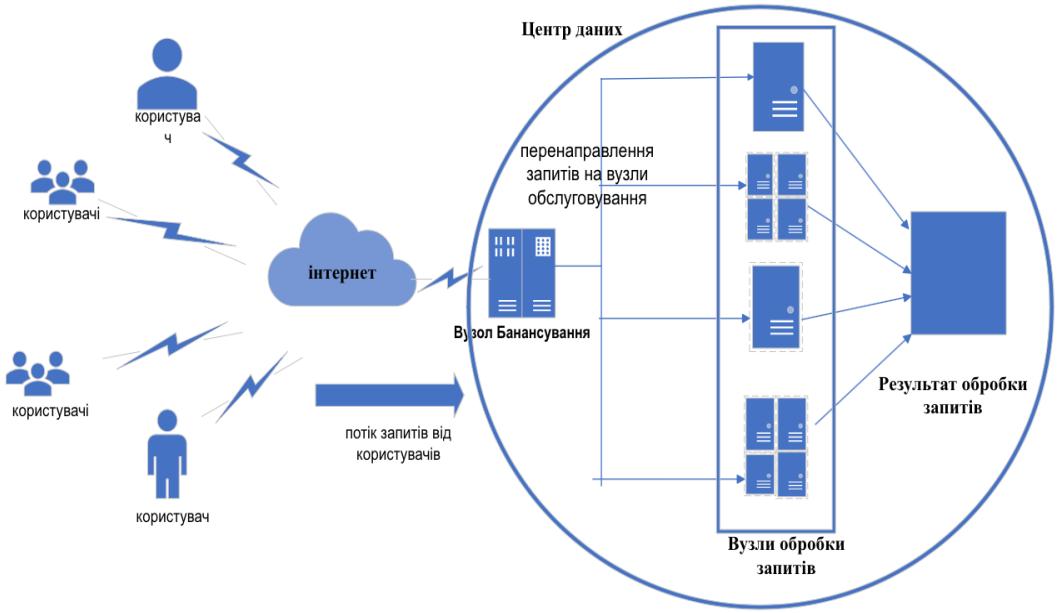


Рис 3.1. Модель роботи неоднорідного центру обробки даних

Варто зазначити той факт, що центр обробки даних являє собою не тільки централізоване сховище, чи центр обробки, а для прикладу, це може будь яка локація в якій існують сервери CDN/Compute мережі. Центр обробки даних, або точка присутності CDN мережі, складається з кешуючих серверів, на яких можуть створюватись динамічні контейнери, в яких будуть працювати певні сервіси, які хоче отримати кінцевий користувач. Він включає також балансувальник навантаження, який працює як основний планувальник для розподілу обчислювальних ресурсів та перенаправлення запитів користувачів на вузли обробки. Усі клієнти надсилають запити до центру обробки даних, використовуючи мережу Інтернет. Запит користувача, згідно концепції роботи CDN мережі, потрапить в точку присутності (центр обробки даних) CDN мережі, яка є

найближчою до локації користувача. Коли завдання(запити користувачів) надходять до центру обробки даних(точка присутності CDN мережі), першочергово вони потрапляють на балансувальних навантаження, який в подальшому відправляє ці запити на вузли обробки. Кількість вузлів обробки запитів може змінюватись в залежності від інтенсивності поступлення запитів до центру обробки даних.

Основний планувальник(Load Balancer) має буфер очікування з кінцевою ємністю, у якому зберігаються завдання, які не можна запланувати чи перенаправити на вузол обробки, негайно. Кінцева ємність буфера означає, що деякі запити не можуть потрапити в буфер очікування у випадку, якщо він переповнений, а замість цього залишать систему відразу після надходження. Кожен вузол обробки отримує всі запити від головного планувальника, балансувальника навантаження та підтримує власну чергу завдань. Після успішного виконання завдання задача, яка виконувалась, залишає систему та результат надсилається назад клієнту.

3.3.1. Система обслуговування черг з великою кількістю вузлів обслуговування та однаковою інтенсивністю обслуговування запитів

Система обслуговування з великою кількістю вузлів обслуговування та однаковою інтенсивністю обслуговування запитів представлена на рисунку 3.2.а

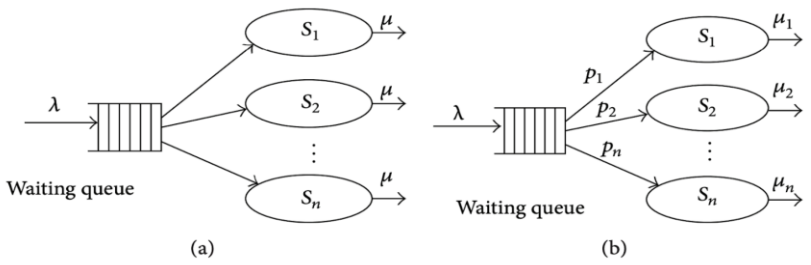


Рис.3.2. Модель обслуговування черг; (а)продуктивність всіх обслуговуючих пристроїв однакова, (b)продуктивність обслуговуючих пристроїв різна

Оскільки кожен сервер (контейнер) має однакову продуктивність і тим самим може забезпечувати однакову інтенсивність обслуговування, ця модель обробки черги розглядає кожен сервер у центрі обробки даних однаковим і призначає йому таку ж саму ймовірність отримання запиту p = 1/n. Ця модель обслуговування черг спрощує аналіз і робить простішим спосіб, як вивести рівняння для важливих показників ефективності таких як середній час відповіді, середня кількість завдань у системі та середній час очікування. Проте процес представлений у цій моделі не стосується процесу обслуговування у хмарному центрі обробки даних, що у нашому випадку представляє собою точку присутності CDN мережі. В даній моделі не враховано балансувальник навантаження, який розподіляє всі запити між серверами обробки, а також різні інтенсивності обробки запитів різними серверами.

3.3.2. Система обслуговування черг з великою кількістю вузлів обслуговування та різною інтенсивністю обслуговування запитів на вузлах обробки

Модель черги, що включає кілька серверів з різними інтенсивностями обслуговування показано на малюнку 3.2(b). Цю систему обробки черг можна використати для моделювання неоднорідного центру обробки даних, що включає багатоядерні сервери з різною швидкодією. Кожен обслуговуючий пристрій в граничній локації CDN мережі, має різну інтенсивність обслуговування μ_i та ймовірність отримання запиту на обробку p_i. Інтенсивність поступлення запитів на обслуговування визначається λ. Діаграма ймовірність переходів стану для N=2 (стан очікування та стан обслуговування запитів)показана на рисунку 3.3

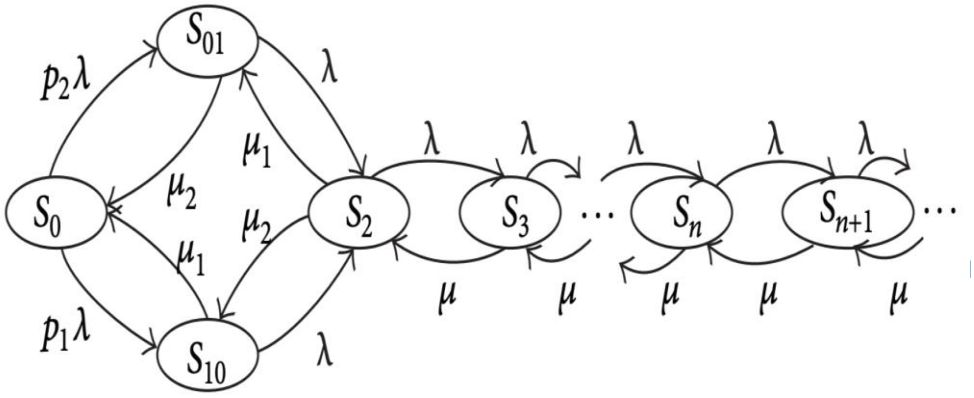


Рис.3.3. Діаграма стану переходів для N=2

Коли запит від користувача надходить в систему на обслуговування та є два сервери які очікують, балансувальник навантаження обирає один із них для обробки запиту що надійшов. Ймовірність вибору сервера 1 дорівнює $p1$ і сервер 2 дорівнює $p2$ ($p2 = 1-p1$). Стан S0 означає, що в системі масового обслуговування немає завдання, яке обробляється; $S10$ означає, що запит для обслуговування направлено на сервер 1, а сервер 2 очікує на надходження запитів; $S01$ вказує на це, що запит для обслуговування було направлено на сервер 2, а сервер 1 перебуває в режимі очікування; $S2$ означає, що є два запити у системі обслуговування, і кожен запит був направлений на один із двох обслуговуючих серверів. $S3$, $S4$,...,Sn вказують, що є 3, 4, . . . , n завдань у системі, а два з них знаходяться на виконанні на двох обслуговуючих серверах. Нехай $\mu1$ і $\mu2$ позначають інтенсивність обслуговування серверів 1 та 2 відповідно. Інтенсивність обслуговування системи в цілому буде:

$$\mu = \sum_{i=2}^{2} \mu_i \tag{3.1}$$

На рисунку 3.3 показано, як ця модель обчислює ймовірність перебування системи в певному стані, коли коефіцієнт завантаженості системи $\rho = \lambda/\mu$ і $\alpha=\mu2/\mu1$

$$\pi_{01=} \frac{\rho}{1+2\rho} \cdot \frac{1+\alpha}{\alpha} (\rho + \rho_2)\pi_0 \tag{3.2}$$

$$\pi_{10} = \frac{\rho}{1+2\rho} \cdot (1+\alpha)(\rho+\rho_1)\pi_0 \quad (3.3)$$

$$\pi_0 = \frac{1-\rho}{1+\rho[1+(1+\alpha^2)\rho-(1-\alpha^2)]\rho_1/\alpha(1+2\rho)} \quad (3.4)$$

3.3.3. Оцінка продуктивності на основі системи обслуговування черг

Модель системи масового обслуговування неоднорідних ЦОД з великою кількістю серверів обробки, які мають різні показники продуктивності показано на рисунку 3.4.

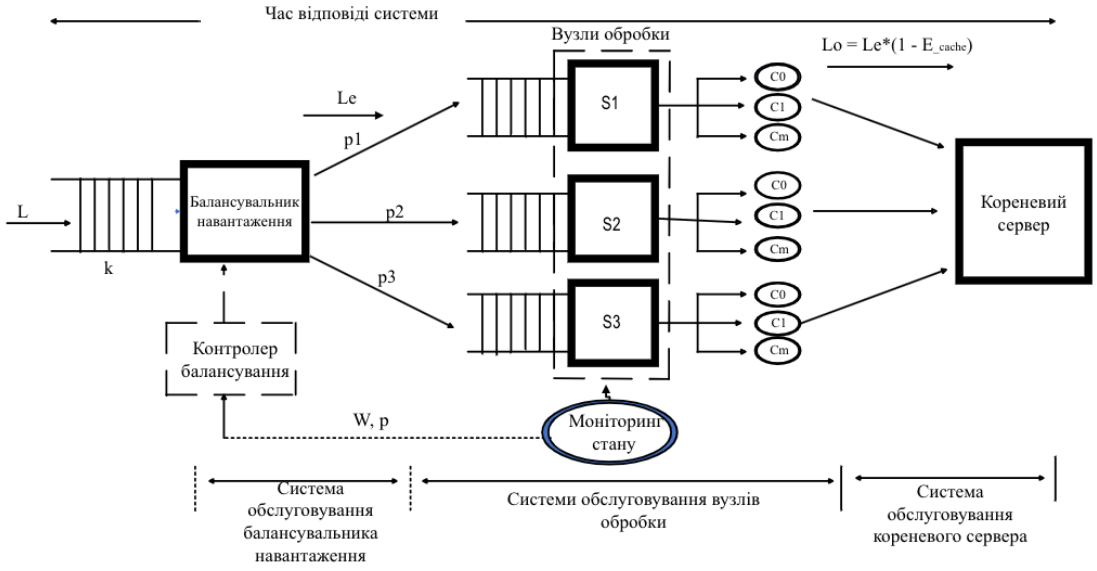


Рис.3.4. Комплексна модель обслуговування запитів в точках присутності CDN мережі

На рисунку 3.4, бачимо три системи обробки черг - система масового обслуговування на рівні балансера навантаження, система масового обслуговування на рівні інстансів обробки, а також система обслуговування на рівні кореневого сервера - складають комплексну модель масового обслуговування, яка характеризує неоднорідність ЦОД та сервісні процеси, які задіяні в хмарних обчисленнях (Edge Compute). Всі підсистеми взаємодіють між собою.

Для ефективного розподілу навантаження, балансувальник повинен отримувати значення показників ефективності та завантаженості з усіх вузлів обробки, а саме кількість завдань в черзі, Lq, середній час очікування завдання в черзі, Wq, і коефіцієнт завантаженості вузла обробки, ρ. Розглянемо більш детально кожну із підсистем.

Головний сервер, функцію якого виконує балансувальник навантаження працює як основний вузол розподілу навантаження і виконує обробку всіх запитів від усіх користувачів. Балансувальник навантаження виступає точкою входу в систему центра обробки даних.

Оскільки процес розподілу ресурсів для виконання завдань у хмарі має враховувати всі ресурси центру обробки даних, головний сервер розглядається як система $M/M/1/k$ - система масового обслуговування з обмеженою продуктивністю k.

Кожен вузол у центрі обробки даних працює як обслуговуючий пристрій, який може бути окремим контейнером в якому запущена аплікація та продуктивність якого є рівною fi. Оскільки кожен вузол обробки може паралельно обробляти кілька завдань, він розглядається як система масового обслуговування $M/M/$n. Для спрощення, кореневий сервер також розглядається як система масового обслуговування $M/M/$n.

Вхідний потік заявок описується розподілом Пуассона із інтенсивністю надходження λ (часові інтервали між поступленням завдань на обслуговування є незалежними).

Оскільки надходження запитів не залежать від стану черги та балансувальник навантаження є системою масового обслуговування планувальника з обмеженою довжиною черги, то завжди існує ймовірність блокування системи розподілу вхідних запитів, яка визначається як pb і буде більше або дорівнює 0 ($pb \geq 0$) і $\lambda \geq \lambda e$

λ - інтенсивність поступлення запитів на вхід системи балансування, λe - сумарна інтенсивність запитів на виході системи балансування(Load Balancer). Основні параметри системи обслуговування визначатимуться наступним чином:

$$\lambda_e = \lambda(1 - \rho_b) \tag{3.5}$$

$$\lambda_i = \lambda_e p_i \quad 1 \le i \le n \quad \Sigma\rho_i = 1 \tag{3.6}$$

$$\mu_i = \frac{1}{\left(\frac{I}{f_i}\right)} = \frac{f_i}{I} \tag{3.7}$$

$$\rho_i = \frac{\lambda_i}{(C_i\mu_i)} = \frac{(\lambda_i I)}{(C_i f_i)} \tag{3.8}$$

3.3.4. Розрахунок параметрів продуктивності балансувальника навантаження

Для розрахунку параметрів балансувальника навантаження вибрано система $M/M/1/k$ - система масового обслуговування з обмеженою довжиною черги k, та 1 обслуговуючим пристроєм. Діаграма станів та переходів для такої системи представлено на рис.3.5.

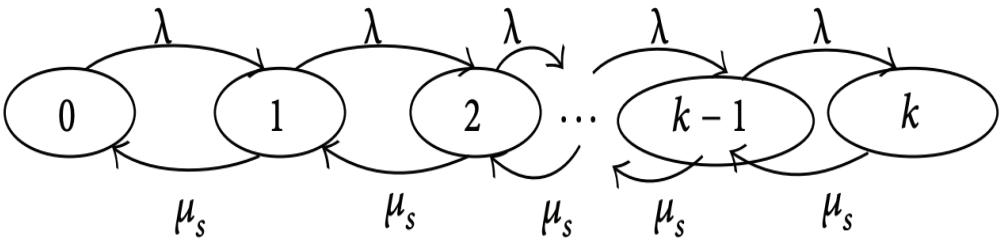


Рис.3.5. Діаграма станів та переходів балансувальника навантаженя

Якщо коефіцієнт завантаженості системи рівний:

$$\rho_s = \frac{\lambda}{\mu_s} \tag{3.9}$$

та $\rho_s < 1$, тоді ймовірність того, що в системі перебуває i задач буде визначатись наступним співвідношенням:

$$\pi_0^{(s)} = \frac{1-\rho_s}{1-\rho_s^{k+1}}$$

$$\pi_i^{(s)} = \frac{1-\rho_s}{1-\rho_s^{k+1}}\rho_s^i = 1,2,\dots,k \tag{3.10}$$

У випадку, якщо $i >$ k , нові задачі не можуть надходити в систему і в такому випадку ймовірність блокування буде рівною:

$$P_b = \pi_k^{(s)} = \frac{1-\rho_s}{1-\rho_s^{k+1}}\rho_s^k \tag{3.11}$$

Враховуючи співвідношення 3.5 та 3.11, ефективна інтенсивність обробки на сервері балансування (тобто та інтенсивність запитів яка буде передаватись на вузли обслуговування), буде визначатись як:

$$\lambda_e = \lambda(1-p_b) = \lambda\frac{1-\rho_s^k}{1-\rho_s^{k+1}} \tag{3.12}$$

Наступним параметром є кількість запитів, які знаходяться в балансувальнику навантаження. Ця кількість включає в себе запити, які знаходяться в самому балансувальному пристрої та відправляються на обслуговуючі вузли, а також запити, які знаходяться в черзі на обслуговування.

