

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Дніпровський національний університет імені Олеся Гончара  
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Дніпровський національний університет імені Олеся Гончара

Кваліфікаційна наукова  
праця на правах рукопису

**БОЖУХА ДАНІЛ ІГОРОВИЧ**

УДК 004.03, 004.04

**ДИСЕРТАЦІЯ**  
**РОЗРОБЛЕННЯ МЕТОДІВ АНАЛІЗУ ТА ПОБУДОВИ СТРУКТУРИ**  
**ХМАРНИХ СИСТЕМ**

12 Інформаційні технології

121 Інженерія програмного забезпечення

Подається на здобуття ступеня доктора філософії. Дисертація містить результати власних досліджень. Використання ідей, результатів та текстів інших авторів мають посилання на відповідне джерело.

\_\_\_\_\_Д.І. Божуха

Науковий керівник:  
Байбуз Олег Григорович  
доктор технічних наук, професор

Дніпро – 2026

## АНОТАЦІЯ

*Божуха Д.І.* Розроблення методів аналізу та побудови структури хмарних систем. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії з галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення – факультет прикладної математики та інформаційних технологій, Дніпровський національний університет імені Олеся Гончара, Дніпро, 2026.

Необхідність розроблення нових методів побудови хмарних систем зумовлена вичерпанням потенціалу традиційних підходів до масштабування ІТ-інфраструктури. В умовах стрімкої цифровізації виникає гостра наукова потреба у створенні теоретико-методологічної бази для проєктування гнучких архітектур, здатних ефективно оперувати великими масивами даних при одночасному забезпеченні високого рівня відмовостійкості та здатності системи до самовідновлення.

Ця дисертаційна робота присвячена розробленню методів побудови хмарних систем в умовах глобальної цифрової трансформації, де особливу увагу приділено методам архітектурного проєктування. Запропоновані методи та практичне значення роботи полягають у розв'язанні суперечності між зростаючими вимогами до продуктивності ІТ-систем та обмеженістю ресурсів. Розроблення таких методів та підходів є актуальною та перспективною задачею, яка має значний потенціал для покращення продуктивності та якості роботи хмарних систем.

Актуальність теми полягає у необхідності створення концептуально нових підходів до організації хмарного середовища, які дозволять нівелювати ризики нестабільності інфраструктури та забезпечити максимальну продуктивність при мінімальних операційних витратах.

Дослідження спрямоване на аналіз методів та підходів побудови хмарної системи при автоматичному управлінні ресурсами та потоками запитів/завдань, що і зумовлює вибір теми дослідження.

**У першому розділі** проведено аналіз спеціалізованої літератури та інформаційних технологій та розглянуто існуючі методи та рішення для виконання поставленої задачі. Наведено архітектурні рішення хмарної системи, поставлено формальну задачу для побудови багаторівневої архітектури хмарної системи та розглянуто характеристики та властивості хмарної архітектури (масштабованість, стійкість, гнучкість та ін.).

**Другий розділ** присвячено ґрунтовному дослідженню архітектури системи хмарної системи на основі запропонованої у розділі 1 масштабованої ієрархії з включенням рівнів Edge/Fog для розгляду її надійності та хмарної покращення взаємозв'язку. Побудовано формалізовану модель хмарної системи з чотирма архітектурними рівнями по визначеним кількостям кінцевих користувачів, граничних серверів, туманних вузлів та віртуальних машин. Розглянуто моделі хмарних систем в термінології мереж масового обслуговування та узагальнені умови розміщення ресурсів, споживання послуг і контролю складності архітектурного рішення.

**У третьому розділі** розглянуто принципи побудови системи на ієрархічній архітектурі та досліджено методологію розвантаження хмарної системи, яка складається з рівнів кінцевих користувачів, граничних серверів, туманних вузлів та хмарного центру. Представлено визначення трьох модифікацій чотирирівневих хмарних систем з достатньо великою кількістю комбінацій зв'язків між вузлами системи та визначеними правилами перенесення запитів/завдань, для однієї з яких проведено ґрунтовне дослідження для обмеженого часу  $[0, T]$  з певними припущеннями.

**У четвертому розділі** запропоновано реалізацію програмних модулів, що дозволяє комплексно оцінити ефективність розгортання хмарних систем, а також визначити шляхи для подальшої оптимізації та поліпшення

архітектурного рішення. Сформовано набори даних різних комбінацій чотирирівневих хмарних систем для апробації розробленого програмного рішення. Проведено аналіз наборів даних відкритих джерел з інформацією про навантаження хмарних ресурсів для подальшого використання часових характеристик надходження запитів/завдань до хмарної системи при імітаційному моделюванні.

Висновки підводять підсумки проведеної роботи, виокремлюють основні наукові та практичні здобутки автора у розробленні методів та підходів побудови архітектурного рішення хмарної системи. Розроблення узагальнених методів та підходів дослідження хмарної системи є актуальною та перспективною задачею. Робота має значний потенціал для подальшого дослідження та розширення у сферах, де важливо відстеження складності побудови та контролю динаміки навантаження, а також у галузі управління хмарними системами.

#### **Наукова новизна одержаних результатів полягає в такому:**

1. **Вперше** проведено комплексний аналіз існуючих архітектурних рішень для побудови структур хмарних систем з позиції теорії систем масового обслуговування та подальшого узагальнення на розподілені системи, що дозволило виявити приховані недоліки наявних підходів, зокрема їхню низьку адаптивність до динамічної зміни інтенсивності вхідного потоку заявок та неефективне використання обчислювальних ресурсів при пікових навантаженнях. На основі отриманих результатів було формалізовано критерії оптимізації структурних параметрів хмарних систем, що забезпечує мінімізацію часу затримки обробки даних та підвищує загальну відмовостійкість системи.

2. **Вперше** розроблено архітектурне рішення побудови хмарної системи з визначеною послідовністю руху запиту/завдання, що відрізняється від існуючих рішень тим, що використовує більшу кількість рівнів (включення парадигм Edge/Fog) представлення зв'язків, яке є масштабованою ієрархією

при використанні розподілених ресурсів для ефективної підтримки та розгортання хмарної системи.

3. **Вперше** запропоновано можливість представлення хмарної системи через чотири рівні з включенням рівнів Edge/Fog та її розгляду, як простого потоку запитів/завдань, що призводить багатофазову систему до стохастичної мережі.

4. **Вперше** запропоновано підхід щодо проектування архітектури хмарної системи з визначеною кількістю рівнів, який поєднує в собі умову розміщення усіх запитів/завдань на ресурсах та швидкодію роботи використаних ресурсів (умови споживання). Зокрема, для чотирирівневої архітектури хмарної системи запропоновано використання перевірки показників ефективності та надійності, як універсальних метрик функціонування системи масового обслуговування.

5. **Запропоновано** програмне рішення щодо об'єднання існуючої умови сталого режиму та вибору критеріїв структури чотирирівневої хмарної системи для подальшої перебудови на етапі її проектування, що незначно збільшує обчислювальну складність роботи запропонованих алгоритмів.

6. **Визначено** три модифікації чотирирівневих хмарних систем з достатньо великою кількістю комбінацій зв'язків між вузлами системи та визначеними правилами перенесення запитів/завдань. Зокрема, наведено ґрунтовне дослідження однієї модифікації чотирирівневої хмарної системи для обмеженого часу  $[0, T]$  з припущенням, що визначено правило балансування  $q^{bal}(t)$  та швидкість  $v_j^{task}(t)$  ( $j = 1, 2, 3, 4$ ) виконання запитів/завдань кожною колекцією вузлів рівнів.

7. **Запропоновано** визначення двох типів вузлів для побудови структури хмарної системи та розроблено програмне рішення, що надало можливість аналізу їх відокремленого функціонування та виявлення проблемних структур при імітації хмарного сервісу. Зокрема, дослідити "вузькі місця" цих блоків за параметрами надійності та отримати гнучкий інструмент використання

розроблених програмних модулів для дослідження поведінки різних структур складної хмарної системи.

8. **Вперше** сформовано набори різних комбінацій чотирирівневих хмарних систем для проведення імітаційних експериментів роботи хмарної системи. Зокрема, сформовано набори з 810000 та 244140625 комбінацій різних структур хмарної системи в залежності від використаних обчислювальних потужностей, що досить суттєво обмежує використання другого набору для аналізу на безкоштовних сервісах.

9. **Розроблено** програмне рішення для побудованої структури хмарної системи, що використовує різні її комбінації (зокрема, побудовано 810000 моделей) та надає методи та підходи аналізу її організаційної структури та функціональних зв'язків.

10. **Підготовлено** набори даних великого обсягу (отримані з відкритих джерел) з зменшеною кількістю параметрів для дослідження різних структур хмарної системи при імітаційному моделюванні їх роботи (зокрема, з 2 наборів у загальній кількості 49/47 параметрів та 2540047/46686579 проведених експериментів відповідно при використанні підходів та технологій підготовки даних машинного навчання – отримано 2 параметри для моделювання двох типів інтервалів).

11. **Досліджено** вбудовування агента-додатку на рівень хмарної системи, який має математичні локальні та глобальні наслідки при зростанні складності управління на рівні та прогнозованості, що в порівнянні з алгоритмом перерозподілу (всередині вузлів) демонструє різницю між глобальною стратегією та локальною тактикою. Зокрема, запропоновано алгоритм перерозподілу за рахунок виконання умов (п.4 наукової новизни) контролюють стабільність вузлів, а агенти-додатки підтримують баланс між рівнями.

12. **Вперше** проведено експерименти моделювання чотирирівневої хмарної системи при додаванні розроблених програмних рішень для кожного з

запропонованих методів та підходів дисертаційного дослідження, що частково підтвердило гіпотезу про наявність "вузького місця" на першому рівні системи та існування порогу (який приблизно дорівнює 50%) зв'язку між вхідним потоком запитів/завдань та швидкістю обробки. Зокрема, збільшення швидкості обслуговування ( $\mu$ ) спочатку різко покращувало пропускну здатність, але після певного порогу приріст ставав незначним, що вказує на переміщення "вузького місця" до інших компонентів системи.

**Практичне значення** робота має на етапі побудови архітектурних рішень хмарної системи для відстеження складності побудови та контролю динаміки навантаження хмарної системи, що дозволяє адаптувати отримані результати та програмні рішення до існуючих алгоритмів у галузі управління хмарними системами.

Розроблено два типи вузлів для побудови структури хмарної системи для дослідження "вузьких місць". Розроблено програмне рішення та сформовано набори різних комбінацій чотирирівневих хмарних систем для проведення імітаційних експериментів роботи хмарної системи. Проведено моделювання на різних комбінаціях хмарних систем з визначеними у дисертаційному дослідженні методами аналізу їх складності та підтверджено ефективність запропонованих підходів. Підготовлені набори даних використано при імітаційному моделюванні для формування динамічного потоку запитів/завдань за рядом даних, за визначеними розподілами та за інтервалами прибуття. Розроблено набір програмних рішень для моделювання чотирирівневої хмарної системи (без/з наявністю зв'язків між вузлами на рівнях) при додаванні умов розміщення ресурсів і споживання послуг, обчисленням основних показників ефективності роботи та умови для обчислення швидкостей навантаження.

Для розроблення програмних рішень використано мову Python та при вирішенні окремих завдань з достатньо високою обчислювальною складністю використано хмарний сервіс (SaaS) з його онлайн-інструментами.

Наукові результати дослідження є внеском у розвиток архітектурних рішень побудови хмарної системи задачі відстеження складності побудови та контролю динаміки навантаження. Дослідження може бути використане як основа для подальших наукових досліджень у галузі управління хмарними системами.

**Ключові слова:** багаторівнева архітектура та шаблони проектування, хмарна система та сценарії розгортання, моделі сервісів та обслуговування, хмарні/обчислювальні вузли, абстрактна модель та складність системи, масштабованість, продуктивність, багатоканальна модель обслуговування, інформаційні потоки, алгоритми обробки даних, імітаційне моделювання, інформаційні технології, розподілена система, моніторинг, балансування навантаження

## ANNOTATION

*Bozhukha D.I.* Development of methods for analyzing and building the structure of cloud systems. – Qualifying scientific work as a manuscript.

Dissertation for the degree of Doctor of Philosophy in specialty 121 Software Engineering – Oles Honchar Dnipro National University, Dnipro, 2026.

The need to develop new methods for building cloud systems is driven by the exhaustion of the potential of traditional approaches to scaling IT infrastructure. In the context of rapid digitalization, there is an urgent scientific need to create a theoretical and methodological basis for designing flexible architectures that can effectively handle large datasets while ensuring a high level of fault tolerance and the system's ability to self-recover.

This thesis is dedicated to the development of methods for building cloud systems within the context of global digital transformation, with a particular focus on architectural design methods. The proposed methods and the practical significance of the work lie in resolving the contradiction between the growing demands on the performance of IT systems and resource constraints. The development of such methods and approaches is a timely and promising task that has significant potential for enhancing the performance and quality of cloud systems.

The relevance of the topic lies in the need to create conceptually new approaches to organizing the cloud environment, which will mitigate the risks of infrastructure instability and ensure maximum performance with minimal operational costs.

The research is aimed at analyzing methods and approaches for building a cloud system with automatic management of resources and request/task flows, which determines the choice of the research topic.

**The first chapter** analyses the specialist literature and information technologies and examines existing methods and solutions for accomplishing the set task. Architectural solutions for a cloud system are presented, a formal problem is

posed for building a multi-level cloud system architecture, and the characteristics and properties of cloud architecture (scalability, stability, flexibility, etc.) are examined.

**The second chapter** is devoted to a thorough investigation of the cloud system architecture based on the scalable hierarchy proposed in Chapter 1, incorporating Edge/Fog layers to examine its reliability and cloud optimization of interconnections. A formalized model of a cloud system with four architectural levels has been constructed based on specified numbers of end users, edge servers, fog nodes and virtual machines. Cloud system models are examined in the terminology of service-oriented networks, along with generalized conditions for resource allocation, service consumption and control of architectural complexity.

**The third chapter** examines the principles of building a system on a hierarchical architecture and investigates a methodology for offloading a cloud system, which consists of end-user, edge server, fog node and cloud center layers. Definitions are presented for three variants of four-level cloud systems with a sufficiently large number of combinations of connections between system nodes and defined rules for transferring requests/tasks; for one of these, a thorough study has been conducted for a limited time frame with certain assumptions.

**The fourth chapter** proposes the implementation of software modules that allow for a comprehensive assessment of the effectiveness of cloud system deployment, as well as the identification of avenues for further optimization and improvement of the architectural solution. Datasets of various combinations of four-tier cloud systems have been compiled to test the developed software solution. An analysis of open-source datasets containing information on cloud resource load has been carried out for the subsequent use of temporal characteristics of request/task arrivals to the cloud system in simulation modelling.

The conclusions summarize the work carried out and highlight the author's main scientific and practical achievements in developing methods and approaches for constructing the architectural design of a cloud system. The development of

generalized methods and approaches for studying cloud systems is a relevant and promising task. The work has significant potential for further research and expansion in areas where it is important to track the complexity of construction and control load dynamics, as well as in the field of cloud system management.

**The scientific novelty of the results obtained lies in the following:**

1. **For the first time**, a comprehensive analysis of existing architectural solutions for constructing cloud system structures has been carried out from the perspective of queuing theory and its subsequent generalization to distributed systems, which has revealed hidden shortcomings in existing approaches, in particular, their low adaptability to dynamic changes in the intensity of the incoming request flow and the inefficient use of computational resources during peak loads. Based on the results obtained, criteria for optimizing the structural parameters of cloud systems were formalized, ensuring the minimization of data processing latency and increasing the overall fault tolerance of the system.

2. **For the first time**, an architectural solution has been developed for building a cloud system with a defined sequence of request/task flow, which differs from existing solutions in that it utilizes a greater number of levels (incorporating Edge/Fog paradigms) for representing relationships, which constitutes a scalable hierarchy when utilizing distributed resources for the effective support and deployment of a cloud system.

3. **For the first time**, the possibility of representing a cloud system across four levels, including Edge/Fog levels, and treating it as a simple flow of requests/tasks is proposed, which reduces a multi-phase system to a stochastic network.

4. **For the first time**, an approach is proposed for designing the architecture of a cloud system with a defined number of levels, which combines the requirement for all requests/tasks to be placed on resources with the performance of the resources used (consumption conditions). In particular, for a four-level cloud system architecture, the use of performance and reliability metrics is proposed as universal metrics for the operation of a queuing system.

5. A software solution **is proposed** for combining the steady-state condition and the selection of criteria for the structure of a four-level cloud system for subsequent restructuring at the design stage, which slightly increases the computational complexity of the proposed algorithms.

6. Three variants of four-level cloud systems **have been identified**, featuring a sufficiently large number of possible combinations of connections between system nodes and defined rules for the forwarding of requests/tasks. In particular, a thorough study is presented of one modification of a four-level cloud system for a limited time  $[0, T]$ , assuming that a balancing rule and the speed of executing requests/tasks by each collection of nodes at each level have been defined.