Сумарна кількість таких запитів рахується за формулою:

$$\overline{C}_\omega^{(s)} = \sum_{i=0}^{k-1} k\pi_{i=1}^{(s)} = \rho_s^2\pi_0^{(s)}\sum_{i=1}^{k-1} k\rho_s^{k-1}$$

$$\overline{C}_{sch}^{(s)} = \left(1-\pi_0^{(s)}\right) = \frac{\rho_s-\rho_s^{k+1}}{1-\rho_s^{k+1}} \tag{3.13}$$

$$\overline{C}_{total}^{(s)} = \overline{C}_\omega^{(s)} + \overline{C}_{sch}^{(s)} = \frac{\rho_s}{1-\rho_s} - \frac{(k+1)\rho_s^{k+1}}{1-\rho_s^{k+1}} \tag{3.14}$$

Враховуючи формулу Літтла та вирази 3.11 3.14 , час відповіді можна отримати за наступним виразом:

$$\overline{T}^{(s)} = \frac{\overline{C}_{total}^{(s)}}{\lambda_e} = \frac{1-(k+1)\rho_s^k+k\rho_s^{k+1}}{\mu_s-(1-\rho_s)(1-\rho_s^k)} = \frac{1}{\mu_s-\lambda} - \frac{k\lambda^k}{\mu_s^{k+1}-\lambda^k} \tag{3.15}$$

3.3.5. Розрахунок параметрів продуктивності вузла обслуговування

Згідно рисунку 3.4, всі запити які опрацьовує балансувальник навантаження потрапляють на вхід обслуговуючого пристрою. Якщо мова йде про стандартний ординарний центр обробки даних, то це буде один вузол обробки даних. В представленій комплексній моделі, фізично це також буде один сервер, одна на самому сервері буде працювати кілька мікро сервісів у вигляді докер контейнерів. Таку систему можна описати у вигляді системи масового обслуговування M/M/C. Враховуючи діаграму станів та переходів, ймовірність того, що на обслуговуванні знаходить 0 задач буде визначається наступним виразом:

$$\pi_{i,0}^{(e)} = \left(\sum_{m=0}^{C_{i-1}} \frac{\rho_{i0}^{m}}{m!} + \frac{\rho_{i0}^{C_i}}{C_i!} \cdot \frac{1}{1-\rho_i} \right)^{-1} \tag{3.16}$$

Ймовірність того, що запити, які приходять від балансувальника навантаження на обслуговуючі пристрої будуть оброблені відразу:

$$q_i = 1 - \sum_{m=C_i}^{\infty} \pi_{i,m}^{(e)} = 1 - \sum_{m=C_i}^{\infty} \frac{C_i^{C_i}}{C_i!} \rho_i^m \pi_{i,0}^{(e)} = \frac{C_i - \rho_{i0} - C_i \pi_{i,C_i}^{(e)}}{C_i - \rho_{i0}} \tag{3.17}$$

Кількість задач, які знаходяться в обслуговуючому пристрої

$$\overline{C}_{total\_i}^{(e)} = \overline{C}_{\omega\_i}^{(e)} + \overline{C}_{exc\_i}^{(e)} = \pi_{i,0}^{(e)} \cdot \frac{\rho_{i0}^{C_{i+1}}}{(C_i-1)!(C_i-\rho_{i0})^2} + \rho_{i0} \tag{3.18}$$

Враховуючи формулу Літтла та вирази 3.18 , час відповіді можна отримати за наступним виразом:

$$\overline{T}_i^{(e)} = \frac{\overline{C}_{total\_i}^{(e)}}{\lambda_i} = \frac{\bar{I}}{f_i} \left(1 + \pi_{i,0}^{(e)} \cdot \frac{\rho_{i0}^{C_i}}{(C_i-1)!(C_i-\rho_{i0})^2} \right) \tag{3.19}$$

Що стосується кореневого сервера, то можливі два варіанти перенаправлення запитів на обробку на цей пристрій. В першому варіанті, всі запити від вузлів обробки в точці присутності будуть перенаправлятись

на кореневий сервер. Наступний випадок, частина запитів буде обслужена на граничних серверах, а частина направлена на кореневий. Інтенсивність надходження запитів, яка буде перенаправлятись на кореневий сервер буде визначатись коефіцієнтом ефективності кешування.

$$\mathrm{E_{cache}} = \frac{\mathrm{N_{edge}}}{\mathrm{N_{total}}}, \tag{3.20}$$

$\mathrm{E_{cache}}$ - ефективність кешування,

$\mathrm{N_{edge}}$ - кількість запитів, відповіді на які були сформовані на граничному сервері (без звернення до кореневого сервера),

$\mathrm{N_{total}}$ - загальна кількість отриманих запитів.

Відповідно, використавши вирази 3.12 та 3.20, визначимо:

$$\lambda_o = \lambda_e(1 - \mathrm{E_{cache}}) = \lambda(1 - p_b)(1 - \mathrm{E_{cache}}) = \lambda\frac{1-\rho_s^k}{1-\rho_s^{k+1}}(1 - \mathrm{E_{cache}}) \tag{3.21}$$

Враховуючи формулу Літтла та вирази 3.18 та 3.21, час відповіді від кореневого сервера можна отримати за наступним виразом:

$$\overline{T}^{(o)} = \frac{\overline{C}^{(e)}_{total\_i}}{\lambda_o} = \frac{\pi^{(e)}_{i,0}\cdot\frac{\rho_{i0}^{C_{i+1}}}{(C_i-1)!(C_i-\rho_{i0})^2}+\rho_{i0}}{\lambda\frac{1-\rho_s^k}{1-\rho_s^{k+1}}(1-\mathrm{E_{cache}})} \tag{3.22}$$

Сумарне значення часу затримки для клієнта буде визначатись значенням часу затримки на рівні балансувальника та часом затримки на рівні вузла обслуговування.

$$T_{sum} = \overline{T}^{(s)} + \overline{T}_i^{(e)} + \overline{T}^{(o)} \tag{3.23}$$

3.4. Результати математичного моделювання

Для перевірки достовірності результатів математичного моделювання, вхідні дані, а саме інтенсивність поступлення заявок на вхід системи λ змінювалось в діапазоні від 1000 до 2200 запитів в секунду. Використовувались ці самі діапазони значень, що і в імітаційно-програмній моделі, яка описує роботи системи обробки даних в точці присутності CDN

мережі. Для генерації запитів було використано програмний комплекс MATLAB, який генерував запити в межах заданого діапазону.

3.4.1. Визначення вхідних параметрів моделі

Для перевірки достовірності результатів математичного моделювання, вхідні дані, а саме інтенсивність поступлення заявок на вхід системи λ змінювалось в діапазоні від 1000 до 2200 запитів в секунду. Використовувались ці самі діапазони значень, що і в імітаційно-програмній моделі, яка описує роботи системи обробки даних в точці присутності CDN мережі. Для генерації запитів було використано програмний комплекс MATLAB, який генерував запити в межах заданого діапазону.

Інтенсивність обробки запитів, яка визначається продуктивністю вузла обробки була задана 2000 запитів/секунду. Інтенсивність обслуговування визначається виходячи із продуктивності роботи одного контейнера, на якому проводилось програмне моделювання. Максимально можливе число запитів які міг опрацювати сервіс, що був запущений в контейнері без втрат, становить 2000 запитів. Інтервали часу між надходженнями запитів були незалежним і відповідають експоненціальному закону розподілу з частотою надходження λ (кількість запитів за секунду). Розроблена модель передбачає використання вагових коефіцієнтів, які визначатимуть ймовірність того, що число запитів, що будуть надходити на вузли обслуговування буде залежати від продуктивності обслуговуючого вузла. Значення вагового коефіцієнта буде визначатись наступним виразом

$$P_i = \sum_{i=1}^{10} \frac{C_i f_i}{C_i f_i} \tag{3.24}$$

і - це к-сть вузлів обслуговування.

З точки зору мікросервісної архітектури, всі вузли обслуговування мають однакову продуктивність (контейнери однакового розміру),

змінюється лише їх кількість в залежності від навантаження на систему. Тому в нашому випадку, вагові коефіцієнти будуть однаковими для всіх обслуговуючих пристроїв.

Довжина черги балансувальника навантаження була визначена як 10% від середньої інтенсивності поступлення запитів на вхід системи обслуговування.

Що стосується вузлів обробки, то варто зазначити, що вузлом роботи з точки зору мікросервісної архітектури вважається контейнер в якому працює певний сервіс. Особливістю роботи є також той факт, що продуктивність всіх контейнерів є однаковою, а для нарощення продуктивності просто збільшується кількість контейнерів. Це дає змогу ефективно використовувати ресурси обслуговуючих пристроїв.

3.4.2. Графічне представлення результатів математичного моделювання

Система обробки запитів, яка складається із двох підсистем та характеризується наявністю черги, в якій зберігатимуться запити які не можуть потрапити на обслуговування відразу після надходження, частину запитів може обслуговувати відразу, частина буде зберігатись у черзі та частина буде блокуватись, тобто втрачатись.

Залежність ймовірності обробки запитів вузлом обслуговування відразу після надходження від інтенсивності поступлення запитів.

На рисунку 3.6. представлено графічні залежності ймовірності бути обслуженим відразу після надходження в систему від інтенсивності поступлення заявок. На графіку представлено результати 3-х варіантів організації вузлів обслуговування. В першому випадку, 1Core_2Gops в схемі вузлів обробки був присутній один вузол, продуктивність якого становить 2000 запитів / секунду. 1Core - рівноцінно одному контейнеру в якому працює сервіс, який запитує кінцевий користувач. Наступний варіант

- 2Core_1Gops - два вузли обробки запитів (два контейнери), продуктивність яких 1000 запитів / секунду. Ваговий коефіцієнт для кожного вузла був рівний 0.5, оскільки продуктивність вузлів є однаковою. Останній варіант - 4Core_0.5Gops - чотири вузли обробки запитів (докер контейнери), продуктивність обробки запитів яких становить 500 запитів / секунду. Ваговий коефіцієнт рівний 0.25, що означає що запити рівномірно розподіляються між усіма вузлами обробки. Варто зазначити, що характерною ознакою для всіх трьох випадків є те, що при наближенні інтенсивності надходження запитів на вузол балансування до граничного значення, тобто коефіцієнт завантаженості вузлів стає близьким до 1, імовірність того, що запит буде обслужений відразу різко знижується та прямує до 0. Це очікуваний результат, оскільки обслуговуючий пристрій працює на максимальній потужності і саме в ці моменти часу різко зростатиме ймовірність блокування запитів.

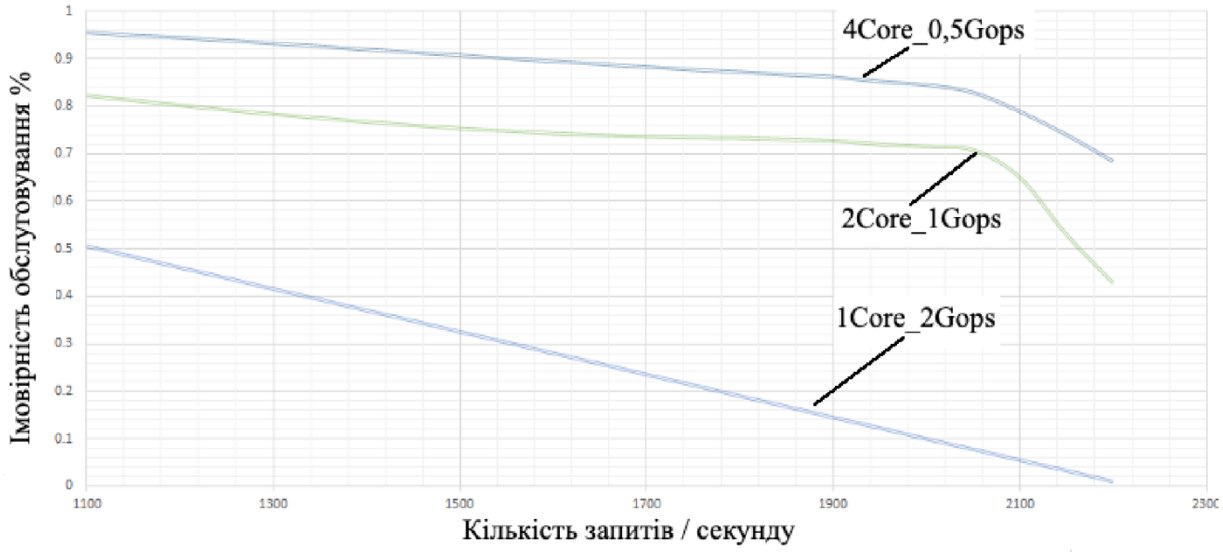


Рис.3.6 Залежність ймовірності бути обслуженим відразу після надходження в систему від інтенсивності поступлення заявок

Що стосується режимів роботи на низькій та середній завантаженості, то цікавим є той факт, що при більшій кількості вузлів обслуговування значення ймовірності обслуговування запитів відразу після надходження є

вищим, навіть при тому, що сумарна продуктивність всіх вузлів залишається без змін. Це можна пояснити тим, що при середній та низькій завантаженості вузли обробки даних працюють краще, а також обробка запитів відбуватиметься швидше.

*Залежність кількості запитів в черзі на обслуговування від інтенсивності поступлення запитів***.**

Наступним параметром, який характеризує систему обслуговування запитів користувачів є довжина черги на обслуговування. Дослідження також було проведено для трьох варіантів роботи системи обслуговування заявок. Результати представлено на рис. 3.7 - 3.8. На рисунку 3.8 представлено значення довжини черги на невеликій завантаженості вузлів обслуговування. Як бачимо, при зростанні інтенсивності надходження заявок, довжина черги збільшується, та починає різко зростати при наближенні інтенсивності поступлення до інтенсивності обслуговування запитів.

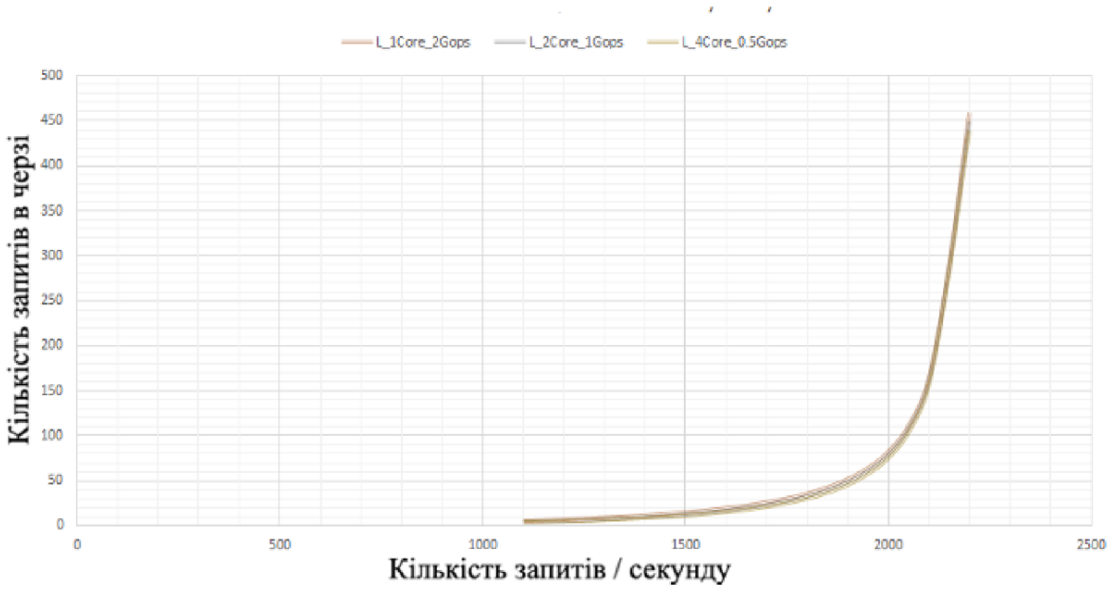


Рис.3.7 Залежність довжини черги обслуговування від інтенсивності поступлення заявок

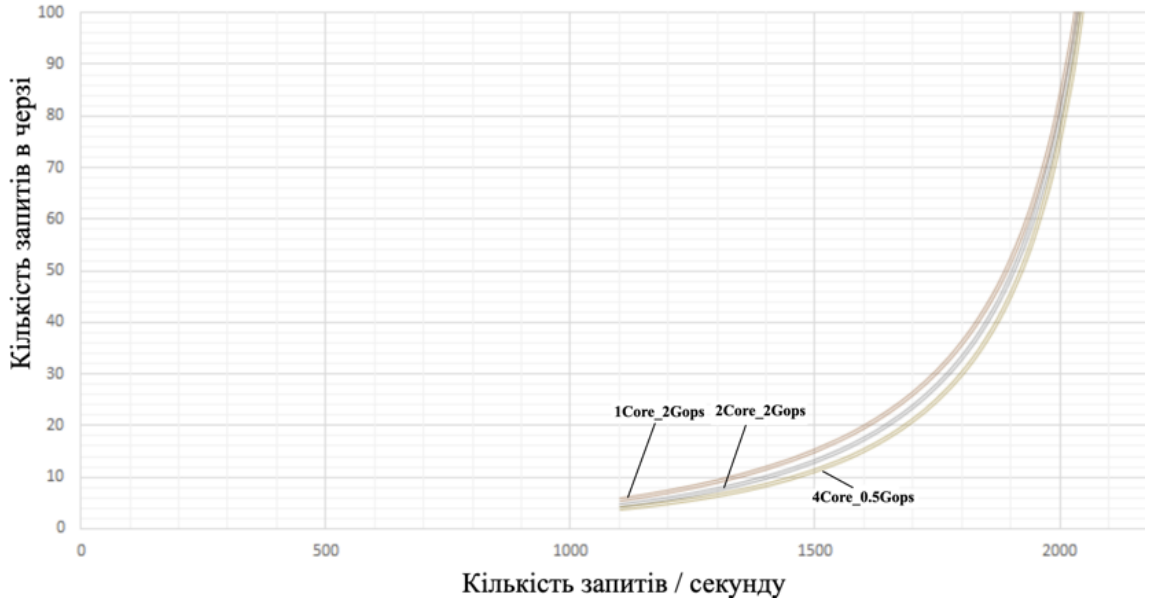


Рис.3.8 Залежність довжини черги обслуговування від інтенсивності поступлення заявок при середніх значеннях завантаженості вузлів

Як бачимо на рисунку 3.8. значення довжини черги дещо відрізняється для трьох варіантів досліджень, а саме для системи в якій є кілька вузлів обслуговування сумарна довжина черги буде дещо меншою, в порівнянні із системою в якій є один вузол обслуговування. Варто зазначити, що сумарна продуктивність системи при цьому залишається без змін.

*Залежність часу відповіді від інтенсивності поступлення запитів***.**

Ще один важливий параметр який характеризує якісні показники роботи системи в цілому є час затримки обслуговування. Враховуючи особливість запропонованої системи, цей час складатиметься із наступних складових.

- час очікування в черзі балансувальника навантаження;
- час обробки запиту балансувальник лом навантаження;
- час перебування в черзі вузла обслуговування;
- час обробки запиту вузлом обслуговування та формування відповіді клієнту.

Результати залежності часу відповіді від інтенсивності надходження запитів представлено на рис.3.9 - 3.10.