7. **We propose** defining two types of nodes for constructing the structure of a cloud system and have developed a software solution that enables the analysis of their isolated operation and the identification of problematic structures when simulating a cloud service. In particular, to investigate the ‘bottlenecks’ of these blocks in terms of reliability and to obtain a flexible tool for using the developed software modules to study the behavior of various structures of a complex cloud system.

8. **For the first time**, sets of different combinations of four-level cloud systems were formed to conduct simulation experiments on the operation of the cloud system. In particular, sets of 810000 and 244140625 combinations of different cloud system structures have been formed depending on the computing power used, which significantly limits the use of the second set for analysis on free services.

9. A software solution has **been developed** for the constructed cloud system structure, utilizing various combinations thereof (in particular, 810000 models have been constructed) and providing methods and approaches for analyzing its organizational structure and functional relationships.

10. Large-scale datasets (obtained from open sources) **have been prepared** with a reduced number of parameters to study various cloud system structures through simulation modelling of their operation (in particular, from 2 datasets comprising a

total of 49/47 parameters and 2540047/46686579 experiments conducted respectively, using machine learning data preparation approaches and technologies – 2 parameters were obtained for modelling two types of intervals).

11. The integration of an agent-application into the cloud system level has **been investigated**, which has mathematical local and global consequences as the complexity of control at the level and predictability increase; compared to the redistribution algorithm (within nodes), this demonstrates the difference between global strategy and local tactics. In particular, a redistribution algorithm is proposed based on the fulfilment of conditions (point 4 of scientific novelty) that control the stability of nodes, whilst application agents maintain the balance between levels.

12. **For the first time**, simulation experiments were conducted on a four-level cloud system by incorporating the developed software solutions for each of the proposed methods and approaches of the dissertation research, which partially confirmed the hypothesis regarding the presence of a bottleneck at the first level of the system and the existence of a threshold (approximately equal to 50%) the relationship between the incoming flow of requests/tasks and the processing speed. In particular, an increase in service rate ( $\mu$ ) initially improved throughput significantly, but beyond a certain threshold, the increase became negligible, indicating a shift of the bottleneck to other system components.

**The practical significance** of the work lies in the stage of developing architectural solutions for cloud systems to track the complexity of construction and monitor the load dynamics of the cloud system, which allows the results and software solutions obtained to be adapted to existing algorithms in the field of cloud system management.

Two types of nodes have been developed to construct the structure of a cloud system for investigating ‘bottlenecks’. A software solution has been developed, and sets of various combinations of four-level cloud systems have been formed to conduct simulation experiments on the operation of the cloud system. Simulations were carried out on various combinations of cloud systems using the methods for

analyzing their complexity defined in the thesis, and the effectiveness of the proposed approaches was confirmed. The prepared datasets were used in simulation modelling to generate a dynamic flow of requests/tasks based on a sequence of data, according to specified distributions and arrival intervals. A set of software solutions has been developed for modelling a four-level cloud system (with or without interconnections between nodes at the levels), incorporating conditions for resource allocation and service consumption, calculating key performance indicators, and conditions for calculating load rates.

The Python language was used to develop the software solutions, and a cloud service (SaaS) with its online tools was utilized to solve specific tasks with sufficiently high computational complexity.

The scientific results of the study contribute to the development of architectural solutions for building a cloud system to track the complexity of construction and control load dynamics. The study can serve as a basis for further scientific research in the field of cloud system management.

**Keywords:** multi-level architecture and design patterns, cloud system and deployment scenarios, service and service models, cloud/computing nodes, abstract model and system complexity, scalability, performance, multi-channel service model, information flows, data processing algorithms, simulation modelling, information technology, distributed system, monitoring, load balancing

## Список опублікованих праць за темою дисертації

*Статті у наукових фахових виданнях України:*

1. **Божуха Д.І., Байбуз О.Г., Мащенко Л.В.** Про підходи дослідження системи хмарних обчислень. *Актуальні проблеми автоматизації та інформаційних технологій.* 2022. Т.26. с. 18-30. DOI: <http://dx.doi.org/10.15421/432203> (**фахове видання категорії Б**) (*особистий внесок Божухи Д.І.: провів аналіз літературних джерел щодо використання методів, технологій, моделей та практичних підходів, які пов'язані з дослідженням хмарної системи; приділив увагу аналізу задачі управління ресурсами IT-інфраструктури; виділив суттєві характеристики ієрархічності IT-інфраструктури; побудував класифікатор для вирішення задачі вибору інструментарію при проектуванні хмарної системи; запропонував для хмарної системи розглянути модель системи масового обслуговування; виділив умови розміщення на ресурсах та умови споживання хмарної системи; Байбуза О.Г.: постановка завдання дослідження, узагальнення отриманих результатів; Мащенко Л.В.: постановка мети дослідження, контроль*).

2. **Божуха Д.І., Байбуз О.Г.** Про імітаційні моделі блоків узагальненої системи хмарних обчислень. *Актуальні проблеми автоматизації та інформаційних технологій.* 2024. Т.28, с. 81-86. DOI: <http://dx.doi.org/10.15421/432407> (**фахове видання категорії Б**) (*особистий внесок Божухи Д.І.: спроектував два допоміжних типи вузла моделі хмарної системи; розглянув сценарії їх вбудовування в хмарну систему для визначення необхідності внутрішньої реконструкції її архітектури; розробив програмне рішення для імітації роботи спроектованих вузлів хмарної системи; провів експерименти роботи блоків з приділенням уваги до їх навантаження; дослідив "вузькі місця" цих блоків за параметрами надійності та отримав гнучкий інструмент використання розроблених програмних модулів для*

дослідження поведінки різних структур складної хмарної системи; ; Байбуза О.Г.: постановка мети дослідження, контроль та узагальнення отриманих результатів).

3. **Божуха Д., Байбуз О.** Модель спільного динамічного розвантаження хмарної архітектури з балансуванням рівнів. *Системні технології*. 2025. Т. 6 № 161, с. 152-157. DOI: <https://doi.org/10.34185/1562-9945-5-161-2025-15> (**фахове видання категорії Б**) (особистий внесок Божухи Д.І.: зосередив увагу на взаємодії рівнів ієрархії хмарної архітектури; розробив архітектурне рішення побудови чотирирівневої хмарної системи з визначеною послідовністю руху запиту/завдання та включенням парадигм Edge/Fog; навів опис математичної моделі чотирирівневої хмарної архітектури для аналізу навантаження; сформував набори різних комбінацій чотирирівневих хмарних систем для проведення імітаційних експериментів; запропонував програмне рішення імітації роботи хмарної системи; Байбуза О.Г.: постановка мети та завдання дослідження, контроль та узагальнення отриманих результатів).

4. **Божуха Д.І., Байбуз О.Г.** Дослідження моделей хмарних систем на прикладі найпростішого потоку. *Актуальні проблеми автоматизації та інформаційних технологій*. 2025. Т.29, с. 84-91. DOI: <http://dx.doi.org/10.15421/432508> (**фахове видання категорії Б**) (особистий внесок Божухи Д.І.: підготував набори даних великого обсягу з зменшеною кількістю параметрів; провів експерименти з моделюванням чотирирівневої хмарної системи; дослідив умову сталого режиму для моделей різних комбінацій чотирирівневих хмарних систем; розробив програмне рішення для аналізу показників продуктивності та візуалізував результати; Байбуза О.Г.: постановка задачі, контроль та узагальнення отриманих результатів).

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

5. **Божуха Д.І., Байбуз О.Г.,** Про формалізацію внутрішніх процесів платформи хмарних обчислень. *Математичне та програмне забезпечення*

*інтелектуальних систем (МПЗІС-2022): тези доповідей XX міжнародної науково-практичної конференції, Дніпро, 23-25 листопада 2022 р., 2022. С. 38. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2022/12/MPZIS-2022-1.pdf> (особистий внесок Божухи Д.І.: набори існуючих інструментарію та технологій для реалізації зв'язків між рівнями; програмна реалізація алгоритму класифікації Байєса; визначення змінних для опису рівнів архітектури; формування навчальної вибірки; Байбуза О.Г.: постановка завдання, узагальнення отриманих результатів).*

6. Vozhukha Daniil. The object of research is the architecture and system of cloud computing. *Міжнародна науково-практична інтернет-конференція: матеріали конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації», Переяслав, 2023, Вип. 95. С. 57-59. URL: <https://0a30397da1.clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1> (особистий внесок Божухи Д.І.: аналіз методів та технологій організаційного управління ІТ-інфраструктурою платформи хмарних послуг).*

7. Божуха Д.І., Байбуз О.Г. Про узагальнену схему складних обчислювальних систем платформи хмарних послуг. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2023): Тези доповідей XXI Міжнародної науково-практичної конференції, Дніпро, 22 – 24 листопада 2023 р., 2023. С. 77. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf> (особистий внесок Божухи Д.І.: логічна архітектура у вигляді схеми системи масового обслуговування; програмний модуль для обчислення показників ефективності; виявлення проблеми існування різновиду архітектурних рішень для формування моделі та її програмної реалізації; Байбуза О.Г.: постановка мети і узагальнення отриманих результатів).*

8. Божуха Д.І. Про методи та технології побудови системи управління. *Ways of Science Development in Modern Crisis Conditions*: тези доповідей V Міжнародної науково-практичної інтернет-конференції, Дніпро, 13-14 червня 2024 р., 2024. С. 37-38. URL: <http://www.wayscience.com/wp-content/uploads/2024/06/Conference-Proceedings-June-13-14-2024.pdf>

*(особистий внесок Божухи Д.І.: проектування системи управління; формування задачі управління чергами; аналіз поведінки об'єктів з часовими властивостями; аналіз структур даних для програмної реалізації).*

9. Божуха Д.І., Байбуз О.Г. Про структуру архітектурного рішення. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2024)*: тези доповідей XXII Міжнародної науково-практичної конференції, Дніпро, 20 – 22 листопада 2024 р., 2024. С. 85. URL: [http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-](http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf)

[1.pdf](http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf) *(особистий внесок Божухи Д.І.: структура архітектурного рішення хмарної системи; узагальнена схема системи хмарних обчислень; програмне рішення структури звітності; вибір структур даних та алгоритмів для програмної реалізації; Байбуза О.Г.: постановка мети і завдання дослідження, контроль результатів).*

10. Божуха Д.І., Байбуз О.Г. Про імітацію роботи вузлів системи. *Міжнародна конференція «Автоматика-2024»*: тези XXVII Міжнародної конференції, Дніпро, 20 – 22 листопада 2024 р., 2024. С. 76-77. URL: [http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

[%](http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf)

*роботи описаних блоків на різних наборах вхідних даних; Байбуза О.Г.: постановка задачі, аналіз результатів).*

11. Божуха Д. Про методи підвищення продуктивності системи. *Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем (MEICS-2024): тези доповідей IX Всеукраїнської науково-практичної конференції, Дніпро, 27-29 листопада 2024 р., 2024. С. 35. URL: <http://meics.dnure.dp.ua/files/MEICS-2024.pdf> (особистий внесок Божухи Д.І.: використання концепції граничних обчислень; метод підвищення продуктивності; розподіл навантаження між вузлами на рівні; оновлено інструмент балансування; проведення експериментів з модельованими даними; Байбуза О.Г.: постановка задачі дослідження, аналіз і узагальнення результатів).*

12. Божуха Д.І., Байбуз О.Г. Про оптимізацію навантаження на інфраструктуру методами машинного навчання та агентними технологіями. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2025): Тези доповідей XXIII Міжнародної науково-практичної конференції, Дніпро, 19 – 21 листопада 2025 р., 2025. С. 87-88. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2025.pdf> (особистий внесок Божухи Д.І.: аналіз підходів використання агентно-орієнтованих систем, технологій аналізу; використання моделей глибокого навчання; дослідження алгоритмів розподілу ресурсів на основі агентної системи; проведення експериментів з модельованими даними; Байбуза О.Г.: постановка задачі дослідження, аналіз і узагальнення отриманих результатів).*

*Наукові доповіді, які засвідчують апробацію матеріалів дисертації:*

13. Байбуз О.Г., Божуха Д.І. Інструментарій та технології багаторівневої архітектури платформи хмарних обчислень. Наукова конференція за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара

за 2022 рік, 2023. URL:  
[https://www.dnu.dp.ua/docs/ndc/2023/Ost\\_var\\_programa.pdf](https://www.dnu.dp.ua/docs/ndc/2023/Ost_var_programa.pdf)

14. Байбуз О.Г., Божуха Д.І. Про характеристики складних обчислювальних систем платформи хмарних послуг. Наукова конференція за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара

за 2023 рік, 2024. URL:  
[https://www.dnu.dp.ua/docs/ndc/2024/Pidsumkova\\_za\\_2023.pdf](https://www.dnu.dp.ua/docs/ndc/2024/Pidsumkova_za_2023.pdf).

15. Байбуз О.Г., Божуха Д.І. Про комбіновані підходи масштабування для управління інфраструктурою. Наукова конференція за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара за 2024 рік, 2025. URL:

[https://www.dnu.dp.ua/docs/ndc/2025/Pidsumkova\\_za\\_2024\\_zavershena.pdf](https://www.dnu.dp.ua/docs/ndc/2025/Pidsumkova_za_2024_zavershena.pdf).

## ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ .....	23
ВСТУП .....	25
РОЗДІЛ 1. ТЕОРЕТИЧНІ, МЕТОДОЛОГІЧНІ ТА ПРАКТИЧНІ ПІДХОДИ РОЗРОБЛЕННЯ ХМАРНИХ СИСТЕМ.....	34
1.1 Про багаторівневу хмарну архітектуру .....	41
1.1.1 Моделі багаторівневої хмарної архітектури .....	41
1.1.2 Сценарії розгортання платформи хмарних обчислень .....	43
1.1.3 Компоненти хмарної архітектури .....	46
1.2 Основні рівні абстрактної моделі хмарної архітектури .....	47
1.3 Парадигми EDGE/FOG .....	49
1.4 Задача масштабованості .....	54
1.5 Про форми використання інформаційних технологій .....	55
1.6 Перспективи використання локальної платформи хмарних обчислень....	56
1.7 Оцінка ефективності .....	56
1.8 Методи перевірки ефективності масштабованих систем .....	57
1.9 Неефективність сучасних систем .....	58
1.10 Способи опису хмарної інфраструктури .....	58
1.11 Стабільність передачі даних .....	59
1.12 Обробка даних з мінімальними затримками .....	60
1.13 Алгоритми балансування .....	60
1.14 Інформаційні технології оптимізації хмарної системи .....	60
Висновки до розділу 1 .....	64
РОЗДІЛ 2. МЕТОДИ АНАЛІЗУ СКЛАДНОСТІ ХМАРНОЇ СИСТЕМИ.....	67
2.1 Інструментарій та технології багаторівневої архітектури платформи хмарних обчислень.....	67
2.2 Оптимізація хмарної архітектури.....	72
2.2.1 Оцінка ресурсів споживання та розміщення.....	72

2.2.2	Оцінка ресурсів хмари.....	76
2.2.3	Задача оптимального розподілу послуг між ресурсами рівнів хмарної архітектури.....	77
2.3	Моделі хмарних систем.....	80
2.3.1	Хмарна система як система масового обслуговування.....	80
2.3.2	Моделі чотирирівневої СХМОбч на прикладах найпростішого потоку.....	85
2.4	Методи дослідження складності системи .....	92
2.5	Показник залежності від часу .....	94
	Висновки до розділу 2 .....	96
	<b>РОЗДІЛ 3. МОДЕЛЬ СПІЛЬНОГО ДИНАМІЧНОГО РОЗВАНТАЖЕННЯ ХМАРНОЇ АРХІТЕКТУРИ.....</b>	<b>98</b>
3.1	Про моделі розвантаження хмарної архітектури.....	98
3.2	Модель чотирирівневої хмарної архітектури з балансуванням рівнів.....	99
3.3	Математичне формулювання стратегії балансування.....	102
3.4	Математичний вираз стратегій розвантаження СХМОбч .....	105
3.5	Дослідження зв'язку між стратегією та функції траєкторією навантаження.....	106
3.6	Використання агентів-додатків в хмарній архітектурі .....	117
	Висновки до розділу 3 .....	121
	<b>РОЗДІЛ 4. ПРОГРАМНІ ЕКСПЕРИМЕНТИ .....</b>	<b>122</b>
4.1	Формування множини моделей хмарних систем.....	122
4.1.1	Параметри генерування моделей хмарних систем .....	122
4.1.2	Програмна реалізація.....	123
4.1.3	Експериментальні дослідження складності побудованих моделей... ..	124
4.1.4	Експериментальні дослідження продуктивності побудованих моделей.....	127
4.2	Імітаційні моделі блоків СХМОбч.....	133
4.2.1	Програмні експерименти роботи першого блоку.....	133
4.2.2	Програмні експерименти роботи другого блоку .....	136

4.2.3 Тактичне планування.....	139
4.3 Підготовка даних для імітації хмарного навантаження.....	142
4.3.1 Набір даних UNSW-NB15 .....	143
4.3.2 Набір даних CICIoT2023 .....	144
4.4 Моделювання роботи хмарної системи .....	145
4.5 Моделювання хмарного навантаження з різними сценаріями розгортання.....	147
4.5.1 Експерименти розгортання хмарних сервісів .....	147
4.5.2 Про інтеграцію агентно-орієнтованих систем DevOps-підходами....	149
Висновки до розділу 4 .....	150
ВИСНОВКИ.....	153
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	157
ДОДАТКИ.....	173
Додаток А Список праць здобувача за темою дисертації.....	173
Додаток Б Акт впровадження результатів роботи в освітній процес .....	177
Додаток В Частина програмного коду .....	180

## ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AC	архітектура системи
CXMOбч	система хмарних обчислень
DC	рівень інфраструктури (Dew Computing)
EC	рівень граничних обчислень (Edge Computing)
FC	рівень туманних обчислень (Fog Computing)
CC	рівень хмарних обчислень (Cloud Computing)
IaaS	<i>інфраструктура як послуга</i> (Infrastructure as a Service)
PaaS	<i>платформа як послуга</i> (Platform as a Service)
SaaS	<i>програмне забезпечення як послуга</i> (Software as a Service)
DaaS	<i>дані як послуга</i> (Data as a service)
XA	хмарна архітектура
VM	віртуальна машина
$\lambda$	інтенсивність надходження запиту/завдання
$\mu$	інтенсивність обслуговування
$\rho$	завантаження (середня кількість запитів/завдань, що приходять за середній час обслуговування одного запиту/завдання, $\rho = \lambda/\mu$ )
Number_Node	кількість вузлів
Additional_node	кількість додаткових вузлів
$N$	кількість користувачів
$ES$	кількість граничних серверів
$FR$	кількість маршрутизаторів навантаження
$U = \{u_1, u_2, \dots, u_N\}$	множина кінцевих користувачів
$S = \{s_1, s_2, \dots, s_{ES}\}$	множина граничних серверів

$R = \{r_1, r_2, \dots, r_{FR}\}$	множина туманних вузлів (маршрутизаторів підключення)
$v_{1,i}^{task}(t)$	швидкість виконання завдань кінцевими пристроями $u_i$ рівня DC позначимо
$v_{2,j}^{task}(t)$	швидкість виконання завдань граничними серверами $s_j$ рівня EC позначимо
$v_{3,k}^{task}(t)$	швидкість виконання завдань туманними вузлами $r_k$ рівня FC позначимо
$x_{ij}^{US}(t)$	частка завдань, які є вивантаженими у момент часу $t \in [0, T]$ з кінцевого пристрою $u_i$ на граничний сервер $s_j$
$x_{jk}^{SR}(t)$	частка завдань, які є вивантаженими у момент часу $t \in [0, T]$ з граничного серверу $s_j$ на туманний вузол $r_k$
$x_{k1}^{RC}(t)$	частка завдань, які є вивантаженими у момент часу $t \in [0, T]$ з туманного вузлу $r_k$ до хмарного центру
$x^{US}(t) = \{x_{ij}^{US}(t)\}_{N \times ES}$	стратегія розвантаження рівня EC ( $t \in [0, T]$ )
$x^{SR}(t) = \{x_{jk}^{SR}(t)\}_{ES \times FR}$	стратегія розвантаження рівня FC ( $t \in [0, T]$ )
$x(t) = \{x_{ik}(t)\}_{N \times FR}$	стратегія розвантаження рівнів DC, EC та FC ( $t \in [0, T]$ )
$q_j^{EC}(t)$	навантаження (довжина черги завдань) граничних серверів $s_j$
$q_k^{FC}(t)$	навантаження (довжина черги завдань) туманних вузлів $r_k$
$q^{bal}(t)$	навантаження хмарного рівня
$Balance(t)$	Функція траєкторії навантаження

## ВСТУП

### **Обґрунтування вибору теми дослідження.**

Основними тенденціями на ринку хмарних технологій є кількісне та якісне зростання пропозицій, а також розбудова розгалужених мережевих структур. Еволюція ІТ-сектору стимулює нарощування потужності хмарних обчислень, що є критичним для контролю мережевих ресурсів і гарантування відмовостійкості систем. Проектування архітектури хмар має базуватися на принципах технічного стандарту, де пріоритетами виступають безпека, продуктивність та економічна ефективність у контексті реалізації стратегічних цілей організації.

Актуальність дослідження методів побудови хмарних систем сьогодні зумовлена глобальною цифровою трансформацією. Хмарні технології перестали бути просто «сховищем» і перетворилися на фундамент для штучного інтелекту, Big Data та критичної інфраструктури.

В сучасних ІТ-системах можна бачити стрімке зростання обсягів даних та запиту на потужності при генеруванні масивів даних, обробка яких у локальних інфраструктурах стає економічно недоцільною. Дослідження методів побудови хмарних систем дозволяє знайти способи ефективного масштабування ресурсів під динамічні потреби бізнесу.

У мовах глобальних кіберзагроз та нестабільності інфраструктури, розробка методів побудови архітектури, що здатна до самовідновлення та безперебійної роботи, є важливою для критичної інфраструктури та фінансового сектору, що формує задачу необхідності забезпечення високої відмовостійкості.

Але перехід до хмарного сервісу не завжди гарантує зменшення витрат, що визначає актуальним напрям пошуку методів проектування, які дозволяють мінімізувати експлуатаційні витрати через автоматизацію, використання безсерверних обчислень та контейнеризації.

Використання сучасними компаніями програмних рішень від кількох провайдерів одночасно призводить до впровадження концепцій Hybrid та Multi-cloud, що формує задачу дослідження методів інтеграції різних хмарних платформ у єдину цілісну систему і є одним з найскладніших викликів сьогодення.

Актуальність теми визначається необхідністю подолання суперечності між зростаючими вимогами до продуктивності ІТ-систем та обмеженістю ресурсів для їх підтримки, що потребує впровадження нових високоефективних методів архітектурного проектування хмарних середовищ.

Враховуючи викладені проблеми, актуальною науково-технічною задачею є розроблення методів побудови хмарної системи, вирішенню якої і присвячена ця дисертаційна робота.

#### **Зв'язок роботи з науковими програмами, планами, темами.**

Запропоновані технологія та модель створені в рамках досліджень наукової школи «Інформаційні технології обробки статистичних даних» на кафедрі інженерії програмного забезпечення та інформаційних технологій Дніпровського національного університету імені Олеся Гончара.

Дисертаційна робота виконана відповідно з поточними та перспективними планами наукової та науково-технічної діяльності Дніпровського національного університету імені Олеся Гончара для подальшого розвитку інженерії програмного забезпечення.

**Мета та завдання дослідження.** Метою дисертаційної роботи є розроблення архітектурного рішення побудови хмарної системи для її ефективної підтримки і розгортання та реалізації відповідної моделі у вигляді програмних модулів.

Результати цього дослідження можуть бути використані для відстеження складності побудови та контролю динаміки навантаження хмарної системи, що дозволяє їх адаптувати до існуючих алгоритмів у галузі управління хмарними системами.

Відповідно до поставленої мети основними задачами дослідження є:

- провести аналіз існуючих підходів побудови хмарних систем, моделей та методів їх представлення та управління, інформаційних технологій для їх опису;
- дослідити характеристики хмарної системи та побудувати узагальнене архітектурне рішення її IT-інфраструктури;
- спроектувати IT-інфраструктуру хмарної системи із застосуванням підходів управління, методів розміщення ресурсів, споживання послуг та прогнозування навантажень хмарної системи;
- розглянути підходи організації роботи систем масового обслуговування та методи побудови складних дискретних систем;
- дослідити існуючі (в термінології систем масового обслуговування) умови сталого режиму і перевірки складності структури хмарної системи;
- розробити програмне рішення щодо роботи вузлів під час навантаження хмарної системи підходами імітаційного моделювання;
- розглянути різні комбінації архітектурних рішень структури хмарної системи;
- розробити програмне рішення для побудованої структури хмарної системи;
- для навантаження хмарних систем провести аналіз даних з відкритих джерел для проведення імітаційних експериментів та підготувати обрані набори даних для імітаційних експериментів роботи хмарної системи;
- провести програмні експерименти та дослідити роботу моделі хмарної системи.

Реалізація цих завдань дозволить створити програмні рішення, які не тільки відповідатимуть академічним вимогам, але й можуть бути адаптованими до існуючих алгоритмів у галузі управління хмарними системами, вносячи значний вклад у розвиток галузі інформаційних технологій.

**Об'єктом дослідження** є процеси побудови хмарних систем автоматизації процесів та взаємодії з комп'ютеризованими пристроями.

**Предметом дослідження** є методи побудови архітектурного рішення та алгоритмічного забезпечення хмарних систем автоматизації процесів та взаємодії з комп'ютеризованими пристроями.

**Методи дослідження:** методи та технології інженерії програмного забезпечення, теорії масового обслуговування, імітаційного моделювання, теорії алгоритмів, підходи проектування архітектурних рішень складних систем.

**Наукова новизна** одержаних результатів полягає у наступному:

1. **Вперше** проведено комплексний аналіз існуючих архітектурних рішень для побудови структур хмарних систем з позиції теорії систем масового обслуговування та подальшого узагальнення на розподілені системи, що дозволило виявити приховані недоліки наявних підходів, зокрема їхню низьку адаптивність до динамічної зміни інтенсивності вхідного потоку заявок та неефективне використання обчислювальних ресурсів при пікових навантаженнях. На основі отриманих результатів було формалізовано критерії оптимізації структурних параметрів хмарних систем, що забезпечує мінімізацію часу затримки обробки даних та підвищує загальну відмовостійкість системи.
2. **Вперше** розроблено архітектурне рішення побудови хмарної системи з визначеною послідовністю руху запиту/завдання, що відрізняється від існуючих рішень тим, що використовує більшу кількість рівнів (включення парадигм Edge/Fog) представлення зв'язків, яке є масштабованою ієрархією при використанні розподілених ресурсів для ефективною підтримки та розгортання хмарної системи.
3. **Вперше** запропоновано можливість представлення хмарної системи через чотири рівні з включенням рівнів Edge/Fog та її розгляду, як простого

потоків запитів/завдань, що призводить багатофазову систему до стохастичної мережі.

4. **Вперше** запропоновано підхід щодо проектування архітектури хмарної системи з визначеною кількістю рівнів, який поєднує в собі умову розміщення усіх запитів/завдань на ресурсах та швидкодію роботи використаних ресурсів (умови споживання). Зокрема, для чотирирівневої архітектури хмарної системи запропоновано використання перевірки показників ефективності та надійності, як універсальних метрик функціонування системи масового обслуговування.

5. **Запропоновано** програмне рішення щодо об'єднання існуючої умови сталого режиму та вибору критеріїв структури чотирирівневої хмарної системи для подальшої перебудови на етапі її проектування, що незначно збільшує обчислювальну складність роботи запропонованих алгоритмів.

6. **Визначено** три модифікації чотирирівневих хмарних систем з достатньо великою кількістю комбінацій зв'язків між вузлами системи та визначеними правилами перенесення запитів/завдань. Зокрема, наведено ґрунтовне дослідження однієї модифікації чотирирівневої хмарної системи для обмеженого часу  $[0, T]$  з припущенням, що визначено правило балансування  $q^{bal}(t)$  та швидкість  $v_j^{task}(t)$  ( $j = 1, 2, 3, 4$ ) виконання запитів/завдань кожною колекцією вузлів рівнів.

7. **Запропоновано** визначення двох типів вузлів для побудови структури хмарної системи та розроблено програмне рішення, що надало можливість аналізу їх відокремленого функціонування та виявлення проблемних структур при імітації хмарного сервісу. Зокрема, дослідити "вузькі місця" цих блоків за параметрами надійності та отримати гнучкий інструмент використання розроблених програмних модулів для дослідження поведінки різних структур складної хмарної системи.

8. **Вперше** сформовано набори різних комбінацій чотирирівневих хмарних систем для проведення імітаційних експериментів роботи хмарної системи.

Зокрема, сформовано набори з 810000 та 244140625 комбінацій різних структур хмарної системи в залежності від використаних обчислювальних потужностей, що досить суттєво обмежує використання другого набору для аналізу на безкоштовних сервісах.

9. **Розроблено** програмне рішення для побудованої структури хмарної системи, що використовує різні її комбінації (зокрема, побудовано 810000 моделей) та надає методи та підходи аналізу її організаційної структури та функціональних зв'язків.

10. **Підготовлено** набори даних великого обсягу (отримані з відкритих джерел) з зменшеною кількістю параметрів для дослідження різних структур хмарної системи при імітаційному моделюванні їх роботи (зокрема, з 2 наборів у загальній кількості 49/47 параметрів та 2540047/46686579 проведених експериментів відповідно при використанні підходів та технологій підготовки даних машинного навчання – отримано 2 параметри для моделювання двох типів інтервалів).

11. **Досліджено** вбудовування агента-додатку на рівень хмарної системи, який має математичні локальні та глобальні наслідки при зростанні складності управління на рівні та прогнозованості, що в порівнянні з алгоритмом перерозподілу (всередині вузлів) демонструє різницю між глобальною стратегією та локальною тактикою. Зокрема, запропоновано алгоритм перерозподілу за рахунок виконання умов (п.4 наукової новизни) контролюють стабільність вузлів, а агенти-додатки підтримують баланс між рівнями.

12. **Вперше** проведено експерименти моделювання чотирирівневої хмарної системи при додаванні розроблених програмних рішень для кожного з запропонованих методів та підходів дисертаційного дослідження, що частково підтвердило гіпотезу про наявність "вузького місця" на першому рівні системи та існування порогу (який приблизно дорівнює 50%) зв'язку між вхідним потоком запитів/завдань та швидкістю обробки. Зокрема, збільшення

швидкості обслуговування ( $\mu$ ) спочатку різко покращувало пропускну здатність, але після певного порогу приріст ставав незначним, що вказує на переміщення "вузького місця" до інших компонентів системи.

**Практичне значення** робота має на етапі побудови архітектурних рішень хмарної системи для відстеження складності побудови та контролю динаміки навантаження хмарної системи, що дозволяє адаптувати отримані результати та програмні рішення до існуючих алгоритмів у галузі управління хмарними системами.

Розроблено два типи вузлів для побудови структури хмарної системи для дослідження "вузьких місць". Розроблено програмне рішення та сформовано набори різних комбінацій чотирирівневих хмарних систем для проведення імітаційних експериментів роботи хмарної системи. Проведено моделювання на різних комбінаціях хмарних систем з визначеними у дисертаційному дослідженні методами аналізу їх складності та підтверджено ефективність запропонованих підходів. Підготовлені набори даних використано при імітаційному моделюванні для формування динамічного потоку запитів/завдань за рядом даних, за визначеними розподілами та за інтервалами прибуття. Розроблено набір програмних рішень для моделювання чотирирівневої хмарної системи (без/з наявністю зв'язків між вузлами на рівнях) при додаванні умов розміщення ресурсів і споживання послуг, обчисленням основних показників ефективності роботи та умови для обчислення швидкостей навантаження.

Для розроблення програмних рішень використано мову Python та при вирішенні окремих завдань з достатньо високою обчислювальною складністю використано хмарний сервіс (SaaS) з його онлайн-інструментами.

Результати дисертаційного дослідження впроваджено в освітній процес кафедри інженерії програмного забезпечення та інформаційних технологій факультету прикладної математики та інформаційних технологій ДНУ під час викладання освітніх компонент «Інженерія надійності систем» та «Імітаційне

модельовання» для здобувачів другого (магістерського) рівня вищої освіти освітньої програми «Інженерія програмного забезпечення» спеціальності 121 Інженерія програмного забезпечення. Окремі теоретичні нароби та результати використано при викладанні освітньої компоненти «Методи теорії масового обслуговування» для здобувачів другого (магістерського) рівня вищої освіти та при виконанні кваліфікаційних робіт здобувачами факультету прикладної математики та інформаційних технологій.

Наукові результати дослідження є внеском у розвиток архітектурних рішень побудови хмарної системи задачі відстеження складності побудови та контролю динаміки навантаження. Дослідження може бути використане як основа для подальших наукових досліджень у галузі управління хмарними системами.

В якості можливих напрямків продовження дослідження можна відмітити актуальність розроблення програмного забезпечення для проєктування хмарної архітектури при подальшому контролі її стану в режимі реального часу.

**Особистий внесок здобувача.** Аналіз літературних джерел, розроблення алгоритмів та програмного забезпечення, обробка отриманих результатів здійснені безпосередньо автором. В роботі [50] здобувачеві належить виділення суттєвих характеристик ієрархічності хмарної структури, побудова класифікатора проєктування хмарних систем та формалізація умов розміщення і споживання ресурсів. В роботах [40], [95] та [103] здобувачем було створено допоміжні вузли та імітаційні моделі, розроблено архітектурне рішення інфраструктури, виконано експериментальні дослідження поведінки системи. Постановка мети і завдань дослідження, а також аналіз і узагальнення отриманих результатів проводились спільно з науковим керівником д.т.н., проф. О.Г. Байбузом.