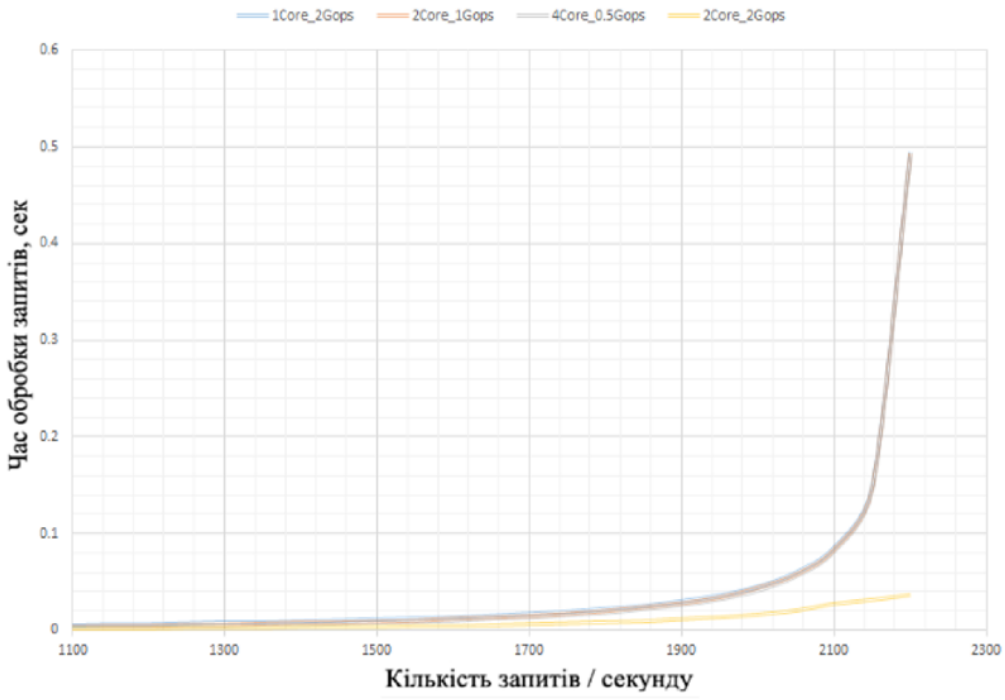


Рис.3.9 Залежність часу відповіді від інтенсивності поступлення заявок

Розрахунок часу затримки проводився для 4-х варіантів роботи підсистеми вузлів обслуговування. Варіант 1 - 1 вузол обробки даних продуктивність якого становить 2000 запитів / секунду, варіант 2 - 2 вузли обробки даних продуктивність яких становить 1000 запитів / секунду кожного, варіант 3 - 4 вузли обробки даних продуктивність яких становить 500 запитів / секунду кожного, та останній варіант - 2 вузли обробки даних продуктивність яких становить 2000 запитів / секунду кожного.

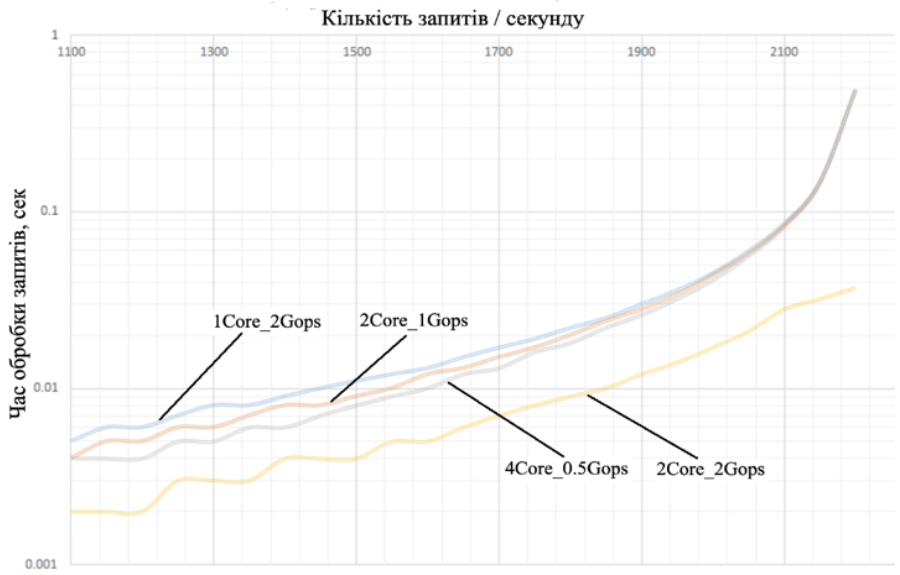


Рис.3.10 Залежність часу відповіді від інтенсивності поступлення заявок(в логарифмічному масштабі)

Для перших трьох варіантів організації підсистеми роботи вузлів обслуговування, бачимо, що час відповіді різко зростає в момент коли завантаженість вузлів обробки сягає > 95% та стає практично однаковим. Однак при завантаженості вузлів в межах до 75%, можна побачити, що система із більшою кількості вузлів обробки показує кращі результати в плані часу відповіді. Це свідчить про те, що середньо завантажені вузли обробки даних працюють стабільніше та показують кращі результати. Також як бачимо із результатів роботи варіанту 4 час затримки при досягненні інтенсивності обслуговування максимального значення 2200 запитів / секунду все ще залишається досить низьким у порівнянні із іншими варіантами. Це можна пояснити тим, що продуктивність системи яка використовувалась в 4-му варіанті була збільшена в 2 рази, тому при максимальному значенні інтенсивності надходження запитів, система все ще працює в режимі низького навантаження.

Залежність часу відповіді від інтенсивності поступлення запитів при різних значеннях коефіцієнта ефективності кешування.

Не менш важливим параметром, який характеризує якісні показники роботи системи в цілому є час затримки обслуговування на кореневому сервері. Враховуючи особливість запропонованої системи, цей час буде залежати від коефіцієнта ефективності кешування даних на граничних серверах. Чим меншим буде значення коефіцієнта ефективності кешування даних на, тим більше запитів буде перенаправлено до кореневого сервера, відповідно тим більшим буде значення часу відповіді системи вцілому. В процесі математичного моделювання було вибрано три значення коефіцієнта ефективності кешування, а саме:

- 0 - кешування відсутнє, в такому варіанті всі запити перенаправляються на кореневий сервер.

- 0.6 (60%) - ефективність кешування, яку забезпечують існуючі методи балансування та обробки даних в граничних локаціях мереж доставки.

- 0.9 (90%) - ефективність кешування, яку забезпечує запропонований в роботі метод розподіленої обробки ресурсозатратних запитів та інтегральний ключ кешування на рівні балансувальника навантаження.

Результати залежності часу відповіді від інтенсивності надходження запитів представлено на рис.3.11.

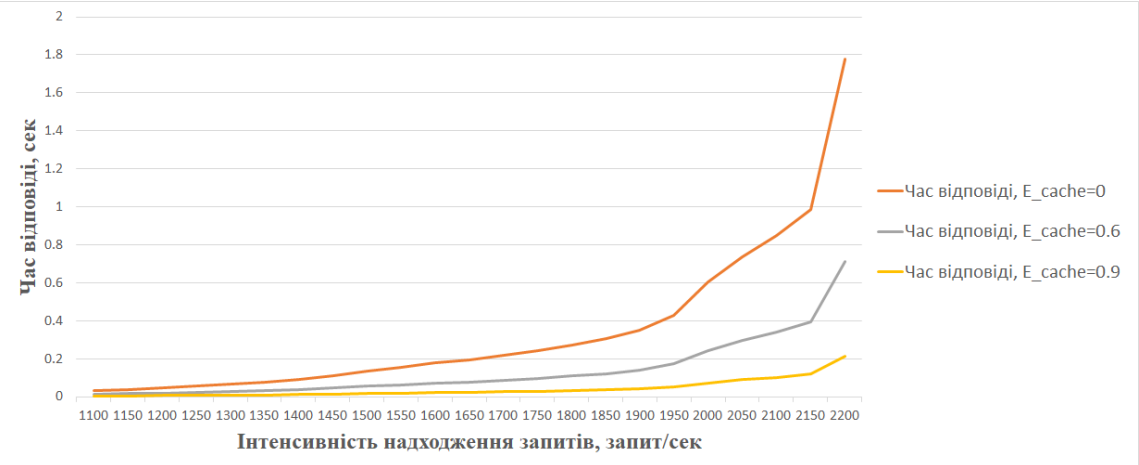


Рис.3.11. Залежність часу відповіді від інтенсивності поступлення заявок при різних значеннях коефіцієнта ефективності кешування

Результати, представлені на рисунку 3.11, показують, що при відсутності кешування даних(коефіцієнт ефективності кешування $E_{\_cache}$ = 0), час відповіді починає суттєво зростати при збільшенні навантаження. Це пов’язано з тим, що всі запити повинні бути оброблені всіма системами (балансувальник навантаження, вузол обробки в граничній локації CDN мережі та кореневий сервер). При використанні кешування даних, час відповіді суттєво зменшується із збільшенням коефіцієнта ефективності кешування. При значенні $E_{\_cache}$ = 0,9, навіть при суттєвому зростанні навантаження, час відповіді залишається в межах 180 - 200мс.

Результати математичного моделювання, підтверджують ефективність застосування запропонованих в роботі методів розподіленої обробки ресурсозатратних запитів, та балансування навантаження із використанням інтегрального ключа кешування та даних аналітики.

3.5. Висновок до розділу 3

Представлено комплексну модель масового обслуговування з трьома об’єднаними системами масового обслуговування для характеристики процесу обслуговування запитів у хмарних сервісах із динамічним виділенням ресурсів. Динамічне виділення ресурсів представляє собою механізм, який дає можливість створювати нові інстанси для обробки даних лише за необхідності, тобто при умові, що існуючі не встигають обробляти всі запити, що надходять в систему, або якість послуг, які забезпечують існуючі, стає недопустимою для забезпечення хорошого сервісу який отримує кінцевий користувач. Три системи масового обслуговування складають складну модель яка характеризує неоднорідність ЦОД та сервісні процеси, які задіяні в хмарних обчисленнях (Edge Compute). Перша - це рівень входу в точку присутності сервісу в Хмарі, друга - виконання обробки даних на прикладі системи масового обслуговування на рівні вузла обробки запитів, який є найближче до користувача та має для цього наявні

ресурси, третя – виконання обробки даних на кореневому сервері. В якості математичної моделі першої підсистеми, а саме балансувальника навантаження, запропоновано систему масового обслуговування класу $M/M/1/k$ - система масового обслуговування з обмеженою довжиною черги k, та 1 обслуговуючим пристроєм. Оскільки кожне вузол обробки може паралельно обробляти кілька завдань, він розглядається як система масового обслуговування $M/M/\text{n}$. Вхідний потік заявок описується розподілом Пуассона із інтенсивністю надходження λ (часові інтервали між поступленням завдань на обслуговування є незалежними).

На основі математичного моделювання проведено оцінку параметрів якості обслуговування, а саме ймовірності миттєвого обслуговування, довжини черги обслуговування та часу відповіді в залежності від інтенсивності надходження запитів та продуктивності вузлів обслуговування.

В результаті моделювання отримано графічні залежності, які показують, що системи обслуговування даних працюють досить добре при середніх навантаженнях вузлів обслуговування. При пікових навантаженнях, ефективність роботи системи різко знижується. Було встановлено, що система яка складається із більшої кількості менш продуктивних вузлів обробки даних, буде працювати краще, ніж система яка складається із одного вузла більшої продуктивності за умови, що особливість трафіку не вимагає складних обчислень та великих затрат процесорного часу. Також з точки зору надійності та відмовостійкості така система буде більш стабільна та довговічна, оскільки використання вузлів обробки при роботі на максимальній потужності призводить до швидкого зношування ресурсів. Що стосується мікросервісної архітектури, завжди ефективнішим є використовувати більшу кількість маленьких за продуктивністю вузлів обслуговування, та в будь який час мати змогу здійснити швидке масштабування.

Результати математичного моделювання підтверджують ефективність застосування запропонованих в роботі методів розподіленої обробки запитів, що потребують значних обчислювальних ресурсів та балансування навантаження із використанням інтегрального ключа кешування та даних аналітики.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ МЕТОДУ АДАПТИВНОГО РОЗГОРТАННЯ МІКРОСЕРВІСУ В ТОЧЦІ ПРИСУТНОСТІ CDN МЕРЕЖІ

Зростаюча конкуренція між постачальниками послуг, які використовують CDN мережі для доставки даних, мотивує вдосконалення технологій оптимізації трафіку, розширення функціональних можливостей платформ та зниження затримок у передачі даних. Водночас ключовими викликами залишаються управління масштабованістю, безпекою даних та енергоефективністю інфраструктури, особливо в умовах глобального розширення IoT-екосистем. Доступність сервісів і забезпечення гарантованої якості обслуговування (QoS) залишаються ключовими вимогами сучасної цифрової інфраструктури. Проте в умовах стрімкого розвитку технологій і зростання вимог користувачів стандартних підходів, таких як кешування даних, вже недостатньо для утримання їхньої уваги.

Постачальникам послуг необхідно зосередитися на впровадженні інноваційних рішень, таких як персоналізовані сервіси, адаптивне управління трафіком, інтеграція з технологіями штучного інтелекту (AI) та машинного навчання (ML), а також розширення функціональності платформ за рахунок аналітичних інструментів.

4.1. Концепція використання технології розподілених граничних обчислень

У звичайних корпоративних мережах дані виникають або генеруються на клієнтських пристроях, таких як комп'ютер користувача, мобільний пристрій або будь-який пристрій Інтернету речей. Ці дані в подальшому переміщуються через мережу WAN, таку як Інтернет, або через корпоративну локальну мережу, де вони проходять подальшу обробку

та зберігаються за допомогою різних сервісів. Після обробки, результати повертаються до кінцевого користувача.

Цей підхід є стандартним у клієнт-серверних обчисленнях та широко застосовується у більшості типових бізнес-додатків.

Однак, кількість пристроїв, які підключені до локальних мереж, чи глобальної мережі Інтернет, та обсяг створюваних ними даних швидко зростає, що створює проблеми для традиційної інфраструктури та центрів обробки даних. Згідно з прогнозами Gartner, до 2025 року 75% даних буде генеруватися за межами централізованих центрів обробки даних. Це ставить під загрозу глобальний Інтернет, який часто стикається з перевантаженнями і збоями. Відповідно, IT-архітектори звертають увагу на логічно-розподілену інфраструктуру, відводячи ресурси зберігання та обчислювальні можливості з центрів обробки даних і переміщуючи їх до місця, де генеруються ці дані або запити. Принцип простий: якщо не можна наблизити дані до центру обробки даних, то наблизьте центр обробки даних до даних. Ця концепція не є новою і базується на ідеї віддаленого обчислення, коли ресурси обробки даних розміщуються ближче до їхнього джерела, що є більш надійним і ефективним в порівнянні з централізованим обслуговуванням та зберіганням.

4.2. Основні підходи впровадження технології розподілених граничних обчислень

У сучасних екосистемах Інтернету речей (IoT) активно впроваджується технологія Edge Computing, яка широко використовується для розподіленої обробки даних та забезпечення швидкого аналізу інформації, що надходить від різних джерел, таких як IoT пристрої. Edge Compute визначається як архітектура, в рамках якої обробка даних відбувається на граничних локаціях мережі, якомога ближче до місця де генеруються самі дані. Дані стають цінним ресурсом для сучасних

підприємств, що дозволяє їм в реальному часі здійснювати контроль за ключовими бізнес-процесами та операціями. Відбувається нескінченне зростання обсягів даних у світі сучасного бізнесу, які регулярно збираються з датчиків та IoT-пристроїв, що працюють у реальному часі, надходять з різних кінців світу. Такий потік даних вимагає ефективної обробки та аналізу, щоб використовувати їхній потенціал у розвитку та управлінні бізнесом [109-110].

Цей підхід диктує нові вимоги щодо обчислення та передачі таких даних з гарантованою якістю. У сучасному світі такий потік інформації вимагає миттєвої обробки та мінімізації часу доставки результатів, які йдуть у обидва напрямки одночасно.

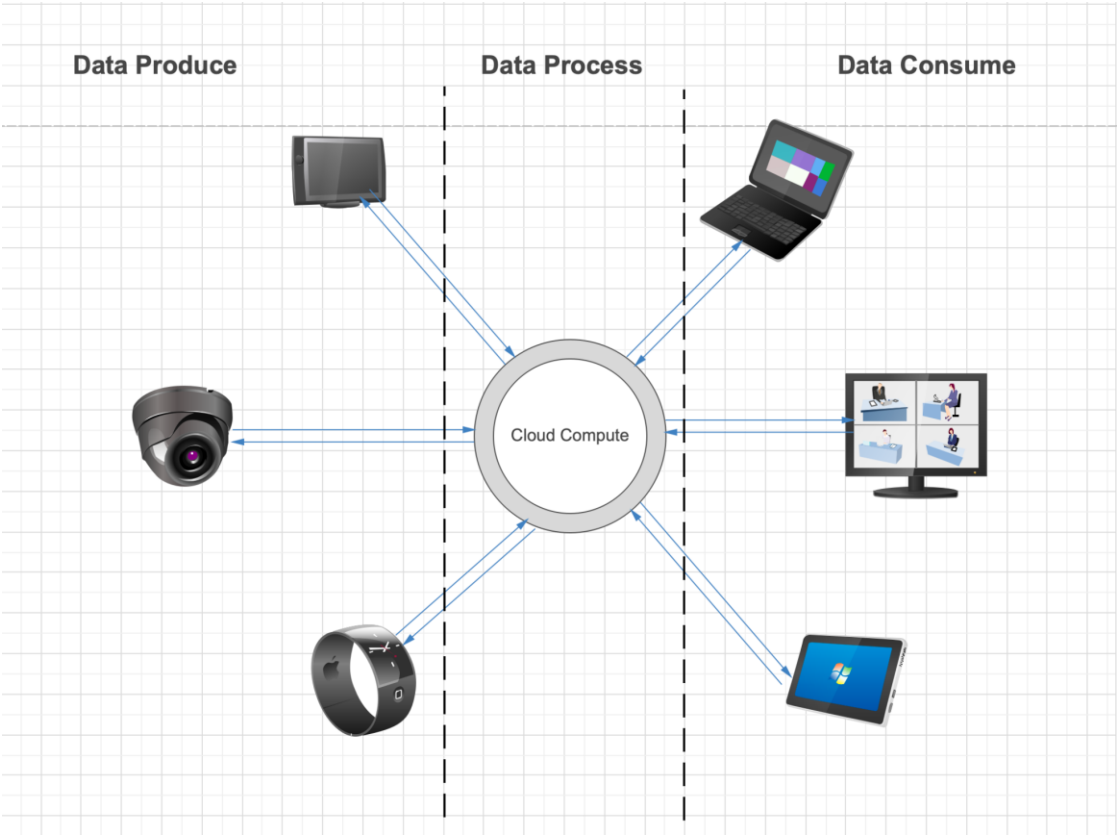


Рис.4.1. Базова схема використання технології розподілених граничних обчислень

Звичайний підхід до обробки та передачі даних, що базується на централізованому центрі обробки даних, не є оптимальним для великих обсягів інформації, які постійно зростають. В цьому контексті виникають проблеми з обмеженою пропускною здатністю, затримками, джитером та іншими непередбачуваними недоліками мережі передачі даних. У відповідь на ці виклики компанії використовують архітектуру розподілених обчислень [111].

Іншими словами, основні процеси аналізу та обробки даних розташовуються якомога ближче до джерела генерації цих даних. Замість того, щоб передавати необроблену інформацію до централізованого центру обробки даних для подальшого аналізу, ці завдання виконуються там, де дані по суті генеруються. Це може бути магазин роздрібної торгівлі, виробничий цех, розподілена мережа водопостачання або ж інтелектуальне місто. Тільки результати цих обчислень, такі як прогнози технічного обслуговування обладнання чи інші висновки, надсилаються назад до центрального центру обробки даних для подальшої обробки. Таким чином, технологія граничних розподілених обчислень змінює інформаційні технології та підходи, що використовуються у бізнес-аналізі.

4.3. Застосування методу адаптивного створення мікросервісу для обробки даних в CDN мережі

Базова схема CDN мережі, описує принципи роботи CDN мережі розкриває основні аспекти її функціонування, зокрема кешування даних від сервера походження та обробку запитів від кінцевих користувачів. Однак граничний сервер може не лише зберігати дані, але й частково опрацьовувати запити користувачів, тобто виконувати "Computing", або обробку цих даних. У такому випадку ми можемо використовувати принцип Edge Compute в оберненому напрямку, коли дані знаходяться на кореневому сервері, а клієнт робить запит щоб їх отримати. Отже,

граничний сервер кешування, виконуватиме не лише функцію збереження даних, але і їхню обробку, якщо це можливо. Схема роботи такого підходу показана на рисунку 4.2. [112].

Опис роботи алгоритму адаптивного створення мікросервісу для обробки даних в CDN мережі наступний:

- Кінцевий користувач, користуючись тим чи іншим сервісом, надсилає запит на отримання певних даних цього сервісу. Одним із найяскравіших прикладів може бути список товарів на сторінці інтернет-магазину.
- Відповідно до принципів функціонування CDN, такий запит потрапить до граничного кешуючого сервера, який знаходиться максимально близько до користувача, або прямо в його географічній локації.
- Наступним кроком, граничний сервер перенаправить цей запит до кореневого сервера на якому працює відповідний тип сервісу, з яким працює користувач.
- Кореневий сервер (сервер походження - Origin) починає виконувати обробку запиту який надійшов від граничного сервера та формує відповідь. Цей процес звичайно займає певний час, який включатиме як мережеву затримку так і час самого виконання.
- В той самий час, модуль аналізу граничних обчислень Edge Compute, періодично опитує кореневий сервер та збирає метрики про його стан, завантаженість а також проводить аналіз часу відповіді на запити які він опрацьовує. Окремий компонент проводить аналіз цих значень та приймає рішення чи якість надання послуг кінцевому користувачу буде задовільною при певному значенні завантаженості сервера походження та часу відповіді на запити клієнтів.
- При умові, що значення всіх параметрів є задовільними, граничний сервер кешування формує відповідь для клієнта.

- У випадку, коли значення починають зростати, а саме час обробки запитів, завантаженість кореневого сервера, модуль аналізу граничних розподілених обчислень приймає рішення про перенесення процесу обробки запитів клієнтів на сторону граничного сервера. Таким чином певна частина бізнес логіки роботи аплікації, яка виконувалась на кореневому сервері, переноситься на сторону граничного сервера. Однак все ще залишається певна частина даних, або запити, попередньо оброблені на граничному сервері які надсилаються на сервер походження.
- Отож, модуль аналізу граничних розподілених обчислень забезпечує зменшення навантаження на граничний сервер а також час відповіді для клієнта, за рахунок перенесення частини функціоналу та бізнес логіки роботи сервісу на свою сторону. Також він може виконувати збір статистики, та на основі аналізу цих даних наперед спрогнозувати час можливого зниження QoS та заздалегідь ініціювати процес обробки запитів клієнтів на своїй стороні. За певний проміжок часу, такий модуль адаптується до особливостей роботи певного сервісу та зможе заздалегідь делегувати обробку даних на кешуючі сервери, які розташовані максимально близько до клієнта.

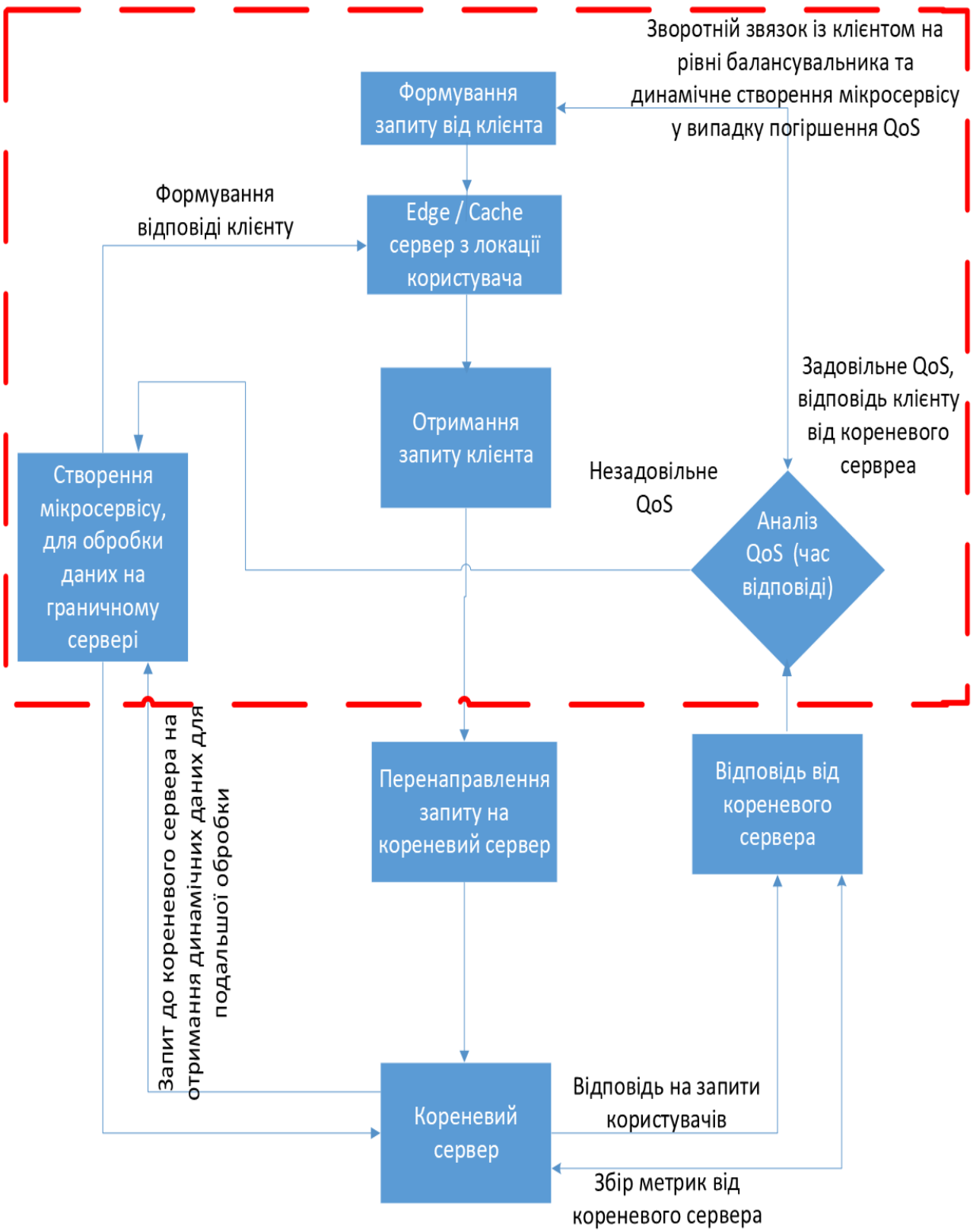


Рис.4.2. Блок схема роботи методу адаптивного створення мікросервісу для обробки даних в CDN мережі

4.4. Аналіз результатів роботи методу адаптивного створення мікросервісу для обробки даних в CDN мережі

Схема методу представлена на рисунку.4.3

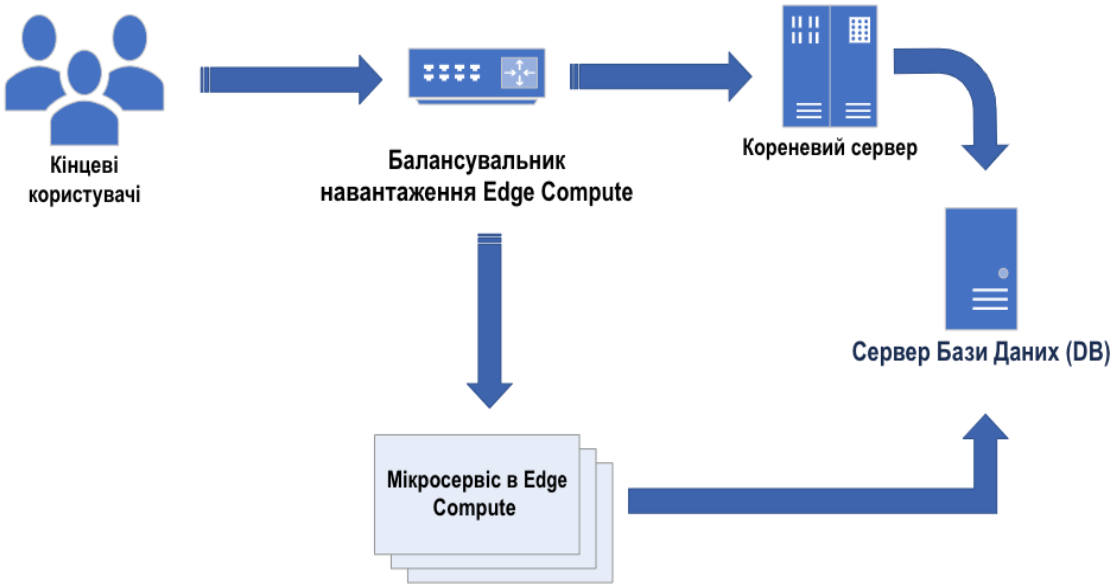


Рис.4.3. Схема організації методу адаптивного створення мікросервісу в CDN мережі із використанням централізованого сховища даних

Роль кінцевих користувачів виконувала аплікація Apache Jmeter. Модель адаптивних граничних обчислень працює згідно описаного вище алгоритму. Edge Compute Service - за нормальних умов роботи цей сервіс не задіюється. Його мета - це швидкий старт при умові, що к-сть трафіку та навантаження різко зростає і сервер походження не встигає обробляти всі задані запити із задовільною для користувача якістю. Origin Server - це сервер на якому працює сама аплікація та вся бізнес логіка. Запропонована модель може використовуватись у двох варіантах. Перший представлено на рисунку 4.3. В даному варіанті, сервіс Edge Compute так само як і кореневий сервер звертатимуться до спільної бази даних. Перевагою такого підходу є те, що всі дані централізовано зберігаються в одному місці, не потрібно використовувати додаткові технології для синхронізації розподілених систем збереження даних. Недоліком може бути час доступу до сховища

даних з географічно розподілених Compute серверів. Другий варіант представлено на рисунку 4.4

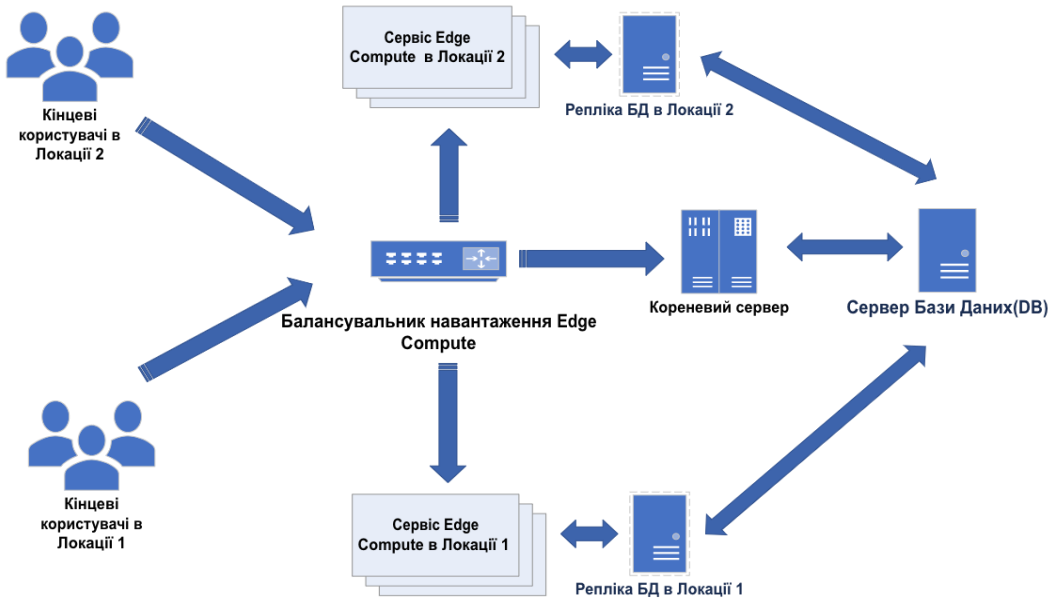


Рис.4.4. Схема організації методу адаптивного створення мікросервісу в CDN мережі із використанням розподіленого сховища даних

Особливістю роботи даної схеми є те, що джерело даних є не централізованим а розподіленим. Розподіленими будуть тільки ці дані, які використовуються Compute сервісом для проведення розрахунків та повторення бізнес логіки роботи аплікації в цілому. Для організації такого підходу потрібно в кожній Edge локації розгортати сервіс для реплікації частини даних із центрального сховища. У випадку баз даних, це може бути read репліка бази, в яку реплікуються тільки необхідні дані. Перевагою такого підходу є те, що в момент коли активується сервіс адаптивних граничних обчислень дані вже є доступні в локальному сховищі. Це дає можливість зменшити час відповіді за рахунок значного зменшення мережевої затримки між сервером обробки та централізованим сховищем. Недоліками такого рішення є:

- безпека - зберігання даних за межами локації постачальника послуги. Тут звичайно варто зазначити, що варто реплікувати тільки дані,

які не є критичними з точки зору безпеки та бізнесу. Наприклад інформацію про товар, опис товару, ціна на товар, наявність на складі та інше. Зрозуміло, що не варто виконувати реплікацію приватних даних користувачів, номери банківських рахунків та інші чутливі з точки зору безпеки дані.

- забезпечення синхронізації даних між централізованим сховищем та розподіленими репліками в режимі реального часу. Завжди є ймовірність виникнення ситуації, коли для прикладу ціна на товар обновилась в централізованому сховищі а синхронізація в конкретний момент часу ще не відбулась на одну із реплік. В такому випадку може трапитись ситуація, що користувач який був перенаправлений на Edge Compute в найближчій до нього локації побачить ціну на товар, яка відрізнятиметься в певний момент часу від ціни, яку бачитиме користувач, запит якого буде перенаправлений на кореневий сервер.

В роботі було проведено моделювання кількох сценаріїв. Моделювання проводилось на створеному прототипі мережі доставки контенту, який представлений на рисунку 4.5.

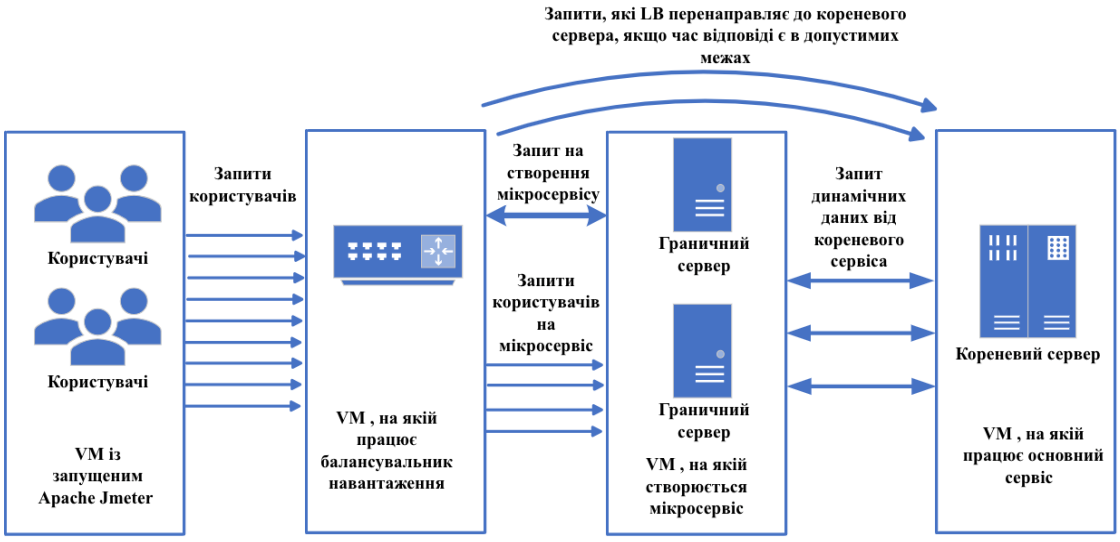


Рис.4.5. Прототип мережі доставки контенту CDN із використанням балансувальника навантаження в граничній локації

Кожен сценарій розглянемо більш детально.

Сценарій 1.

Кількість запитів становить 1500 запитів/секунду. Під час моделювання, було проведено чотири стрибки навантаження до 2000 запитів/секунду різної тривалості. Під час першого сценарію, весь трафік оброблявся сервером походження, адаптивні граничні обчислення не було задіяно. Результати моделювання представлені на рисунках 4.6-4.7.