**Апробація результатів дисертації.** Основні положення та результати дисертаційної роботи доповідалися й обговорювалися на XX та XXIII

Міжнародних науково-практичних конференціях «Математичне та програмне забезпечення інтелектуальних систем (MPZIS)» (м. Дніпро, 2022, 2023, 2024, 2025), XXVII Міжнародній конференції «Автоматика-2024» (м. Дніпро, 2024), V Міжнародній науково-практичній інтернет-конференції «Ways of Science Development in Modern Crisis Conditions» (м. Дніпро, 2024), IX Всеукраїнській науково-практичній конференції «Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем (MEICS-2024)» (м. Дніпро, 2024), наукових конференціях за підсумками науково-дослідної роботи Дніпровського національного університету імені Олеся Гончара (м. Дніпро, 2023, 2024, 2025), Міжнародній науково-практичній інтернет-конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації» (Переяслав, 2023).

**Публікації.** Основні положення й результати дисертаційної роботи опубліковано у 12 роботах: 4 статті у наукових фахових виданнях України категорії Б та 8 тез доповідей у збірниках матеріалів наукових конференцій.

**Зв'язок роботи з науковими програмами, планами, темами.** Дисертаційна робота виконувалась у відповідності з індивідуальним планом підготовки аспіранта кафедри інженерії програмного забезпечення та інформаційних технологій Дніпровського національного університету імені Олеся Гончара. Дослідження за темою дисертації здійснювалися також в рамках науково-дослідних робіт № ФПМ-2-22 «Розроблення програмного забезпечення аналізу та кластеризації часових рядів» 2022-24 рр. № держреєстрації 0122U001465 та № ФПМ-2-25 «Розроблення інформаційної технології обробки статистичних даних» 2025-27рр. номер держреєстрації 0125U002280.

**Структура та обсяг дисертації.** Робота містить вступ, 4 розділи, висновки та список використаних джерел, що містить 117 найменувань на 16 сторінках, додатки. Загальний обсяг дисертації – 189 сторінки, обсяг основного тексту – 132 сторінки. Робота містить 44 рисунки та 7 таблиць.

## РОЗДІЛ 1. ТЕОРЕТИЧНІ, МЕТОДОЛОГІЧНІ ТА ПРАКТИЧНІ ПІДХОДИ РОЗРОБЛЕННЯ ХМАРНИХ СИСТЕМ

Під архітектурою системи (АС) розуміємо формалізований опис структури програмного забезпечення або обчислювальної системи, що включає її основні компоненти, їхні функціональні призначення, способи взаємодії та інтеграції. Архітектура може бути розглянута з позицій кінцевого користувача, замовника та архітектора системи, і визначає логічну організацію системи, принципи побудови, а також технологічні рішення, що забезпечують реалізацію функціональних та нефункціональних вимог.

Характеристиками АС є компоненти, способи взаємодії між компонентами, вимоги до середовища, архітектурні шаблони та функціональні/нефункціональні вимоги.

Для опису АС можна виділити види робочих моделей:

- операційна (Operation View), яка пов'язана з концепцією операцій та містить етапи у вигляді графіки операцій високого рівня, опису з'єднання операційних вузлів та матриці обміну оперативною інформацією (Resource Flow), організаційної схеми (organization charts), моделей операційних діяльності та правил, описів переходу станів та трасування подій, логічної моделі даних;

- логічна (Logical View), яка пов'язана з концепцією абстрактного представлення системи через компоненти сервісу, операції, компоненти інфраструктури та залежності у вигляді функціональної залежності, діаграм потоків даних, схем поведінкового та часового представлення [1].

- фізична (Physical View), яка пов'язана з концепцією доступності фізичних ресурсів та їх розташування та включає розміщення компонентів на фізичних серверах, мережеві з'єднання, бази даних, операційні системи та інші апаратні та програмні засоби, які використовуються для реалізації системи.

Існують різні типи архітектурних стилів та моделей систем, вибір яких зазвичай проходить серед з найпоширеніших: монолітна, багаторівнева, сервіс-орієнтована та мікросервісна (рис. 1.1).

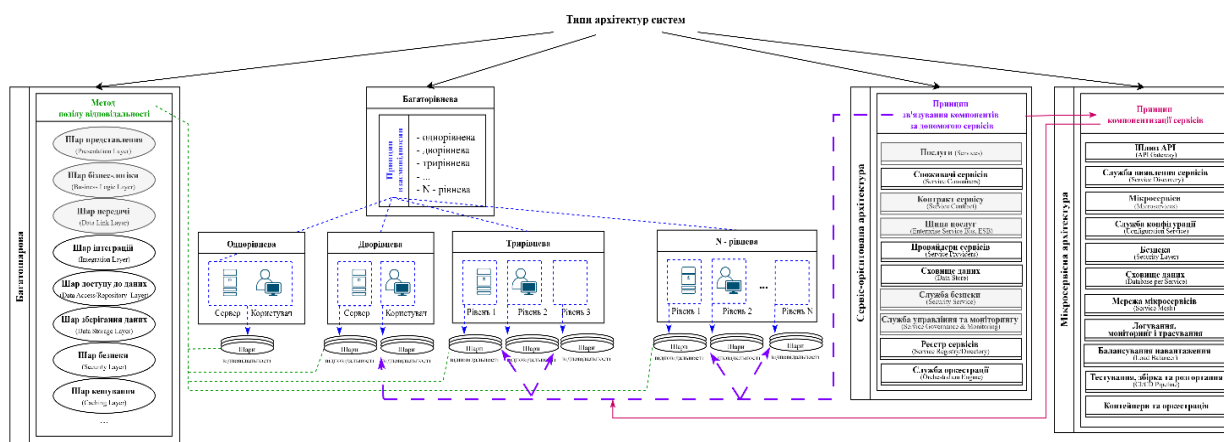


Рисунок 1.1 - Типи архітектурних стилів системи

*Монолітна (багатошарова) АС* працює за методом поділу відповідальності та використовує підхід, при якому всі компоненти програми є об'єднаними в єдиний, великий модуль. В АС виділяють шари (рис. 1.1), базовими з яких є шар представлення (Presentation Layer), шар бізнес-логіки (Business Logic Layer) та шар передачі/комунікації (Data Link Layer).

*Багаторівнева (багатоярусна) АС* ділить програмне забезпечення на рівні за принципом взаємовідносини, які розмежовують відповідальність об'єктів системи (рис. 1.1). Наприклад, однорівнева система є зразком монолітної АС для нескладних програм, яка виключає міжсистемну взаємодію та забезпечує простоту розгортання. Дворівнева система є зразком клієнт-серверної архітектури, що забезпечує відособленість операцій з управління даними, операцій подання та обробки даних. Кожний з побудованих рівнів може включати шари поділу відповідальності. Створення N-рівневої системи надає архітектурі можливості горизонтального та/або вертикального масштабування та надає можливості відповідно використання вузлів та покращення їх продуктивності.

*Сервіс-орієнтована АС* (Service-oriented architecture, SOA) базується на принципі використання та взаємодії набору сервісів через стандартні інтерфейси (рис. 1.1). Основними елементами АС такого типу є послуги (Services), контракти сервісу (Service Contract), сполучне програмне забезпечення або сервісна шина (Service Bus) та служба управління та моніторингу (Service Governance & Monitoring).

*Мікросервісна АС* ґрунтується на принципі компонентизації незалежних сервісів (рис. 1.1) з відокремленою бізнес- функцією на при розробці програмного забезпечення, основними компонентами якої є мікросервіси (Microservices), шлюз API (API Gateway), служба конфігурації (Configuration Service), Безпека (Security Layer) та ін.

В дисертаційній роботі приділена увага моделям багаторівневої архітектури за різновидом моделей обслуговування та сценаріями розгортання, зокрема компонентам хмарної архітектури та рівням абстракції, концепціям створення хмарних послуг постачальників.

Для базової побудови багаторівневої АС використовують три шари відповідальності на кожному з  $N$  – рівнів взаємовідносин (рис. 1.2).

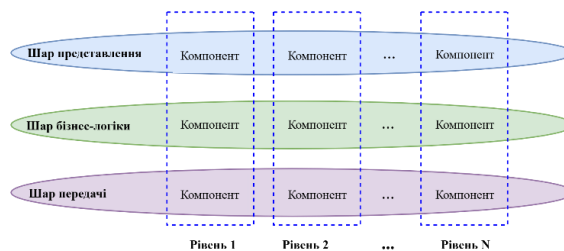


Рисунок 1.2 - Приклад багаторівневої архітектури

Для забезпечення незалежної розробки та підтримки компонентів системи за багаторівневим шаблоном розробники поділяють компоненти на групи рівнів (модулів), що надає взаємопов'язаний набір сервісів з визначеною формою доступу через інтерфейс. Поділ на рівні в системі формує ефективні ролі і зони відповідальності, технічний поділ на групи компонентів та руху будь-якого запиту.

Визначена та зафіксована послідовність руху для виконання компонентів довільного шару не є ефективним, що додатково ускладнюється збільшенням кількості рівнів (рис. 1.3).

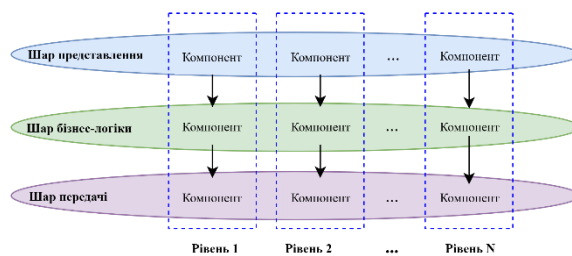


Рисунок 1.3 - Приклад багаторівневої архітектури з визначеною послідовністю руху запиту

При розподіленому розгортанні системи для представлення набору незалежних компонентів часто необхідно розділити інфраструктуру системи на окремі структурні підмножини, які об'єднані засобами зв'язку. Архітектури багатьох систем організовані як набори логічних груп компонентів (рис. 1.4).

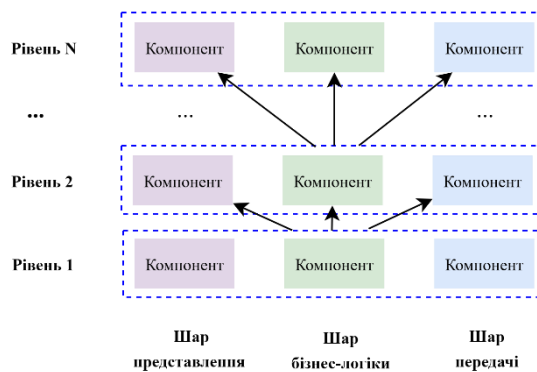


Рисунок 1.4 - Приклад багаторівневої архітектури

В архітектурі програмного забезпечення для представлення багаторівневої розподіленої архітектури з рисунку 1.4 використовується «канали та фільтри» (Pipe and Filter) для показу перетворення потоків дискретних елементів даних від введення до виведення. Слабо пов'язані компоненти з узагальненою взаємодією для повторного використання

приводять до використання підходів та алгоритмів паралельного програмування. Для побудови АС за представленим шаблоном виділяють фільтр-джерело (source) як відправна точка процесу, фільтр-перетворювач (map) як перетворювач даних, фільтр-випробувач (reduce) як перевірка критеріїв, фільтр-споживач (sink) як кінцева точка процесу (рис. 1.5). Вибір такого підходу не є правильним для інтерактивних систем, оскільки така АС орієнтована на перетворення даних через фільтри, що знижує продуктивність та може ускладнити написання фільтрів.

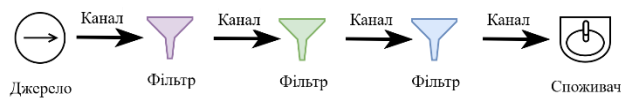


Рисунок 1.5 - Використання підходу «канали-фільтри»

В хмарних середовищах постійно виникає необхідність використання розподілених ресурсів для ефективної підтримки та розгортання, що приводить до створення застосунків мікросервісної архітектури (рис. 1.6).

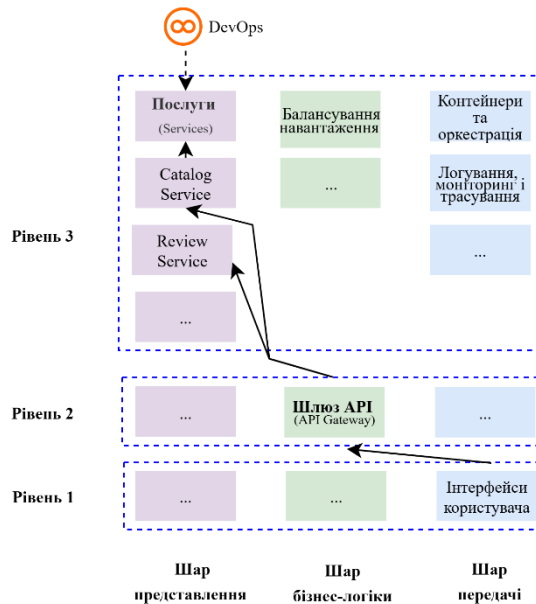


Рисунок 1.6 - Приклад мікросервісної архітектури у хмарному середовищі

*Програмно-конфігурована мережа* (Software-defined Networking, SDN) є мережею передачі даних, яка дозволяє централізовано керувати мережею за допомогою програмного забезпечення, а не через традиційні, апаратні налаштування. В SDN рівень управління мережею відокремлений від пристроїв передачі даних, і реалізується програмно, що забезпечує більшу гнучкість та ефективність.

SDN ефективні для побудови інфраструктурних хмарних сервісів для автоматичного і швидкого створення віртуальних вузлів та виділення для них віртуальних мережевих ресурсів в залежності від запитів споживачів послуг. В SDN управління трафіком здійснюється за допомогою програмного забезпечення, що зменшує обчислювальне навантаження на вузли системи, але може знизити надійність системи.

В порівнянні з мікросервісною архітектурою у хмарному середовищі (рис. 1.6) архітектура SDN складається з рівня застосунків, рівня керування та рівня інфраструктури (рис. 1.7), кожен з яких забезпечує автоматизацію мережевих функцій, гнучку взаємодію мережевих пристроїв та виконання інструкцій контролера відповідно. Зміна зв'язків між компонентами та більша деталізація кожного з наведених рівнів формують різновид SDN-архітектур.

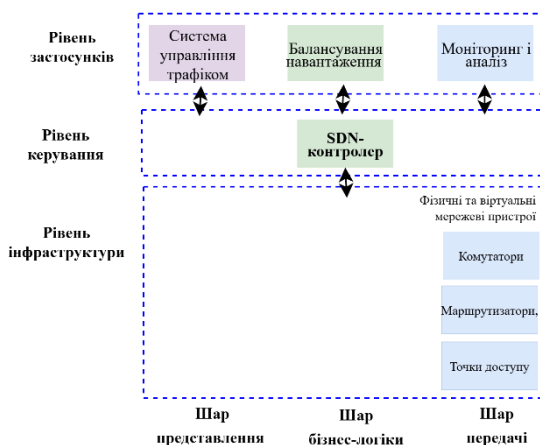


Рисунок 1.7 - Приклад SDN-архітектури

На ринку ІТ-індустрії в останні роки можна зафіксувати зацікавленість у використанні хмарних технологій та вже розроблених сервісів для їх швидкого впровадження. Розвиток інформаційних технологій вимагає наявності архітектури системи, яка може бути комбінацією всіх класичних традиційних архітектур систем. Наприклад, ІТ-компанія PayPro Global пропонує окремий підхід до розробки та розгортання додатків у хмарних середовищах, що передбачає використання мікросервісів, масштабованість і еластичність хмари та інших хмарних технологій для створення гнучких та адаптивних систем. ІТ-компанія визначає термінологію хмарно-рідної архітектури, яка включає функції для полегшення роботи в нестабільних хмарних середовищах для швидкого реагування та адаптації до змін. Відмінністю хмарно-рідної архітектури системи є використання слабо пов'язаних служб мікросервісної архітектури, які можуть бути незалежно розроблені та масштабовані для ефективного використання ресурсів [2].

*Хмарною архітектурою* є структура організації хмарних ресурсів та інфраструктури.

*Хмарна інфраструктура* включає апаратні компоненти для забезпечення роботи хмарних сервісів (зокрема, центральний процесор, графічний процесор (GPU), мережеві пристрої та інші апаратні компоненти, необхідні для безперебійної роботи систем) та програмне забезпечення для запуску та управління.

*Хмарні ресурси (або хмарні обчислення)* є моделлю забезпечення доступу до обчислювальних ресурсів (наприклад, серверів, сховищ даних, програм та послуг) через інтернет, зазвичай на вимогу, з мінімальними зусиллями з боку користувача.