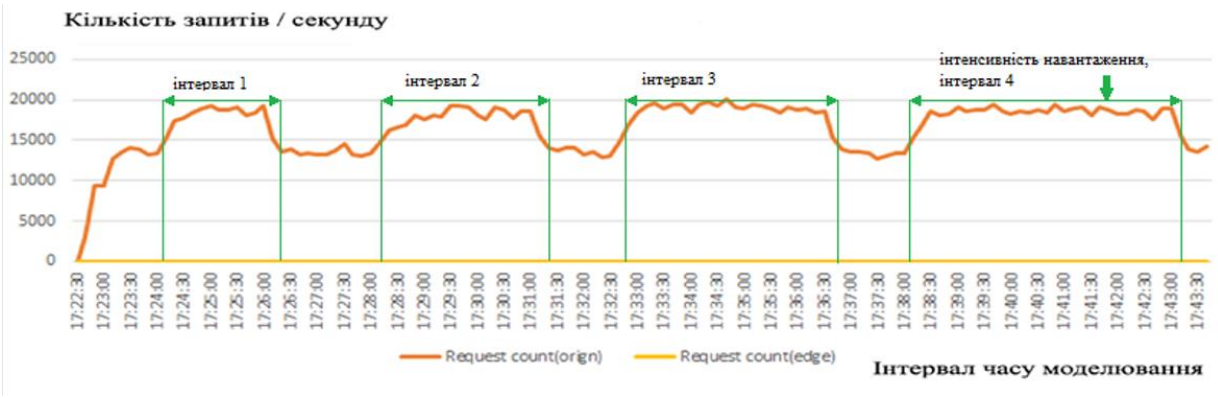


Рис.4.6. Кількість запитів на проміжку часу моделювання

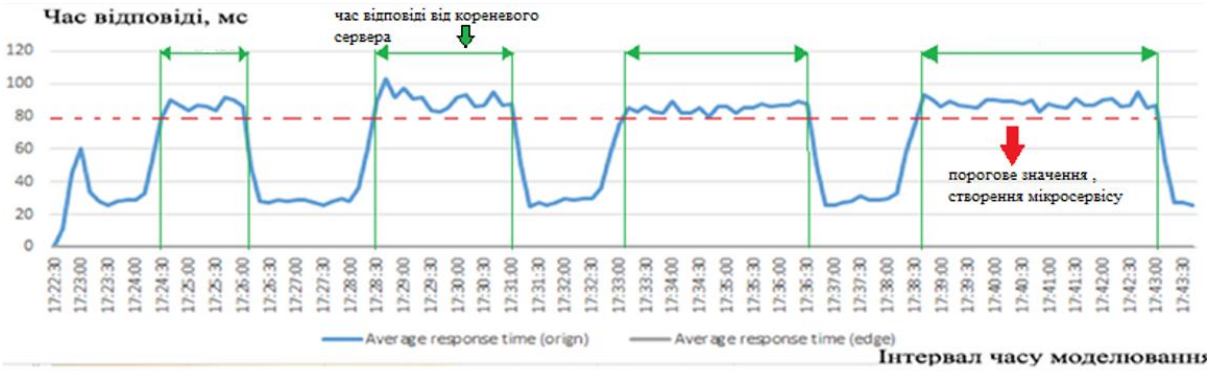


Рис.4.7. Час відповіді від кореневого сервера на проміжку часу моделювання

Як можна побачити із рисунків 4.6.-4.7. при звичайному навантаженні, 1500 запитів / секунду, час відповіді був в межах 30-40 мс. Перший стрибок навантаження тривав 2 хв, при цьому весь трафік Edge

Compute Load Balancer надсилав тільки на кореневий сервер. Час відповіді при такому навантаженні був в межах 90-100 мс. Наступні стрибки були дещо довші, тривали 3, 4 та 5 хв. Навантаження було в межах 2000 запитів / секунду, а час відповіді теж зберігався в межах 90-100мс. Підсумовуючи результати першого експерименту, можна стверджувати, що при зростанні навантаження на третину, час відповіді зростав більше як у два рази без використання Edge Compute сервісу.

Сценарій 2.

На початку експерименту, кількість запитів становила 1500 запитів/секунду. Параметром, який визначив якість надання послуг було прийнято час відповіді. Поріг допустимих значень було визначено 80мс. При умові, що навантаження на сервер походження зростало і час відповіді становив більше 80 мс, Edge Compute Load Balancer запускав аплікацію на кешуючому сервері і починав перенаправляти запити на цей кешуючий сервер для їх обробки. В даному експерименті, частина запитів які потрапляли на кешуючий Edge Compute сервер становила 10% від загальної кількості запитів які потрапляли на Load Balancer. Кожні 30 сек к-сть запитів які Load Balancer перенаправляє на EC збільшувалась на +10% від загальної кількості. Як і в попередньому експерименті було проведено чотири стрибки навантаження до 2000 запитів/секунду тривалості 2, 3, 4 та 5х. Результати моделювання представлено на рисунках 4.8-4.10.

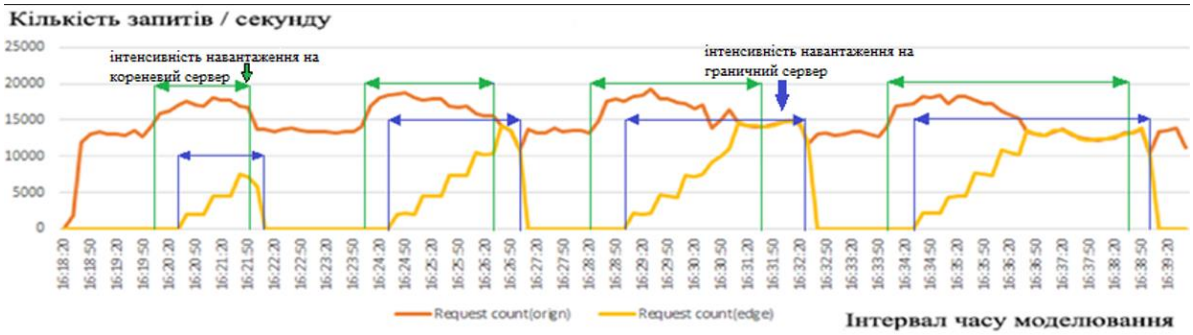


Рис.4.8. Кількість запитів на проміжку часу моделювання на кореневий та граничний сервер

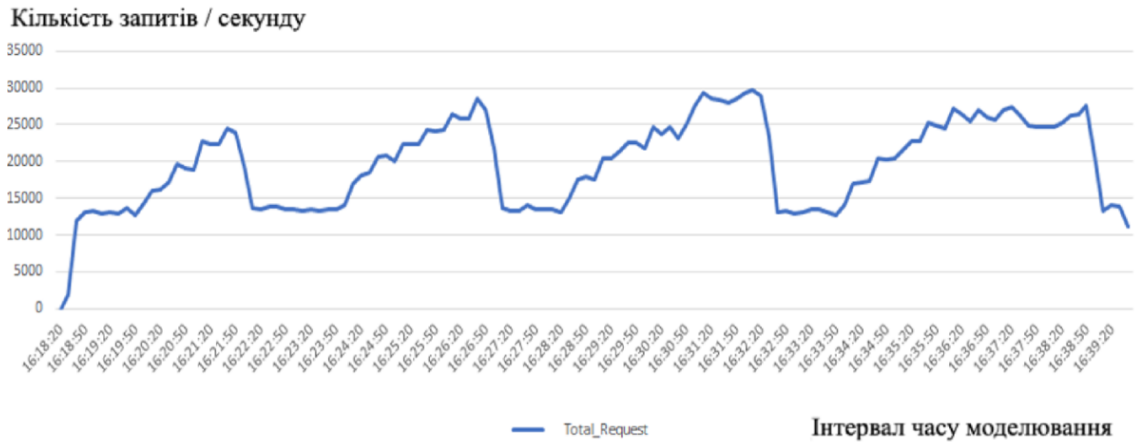


Рис.4.9. Сумарна кількість запитів на проміжку часу моделювання

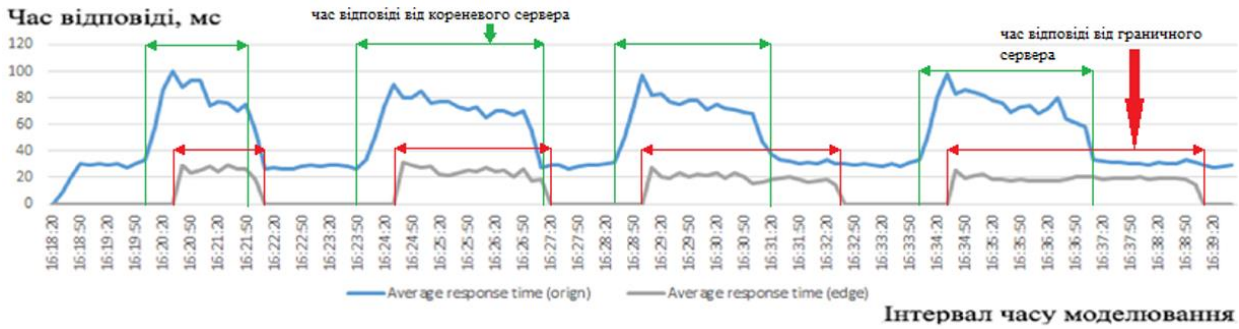


Рис.4.10. Час відповіді від кореневого та граничного сервера на проміжку часу моделювання

З представлених вище рисунків можна побачити як змінювався час відповіді сервера походження, та в які моменти часу активувався Edge Compute. Як тільки значення часу відповіді від Origin сервера перевищувало 80мс, Edge Compute Load Balancer активував Edge Compute на кешуючому сервері і 10% запитів перенаправив на нього. При цьому, число запитів на сервер походження починає зменшуватись та відповідно зменшувався і час відповіді. Значення часу відповіді на EC сервері становить 20-30мс. Як можна побачити з рисунку 4.10, при запуску аплікації час відповіді є дещо більшим, однак вже за кілька секунд стає в межах 20мс. Це спричинено запуском самої аплікації, підвантаження даних для обробки (“холодний старт”). Ще стосується сервера походження, то на графіку 4.10. чітко видно,

що під час зростання навантаження, час відповіді різко зростає і становить більше 80мс. В момент часу коли активується ЕС, частина навантаження перенаправляється на кешуючий Edge сервер, відповідно час відповіді сервера походження поступово зменшується. Як тільки половина всіх запитів перенаправляється на сервер ЕС, час відповіді від сервера походження вертається практично до початкового значення. На рисунку 4.9 можна побачити як зростало сумарне навантаження у вигляді кількості запитів на Load Balancer.

Сценарій 3

Цей сценарій моделювання був дуже наближеним до попереднього. Вхідні параметри задавались такими ж, основною відмінністю було те, що Load Balancer відразу 50% запитів перенаправив на сервер ЕС. Результати моделювання представлено на рис 4.11-4.13.

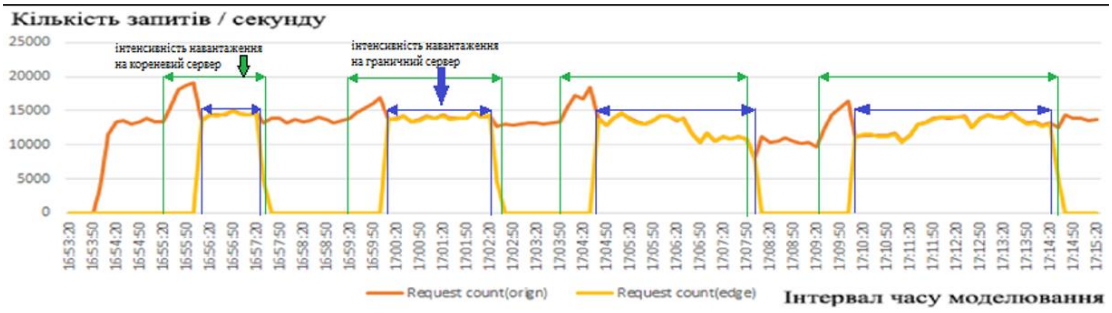


Рис.4.11. Кількість запитів на проміжку часу моделювання на кореневий та гарничний сервер

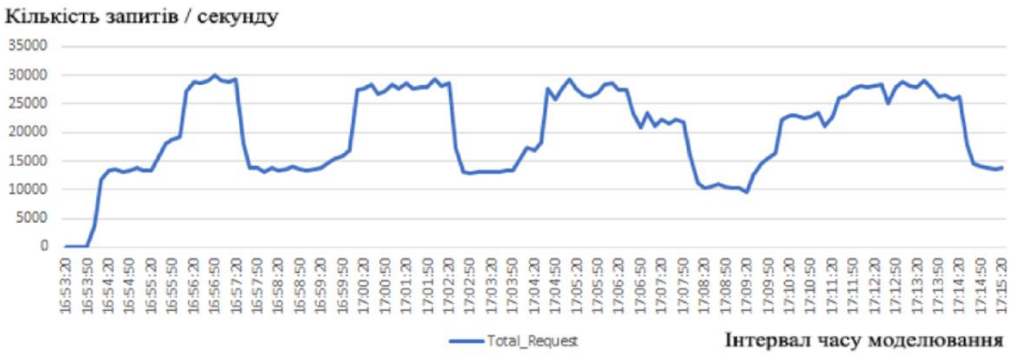


Рис.4.12. Сумарна кількість запитів на проміжку часу моделювання на балансувальник навантаження

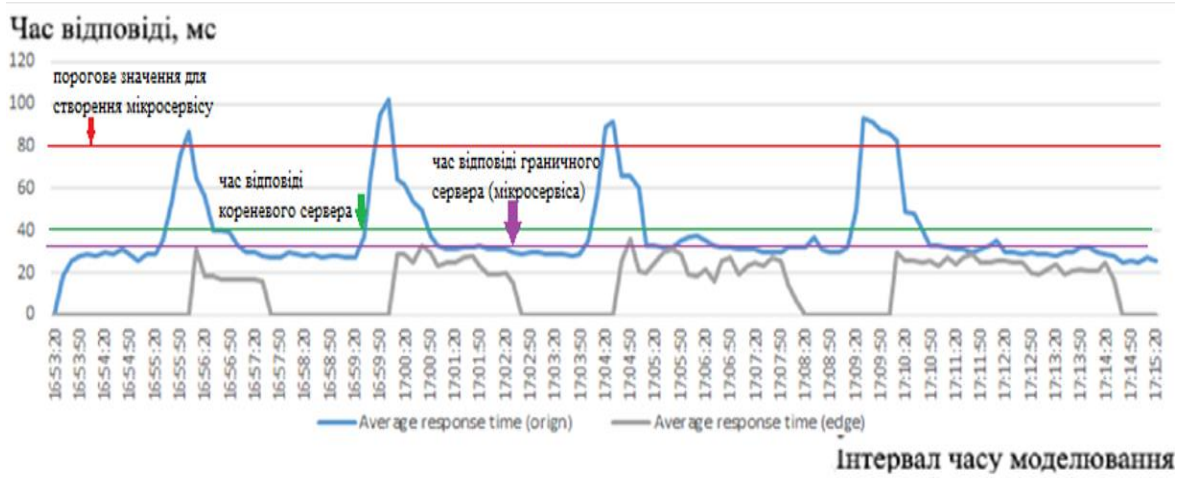


Рис.4.13. Час відповіді від кореневого та граничного сервера на проміжку часу моделювання

Аналізуючи результати моделювання, можна побачити, що як тільки час відповіді від сервера походження досягав встановленого порогового значення, а саме 80 мс, балансувальник навантаження половину всіх запитів перенаправив на ЕС. На протязі хвилини, час відповіді від сервера походження вернувся до початкових значень, а час відповіді від сервера ЕС був у межах 20-30 мс. Якщо порівняти результати із попереднім сценарієм моделювання, то перевагою є те, що навантаження на сервер походження зменшується практично відразу, час відповіді також вертається в межі 30 - 40 мс. Єдиним недоліком такого підходу буде те, що якщо к-сть запитів буде надто великою, то перенаправивиши відразу велику частину на ЕС можна спричинити миттєва перевантаження цього сервера і зробить неможливим якісне обслуговування запитів які на нього надходять.

Якщо навантаження перенаправляти поступову, то це дозволить серверу ЕС з великою ймовірністю обробляти всі запити без втрати якості обслуговування.

Варто зазначити, що досить часто виникають сплески навантаження які тривають кілька хвилин. В таких випадках, активовувати модуль Edge Compute є недоцільно, оскільки варто враховувати час на створення

ресурсу, запуск сервісу та кешування частини даних від сервера походження. Клієнт у вигляді Jmeter імітував звернення користувачів до інтернет-магазину щоб отримати інформацію про певний товар, а саме опис товару, наявність на складі, категорію товару, зображення, відгуки про цей товар, а також додаткові атрибути товару. Модуль граничних розподілених обчислень, в даній моделі - це окремий сервіс який виконував частину бізнес логіки основного сервісу, таким чином зменшуючи навантаження на основний сервіс та тим самим і час відповіді клієнту.

Як показують результати моделювання, ефективність застосування даного модуля є очевидною, оскільки значення часу затримки для кінцевого користувача зменшилось навіть при умові, що к-сть запитів зросла у два рази. Також слід розуміти шо у випадку якщо одного модуля граничних розподілених обчислень буде недостатньо, то балансувальник навантаження в будь який момент часу може активувати додаткові і частину запитів спрямовувати на них. Варто зазначити також, що всі моменти активації / деактивації модуля зберігаються з метою подальшого аналізу та можливості наперед прогнозувати періоди зростання навантаження та попередньої активації із збереженням параметрів якості обслуговування в допустимих межах. Це в свою чергу впливає на якість сприйняття роботи вашого сервісу кінцевими споживачами.

4.4.1. Приклад використання інфраструктури як коду для організації методу адаптивного створення мікросервісу в CDN мережі

Розглянемо роботу одного із методів створення сервісу та дослідження його роботи на прикладі хмарного сервісу Google Cloud Run.

Cloud Run - це керована обчислювальна платформа, яка дає змогу запускати контейнери без збереження стану, які можна викликати через запити HTTP. Cloud Run є безсерверним: він абстрагує все керування

інфраструктурою, тому ви можете зосередитися на найважливішому - створенні чудових програм [113].

Оскільки образ докера є обов'язковим як вхідний елемент для Cloud Run, розробник може вільно використовувати будь-яке програмне забезпечення для забезпечення функціональності та обробки запитів [114-115].

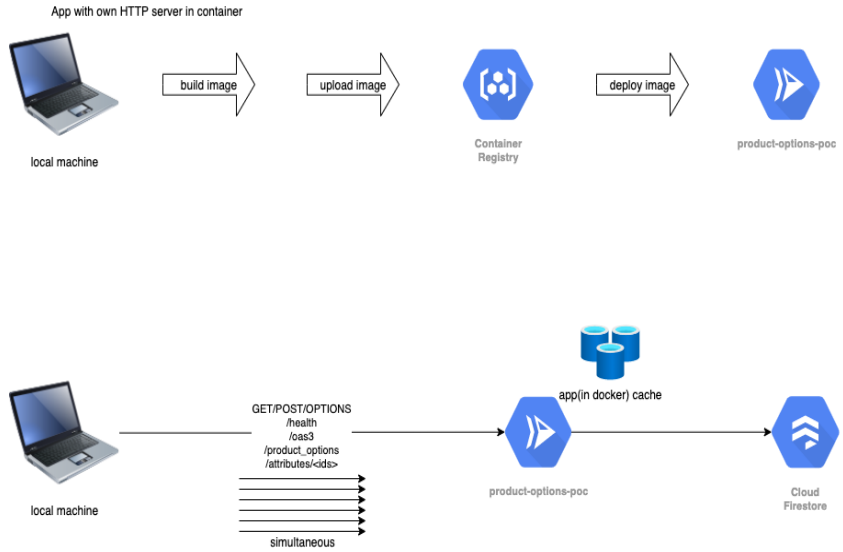


Рис.4.14. Схема розгортання та роботи сервісу Cloud Run

Щоб запустити GCP Cloud Run, потрібно виконати 2 кроки. Перший — створити образ докера з вихідного коду та відправити його в реєстр Google. Наступний – розгортання образу програми з GCR в Cloud Run. Cloud Run приймає образи контейнерів, створені за допомогою будь-якого із наявних для цього інструментів. Особливість роботи Cloud Run в тому, що при відсутності запитів цей сервіс не використовує ресурси. У випадку появи запитів, створюється робочий інстанс, який виконує обробку цих запитів. Під час створення інстансу оброки, виникає проблема "холодного старту", яка в перші моменти роботи сервісу не завжди забезпечує

задовільні значення часу обробки. Результати обробки запиту при “холодному старті”:

```
$ # Холодний старт аплікації
$ time curl --location --request GET 'https://test-poc-k5kabakx3a-uc.a.run.app/health'
{
  "run": "COLD",
  "status": "OK"
}
real    0m3,776s
```

Результат обробки такого ж запиту при тривалій роботі сервісу:

```
$ # Робочий старт 1
$ time curl --location --request GET 'https://test-poc-k5kabakx3a-uc.a.run.app/health'
{
  "run": "WARM",
  "status": "OK"
}
real    0m0,364s
$ # Робочий старт 2
$ time curl --location --request GET 'https://test-poc-k5kabakx3a-uc.a.run.app/health'
{
  "run": "WARM",
  "status": "OK"
}
real    0m0,304s
```

Як можна побачити із результатів, тривалість обробки запитів при “холодному старті” становить 3.7сек, а при тривалій робі “робочому старті” 0.3 - 0.35с

В таблиці 4.1 представлено результати експериментальних досліджень роботи аплікації, а саме, як залежить час відповіді від моменту коли здійснюється обробка запиту.

Таблиця 4.1.

Час відповіді аплікації, яка запущена в Cloud Run

	GCP Cloud Run	
	Відповідь клієнту, мс	Логування, мс
Перший запит	5254	3574
Другий запит	1376	290
Середнє значення для 40 запитів	414	83
40 запитів / секунду	352	69

Існує період часу, коли аплікація, що працює у вигляді мікросервісу розігрівається. У цей момент, час відповіді набагато довший, ніж може бути загалом. Це так звана проблема “холодного старту”.

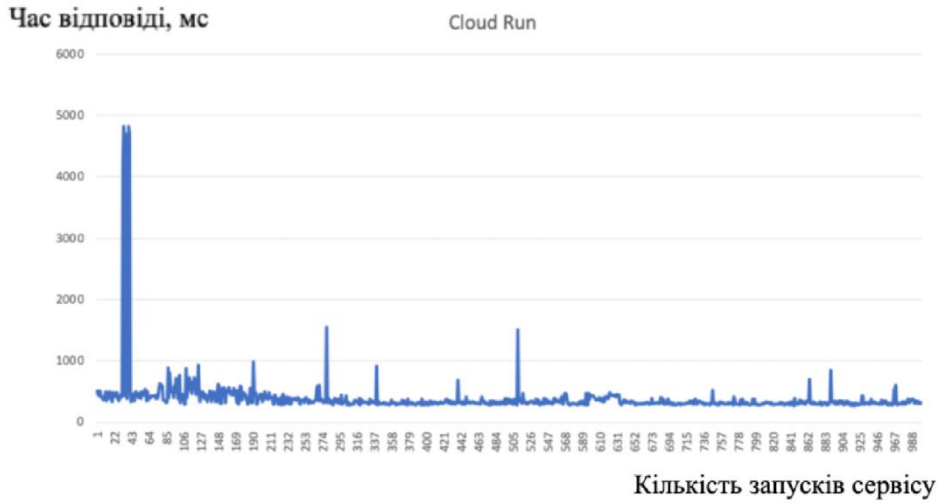


Рис.4.15. Залежність часу відповіді від кількості запусків Cloud Run

На рисунку 4.15 представлено графічну залежність, яка показує як змінюється час затримки. Як можна побачити, на початку, після створення сервісу Cloud Run, спостерігається проблема “холодного старту”, що характеризується великими значеннями часу відповіді сервісу. Однак вже через кілька запитів, час затримки стає в межах 300 - 400 мс.

Варто також зазначити, що використовуєчи підходи інфраструктури, як коду та метод розгортання аплікації на виробництві за допомогою сервісу Cloud Run, який пропонує один із найбільших постачальників хмарних послуг Google, розробник має можливість інтегрувати моніторинг аплікації та проінструментувати код самого сервісу APM (Application monitoring) в самій аплікації. Це в свою чергу дає можливість ефективно використовувати сучасні системи моніторингу аплікацій та збирати всі дозволені метрики для подальшого аналізу стану роботи сервісу та виявлення будь яких збоїв в роботі. Приклади таких метрик представлено на рис 4.16-4.17.

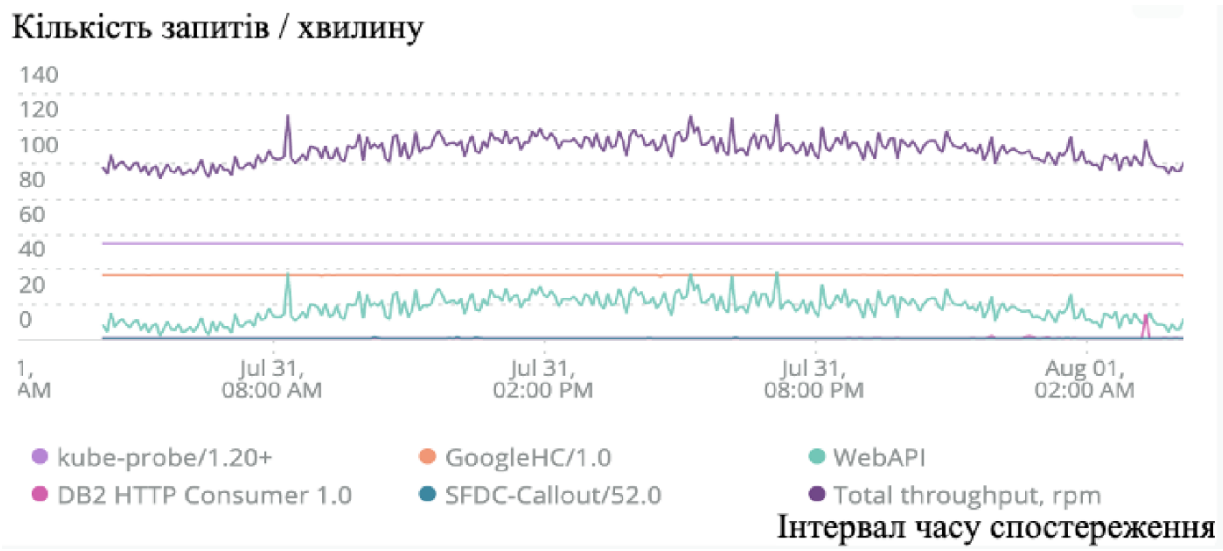


Рис.4.16. Розподіл числа запитів між сервісами, які звертаються до аплікації

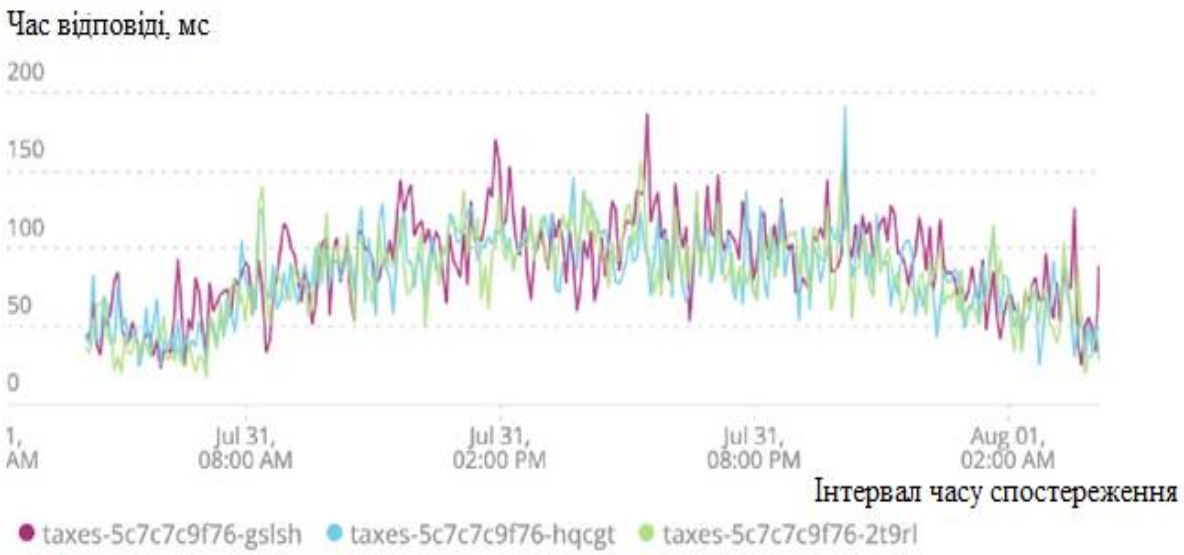


Рис.4.17. Час відповіді в мс, від різних інстансів запущеної аплікації

Із представлених результатів, можна побачити як розподіляється загальна кількість запитів до аплікації. На рис 4.16 бачимо, що є різні клієнти, які роблять запити до аплікації, а також є частина сервісних запитів для оцінки стану роботи аплікації. У випадку аномалій, відразу буде очевидно який клієнт, чи який сервісний запит спричинив стрімкий ріст навантаження та легко його виявити і в подальшому виявити причину цього. Що стосується рисунку 4.17, то такі метрики показують загальне число інстансів, які використовує певний мікросервіс (аплікація), розподіл навантаження між всіма інстансами, що виконують обробку даних, та час відповіді по кожному із них. Використовуючи ці метрики досить легко можна ідентифікувати проблему із певним інстансом обробки чи сервісом в цілому, неефективність роботи балансувальника навантаження, а також будь які проблеми всередині самої аплікації чи її взаємодії із зовнішніми сервісами.

Виходячи з результатів представлених вище, можна сказати, що основною проблемою є “холодний старт”, який займає занадто багато часу, що може бути неприйнятним для деяких програм та сервісів. Однак з іншої

сторони завдяки розгортанню мікросервісу у Cloud Run ми можемо досягти хорошої продуктивності програми та зменшити витрати, оскільки ресурси GCP у більшості випадків будуть використовуватись лише за запитом.

Мікросервіси, що працюють як Cloud Run, постійно передають ключові показники та журнали в Cloud Log і Cloud APM. Ці показники та журнали використовуватимуться для AI / Ops і дозволять нам робити прогнози в реальному часі та запобігати можливих помилок, використовуючи здатність штучного інтелекту виконувати безперервний аналіз журналів, виявлення аномалій, прогнозоване обслуговування, діагностику першопричин та інші важливі функції.

4.5. Висновок до розділу 4

У цьому розділі представлено результати роботи та використання технології розподілених граничних обчислень, які використовуються для швидкої обробки даних в найближчій до користувача локації. Розроблений метод адаптивного створення мікросервісу в граничній локації CDN мережі, який призначений для забезпечення задовільних параметрів якості обслуговування в мережах передачі даних. Даний метод ґрунтується на використанні алгоритму, який здійснює збір метрик і аналіз параметрів якості обслуговування. Алгоритм працює на рівні балансувальника навантаження, який приймає рішення про початок обробки даних на граничному сервері на основі значення параметрів якості обслуговування. Використання системи, що працює на базі даного алгоритму дає змогу завжди мати актуальну інформацію про стан кореневого сервера, актуальні значення параметрів якості обслуговування, та гранично допустимі межі, при яких ефективно застосовувати розподілені обчислення в точках присутності CDN мережі. Також на основі зібраних статистичних даних, є можливість заздалегідь прогнозувати періоди росту навантаження та

використовувати запропонований метод розподілених обчислень для збереження параметрів якості обслуговування в допустимих межах.

Проведено імітаційне моделювання роботи методу адаптивного створення мікросервісу в точках присутності CDN мережі, враховуючи граничні значення часу затримки при певній кількості запитів від клієнтів. Розроблено прототип CDN мережі із використанням балансувальника навантаження у граничній локації, на якому проводилось дослідження. У результаті моделювання, було створено графічну інтерпретацію отриманих результатів, що відображає, як змінюється час затримки для користувача при зміні навантаження на ресурси мережі доставки контенту. З аналізу результатів моделювання, можна зробити висновок, що запропонований метод є досить ефективним, оскільки при подвоєнні кількості запитів, він забезпечує необхідні значення якості обслуговування QoS. Із результатів експериментальних досліджень можна також зазначити, що використання методу адаптивного створення мікросервісу в граничних локаціях CDN мережі, дозволяє значно зменшити навантаження на кореневий сервер, скоротити час затримки для кінцевого користувача при отриманні контенту, а також знизити ймовірність втрати даних під час їх передачі, що забезпечує покращення якості обслуговування.

ВИСНОВКИ

В дисертаційній роботі розв’язано науково-практичне завдання підвищення якості обслуговування користувачів в умовах обмеженої обчислювальної інфраструктури, шляхом розроблення методів і алгоритмів обробки запитів, що потребують значних обчислювальних ресурсів, кешування даних, реалізації балансування навантаження та адаптивного розгортання мікросервісів у точках присутності CDN мереж.

Основні наукові та практичні результати полягають у наступному:

1. Проведено аналіз існуючих підходів та архітектурних рішень, які використовуються під час проєктування та впровадження сучасних сервісів і послуг у мережах доставки контенту. Встановлено, що для ефективного використання ресурсів CDN-мереж, а також забезпечення можливості застосування розподілених граничних обчислень у найближчих до користувача локаціях, доцільно застосовувати мікросервісну архітектуру. Такий підхід дозволяє гнучко управляти компонентами аплікацій і використовувати методи їх швидкого розгортання, що підвищує масштабованість, стійкість до збоїв і можливість адаптації системи до змін у моменти зростання навантаження.

2. Удосконалено метод обробки запитів, що потребують значних обчислювальних ресурсів у точках присутності CDN мережі, який враховує дані аналітики щодо популярності певних типів контенту, таких як веб-сторінки, мультимедійний контент, серед списку найбільш запитуваних, що дало змогу забезпечити ефективне використання кешованих даних, зменшити навантаження на кореневий сервер та час відповіді на запити кінцевого користувача. Застосування даного методу дозволило скоротити час відповіді для кінцевого користувача на 9% та на 10% завантаженість кореневого сервера, що сприяло покращенню загальної продуктивності системи та підвищенню якості взаємодії з користувачем.

3. Запропоновано інтегральний ключ кешування, що являє собою унікальний ідентифікатор, який формується шляхом поєднання кількох параметрів або компонентів, чим дає можливість деталізувати критерії вибору даних, які зберігатимуться як одне ціле та забезпечує ефективне кешування статичних даних в мережах доставки контенту. Інтегральний ключ кешування може бути адаптований для роботи з різними типами мережевого трафіку, а його складові можуть змінюватися відповідно до специфіки застосування. Правильно сформований ключ кешування, який також враховує аналітичні дані щодо популярності контенту в локаціях CDN мережі, забезпечує підвищення коефіцієнта ефективності кешування, зменшення навантаження на кореневі сервери та зниження затримок при доступі до даних, що є важливим аспектом підвищення якості обслуговування у мережі доставки контенту.

4. Удосконалено метод балансування навантаження у мережах доставки контенту, який враховує значення інтегрального ключа кешування сформованого на основі розробленого методу обробки запитів, локацію клієнта, наявність контенту на граничному сервері та стан функціонування доступних серверів, для забезпечення ефективного використання кеш-пам'яті та ресурсів обчислювальної інфраструктури. Застосування такого типу балансування у граничній локації CDN мережі, забезпечує ефективність кешування даних до 90% та зменшення часу відповіді із 1600 мс до 800 мс для кінцевого користувача. Забезпечення таких показників є важливим не тільки для кінцевого користувача, а безумовно і для збереження хорошого рейтингу ресурсу в пошукових системах, які використовуються в глобальних мережах.

5. Запропоновано використання комплексної математичної моделі з трьома об’єднаними системами масового обслуговування для характеристики процесу обробки запитів у хмарних сервісах із динамічним виділенням ресурсів. Встановлено, що система, яка складається із більшої

кількості менш продуктивних вузлів обробки даних, буде працювати краще, ніж система, яка складається із одного вузла більшої продуктивності за умови що особливість трафіку не вимагає складних обчислень та великих затрат процесорного часу.

6. Розроблено метод адаптивного розгортання мікросервісів для обробки динамічних даних у режимі реального часу в точках присутності CDN-мереж, який частково дублює бізнес-логіку сервісу з кореневого сервера, здійснює в режимі реального часу аналіз параметрів, які визначають якість послуги, та адаптивно розподіляє запити на основі оцінювання рівня завантаженості граничних серверів для забезпечення необхідної якості обслуговування. Результати застосування методу демонструють зменшення часу затримки для кінцевого користувача, навіть за умов двократного зростання кількості запитів.

7. Наведено результати практичного використання методу адаптивного розгортання мікросервісу та дослідження його роботи на прикладі хмарного сервісу Google Cloud Run, що забезпечує хорошу продуктивність програм та аплікацій. Із результатів досліджень можна зазначити, що використання методу адаптивного розгортання мікросервісу в граничних локаціях CDN мережі, дозволяє значно зменшити навантаження на кореневий сервер, скоротити час затримки для кінцевого користувача при отриманні контенту, а також знизити ймовірність втрати даних під час їх передачі, що забезпечує покращення якості обслуговування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Monolithic Architecture [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу до ресурсу: https://microservices.io/patterns/monolithic.html
2. Microservices [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу до ресурсу: https://martinfowler.com/articles/microservices.html
3. Chris Richardson, Manning Publications: Microservices Patterns, 428 – 471 (2018). https://microservices.io/book
4. Sam Newman, O'Reilly Media: Building Microservices 2nd edition, 14– 170 (2021).
5. Microservice Premium [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу до ресурсу: https://martinfowler.com/bliki/MicroservicePremium.html
6. M. Kyryk, O. Tymchenko, N. Pleskanka, and M. Pleskanka, "Methods and process of service migration from monolithic architecture to microservices," in *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2022.
7. Dev Ops Culture [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу до ресурсу: https://martinfowler.com/bliki/DevOpsCulture.html
8. Container runtime contract. [Online]. Available: https://cloud.google.com/run/docs/reference/container-contract
9. Observability Patterns [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу до ресурсу: https://microservices.io/patterns/observability/health-check-api.html

10. TR-126 «Triple-play services quality of experience (QoE) requirements», Broadband Forum Architecture and Transport Working Group Tech. Rep., December, 2006.
11. Akira Takahashi. Overview of ITU-T and its standardization of QoE assessment methodologies, IEICE Tech. Rep., vol. 110, no. 118, CQ2010-28, pp. 65-69, July 2010.
12. Fleming S. DevOps And Microservices Handbook: Non-Programmer's Guide to DevOps and Microservices (Continuous Delivery) / Fleming., 2018. – 246 c.
13. Morris K. Infrastructure as Code: Managing Servers in the Cloud. 1st Edition / Kief Morris., 2016. – 362 c.
14. P. Gupta, M. K. Goyal, and N. Gupta, “Reliability aware load balancing algorithm for content delivery network,” in *Advances in Intelligent Systems and Computing*, Cham: Springer International Publishing, 2015, pp. 427–434.
15. K. Nakanishi, F. Suzuki, S. Ohzahata, R. Yamamoto, and T. Kato, “A container-based content delivery method for edge cloud over wide area network,” in *2020 International Conference on Information Networking (ICOIN)*, 2020.
16. Y. Yuan *et al*., "Distributed Learning for Large-Scale Models at Edge With Privacy Protection," in *IEEE Transactions on Computers*, vol. 73, no. 4, pp. 1060-1070, April 2024, doi: 10.1109/TC.2024.3352814.
17. Y. Dong, G. Xu, M. Zhang, and X. Meng, “A high-efficient joint ’cloud-edge’ aware strategy for task deployment and load balancing,” *IEEE Access*, vol. 9, pp. 12791–12802, 2021.
18. T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, “Horizontal pod autoscaling in Kubernetes for elastic container orchestration,” *Sensors (Basel)*, vol. 20, no. 16, p. 4621, 2020.