Актуальним питанням є дослідження характеристик та властивостей хмарної архітектури, наприклад таких, як масштабованість, стійкість, гнучкість та ін. Якщо горизонтальне масштабування може бути за рахунок додавання/видалення кількості ресурсів (вузлів), то вертикальне

масштабування може бути за рахунок додавання/видалення сервісів залежно від робочого навантаження. Зміна розподілу ресурсів на основі фактичного використання може бути інструментом для оптимізації витрат. Для досягнення стійкості системи можна використовувати підходи «ізоляції» та самовідновлення, для мінімізації та запобіганню простою у випадках збою. Гнучкість системи може бути закладена у виборі її архітектури для виконання розробки та розгортання.

Для хмарної розробки виділяють технології організації мікросервісів, формування контейнерів, оркестрування, безпервної інтеграції/поставки (CI/CD) та DevOps.

## 1.1 Про багаторівневу хмарну архітектуру

### 1.1.1 Моделі багаторівневої хмарної архітектури

Розглянемо моделі хмарної архітектури, які формують багаторівневу архітектуру [3] за принципом взаємовідносин між ресурсами рівнів (рис. 1.7).

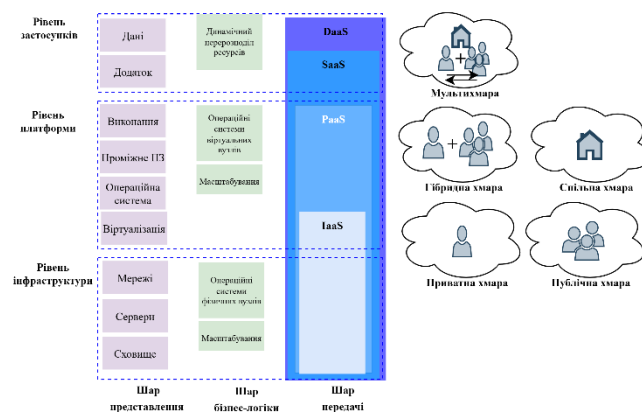


Рисунок 1.8 - Багаторівнева архітектура моделей обслуговування

З погляду споживачів виділяють основні три *моделі обслуговування* (називають ще видами хмарних послуг) [4]:

- *Інфраструктура як послуга* (Infrastructure as a Service, IaaS). Модель забезпечує доступ на вимогу до хмарної інфраструктури, такої як сервери, сховища та мережі, що усуває необхідність закупувати, керувати та обслуговувати локальну інфраструктуру.

- *Платформа як послуга* (Platform as a Service, PaaS). Модель розрахована на розробників програмного забезпечення та пропонує обчислювальну платформу з усією базовою інфраструктурою та програмними інструментами, необхідними для розроблення, запуску та управління додатками.

- *Програмне забезпечення як послуга* (Software as a Service, SaaS). Модель передбачає надання сформованого рішення користувачу з мінімальною кількістю рекомендацій для налаштування, що усуває необхідність для кінцевих користувачів розгортати програмне забезпечення локально.

Для організації аналізу та управління отриманими даними для надання даних за вимогою та незалежно від місцезнаходження чи інфраструктури споживача з'явилася ще одна модель обслуговування *Дані як послуга* (Data as a service, DaaS). Завданнями цієї моделі є збір інформації із зовнішніх та внутрішніх джерел, транспортування даних, зберігання даних у файлових сховищах провайдерів хмарних технологій, подальший аналіз даних та доставка даних до клієнта.

Всі перелічені види хмарних послуг надаються за моделлю підписки та використовуються тільки при необхідності. Сутність хмарних послуг пояснює концепція сервісного підходу до управління ІТ інфраструктури [5].

## 1.1.2 Сценарії розгортання платформи хмарних обчислень

Фізичне розташування та форма власності хмарної інфраструктури може суттєво впливати на побудову рішення [6].

Існує чотири сценарії розгортання (рис. 1.8) платформи хмарних обчислень ([7, 8, 9]): приватний сценарій (private), публічний сценарій (public), гібридний (hybrid), спільний сценарій (community) та мультисценарій (multi).

*Архітектура приватної хмари (Private Cloud)* стосується формуванню хмари, яка належить і управляється визначеною організацією, яка може приватно розміститися локально у власному центрі обробки даних, що забезпечує більший контроль над ресурсами, більший захист даних та інфраструктури та кращу продуктивність з меншою кількістю проблем перенавантаження та затримки. Однак така архітектура значно дорожча та вимагає більшої ІТ-експертизи для обслуговування.

*Архітектура публічної хмари (Public Cloud)* використовує ресурси хмарних обчислень та фізичну інфраструктуру, якими володіє та управляє сторонній постачальник хмарних послуг, що дозволяє легко масштабувати ресурси без необхідності інвестувати у власне обладнання чи програмне забезпечення, але використовують багатокористувацькі архітектури, які одночасно обслуговують інших клієнтів. Ризиками використання архітектури публічної хмари може бути витік даних та збоїв в роботі самої хмарної системи.

*Архітектура гібридної хмари (Hybrid Cloud)* використовує як публічну, так і приватну хмарну архітектуру для забезпечення гнучкого поєднання хмарних сервісів, що дозволяє переносити робочі навантаження між середовищами для використання найкращих сервісів для бізнес-потреб та робочому навантаженню [4, 10]. Ця архітектура при використанні одночасно локальних і зовнішніх ресурсів надає можливість створення більш надійної системи з меншою ймовірністю простоїв. Але недоліками архітектури

гібридної хмари є вартість локальної інфраструктури і хмарних послуг, складність керування та питання безпеки зберігання даних.

*Архітектура спільної хмари (Community Cloud)* використовує обчислювальні ресурси кількома організаціями, які мають спільні інтереси або спільні вимоги.

*Архітектура мультихмари (Multi-Cloud)* використовує хмарні сервіси від кількох постачальників хмарних послуг, що надає гнучкості та здатності краще підбирати варіанти використання для конкретних пропозицій, незалежно від постачальника.

Найпростіша базова хмарна архітектура може бути представлена на рисунку 1.9.

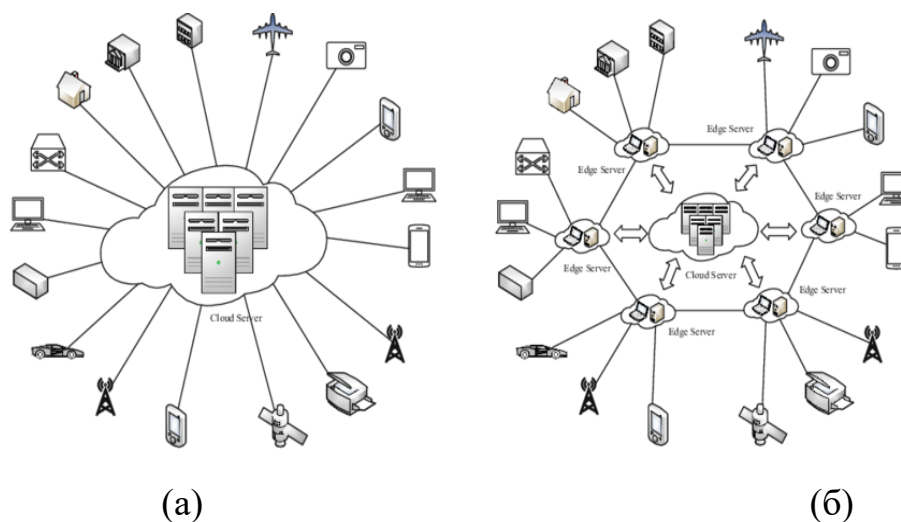


Рисунок 1.9 - Порівняння базових хмарних архітектур  
(a) Cloud і (б) Edge Computing [11]

Різновид типів хмарної архітектури обумовлюється контрольованими параметрами безпеки (рис. 1.10).

Периметр безпеки локальної приватної хмари охоплює як локальні ресурси споживача, так і ресурси приватної хмари. Модель аутсорсингової приватної хмари має два периметри безпеки, один з яких реалізований споживачем хмари, а інший реалізований постачальником.

Існують різні механізми для досягнення безпечного розділення між ресурсами приватної хмари та іншими хмарними ресурсами залежить від компромісів вибору механізмів, що можуть розділяти клієнтів як у публічній хмарі і змінювати сценарій [12].

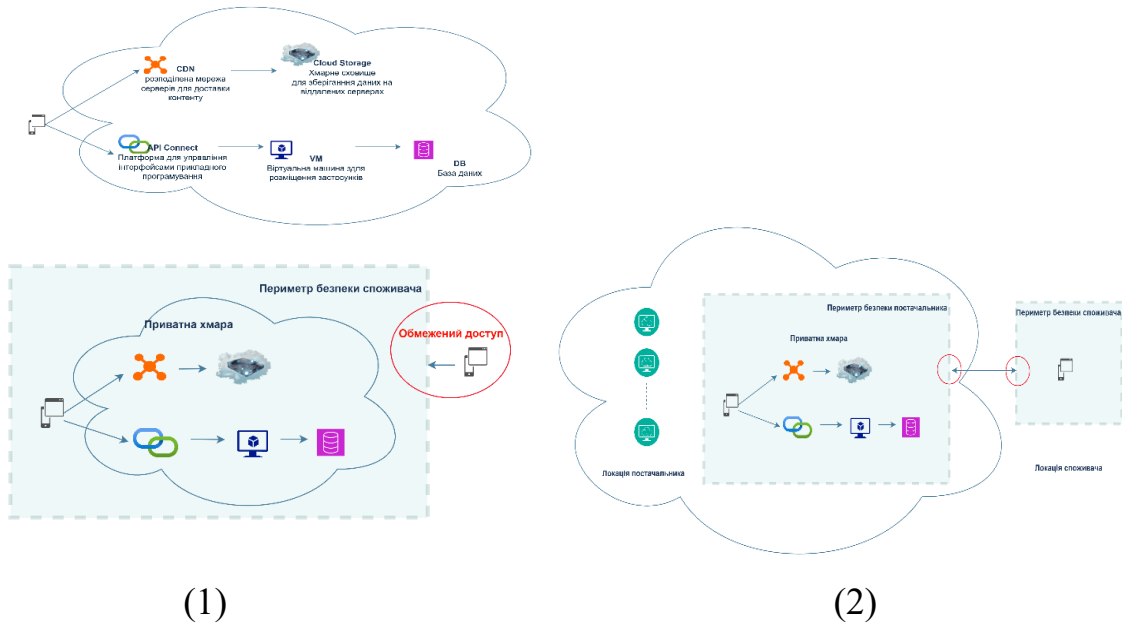


Рисунок 1.10 - Різновид хмарних архітектур за параметрами безпеки [13]: (1) – локальна приватна хмара (On-site Private Cloud), (2) – приватна хмара з передачею управління постачальнику (Outsourced Private Cloud)

В роботах [14, 81] автори виявляли найбільш ефективні підходи до моделювання хмарних ресурсів та пропонували ефективні стратегії автоматизації розгортання та управління хмарною інфраструктурою на основі платформи Microsoft Azure Cloud та інструменту Terraform. Огляд такого типу питань підкреслює важливість інтеграції автоматизованих інструментів управління для підвищення ефективності використання хмарних ресурсів з аналізом поточних викликів у масштабованих хмарних ресурсах з алгоритмами балансування навантаження [15, 52], стратегіями забезпечення неперервної доступності сервісів та методами оптимізації використання ресурсів.

Розподіленою хмарною архітектурою системи називають модель, в якій обчислювальні ресурси рівня платформи розподілені між кількома географічно рознесеними та незалежними один від одного дата-центрами. Сервіси Google Cloud, Amazon Web Services (AWS) та Microsoft Azure використовують розподілену архітектуру публічної хмари для надання послуг споживачам. Організації можуть створювати розподілені приватні хмари для внутрішнього використання та забезпечення контролю над своїми даними та ресурсами (рис. 1.11).

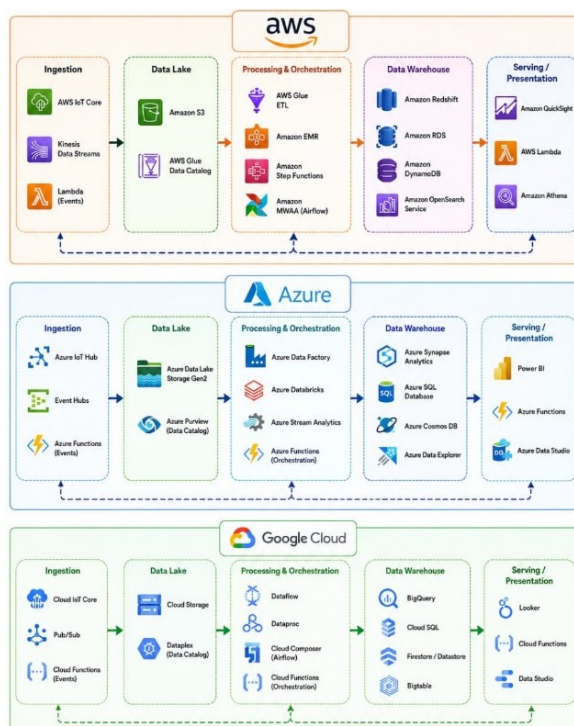


Рисунок 1.11 - Приклад архітектури конвеєра даних на хмарних платформах AWS-Azure-GoogleCloud [16, 37]

### 1.1.3 Компоненти хмарної архітектури

Компонентами хмарної архітектури є фронтенд-платформа, бекенд-платформа, хмарна модель доставки та мережа [4, 9, 17].

Кожен з цих компонентів визначає організацію хмарних ресурсів та інфраструктури [18, 19, 20, 21].

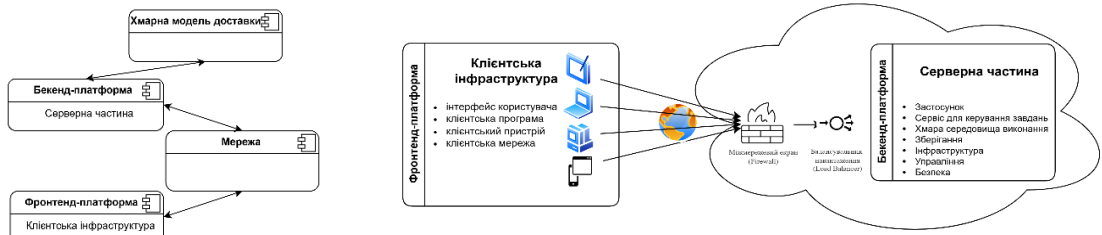
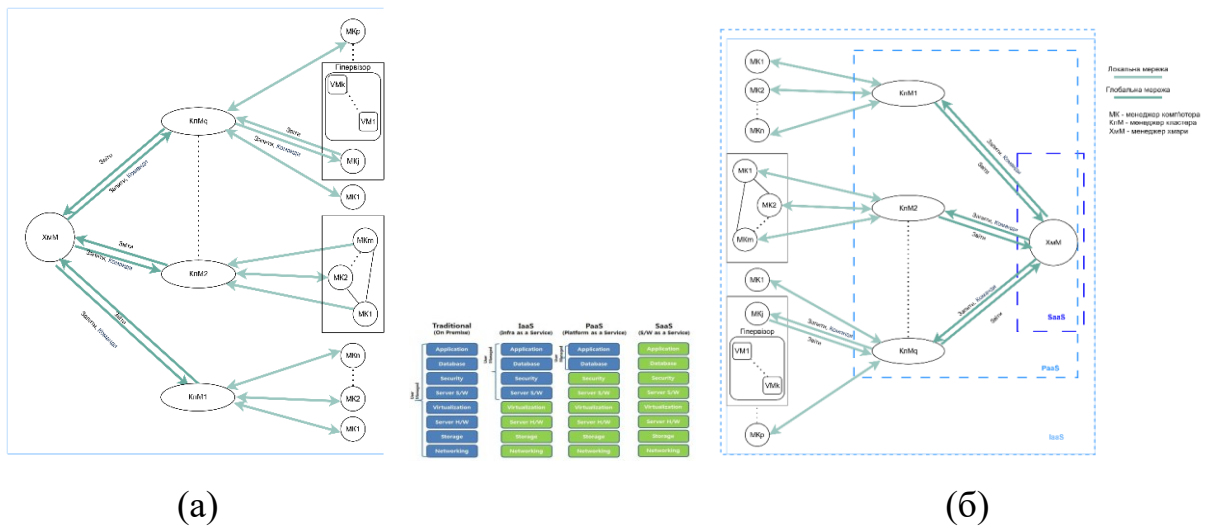


Рисунок 1.12 - Організація хмарних ресурсів та інфраструктури.

## 1.2 Основні рівні абстрактної моделі хмарної архітектури

У роботі [9] виділено три основні рівні (шари) хмарної архітектури: апаратне забезпечення, віртуалізація та додатки/сервіси.



(а)

(б)

Рисунок 1.13 - Хмарна архітектура моделі IaaS [9]

(а) - з сторони користувача, (б) – з сторони менеджера хмари

Розглянемо модель IaaS (рис. 1.8). На нижньому рівні апаратного забезпечення можна виділити менеджерів комп'ютерів. На середньому рівні віртуалізації знаходяться менеджери кластерів, які є відповідальними за взаємо з'єднання комп'ютерів та за локальне сховище. На верхньому рівні додатків/сервісів можна виокремити менеджера хмари, який відповідає за

облікові записи користувачів та розподіл ресурсів високого рівня в межах загальної хмари.