19. Y. Niu, F. Liu and Z. Li, "Load Balancing Across Microservices," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Honolulu, HI, USA, 2018, pp. 198-206, doi: 10.1109/INFOCOM.2018.8486300.
20. Kumari N., Lugones D., Analysis of dynamic application load balancing in Kubernetes using CDN. NORMA eResearch @NCI Library, 2023.
21. The Art of Service - Infrastructure As A Code Publishing. Infrastructure As A Code A Complete Guide - 2021 Edition / The Art of Service - Infrastructure As A Code Publishing, 2020, 319 c.
22. M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "DevOps: Introducing Infrastructure-as-Code," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017.
23. Young A. Infrastructure as Code: A Comprehensive Guide to Managing Infrastructure as Code Kindle Edition / Austin Young., 2019. – 118 c.
24. S. Chakraborty and D. Sarddar, "An efficient edge server selection in content delivery network using Dijkstra's shortest path routing algorithm with euclidean distance," *Int. J. Comput. Appl.*, vol. 117, no. 4, pp. 24–26, 2015.
25. Morris K. *Infrastructure as Code: Dynamic Systems for the Cloud Age*. 2nd ed. Sebastopol,, , CA: O'Reilly Media, 2020.
26. B. Lubanovic, *Introducing python: Modern computing in simple packages*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2019.
27. O. Lemeshko, O. Yeremenko, M. Yevdokymenko, and A. M. Hailan, "Tensor multiflow routing model to ensure the guaranteed quality of service based on load balancing in network," in *Advances in Computer Science for Engineering and Education III*, Cham: Springer International Publishing, 2021, pp. 120–131.

28. К.Л. Горященко, М.Д. Доротюк, “Огляд систем імітаційного моделювання телекомунікаційних мереж,” *Вісник Хмельницького національного університету*, №5, с. 115–118, 2014.
29. Y.-C. Chen and C.-Y. Liao, “Improving quality of experience in P2P IPTV,” in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016.
30. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Analysis of the technologies and methodologies of data transmission in distributed information systems,” in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, 2018.
31. М.В. Кайдан, В.С. Андрущак, М.В. Піцик, В.З. Пашкевич, “Аналіз енергетичного балансу оптичної транспортної мережі з урахуванням технологічних і архітектурних підходів,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* №818, с. 120-129, 2015.
32. M. Kaidan, V. Andrushchak, and M. Pitsyk, “Calculation model of energy efficiency in optical transport networks,” in *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, 2015.
33. T. Wauters, J. Coppens, B. Dhoedt, and P. Demeester, “Load balancing through efficient distributed content placement,” in *Next Generation Internet Networks, 2005*, 2005.
34. Campbell B. The Definitive Guide to AWS Infrastructure Automation: Craft Infrastructure-as-Code Solutions 1st ed. Edition / Bradley Campbell., 2019, с. 367.
35. Joyner . «*DevOps For Beginners: DevOps Software Development Method Guide For Software Developers and IT Professionals*».

36. Y. Bai, B. Jia, J. Zhang, and Q. Pu, “An Efficient Load Balancing Technology in CDN,” in *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, 2009.
37. G. Pallis, K. Stamos, A. Vakali, D. Katsaros, and A. Sidiropoulos, “Replication based on objects load under a content distribution network,” in *22nd International Conference on Data Engineering Workshops (ICDEW’06)*, 2006.
38. *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*.
39. F. T. Lin and T. S. Shih, *Cloud computing: The emerging computing technology*.
40. M. Kyryk, N. Pleskanka, M. Pleskanka, “Adaptive Edge Compute module in CDN networks,” *Advanced Information and Communication Technologies: Proceedings of the 5th IEEE International Conference*, Lviv, Ukraine, 2023, pp. 41–43.
41. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Content delivery network usage monitoring,” in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017.
42. Y. Brikman, *Terraform: Up & Running: Writing Infrastructure as Code*.
43. S. Sivasubramanian et al., “Replication for web hosting systems,” 2002.
44. Brikman Y. Why we use Terraform and not Chef, Puppet, Ansible, SaltStack, or CloudFormation/Yevgeniy Brikman. – 2016.
45. M. Klymash, M. Kyryk, N. Pleskanka, and V. Yanyshyn, “Data buffering multilevel model at a multiservice traffic service node,” *Thesmart Comput. Rev.*, vol. 4, no. 4, 2014.
46. О. Тимченко, М. Кирик, Н. Плесканка, “Аналіз проходження мультимедійного трафіку в мережі доставки контенту,” Комп`ютерні

технології друкарства. Зб. наук. пр. – Львів: Українська Академія Друкарства, 2011. – Вип. 25. – С.109-115.

47. М. І. Кирик, Н.М. Плесканка, “Алгоритм обслуговування черг у безпровідних мережах,” *Інститут проблем моделювання в енергетиці ім. Г.Є.Пухова. Збірник наукових праць*. Вип. 70. – с. 159-162, 2014.
48. Introduction to Infrastructure as Code with Terraform. [Online]. Available: https://learn.hashicorp.com/tutorials/terraform/infrastructure-as-code?in=terraform/aws-get-started
49. Кирик М.І., Плесканка Н.М. Визначення залежності якості відеосигналу від параметрів кодек H264. *Матеріали конференції, науково-методична конференція* «*Сучасні проблеми телекомунікацій і підготовка фахівців в галузі телекомунікацій* 2011», 27-30 жовтня 2011 р., Львів, с.9-11.
50. Kyryk M., Kostiv O. Quality of Experience for IPTV // Modern problems of radio engineering, telecommunications and computer science: Proc. Int. Conf TCSET'2010. – Lviv: Publishing house of Lviv Polytechnic, 2010. – P. 214.
51. ITU-T Recommendation ITU-T Y.1901 (2009),Requirements for the support of IPTV services
52. RFC 1889 – An outline of RTP and RTCP, provisioning a method of transportation independent of transport and network layers
53. S. Winkler and P. Mohandas, “The evolution of video quality measurement: From PSNR to hybrid metrics,” *IEEE Trans. On Broadcast.*, vol. 54, no. 3, pp. 660–668, 2008.
54. В.І. Романчук, Ю.В. Антонюк, А.В. Поліщук, “Дослідження технології ip/mpls за різних мережевих топологій,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації*, № 645. с. 95-102, 2009.

55. M. Kyryk, N. Pleskanka, and M. Pitsyk, “QoS mechanism in content delivery network,” in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, 2016
56. "Testing MPEG based IP video QoE/QoS" // Shenick Network Systems. [Електронний ресурс]. – Режим доступу: https: //www.scribd.com/ document/ 57128274/Testing-Mpeg-Iptv-Vod-Qoe/
57. RFC 2250 – A Packetization method for transportation of MPEG-TS utilizing RTP protocols.
58. ISO/IEC 11172: 'Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s'.
59. М.І. Кирик, В.Б. Янишин, М.В. Плесканка, “Оцінка ефективності методів спектральної мобільності у когнітивних радіомережах,”. *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* №849, с. 194 – 202, 2016.
60. Т.А. Максимюк,О.М. Яремко, М.В. Піцик, “Моделі конвергенції гетерогенних мереж мобільного зв’язку 5-го покоління на основі технології D2D,”. *Телекомунікаційні та інформаційні технології, Київ, ДУТ,* № 3, с. 91-102, 2016.
61. М.І. Кирик, Н.М.Плесканка, “Дослідження впливу параметрів кодеку x264 на якість відеосигналу” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* № 885 № 705, с. 161–166, 2011.
62. M. Kyryk, O. Tymchenko, N. Pleskanka, and M. Pleskanka, “Methods and process of service migration from monolithic architecture to microservices,” in *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2022.

63. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Analysis of the technologies and methodologies of data transmission in distributed information systems,” in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, 2018.
64. Biliris, A., Cranor, C., Douglis, F., Rabinovich, M., Sibal, S., Spatscheck, O., and Sturm, W. CDN brokering. Computer Communications, 25(4), pp. 393–402, 2002.
65. Buyya, R., Pathan, A.-M., Broberg, J., & Tari, Z. (2006). A case for peering of content delivery networks. *IEEE Distributed Systems Online*, *7*(10), 3–3.
66. G. Pallis, K. Stamos, A. Vakali, D. Katsaros, and A. Sidiropoulos, “Replication based on objects load under a content distribution network,” in *22nd International Conference on Data Engineering Workshops (ICDEW’06)*, 2006.
67. David Farley, Jez Humble. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional 2011. – 464 с.
68. “The Zettabyte Era: Trends and Analysis.” Cisco, 7 June 2017. [Електронний ресурс]. – Режим доступу: www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/ vni-hyperconnectivity-wp.html
69. М.М. Климаш, М.І. Кирик, Н.М. Плесканка, І.О. Кагало, “Багаторівнева модель буферизації даних у вузлах обслуговування мультисервісного трафіку,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації*, №796, с. 182-194, 2014.
70. M. S. Taqqu, W. Willinger, and R. Sherman, “Proof of a fundamental result in self-similar traffic modeling,” *Comput. Commun. Rev.*, vol. 27, no. 2, pp. 5–23, 1997.

71. Thomas Karagiannis, Mart Molle, Michalis Faloutsos, “Long-Range Dependence Ten Years of Interent Traffic Modeling,” *IEEE Interenet Computing,*September-October 2004
72. Торошанко Я. І.,Якимчук Н. М., “Аналіз і моделювання різнорідного самоподібного трафіку комп’ютерних мереж,” *ISSN2412-4338Телекомунікаційні та інформаційні технології*.2017. №4(57) С. 42-49
73. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, “On the self-similar nature of Ethernet traffic (extended version),” *IEEE ACM Trans. Netw.*, vol. 2, no. 1, pp. 1–15, 1994.
74. *Data Networks as Cascades: Investigating the multifractal nature of Internet WAN traffic.*
75. J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, “On the nonstationarity of Internet traffic,” in *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2001.
76. Воробйов О. В. “Моделювання самоподібного трафіку синтезом ансамблю стохастичних квазіперіодичних джерел та ON/OFF-моделі,” *Системи озброєння і військова техніка*. 2006. Вип. 3. с. 97-104.
77. W. E. Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, “Про самоподібний характер трафіку Ethernet”, *IEEE / ACM Trans. Networking* Vol. 2, арк.1-15. 1994 рік.
78. Ryu B.K. Fractal Network Traffic: From Understanding to Implications. Ph.D. thesis. – Columbia University,1996. – 143 p.
79. W. E. Leland, M. S. Taqqu, and W. Willinger, “On the self-similarnature of ethernet traffic,” IEEE/ACM Transactions of Networking, vol. 2, no. 1, pp. 1–15, 1994.

80. Shaabany, A., & Jamshidi, F. (2012). Network traffic deviation detection based on fractal dimension. *Journal of Computing and Information Technology*, *20*(1). https://doi.org/10.2498/cit.1002007.

81. M. Kyryk, N. Pleskanka and M. Pleskanka, "The Analysis of the Optimal Data Distribution Method at the Content Delivery Network*," 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, Polyana, Ukraine, 2019, pp. 1-4.

82. Кирик М. І., "Дослідження механізмів управління контентом у мережах CDN," *Вісник Національного університету "Львівська політехніка". Радіоелектроніка та телекомунікації*, № 849, с. 229–236, 2016.

83. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, "Аналіз роботи методу оптимізованого кешування даних в мережі доставки," *Проблеми телекомунікацій.* № 1 (22), с. 43–55, 2018.

84. Ложковський А.Г., "Нові методи теорії телетрафіка" Одеса: ОНАЗ ім. О.С.Попова, 2018. – 80 с.

85. Воробйов О.С., "Методи дослідження самоподібного трафіку у телекомунікаційних мережах," *XII Міжнародна науково-практична конференція магістрантів та аспірантів : матеріали конф.*, Харків. 2018. – С. 30.

86. Куліш Є.Б., Ложковський А.Г., Гордієнко В.Ю, "Оптимальна маршрутизація ефективного використання пропускної здатності мультисервісної мережі зв'язку," *Наукові праці ОНАЗ ім. О.С. Попова*, 2013, № 1 – С. 106–109.

87. M. Kyryk, M. Pleskanka and N. Pleskanka, "The efficiency and productivity of the CDNs," *2017 2nd International Conference on Advanced Information and Communication Technologies (AICT)*, Lviv, Ukraine, 2017, pp. 270-273.

88. The wavelet-based synthesis for the fractional Brownian motion proposed by F. Sellan and Y. Meyer: Remarks and fast implementation / P. Abry, F. Sellan // Applied and Computational Harmonic Analysis – Vol. 3(4). – 1996. – PP. 377–383.
89. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, "Дослідження ефективності використання мережі CDN," *Вісник Національного університету "Львівська політехніка". Радіоелектроніка та телекомунікації,* № 885, с. 31–40, 2017.
90. Crovella, M. E., & Bestavros, A. (1997). Self-similarity in World Wide Web traffic: evidence and possible causes. *ACM Transactions on Networking [a Joint Publication of the IEEE Communications Society, the IEEE Computer Society, and the ACM with Its Special Interest Group on Data Communication], 5*(6), 835–846. https://doi.org/10.1109/90.650143.
91. Voorsluys, W., Broberg, J., & Buyya, R. (2011). Introduction to cloud computing. In *Cloud Computing* (pp. 1–41). John Wiley & Sons, Inc.
92. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. https://doi.org/10.1145/1721654.1721672.
93. Довгий Б.П. Використання математичного пакета MATLAB для розв'язування прикладних задач /Б.П.Довгий, Є.С.Вакал, Ю.Є.Вакал, А.В.Попов. – К.:Фітосоціоцентр,2012. – 77 с.
94. Budkova, L.V., Korniyenko, V.I. (2013) "Complex estimation of characteristics and traffic identification in information telecommunication networks", *Information processing systems*, no. 2(109), pp. 207–211, Ukraine.
95. Gross, D., Shortle, J. F., Thompson, J. M., & Harris, C. M. (2013). *Fundamentals of queueing theory* (4th ed.). Wiley-Interscience.

96. Комплексна оцінка характеристик та ідентифікація трафіку в інформаційних телекомунікаційних мережах / Л.В. Будкова, В.І. Корнієнко // Системи обробки інформації. – 2013. – № 2 (109). – С. 207–211.
97. Buyya, R., Yeo, C. S., & Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. *2008 10th IEEE International Conference on High Performance Computing and Communications*.
98. Khazaei, H., Misic, J., & Misic, V. B. (2012). Performance analysis of cloud computing centers using M/G/m/m+r queuing systems. *IEEE Transactions on Parallel and Distributed Systems: A Publication of the IEEE Computer Society*, 23(5), 936–943. https://doi.org/10.1109/tpds.2011.199
99. Vilaplana, J., Solsona, F., Teixidó, I., Mateo, J., Abella, F., & Rius, J. (2014). A queuing theory model for cloud computing. *The Journal of Supercomputing*, *69*(1), 492–507. https://doi.org/10.1007/s11227-014-1177-y.
100. Guo, L., Yan, T., Zhao, S., & Jiang, C. (2014). Dynamic performance optimization for cloud computing using M/M/m queueing system. *Journal of Applied Mathematics*, *2014*, 1–8. https://doi.org/10.1155/2014/756592.
101. Nan, X., He, Y., & Guan, L. (2011). Optimal resource allocation for multimedia cloud based on queuing model. *2011 IEEE 13th International Workshop on Multimedia Signal Processing*.
102. Nan, X., He, Y., & Guan, L. (2014). Queueing model based resource optimization for multimedia cloud. *Journal of Visual Communication and Image Representation*, *25*(5), 928–942. https://doi.org/10.1016/j.jvcir.2014.02.008.
103. Norros, I. (1994). A storage model with self-similar input. *Queueing Systems*, *16*(3–4), 387–396. https://doi.org/10.1007/bf01158964

104. Q. Li, S. Wang, Y. Liu, H. Long, and J. Jiang, “Traffic self-similarity analysis and application of industrial internet,” Wirel. Netw., vol. 30, no. 5, pp. 3571–3585, 2024.
105. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, “Дослідження ефективності використання мережі CDN,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* № 885, с. 31–40, 2017.
106. Я.І. Соколовський,В.М. Фірман, “Використання мережі Петрі для моделювання складних систем” – Львів: Вісник ЛДУ БЖД , № 1. 2007 С. 150–154.
107. P. Abry and F. Sellan, “Wavelet-based synthesis for fractional Brownian motion,” *Appl. Comput. Harmonic Anal.*, vol. 3, no. 4, pp. 377–383, 1996.
108. Cardellini, V., Colajanni, M., & Yu, P. S. (2003). Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems: A Publication of the IEEE Computer Society*, *14*(4), 355–368. https://doi.org/10.1109/tpds.2003.1195408.
109. A. Mosenia and N.K. Jha, “A comprehensive study of security of internet-of-things,” *IEEE Trans. Emerging Topics Comput.*, vol. 5, no. 4, pp. 586–602, 2017.
110. Barman B K, Yadav S N and Kumar S 2018 IOT based smart energy meter for efficient energy utilization in smart grid 2nd Int. Conf. on Power, Energy and Environment: Towards Smart Technology (ICEPE)
111. M. Kyryk, N. Pleskanka, M. Pleskanka and P. Nykonchuk, “Load Balancing Method in Edge Computing,” *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, Lviv-Slavske, Ukraine, 2020, pp. 978-981.
112. М.В. Плесканка, “Покращення параметрів якості обслуговування QoS в CDN мережі за рахунок використання модуля Edge Compute,”

Інфокомунікаційні технології та електронна інженерія, Випуск.3, № 2, с. 64-73, 2023Google Cloud Run documentation. [Online]. Available: https://cloud.google.com/run/docs#training-and-tutorials

113. M. Kyryk, N. Pleskanka, M. Pleskanka, and V. Kyryk, “Infrastructure as code and microservices for intent-based cloud networking,” in *Lecture Notes in Electrical Engineering*, Cham: Springer International Publishing, 2022, pp. 51–68.
114. Google Cloud Run documentation. [Online]. Available: https://cloud.google.com/run/docs#training-and-tutorials