Деталізація архітектури системи по кожному рівню у вигляді ієрархії забезпечує можливість подальшої формалізації внутрішніх процесів платформи хмарних обчислень із застосуванням математичних методів та забезпечує спрощення розуміння внутрішньої будови систем з хмарною інфраструктурою [5].

Рівень архітектури	Загальна нумерація рівнів	Рівень	Опис та функції	Сучасні приклади (Інструментарій)
Рівень застосунків	L9	Дані	Керування інформацією користувача, аналітика та бази даних	Google BigQuery, AWS S3, Snowflake, MongoDB Atlas.
	L8	Додатки	Кінцеве програмне забезпечення, доступне через браузер або API	Microsoft 365, Salesforce, Slack, Zoom, GitHub.
	L7	Виконання	Середовища для запуску коду (Runtime) та контейнеризація	Docker, Kubernetes (K8s), AWS Lambda (Serverless).
Рівень платформи	L6	Middleware	Проміжне ПЗ: черги повідомлень, автентифікація, шини даних	Apache Kafka, RabbitMQ, Redis, Firebase Auth.
	L5	ОС вірт. вузлів	Операційні системи, на яких працюють хмарні інстанси	Ubuntu Server, Amazon Linux 2, Windows Server Core.
Рівень інфраструктури	L4	Віртуалізація	Технології розділення фізичних ресурсів на логічні одиниці	VMware vSphere, KVM, Hyper-V, Xen.
	L3	Мережі	Логічна топологія: підмережі, шлюзи, балансувальники	AWS VPC, Cloudflare (CDN/WAF), Cisco Meraki.
	L2	Сервери	Обчислювальні потужності (CPU, RAM) у вигляді віртуальних машин	Amazon EC2, Google Compute Engine, Azure VMs.
	L1	Сховище	Системи зберігання даних (блочні, об'єктні, файлові)	Amazon EBS, Azure Blob Storage, Google Cloud Storage.
Фізична інфраструктура	L0	ОС фіз. вузлів	Гіпервізори «заліза» (Bare Metal OS).	ESXi, Proxmox VE, Red Hat Enterprise Linux.

Рисунок 1.14 - Інструментарій багаторівневої хмарної архітектури

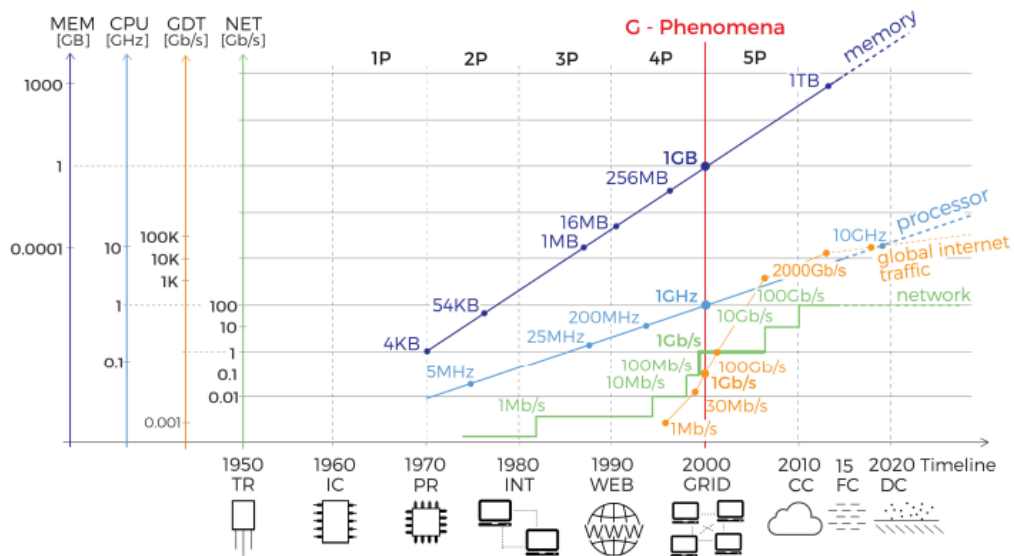
Для кожного рівня моделі багаторівневої архітектури запропоновано велика кількість розробленого інструментарію та технологій [22, 23, 24] для реалізації зв'язків між рівнями (рис. 1.14). На рівні виконання (L7) сучасний тренд дозволяє розробникам взагалі не думати про рівні L2–L5 при звичному завантаженні коду (функції) з наступним виділенням хмарою ресурсів. На даний етапі розвитку інформаційних технологій інфраструктура (L1–L4) описується кодом за допомогою інструментів типу Terraform або Ansible. Що стосується безпеки, то це поняття є актуальним на всіх рівнях: від фізичної охорони дата-центру (L-1) до шифрування даних (L9).

Постає завдання оптимізації ресурсних параметрів масштабування та аналізу залежності швидкодії програми від кількості паралельних процесів [12, 57, 25]. Мета розгортання такої системи полягає у створенні відтворюваного, високопродуктивного обчислювального середовища на базі сучасних технологій для впровадження як на хмарних, так і на локальних кластерних платформах.

### **1.3 Парадигми EDGE/FOG**

З появою парадигм EDGE/FOG [13, 26, 27] оптимізація є невід'ємною частиною хмарних обчислень, які вимагають не лише переоцінки хмарної оптимізації [28] та дослідження рішень [29, 30, 31], але й значного зсуву цілей від розгляду лише затримки до енергоспоживання, безпеки, надійності та вартості [32- 36].

Парадигма розподілених обчислень розвивається як альтернатива потужним суперкомп'ютерам (рис. 1.15), великим, дорогим і негнучким машинам та формують стратегію розвитку розподілених хмарних систем, які мають багато обчислювальних вузлів, що з'єднані швидкими локальними мережами [12, 37, 38].



G (Giga) – гіга феномен,  
 MEM – об'єм оперативної пам'яті,  
 CPU – тактова частота процесора,  
 GDT – глобальний трафік даних (Global Data Traffic),  
 NET – спосіб взаємодії та передачі інформації між різними мережами,  
 1P-5P – фази розвитку,  
 TR – транзистори,  
 IC – інтегральні мікросхеми,  
 PR – процесор,  
 INT – Інтернет,  
 WEB – служби WWW,  
 GRID – розподілені Grid-мережі,  
 CC – хмарні обчислення (Cloud Computing),  
 FC – туманні обчислення (Fog Computing),  
 DC – поєднання концепції хмарних обчислень з можливостями кінцевих пристроїв (Dew Computing)  
 Рисунок 1.15 - Еволюція розподілених обчислень [33]

На рисунку 1.16 представлено масштабовану ієрархію хмарної системи з можливими існуючими рівнями, задачами, ресурсами та сценаріями розгортання.

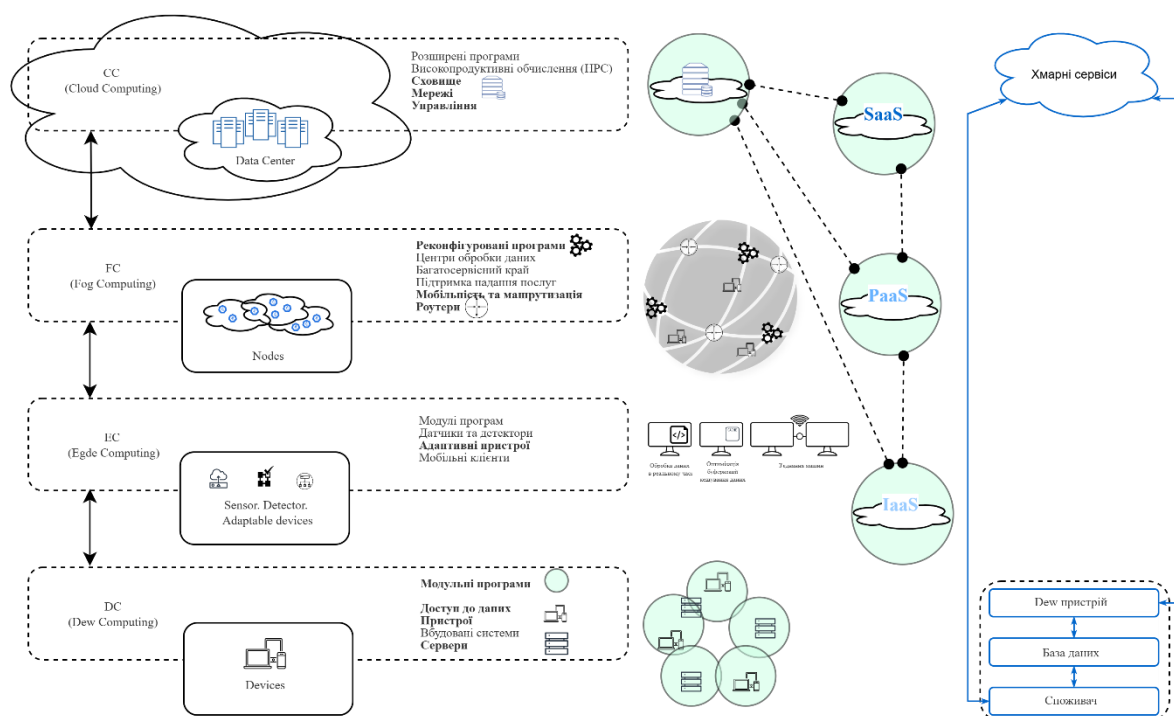


Рисунок 1.16 - Масштабована ієрархія розподілених обчислень [40] з включеним рівнем Edge/Fog Computing

DC використовує концепцію мікросервісів у вертикально-розподіленій ієрархії обчислень та зсуває межі мережі до додатків та даних у підключених до мережі кінцевих пристроях при використанні інформаційних технологій (mobile data acquisition, peer-to-peer ad-hoc networks та ін.).

EC (Edge Computing) є моделлю розподілених обчислень та наближає обчислення та зберігання даних до джерел даних.

Функції FC дозволяють створювати повторювані структури в концепції периферійних обчислень EC (Edge Computing) та розширюють платформу CC з її послугами до межі мережі EC. Метою FC є збільшення ефективності шляхом обробки даних у місці їх отримання та зниження кількості даних, що передаються у зовнішню мережу. Концепція FC підтримує зростаючу складність програмних застосунків (горизонтальне масштабування).

Властивостями технологій СС є забезпечення для споживачів ефективного управління дата-центрами та віртуалізації обчислювальних та мережевих ресурсів (зокрема, із застосуванням сучасних методів маршрутизації [39]) при використанні споживачами хмарних послуг за моделями IaaS, PaaS, SaaS та наявності доступу до Інтернету, що дозволяє послугам СС масштабуватися залежно від вимог з забезпеченням гнучкого середовища для роботи застосунків (табл. 1.1).

Таблиця 1.1 Характеристики та особливості рівнів ХА

Характеристики	Особливості рівнів хмарної архітектури			
	СС	FC	ЕС	DC
Місце обробки даних	центральний хмарний сервер	вузли	вузли	пристрій споживача послуг
Обчислювальна потужність та можливості зберігання	найкраща			найгірша
Розподіленість		між вузлами	між вузлами	на пристроях
Підключення до інтернету	постійний доступ	періодична синхронізація		періодична синхронізація

У роботах [41] і [42] авторами запропоновані математичні моделі для планування застосунків реального часу тривірнєвої архітектури та з використанням рівня Dew Computing. У роботі [41] проведений аналіз та порівняння трьох режимів обчислення СС, ЕС і FC, побудована архітектура сервісів туманних обчислень.

Визначення швидкодії хмарної системи дуже часто пов'язують не тільки з загальною продуктивністю програмного забезпечення та взаємодією між різними компонентами багаторівневої хмарної системи, але і з часом реакції користувачів та часом виконання запитів/завдань. На швидкодію хмарної системи суттєвий вплив має використані алгоритми та структури даних для

основних/додаткових модулів програмного додатку з технологіями паралельної обробки даних.

Ефективність роботи хмарної системи залежить від зовнішніх факторів (обсяг даних [43], тип запитів/завдань та інше), що є причиною перевантаження ресурсів системи [44], особливо при виконанні операцій з обробки та аналізу даних [45] (запропоновані існуючі рішення у вигляді програмних додатків).

Структура додатка для інтелектуальної оптимізації хмарної інфраструктури базується на модульному підході та охоплює повний цикл (від збору метрик до автоматичного реагування) і складається з таких компонентів:

- *блок накопичення даних* відповідає за отримання інформації від сенсорів та агентів з подальшим її збереженням у базах даних (SQLite, PostgreSQL або MongoDB);
- *аналітичний блок* здійснює попередню обробку (ETL-процеси) та глибокий аналіз зібраних показників;
- *ядро машинного навчання* містить моделі прогнозування (класифікація, регресія), які навчаються передбачати майбутнє навантаження;
- *блок інтелектуального масштабування* приймає рішення щодо коригування ресурсів у середовищах AWS, Azure чи Google Cloud на основі прогнозів;
- *система нагляду* забезпечує безперервний моніторинг стану системи та контроль за виконанням операцій;
- *контур захисту*: відповідає за безпеку даних та загальну відмовостійкість усієї архітектури.

Для візуалізації такої архітектури найкраще підходить схема конвеєра (pipeline), де дані проходять шлях від фізичних/віртуальних показників до конкретних дій у хмарі (рис. 1.17).

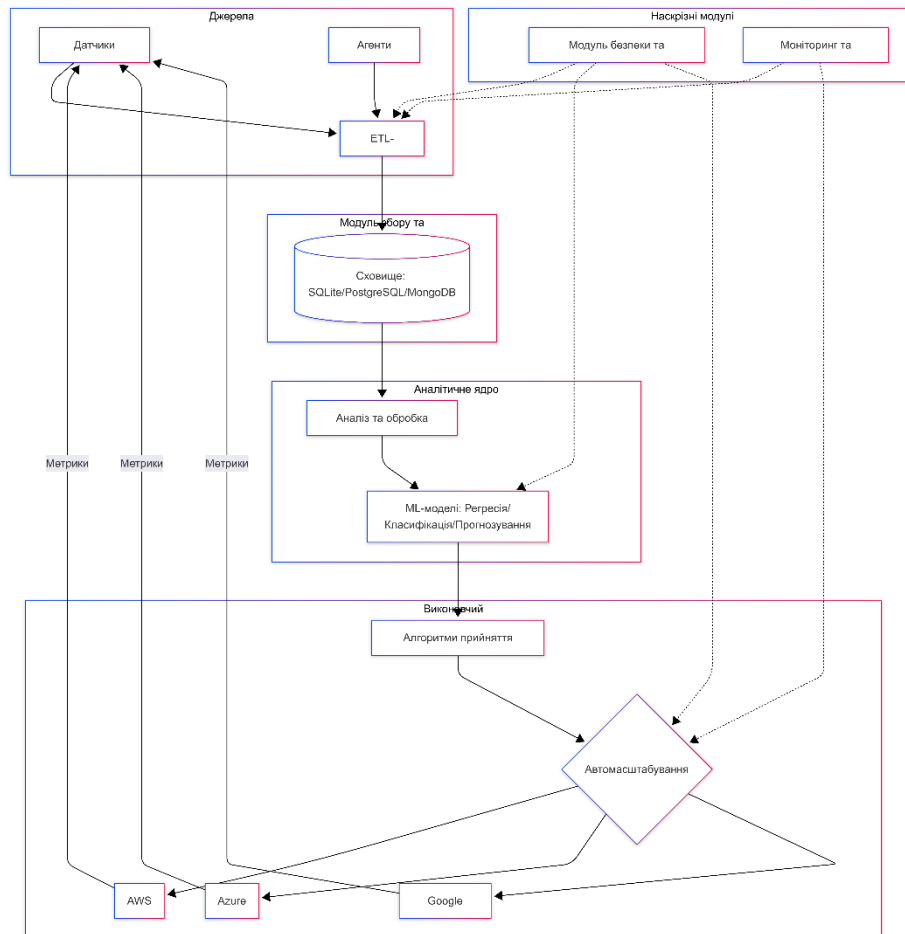


Рисунок 1.17 - Структура додатка для інтелектуальної оптимізації хмарної інфраструктури [46]

#### 1.4 Задача масштабованості

Перевагами масштабованих та доступних систем, які є затребуваними на даний час і стимулюють розвиток хмарних архітектурних рішень [12, 47], можна вважати децентралізацію обчислень, економію ресурсів та гнучкість масштабування.

При проєктуванні хмарної архітектури (ХА) є потреба у виборі різних компонентів хмарних технологій, які взаємодіють і підключаються для створення середовища хмарних обчислень [48]. ХА є рішенням, що пропонує спосіб стратегічного об'єднання ресурсів для створення хмарного середовища для вирішення прикладних задач.

У роботі [49] авторами запропоновано інформаційну технологію масштабування хмарних додатків на поєднанні реактивного та проактивного масштабування, що може підвищити загальну ефективність вебдодатку на 8% та підвищити адаптивність хмарної структури і зменшити витрати на хостинг додатків.