ДОДАТОК 1. Програмний код імітаційної моделі EdgeCompute Middleware:

```
using EdgeCompute.Configuration;
using EdgeCompute.State;

namespace EdgeCompute.Middleware
{
   public class ReverseProxyMiddleware
   {
      private static readonly HttpClient _httpClient = new HttpClient();
      private readonly RequestDelegate _nextMiddleware;
      private readonly ApplicationConfiguration _configuration;
      private readonly IState _state;
      private readonly ILogger<ReverseProxyMiddleware> _logger;

      public ReverseProxyMiddleware(RequestDelegate nextMiddleware, ApplicationConfiguration configuration, IState state, ILogger<ReverseProxyMiddleware> logger)
      {
         _nextMiddleware = nextMiddleware;
         _configuration = configuration;
         _state = state;
         _logger = logger;
      }

      public async Task InvokeAsync(HttpContext context)
      {
         var targetUri = BuildTargetUri(context.Request);

         if (targetUri != null)
         {
            var targetRequestMessage = CreateTargetMessage(context, targetUri);

            using (var responseMessage = await _httpClient.SendAsync(targetRequestMessage, HttpCompletionOption.ResponseHeadersRead, context.RequestAborted))
            {
               context.Response.StatusCode = (int)responseMessage.StatusCode;
               CopyFromTargetResponseHeaders(context, responseMessage);
               await responseMessage.Content.CopyToAsync(context.Response.Body);
```

```
            }
            return;
        }
        await _nextMiddleware(context);
    }
    private HttpRequestMessage CreateTargetMessage(HttpContext context, Uri targetUri)
    {
        var requestMessage = new HttpRequestMessage();
        CopyFromOriginalRequestContentAndHeaders(context, requestMessage);

        requestMessage.RequestUri = targetUri;
        requestMessage.Headers.Host = targetUri.Host;
        requestMessage.Method = GetMethod(context.Request.Method);

        return requestMessage;
    }

    private void CopyFromOriginalRequestContentAndHeaders(HttpContext context,
HttpRequestMessage requestMessage)
    {
        var requestMethod = context.Request.Method;
        if (!HttpMethods.IsGet(requestMethod) &&
          !HttpMethods.IsHead(requestMethod) &&
          !HttpMethods.IsDelete(requestMethod) &&
          !HttpMethods.IsTrace(requestMethod))
        {
            var streamContent = new StreamContent(context.Request.Body);
            requestMessage.Content = streamContent;
        }

        foreach (var header in context.Request.Headers)
        {
            requestMessage.Content?.Headers.TryAddWithoutValidation(header.Key,
header.Value.ToArray());
        }
    }

    private void CopyFromTargetResponseHeaders(HttpContext context, HttpResponseMessage
responseMessage)
    {
```

```
            foreach (var header in responseMessage.Headers)
            {
                context.Response.Headers[header.Key] = header.Value.ToArray();
            }

            foreach (var header in responseMessage.Content.Headers)
            {
                context.Response.Headers[header.Key] = header.Value.ToArray();
            }
            context.Response.Headers.Remove("transfer-encoding");
        }
        private static HttpMethod GetMethod(string method)
        {
            if (HttpMethods.IsDelete(method)) return HttpMethod.Delete;
            if (HttpMethods.IsGet(method)) return HttpMethod.Get;
            if (HttpMethods.IsHead(method)) return HttpMethod.Head;
            if (HttpMethods.IsOptions(method)) return HttpMethod.Options;
            if (HttpMethods.IsPost(method)) return HttpMethod.Post;
            if (HttpMethods.IsPut(method)) return HttpMethod.Put;
            if (HttpMethods.IsTrace(method)) return HttpMethod.Trace;
            return new HttpMethod(method);
        }

        private Uri BuildTargetUri(HttpRequest request)
        {
            string url = _configuration.MainHost;
            var mainState = _state.GetState(url);
            var secondaryState = _state.GetState(_configuration.SecondaryHost);
            if (_configuration.EnableEdge && ResolveMaxAverage() < mainState.Average &&
IsMaxRequests(mainState) && CalcMaxRedirectRequests(mainState, secondaryState))
            {
                url = _configuration.SecondaryHost;
            }
            request.HttpContext.Items["redirect_url"] = url;
            url += request.Path;
            url += request.QueryString;

            return new Uri(url);
        }
```

```
        private bool IsMaxRequests(IHostState mainState)
        {
            //return _configuration.RequestLimit <= 0 || _configuration.RequestLimit <
mainState.RequestCount;
            return true;
        }

        private bool CalcMaxRedirectRequests(IHostState mainState, IHostState secondaryHost)
        {
            if (_configuration.RedirectPercent > 0)
            {
                var percent = _configuration.RedirectPercent * mainState.OverloadPeriods;
                if (percent > _configuration.MaxRedirectPercent)
                    percent = _configuration.MaxRedirectPercent;
                if (_state.RequestCount > 0 & percent > 0)
                {
                    var redirectedPercent = secondaryHost.RequestCount == 0 ? 0 :
(double)secondaryHost.RequestCount / (double)_state.RequestCount * 100;
                   // _logger.LogWarning($"{redirectedPercent}");
                    return redirectedPercent < percent;
                }

                return false;
                //var redirectMaxRequests = _state.RequestCount * ((double)percent / 100);
                //var mainRequests = _state.RequestCount * ((double)percent / 100);
                //var mainRequestDif = mainState.RequestCount - _configuration.RequestLimit;
                //return Math.Min(requestPercent, mainRequestDif);
            }
            else
            {
                return ((((double)_configuration.LimitAverage / (double)mainState.Average + 0.3) *
mainState.RequestCount) + mainState.RequestCount) < secondaryHost.OverloadPeriods;
            }
        }

        private long ResolveMaxAverage()
        {
            if (_configuration.RedirectPercent > 0)
                return (long)(_configuration.LimitAverage * (1 + ((double)_configuration.RedirectPercent /
100)));
```

```
            return _configuration.LimitAverage;
        }
    }
}
```

EdgeCompute Properties:

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:23583",
      "sslPort": 0
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "weatherforecast",
      "applicationUrl": "http://localhost:5062",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "weatherforecast",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
}
```

EdgeCompute Configurations:

```
namespace EdgeCompute.Configuration;
public class ApplicationConfiguration
{
   private readonly IConfiguration _configuration;

   public ApplicationConfiguration(IConfiguration configuration)
   {
      _configuration = configuration;
   }
   public long LimitAverage => _configuration.GetValue<long>("average_limit");

   public string MainHost => _configuration.GetValue<string>("main_host");
   public string SecondaryHost => _configuration.GetValue<string>("secondary_host");
   public int RequestLimit => _configuration.GetValue<int>("request_limit");
   public int RedirectPercent => _configuration.GetValue<int>("redirect_percent");
   public int MaxRedirectPercent => _configuration.GetValue<int>("max_redirect_percent");
   public int StatePeriod => _configuration.GetValue<int>("period");
   public int IncreaseRedirectPeriod => _configuration.GetValue<int>("increase_redirect_period");
   public bool EnableEdge=>_configuration.GetValue<bool>("enable_edge");
}
```

EdgeCompute Config Values:

```
average_limit: 40
request_limit: 14150
increase_redirect_period: 30
redirect_percent: 50
max_redirect_percent: 50
period: 10
main_host: 'http://'
secondary_host: 'http://'
enable_edge: false
Logging:
     LogLevel:
       Default: Warning
```

EdgeCompute Program:

```
using EdgeCompute.Configuration;
using EdgeCompute.Middleware;
```

```
using EdgeCompute.State;
using Serilog;
var builder = WebApplication.CreateBuilder(args);
if (builder.Environment.IsDevelopment())
{
   builder.Configuration.AddYamlFile("config.yaml");
}
else
   builder.Configuration.AddYamlFile("config/config.yaml", optional: true, reloadOnChange: true);
builder.Services.AddSingleton<IState, State>();
builder.Services.AddSingleton<ApplicationConfiguration>();

var app = builder.Build();
app.UseMiddleware<StateMiddleware>();
app.UseMiddleware<ReverseProxyMiddleware>();

app.Run();
```

EdgeCompute Dockerfile:

```
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /app

COPY src/. ./src/
COPY *.sln .

ENV DOTNET_CLI_HOME=/tmp/

RUN dotnet restore --no-cache
RUN dotnet build --no-restore -c Release

FROM build as publish
RUN dotnet publish /app/src/EdgeCompute/EdgeCompute.csproj -c Release -r linux-x64 --self-contained
false -p:PublishReadyToRun=true -o /out
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
COPY --from=publish /out/. /app/
RUN chmod +x /app/

ENTRYPOINT ["dotnet", "EdgeCompute.dll"]
```

ДОДАТОК 2. Акти впровадження результатів дисертаційної роботи

«ЗАТВЕРДЖУЮ»
Директор
ТзОВ «Телекомунікаційна компанія»

Пентак І.М.
“ 27 ” 06 2024 р.

АКТ

Про використання результатів кандидатської дисертаційної роботи

Плесканки Мар’яни Вікторівни

“Підвищення якості обслуговування у мережі доставки контенту”

Даний АКТ складений про те, що у ТзОВ “Телекомунікаційна компанія” для покращення якості доставки контенту у телекомунікаційних мережах використані результати дисертаційної роботи Плесканки М.В. “Підвищення якості обслуговування у мережі доставки контенту” представленої на здобуття наукового ступеня кандидата технічних наук, а саме:

метод обробки “складних” запитів, що забезпечує можливість розподіленої обробки даних на рівні точки присутності CDN мережі, а також враховує дані аналітики щодо популярності певних ресурсів, тим самим зменшує навантаження на Origin сервер, забезпечує ефективне використання кешованих даних, а також зменшує час відповіді для кінцевого користувача до 9%.

методи балансування трафіку та обробки запитів у точках присутності CDN мережі, які забезпечують зменшення часу відповіді для кінцевого користувача, та зростання ефективності кешування даних (Cache Affinity) до рівня 95% та зменшення навантаження на Origin сервер.

Результати експериментальних досліджень, на виробничих потужностях ТзОВ “Телекомунікаційна компанія”, відповідають результатам досліджень, що представлені у дисертаційній роботі, похибка не перевищує 3% .

Головний інженер
ТзОВ “Телекомунікаційна компанія” Рубаха І.М.

«ЗАТВЕРДЖУЮ»
Директор
ТзОВ ВТФ «Контех»

Смольницький Є.С.
"24" 06 2024 р.

АКТ

Про використання результатів кандидатської дисертаційної роботи

Плесканки Мар'яни Вікторівни на тему:

"Підвищення якості обслуговування у мережі доставки контенту"

Даний АКТ складений про те, що у ТзОВ ВТФ «Контех» для покращення якості надання мультисервісних послуг кінцевим користувачам використані результати дисертаційної роботи Плесканки М.В. "Підвищення якості обслуговування у мережі доставки контенту" представленої на здобуття наукового ступеня кандидата технічних наук, а саме:

метод адаптивного створення мікросервісу в точці присутності CDN мережі, що надає можливість динамічно розпочати обробку запитів користувачів та формування відповіді граничними (Edge) серверами, впровадження якого дало змогу:

- зберегти значення часу відповіді на запити користувачів на рівні 25мс у тих випадках, якщо тривалість обробки запитів клієнтів була більшою за порогові значення і тим самим забезпечити задовільну якість обслуговування.
- знизити час відповіді на запити користувачів до 7% у моменти пікового навантаження у порівнянні із часом, який отримує клієнт в цій же локації від Origin сервера і тим самим покращити якість надання послуг у мультисервісних телекомунікаційних мережах;

Внаслідок перевірки роботи запропонованого методу на мережевому обладнанні у ТзОВ ВТФ «Контех», встановлено, що результати знаходяться в межах п'ятивідсоткового середньоквадратичного відхилення від поданих у дисертаційній роботі.

Керівник відділу
обслуговування мереж Смольницький О.Є.

"ЗАТВЕРДЖУЮ"
Директор
ТзОВ "МаксіТех"
Заблоцький С.О.
27.06.2024 р.

АКТ

Про використання результатів кандидатської дисертаційної роботи

Плесканки Мар'яни Вікторівни

"Підвищення якості обслуговування у мережі доставки контенту"

Даний АКТ складений про те, що у ТзОВ "МаксіТех" для забезпечення ефективного використання та врповадження мікросервісної архітектури використані результати дисертаційної роботи Плесканки М,В. "Підвищення якості обслуговування у мережі доставки контенту" представленої на здобуття наукового ступеня кандидата технічних наук, а саме:

- схему міграції архітектурних рішень до моделі мікросервісів та способи швидкого їх розгортання, для забезпечення можливості ефективного кешування та адаптивної обробки даних в точках присутності CDN мережі.
- алгоритм роботи адаптивного Edge Compute, який забезпечує обробку даних в найближчій локації клієнта при умові що параметри якості обслуговування для нього стають незадовільними і тим самим покращує якість надання послуг у мультисервісних телекомунікаційних мережах за рахунок зменшення часу відповіді на 7%.

Результати експериментальних досліджень, на виробничих потужностях ТзОВ "МаксіТех", відповідають результатам досліджень, що представлені у дисертаційній роботі.

Інженер
ТзОВ "МаксіТех" Матіїшин Л.З.

"ЗАТВЕРДЖУЮ"
Проректор з науково-педагогічної роботи
НУ "Львівська політехніка"

_____ доц. Давидчак О.Р.
"27" 06 2024 р.

АКТ

про використання результатів кандидатської дисертаційної роботи
Плесканки Мар'яни Вікторівни на тему
" Підвищення якості обслуговування у мережі доставки контенту "
у навчальному процесі кафедри телекомунікацій

Даний акт складений комісією у складі:

- д.т.н., проф. Климаш М.М., завідувача кафедри телекомунікацій;
- д.т.н., доц. Озірковський Л.Д., директора Інституту телекомунікацій, радіоелектроніки та електронної техніки;
- к.т.н. доц. Фаст В.М., заступника директора Інституту телекомунікацій, радіоелектроніки та електронної техніки;

про те, що в навчальному процесі кафедри телекомунікацій використано результати кандидатської дисертаційної роботи Плесканки М.В. "Підвищення якості обслуговування у мережі доставки контенту", а саме програмна імітаційна модель роботи Load Balancer, яка дає можливість визначати порогові значення часу затримки та адаптивно перенаправляти обробку запитів користувачів на граничні сервери, а також технології розгортання мікросервісів в найближчій до користувача локації, в навчальному процесі, в лекційному курсі та лабораторних роботах дисциплін «Інфокомунікаційні системи та мережі», «Методи побудови та моделі інформаційно-телекомунікаційної інфраструктури на основі SDN-технологій для систем електронного урядування» і «Методи побудови гетерогенних інформаційно-комунікаційних систем для розгортання програмно-конфігурованих мереж 5G подвійного використання».

Члени комісії: _____ Михайло КЛИМАШ
_____ Володимир Фаст
_____ Леонід ОЗІРКОВСЬКИЙ

ДОДАТОК 3. Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. M. Kyryk, N. Pleskanka, M. Pitsyk, V. Kyryk, “Infrastructure as code and microservices for intent-based cloud networking,” *Lecture Notes in Electrical Engineering: Future intent-based networking. On the QoS robust and energy efficient heterogeneous software defined networks*, vol. 831, pp. 51–68, 2022.

2. М.В. Плесканка, “Покращення параметрів якості обслуговування QoS в CDN мережі за рахунок використання модуля Edge Compute,” *Інфокомунікаційні технології та електронна інженерія,* Випуск.3, № 2, с. 64-73, 2023.

3. Н.М. Плесканка, М.В. Плесканка, Т.С. Слободзян,Б.М. Марко, “Аналіз ефективності використання мікросервісів при розробці web додатків,” *Комп'ютерні системи проектування. Теорія і практика.* Випуск 6, № 2, с. 146-157, 2024.

4. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, “Аналіз роботи методу оптимізованого кешування даних в мережі доставки,” *Проблеми телекомунікацій.* № 1 (22), с. 43–55, 2018.

5. М.І. Кирик, Н.М. Плесканка, М.В. Плесканка, “Дослідження ефективності використання мережі CDN,” *Вісник Національного університету “Львівська політехніка”. Радіоелектроніка та телекомунікації,* № 885, с. 31–40, 2017.

6. М.І. Кирик, В.Б. Янишин, М.В. Плесканка, “Оцінка ефективності методів спектральної мобільності у когнітивних радіомережах,”. *Вісник Національного університету “Львівська*

політехніка". Радіоелектроніка та телекомунікації, №849, с. 194 – 202, 2016.

7. Т.А. Максимюк,О.М. Яремко, М.В. Піцик, "Моделі конвергенції гетерогенних мереж мобільного зв'язку 5-го покоління на основі технології D2D,". *Телекомунікаційні та інформаційні технології, Київ, ДУТ,* № 3, с. 91-102, 2016.

8. М.В. Кайдан, В.С. Андрущак, М.В. Піцик, В.З. Пашкевич, "Аналіз енергетичного балансу оптичної транспортної мережі з урахуванням технологічних і архітектурних підходів," *Вісник Національного університету "Львівська політехніка". Радіоелектроніка та телекомунікації,* №818, с. 120-129, 2015.

<u>Наукові праці, які засвідчують апробацію матеріалів дисертації</u>

9. M. Kyryk, O. Tymchenko, N. Pleskanka, and M. Pleskanka, "Methods and process of service migration from monolithic architecture to microservices," in *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET),* 2022.

10. M. Kyryk, N. Pleskanka, M. Pleskanka, "Adaptive Edge Compute module in CDN networks," *Advanced Information and Communication Technologies: Proceedings of the 5th IEEE International Conference, Lviv, Ukraine,* 2023

11. M. Kyryk, N. Pleskanka, M. Pleskanka, "Dynamic Data Processing on Edge Locations of CDN Network," in *2024 IEEE 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET),* 2024.

12. M. Kyryk, N. Pleskanka, M. Pleskanka, and P. Nykonchuk, "Load balancing method in edge computing," in *2020 IEEE 15th International*

Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 2020.

13. M. Kyryk, N. Pleskanka, and M. Pleskanka, “The analysis of the optimal data distribution method at the content delivery network,” in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2019.

14. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Analysis of the technologies and methodologies of data transmission in distributed information systems,” in *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, 2018.

15. M. Kyryk, M. Pleskanka, and N. Pleskanka, “The efficiency and productivity of the CDNs,” in *2017 2nd International Conference on Advanced Information and Communication Technologies (AICT)*, 2017.

16. M. Kyryk, N. Pleskanka, and M. Pitsyk, “QoS mechanism in content delivery network,” in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, 2016.

17. M. Kyryk, N. Pleskanka, and M. Pleskanka, “Content delivery network usage monitoring,” in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017.

18. M. Kaidan, V. Andrushchak, and M. Pitsyk, “Calculation model of energy efficiency in optical transport networks,” in *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, 2015.

19. Y. Pyrih, M. Kaidan, I. Tchaikovskyi, and M. Pleskanka, “Research of genetic algorithms for increasing the efficiency of data routing,” in *2019 3rd International Conference on Advanced Information and Communications Technologies (AICT)*, 2019