Дослідження [50] присвячено вирішенню задачі горизонтального масштабування. Автори реалізували алгоритми покрокового нарощування кількості паралельних процесів із фіксацією часу виконання, контролем динаміки навантаження на обчислювальний вузол та моніторингом використання оперативної пам'яті.

Для дослідження задачі масштабування при збільшенні розмірності мереж науковці використовують різні методи аналізу для управління мережею та підвищення швидкості та надійності передачі даних. Так у роботі [51] представлений метод динамічної реконфігурації, де досліджувався процес генерації топології програмно-конфігурованої мережі за допомогою випадкових геометричних графів перетину. Автором приділена увага також впливу завантаженості каналу зв'язку на якісні характеристики каналів передачі даних.

## **1.5 Про форми використання інформаційних технологій**

Актуальними питаннями теперішнього часу є формування ідеї реалізації широкодоступного середовища з сучасними і доступними інформаційними технологіями та пошуку рішень для проблем, які виникають при побудові та експлуатації.

Для будь-якої вже розробленої програмної системи проблемним питанням завжди є етап експлуатації та оновлення не тільки програмного забезпечення, але й зміна форми використання інформаційних технологій (наприклад, на форму споживання послуг).

## **1.6 Перспективи використання локальної платформи хмарних обчислень**

Серед ключових недоліків хмарних систем варто виділити відсутність у користувача прав власності на інфраструктуру та неможливість безпосереднього доступу до її внутрішніх компонентів. Зберігання компанією провайдера даних споживача та наявність інструментів для надійного і швидкого доступу є необхідною умовою для отримання якісних послуг.

Стрімкий розвиток хмарних обчислень забезпечує безперервну еволюцію ІТ-індустрії та трансформацію бізнес-процесів. Завдяки впровадженню доступних інструментів організації отримали ефективні методи проєктного менеджменту, які раніше потребували надмірних фінансових витрат. Інтеграція хмарних сховищ та сервісів обробки даних звела до мінімуму стартові капіталовкладення та трудомісткість адміністрування. Домінування хмарних архітектур у сучасному інформаційному просторі вказує на те, що невдовзі гнучкі й легкі системи управління проєктами стануть стандартом для більшості секторів бізнесу.

## **1.7 Оцінка ефективності**

В останні роки приділена увага питанню існування різних видів хмарної інфраструктури та відповідно методикам їх побудови та методам перевірки ефективності масштабованих систем [26, 31, 52, 53, 54].

Питання про фізичне розташування та форми власності платформ хмарних обчислень може суттєво впливати на ефективність застосування хмарних обчислень [4, 55].

Актуальним залишається питання оцінки ефективності систем при використанні різних методів збільшення продуктивності [56, 57] та підходів щодо масштабування [49, 58].

Основними проблемами можуть бути складнощі програмування та питання балансу між витратами та продуктивністю [59, 60, 65].

У роботі [61] приділена увага динамічній реконфігурації системи з урахуванням параметрів якості обслуговування каналів зв'язку, надійності вузлів та підвищення стабільності передачі даних. Універсальність системи може забезпечуватися взаємоінтеграцією отриманих рішень.

## **1.8 Методи перевірки ефективності масштабованих систем**

Для порівнювання стратегій масштабування хмарного додатку у роботі [49] наведено огляд підходів до масштабування хмарних систем, що може бути використано для їх порівняння. Авторами проведено формалізація вимог до симулятора хмарного додатку та спроектовано його архітектуру, яка підтримує взаємодію із зовнішніми модулями масштабування та дозволяє розраховувати метрики хмарного додатку, такі як середній час виконання мережевого запиту, завантаження процесора та пам'яті. Для перевірки ефективності використовуються методи інтерполяції реального тестового навантаження хмарного додатку при використанні хмарних віртуальних машин, що надає можливість оцінити ефективність вибору стратегії масштабування та зменшити фінансові і ресурсні витрати на тестування.

Формальний опис інфраструктури хмарного застосунку та можливих переходів між ними запропоновано у роботах [62, 63, 64, 68]. У якості критерію класифікації поточного стану хмарного застосунку запропоновано підхід на основі методу Пейджа-Хінклі та календаря подій, пов'язаних з робочим станом хмарного застосунку. Авторами розглянуто поточний стан для підвищення точності прогнозування робочого навантаження хмарного застосунку. Проведено порівняння запропонованого критерію зі стандартним офлайн-критерієм для аналізу інформації про весь часовий ряд хмарного

застосунку протягом значного часу після подій, що призводять до піку навантаження і не може бути використаний при класифікації в режимі реального часу. Класифікація стану хмарного застосунку є узгодженою у 92% випадків і отримана інформаційна модель масштабування хмарного застосунку зі змінними піками навантаження може бути використана для більш великих систем.

## **1.9 Неефективність сучасних систем**

Проблема неефективності сучасних систем [66] та горизонтального масштабування як методу підвищення продуктивності [49, 57, 58] є досі актуальною, що надає розвиток наукових досліджень в питаннях щодо обмеження паралелізму, невідповідності між завданням та системою, складністю програмування та питання балансу між вартістю та продуктивністю [67]. Науковцями запропоновано класифікацію рішень як архітектурні та мережеві класи, де увага приділена квантовим обчисленням та парадигмі потоку даних (controlflow та dataflow), що забезпечує непогані топологічні характеристики та підтримку основних сучасних технологій маршрутизації.

## **1.10 Способи опису хмарної інфраструктури**

Різні постачальники пропонують великий спектр розроблених технологій при використанні опису хмарної інфраструктури різними способами. Наприклад, метод об'єднання деталей від різних постачальників приводить до представлення хмарної інфраструктури узагальнено на вищому рівні абстракції [48], зокрема у форматі підходу до опису та автоматизації управління інфраструктурними ресурсами (IaC) для організації хмарних технологій за способом їх взаємодії з користувачем.

## 1.11 Стабільність передачі даних

Кожний з компонентів мікросервісної архітектури (рис. 1.1) є важливим та складним в реалізації щодо використання існуючих інформаційних технологій. Для довільної хмарної інфраструктури актуальним проблемним питанням щодо реалізації може бути модуль видимості (visibility) у вигляді сервісу «Логування, моніторингу, трасування», зокрема при виборі сценарію гібридного розгортання хмарної інфраструктури [69] при ускладненні моніторингу та збільшенні кількості ризиків [70, 71]. Логування є процесом запису подій, повідомлень та помилок під час роботи застосунку для відстеження його стану та діагностики проблем, який включає етап анрегації, зберігання та аналізу даних (логів). Моніторинг використовує для аналізу дані логування та застосовує інструменти для надання метрик. Для гібридної хмарної розподіленої інфраструктури виникають проблеми, які пов'язані несумісністю різних форматів даних, що можуть бути наслідками використання різних інструментів: локальний кластер Kubernetes формує дані у форматі CRI (JSON) з JSON-об'єктами для повідомлень, хмарний сервіс в AWS формує дані у текстовому форматі в S3 визначеного формату, сервіс моніторингу від стороннього постачальника надсилає дані у форматі структурованого тексту CSV. У роботі [69] розглянуто питання централізації систем моніторингу за рахунок зібрання даних з усіх перелічених джерел.

Метою наукових досліджень є підвищення стабільності передачі даних у хмарній інфраструктурі при зростанні навантажень не тільки на канали зв'язку при умові зростання навантажень та при відмові вузлів [61]. Запропоновані алгоритми, які є вбудованими в інструментарій провайдера не в повному обсязі забезпечують комплексне вирішення проблеми визначення найоптимальнішого шляху щодо забезпечення з'єднання та можуть не враховувати специфічні параметри побудованої системи [55, 72]. В умовах зростання кількості користувачів хмарних сервісів та кількості вузлів та

запропонованих інструментів виникає актуальна проблема підвищення стабільності роботи всієї системи.

### **1.12 Обробка даних з мінімальними затримками**

Наразі приділена увага потребам швидкої обробки даних з мінімальними затримками при зростанні кількості пристроїв хмарної інфраструктури, що може привести до зменшення ефективності хмарних обчислень та формує концепцію периферійних обчислень для підвищення продуктивності та балансування навантаження для забезпечення балансу [73].

У роботі [74] наведено результати теоретичних та експериментальних досліджень методів формування та організації потоків даних у розподілених комп'ютерних системах.

### **1.13 Алгоритми балансування**

Для вирішення задачі балансування науковцями запропоновано безліч емпіричних методів з використанням щільності з'єднань, розподілу навантаження [65, 71, 75, 76], пріоритету завдань [77] та вибору кількості серверних служб [78].

### **1.14 Інформаційні технології оптимізації хмарної системи**

Основні постачальники послуг при різних назвах їх мають однакову концепцію, що може об'єднувати їх у відповідні категорії за ключовими словами (табл. 1.2).

Таблиця 1.2 Категорії постачальників послуг за ключовими словами

	<b>IaaS</b>	<b>SaaS</b>	<b>PaaS</b>
Обчислення	VMs, Functions, App Services, Cloud Services	Functions, App Services, Cloud Services	
	K8ds, Container Services, Batch, Delicated Host	K8S, Container Services, Batch, Fabric Services	
Сховище	Disk, Blob, File, Box	SQL DB, NoSQL DB	
	Data Lake, Ledger	Redis, Caching	
Мережі	Routing, Security, LB, DDoS, Firewall, DNS, CDN		
	WAN, VPN, GateWay, Routing		
	5G, Edge		
Аналітика			Data, Cloud Pak, Data Catalog, Reporting
			Machine Learning, Model Creation
Розроблення		Routing, Security, LB, DDoS, Firewall, DNS, CDN	DevOps, DevSecOps, IT Ops Services
		App Config, Development, Spring Cloud, IDE Services	Dev Tools, IDE
		Test, Lab Services & Load Services	
Безпека	Encryption, Vault, Protection, Gateway	Identify, Access Management, Encryption	Defender, Sentinel, Protection
	Confidential, Ledger, Attestation	Confidential, Ledger, Attestation	
Гібридний	Openshift, KBS		
Управління			Advisor, Backup, Portal Shell
			Cost Management Services
AI / ML		AI, BOT, Spech, Text	

Особливу увагу можна приділити підходам щодо поширення програмного забезпечення.

На сучасному етапі розвитку галузі інформаційних технологій у великій кількості організацій можна замітити тенденцію відмови від розгортання локальних програмних комплексів до використання мережевих сервісів, зокрема хмарних сховищ, систем управління взаємовідносинами з клієнтами (CRM) та планування ресурсів підприємства (ERP).

Протягом останніх років активно формується глобальне розподілене середовище надання затребуваних ІТ-послуг у формі хмарних сервісів [5]. Зі становленням цього сегмента споживачі отримують повний спектр можливостей: від базових інфраструктурних рішень до готових розподілених програмних платформ розробки та розгортання.

Хмарні інфраструктури містять механізм багаторівневого абстрагування та різні сценарії розгортання, що ускладнює розуміння. Збільшення попиту на такі хмарні послуги породжує велику кількість сценаріїв, що приводить іноді до хибних тверджень щодо використання обраної архітектури та інструментарію для розв'язання поставленої задачі.

Найпопулярнішими технологіями, що використовуються в хмарній системі, є Hadoop, Dryad та інші фреймворки для скорочення карт. Для оптимізації продуктивності хмарної системи можуть використовуватися Cap3, HEP та Cloudburst. У дослідженнях того періоду розглядаються різні підходи до побудови хмарної інфраструктури, а також проводиться аналіз інструментів і технологій її реалізації з урахуванням таких характеристик, як швидкість розгортання, ефективність, переваги та обмеження застосування [79].

Велику кількість різних CI/CD-рішень (комбінація безперервної інтеграції (Continuous Integration) та безперервної доставки (Continuous Delivery) програмного забезпечення в процесі його розроблення), які є одними з найкращих практик DevOps-технологій, пропонують AWS, CircleCI,

CloudBees, CodeFresh, GitLab, IBM, JFrog, Micro Focus, Microsoft, Red Hat, Xebia Labs та ін.

Методологія DevOps базується на інтеграції двох раніше ізольованих сфер: розроблення програмного забезпечення (Dev) та його системної експлуатації (Ops). Підхід засновано на міжфункціональній взаємодії обох підрозділів для оптимізації робочих процесів та підвищення якості програмних продуктів на етапі релізу. Спільна діяльність дозволяє оперативну усувати інженерні бар'єри та прискорювати виведення рішень на ринок.

Практичне впровадження DevOps забезпечує такі переваги: скорочення термінів реалізації завдань, підвищення надійності, ефективне керування складними чи динамічними середовищами розробки, тестування та експлуатації з мінімальними ризиками. Це досягається завдяки автоматизації політик відповідності, точним інструментам контролю конфігурацій та застосуванню концепції «інфраструктура як код» (IaC).

Ключовим елементом DevOps є автоматизація інфраструктури та процесів конфігурування, а найбільш критичним завданням — розгортання програмного забезпечення. Цей підхід гарантує своєчасну доставку сервісів на різноманітні платформи. Автоматизація підвищує швидкодію хмарних систем, точність міжвузлових зв'язків, послідовність операцій, надійність архітектури та, як наслідок, частоту релізів. Вона охоплює етапи створення, розгортання та моніторингу програмного забезпечення.

Автоматизація інфраструктури та деплою (deployment) є критично важливими елементами безперебійної доставки сервісів на цільові платформи. Оптимізація процесів збирання, розгортання та моніторингу підвищує продуктивність хмарного середовища, точність взаємодії вузлів та частоту релізів (deployment frequency). Ефективна реалізація цих DevOps-практик прямо залежить від зваженого вибору інструментарію, оскільки сучасний ІТ-ринок пропонує широкий спектр як комерційних, так і відкритих (open-source) програмних рішень для кожного етапу життєвого циклу розробки.

Останні роки розвивається технологія DevSecOps, яка є еволюцією DevOps, що інтегрує практики безпеки, тестування та відповідність безпосередньо в конвеєр CI/CD з самого початку («зсув ліворуч»), а не розглядає безпеку як окремий завершальний етап.

Хоча DevOps зосереджується на швидкості та співпраці між розробкою та операціями, DevSecOps гарантує, що безпека є спільною відповідальністю, зменшуючи вразливості та уникаючи "вузьких місць" у розгортанні.

Ключовими відмінностями між DevOps та DevSecOps є фокус на безпеці. DevOps зосереджується на швидкості та співпраці між розробкою та операціями, а DevSecOps впроваджує безпеку, сканування вразливостей та перевірки відповідності протягом усього життєвого циклу розробки програмного забезпечення (SDLC).

Суттєвою різницею в цих двох підходах є час забезпечення безпеки. Якщо в DevOps перевірки безпеки часто відбуваються в кінці конвеєра, що може приводити до затримок, то в DevSecOps безпека інтегрована на кожному етапі.

Різниця полягає ще і в зоні відповідальності, де DevOps в першу чергу охоплює команди розробників та операцій, тоді як DevSecOps розширює цю зону та включає команди безпеки як активних учасників процесу розробки.

Але спільним при використанні цих послуг є використання автоматизації з розширенням на завдання безпеки для DevSecOps.

Надалі у роботі буде приділена увага цим технологіям на рівні аналізу швидкості та співпраці між розробкою та операціями.

## **Висновки до розділу 1**

В першому розділі було проведено аналіз спеціалізованої літератури та інформаційних технологій для визначення актуальності теми дослідження

даної дисертаційної роботи. Розглянуто існуючі методи та рішення для виконання поставленої задачі.

Наведено різновид архітектури системи та висвітлено проблематику задач проєктування систем з описом можливих сценаріїв розгортання.

Поставлено формальну задачу для побудови багаторівневої архітектури хмарної системи та розглянуто існуючі технології організації хмарної архітектури за допомогою як традиційних (мікросервісна, формування контейнерів, оркестрування, безперервна інтеграція/поставка), так і комбінованих (хмарно-рідна, DevOps, DevSeqOps).

При врахуванні проведеного аналізу переваг та недоліків кожного з описаних підходів, для розроблення методу (технології) за базову концепцію обрано багаторівневе архітектурне рішення з визначеною послідовністю руху запиту/завдання. Даний вибір ґрунтується на необхідності використання розподілених ресурсів для ефективної підтримки та розгортання хмарної системи, що дає можливість реалізувати контроль навантаження вузлів системи та формування стратегії динамічної перебудови її архітектури за необхідністю.

Для задачі побудови хмарної архітектури розглянуто характеристики та властивості хмарної архітектури (масштабованість, стійкість, гнучкість та ін.) та визначено, що найбільше відповідає темі дослідження.

Приділено увагу багаторівневій архітектурі, які є організованими за принципом взаємовідносин між ресурсами рівнів, та існуючим сценарієм розгортання хмарної системи (приватна, публічна, гібридна, спільна, мультихмара). Для подальшого дослідження та проведення експериментів обрано приватний та публічний сценарій розгортання хмарної системи.

Розглянуто парадигми розподілених обчислень Edge/Fog та запропоновано масштабовану ієрархію хмарної системи з включенням цих рівнів для розгляду надійності хмарної системи та її хмарної оптимізації. Характеристики та особливості рівнів хмарної архітектури показують існуючі

проблеми на рівнях Edge/Fog щодо обчислювальної потужності та можливостей зберігання, що надає надалі у дисертаційному дослідженні поставити задачу побудови методів та інструментарію для їх вирішення.

Оглянуто різні підходи щодо побудови рівнів абстрактної моделі хмарної архітектури та існуючі методи аналізу для управління хмарною системою та підвищення швидкості та надійності передачі даних. Обрано задачу побудови імітаційної моделі (окремих вузлів/модулів) щодо використання стратегії масштабування хмарної системи.

## РОЗДІЛ 2. МЕТОДИ АНАЛІЗУ СКЛАДНОСТІ ХМАРНОЇ СИСТЕМИ

### 2.1 Інструментарій та технології багаторівневої архітектури платформи хмарних обчислень

Для кожного рівня моделі багаторівневої архітектури платформи хмарних обчислень запропоновано велика кількість розробленого інструментарію та технологій для реалізації зв'язків між рівнями. Для забезпечення процедур відтворювання та масштабування із застосуванням сучасних технологічних рішень, розрахованих як для хмарних (OpenStack, AWS, Google), так і для кластерних платформ (Slurm), прийнято до уваги інформацію про досвід тестового розгортання та використання визначених хмарного сервісу та віртуального кластерного середовища [80].

На початку власного дослідження виконано модифікацію архітектури СХМОбч [80] за моделями обслуговування. Зокрема, у роботі [50] досліджено семирівневу модель класифікації хмарних обчислень за типами послуг. Основну увагу зосереджено на пошуку оптимальних за ресурсами параметрів масштабування інфраструктури. Для подальшого вибору раціональної архітектури системи хмарних обчислень застосовано наївний байєсівський класифікатор.

На рисунку 2.1 (а) представлені варіанти вибору інструментів та технологій при розв'язанні задачі побудови багаторівневої платформи хмарних обчислень. Червоним кольором (рис. 2.1, б) представлена інформація використаних інструментарію та технологій щодо створення масштабованого середовища.

Розглянуто модель для приватного сценарію розгортання ХА.

Нехай  $A$  і  $B$  є незалежними подіями,  $P(A)$  – апіорна (що передусе досвіду) ймовірність події  $A$ ,  $P(B)$  – повна ймовірність події  $B$ ;  $P(A)$  –

ймовірність спостереження події  $B$  за умови істинності події  $A$ ;  $P(B)$  – апостеріорна (умовна, що впливає з досвіду) ймовірність, ймовірність події  $A$  за умови істинності події  $B$ .

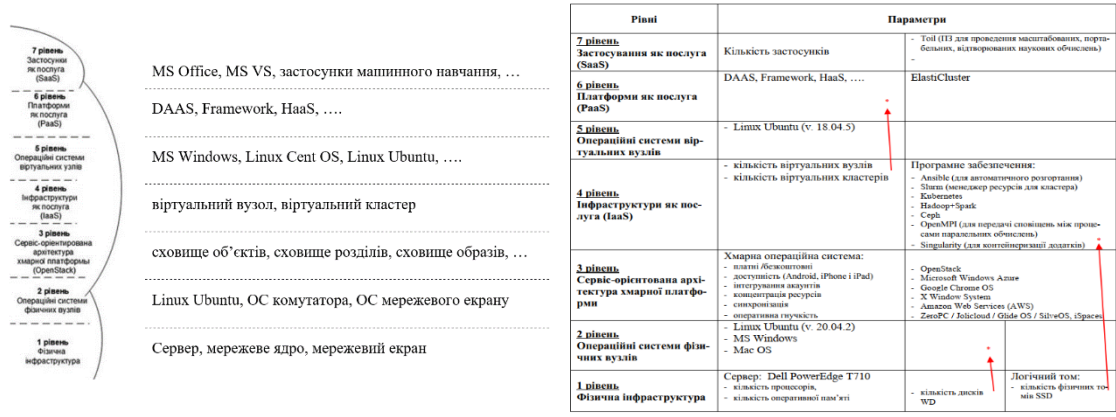


Рисунок 2.1 -

- (а) Набір інструментарію та технологій багаторівневої архітектури СХМОбч
- (б) Вибір параметрів моделі Байєса ( примітка - для розміщення / розгортання)

Використовуючи основу теореми Байєса та формулу наївного байєсівського класифікатора (зі строгим припущенням про незалежність подій) можна записати наступним чином:

$$P(y_i | c_1, c_2, \dots, c_N) = \frac{P(c_1, c_2, \dots, c_N | y_i) P(y_i)}{P(c_1, c_2, \dots, c_N)}$$

де

$i = 1, 2, \dots, M$ ,  $P(y_i)$  – ймовірність події  $y_i$ ,

$P(c_1, c_2, \dots, c_N)$  – повна ймовірність, яка завжди залишається однаковою при розрахунку наївного класифікатору Naive Bayes;

$P(c_1, c_2, \dots, c_N | y_i)$  – ймовірність значення ознак  $c_1, c_2, \dots, c_N$ , за умови відповідності їх класу  $y_i$ ;

$P(y_i | c_1, c_2, \dots, c_N)$  – умовна ймовірність даних, які є включеними до класу  $y_i$ , при врахуванні їх особливостей  $c_1, c_2, \dots, c_N$ .

Кожний з одновимірних граничних розподілів  $(c_j | y_i), j = \overline{1, N}, i = \overline{1, M}$  оцінюється окремо. Багатовимірна задача зводиться до оцінки одновимірних, для яких потрібна навчальна вибірка меншого розміру.

Розглянемо типовий випадок бінарного стану системи: «Стабільний (ресурсів достатньо)» та «Критичний (потрібне негайне масштабування)», для якого визначено два відповідні класи:  $y_1$  та  $y_2$ . Припущення по вибору існуючого програмного забезпечення застосовується без прив'язки до фінансової сторони використання будь-яких технологій. Калькулятор вибору інструментів створення хмарної мережі є актуальним і головною метою його існування є вартість [81].

Нові об'єкти будуть класифіковані шляхом порівняння їх оцінок із заданим пороговим значенням. Новий об'єкт з вищою оцінкою буде віднесено до класу  $y_1$ , у протилежному випадку об'єкт буде віднесено до класу  $y_2$ :

$$\frac{P(y_1 | c_1, c_2, \dots, c_N)}{P(y_2 | c_1, c_2, \dots, c_N)} = \frac{P(c_1 | y_1) P(c_2 | y_1) \dots P(c_N | y_1) P(y_1)}{P(c_1 | y_2) P(c_2 | y_2) \dots P(c_N | y_2) P(y_2)}$$

Нехай

$c_1 = (c_{11}, c_{12}, c_{13}, c_{14}, c_{15})$  - змінна першого рівня архітектури, де

$c_{11}$  – кількість серверів (припустимо надалі, що всі сервери мають однакові характеристики), множина значень  $\{N_{11}, \text{де } N_{11} - \text{цілі додатні числа}\}$ ,

$c_{12}$  – кількість процесорів сервера, множина значень

$$\{1, 2, 4, 8\} = \{2^{N_{12}}, N_{12} = 0, 1, 2, 3\},$$

$c_{13}$  – кількість оперативної пам'яті сервера, множина значень

$$\{8, 16, 32, 64, 128, 256, 512\} = \{2^{N_{13}}, N_{13} = 3, 4, \dots, 9\},$$

$c_{14}$  - кількість дисків Western Digital (WD) або фізичні порти (SATA або M.2 на материнській платі комп'ютера), яка є обмеженою лише технічними характеристиками системи (материнської плати, NAS-сховища або RAID-контролера) та обсягом вільного простору, множина значень  $\{N_{14}, \text{де } N_{14} - \text{цілі додатні числа}\}$ ,

$c_{15}$  - кількість фізичних томів SSD, множина значень  $\{N_{15}$ , де  $N_{15}$  – цілі додатні числа};

$c_2$  - змінна другого рівня архітектури, яка відповідає за вибір операційної системи фізичних вузлів, множина значень

$\{Linux\ Ubuntu, MS\ Windows, Mac\ OS\ \};$

$c_3 = (inf_{31}, inf_{32}, inf_{33}, inf_{34}, inf_{35}, inf_{36})$  - змінна третього рівня архітектури, складовими якої є

$inf_{31}$  – інформація про вибір хмарної операційної системи, множина значень

$\{OpenStack, Microsoft\ Windows\ Azure, Google\ Chrome\ OS, X\ Window\ System, Amazon\ Web\ Services\ (AWS), ZeroPC, Jolicloud, Glide\ OS, SilveOS, iSpaces, \dots\} =$   
 $= \{N_{31}$ , де  $N_{31}$  – цілі додатні числа};

$inf_{32}$  – інформація про платність, множина значень  $\{0, 1\}$ ,

$inf_{33}$  – інформація про доступність, множина значень  $\{0, 1\}$ ,

$inf_{34}$  – інформація про інтегрування аккаунтів, множина значень  $\{0, 1\}$ ,

$inf_{35}$  – інформація про можливість синхронізації, множина значень  $\{0, 1\}$ ,

$inf_{36}$  – інформація про наявність оперативної гнучкості, множина значень  $\{0, 1\}$ ;

$c_4 = (R_{41}, R_{42})$  - змінна четвертого рівня архітектури (IaaS), де

$R_{41}$  – кількість віртуальних вузлів, множина значень  $\{N_{41}$ , де  $N_{41} = 0, 1, 2, \dots\}$ ,

$R_{42}$  – кількість віртуальних кластерів, множина значень  $\{N_{42}$ , де  $N_{42} = 0, 1, 2, \dots\}$ ;

$c_5$  - змінна п'ятого рівня архітектури, яка відповідає за вибір платформи як послуги (PaaS), множина значень

$\{Linux\ Ubuntu, \dots\} = \{N_{51}$ , де  $N_{51}$  – цілі додатні числа};

$c_6$  - змінна шостого рівня архітектури, яка відповідає за вибір операційної системи віртуальних вузлів, множина значень

$$\{DAAS, Framework, HaaS, ElasticCluster\} = \{N_{61}, \text{де } N_{61} - \text{цілі додатні числа}\};$$

$c_7 = (c_{71}, c_{72}, c_{73})$  - змінна сьомого рівня архітектури, складовими якої є

$c_{71}$  - кількість вебзастосунків, множина значень  $\{N_{71}, \text{де } N_{71} - \text{цілі додатні числа}\}$ ,

$c_{72}$  - кількість агентів-додатків для проведення масштабування, множина значень  $\{N_{72}, \text{де } N_{72} - \text{цілі додатні числа}\}$ ,

$c_{73}$  - кількість локальних застосунків, множина значень  $\{N_{73}, \text{де } N_{73} - \text{цілі додатні числа}\}$ .

Сформована навчальна вибірка з експертною оцінкою відповідних комбінацій змінних (табл. 1.3).

Таблиця 2.1 - Приклад сформованої навчальної вибірки для ХА

№	$c_1$	$c_2$	$c_3$	$c_4$	$c_5$	$c_6$	$c_7$	Клас $y$
1	(2, 8, 128, 4, 2)	Linux Ubuntu	(AWS, 1, 1, 1, 1)	(10, 2)	Linux Ubuntu	Framework	(5, 2, 10)	$y_1$
2	(1, 4, 32, 2, 1)	MS Windows	(Azure, 1, 1, 0, 1, 1)	(20, 1)	Linux Ubuntu	ElasticCluster	(15, 1, 5)	$y_2$
3	(4, 16, 512, 8, 4)	Linux Ubuntu	(OpenStack, 0, 1, 1, 1, 1)	(50, 5)	Linux Ubuntu	HaaS	(8, 4, 20)	$y_1$
4	(2, 2, 16, 2, 1)	Linux Ubuntu	(Google OS, 1, 1, 1, 1, 0)	(5, 1)	Linux Ubuntu	DAAS	(12, 1, 2)	$y_2$
5	(8, 8, 256, 12, 8)	Linux Ubuntu	(AWS, 1, 1, 1, 1, 1)	(100, 10)	Linux Ubuntu	Framework	(20, 10, 50)	$y_1$

Отримано результати роботи моделі на різних наборах вхідних параметрів.

## 2.2 Оптимізація хмарної архітектури

### 2.2.1 Оцінка ресурсів споживання та розміщення

Актуальним є завдання оцінювання обсягу ресурсів, необхідних для реалізації сценарію розгортання платформи хмарних обчислень [58, 82-86]). У дослідженні [50] додатково проаналізовано підходи до моделювання багаторівневої архітектури хмарних систем для розв'язання задачі горизонтального масштабування шляхом ітераційного нарощування кількості паралельних процесів із фіксацією часу виконання, збереженням динаміки навантаження на обчислювальний вузол та контролем використання оперативної пам'яті.

Ця математична модель описує динамічне керування парком віртуальних машин (VM) на фізичних серверах. Основна логіка полягає у розрахунку коефіцієнтів відповідності між запитами VM ( $c_4$ ) та фізичними можливостями серверів ( $c_1$ ). Логічна схема алгоритму (рис. 2.2) відображає процес оновлення стану активних VM на основі планових показників, видалення та додавання нових вузлів.

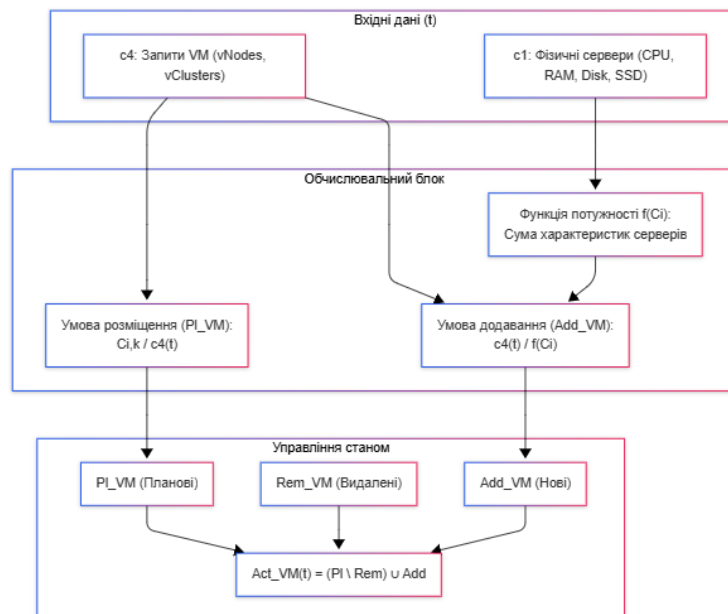


Рисунок 2.2 - Логічна схема процесу стану активних VM

У наших позначеннях припустимо, що використовується  $N_{11}$  серверів з визначеними характеристиками ( $c_{12}, c_{13}, c_{14}, c_{15}$ ).

Кожний сервер  $S_i, i = 1, 2, \dots, N_{11}$  характеризується для кожного ресурсу відомою ємністю  $C_{ik} = c_{1k}^{(i)}$  ( $i = 1, 2, \dots, N_{11}, k = 2, 3, \dots, 5$ ), яка є сукупністю ресурсів та граничних можливостей (обчислювальна потужність, пам'ять, мережева швидкість) для визначення кількості запитів/завдань для одночасного оброблення без втрати продуктивності.

У хмарній системі четвертий рівень (рис. 2.1) інфраструктури як послуги (IaaS) вимагає певну кількість ресурсів  $c_4^{(i)} = (R_{41}^{(i)}, R_{42}^{(i)})$  для обробки запитів/завдань з кожного серверу  $S_i, i = 1, 2, \dots, N_{11}$ .

При відсутності мінімальної кількості ресурсів на сервері віртуальна машина не може бути розгорнута.

Кількість ресурсів  $R_{4j}^{(i)}$  ( $i = 1, 2, \dots, N_{11}, j = 1, 2$ ) є статичним параметром. Запит/завдання є динамічним параметром, який визначає реальну потребу в ресурсах.

В кожний момент часу  $t$  активні віртуальні машини  $Act\_VM(t)$  розділяються на вже розміщені  $Pl\_VM(t)$  та ті, які необхідно розмістити  $Add\_VM(t)$  або є виключеними  $Rem\_VM(t)$ :

$$Act\_VM(t) = \{Pl\_VM(t) | Rem\_VM(t)\} \cup Add\_VM(t).$$

Розміщення віртуальної машини у хмарі характеризується двома змінними, які відповідають за споживання процесорного ресурсу (кількість віртуальних процесорів) та розміщення на обчислювальному сервері (зв'язок між обчислювальним сервером та віртуальною машиною):

$$Condition\_Pl\_VM(t) = Condition\_Pl\_VM(t, c_1, c_4)$$

Умова споживання процесорного ресурсу  $c_4^{(i)} = (R_{41}^{(i)}, R_{42}^{(i)})$  та розміщення на обчислювальному сервері  $S_i, i = 1, 2, \dots, N_{11}$  в момент часу  $t$  на обчислювальному сервері

$$Condition\_Pl\_VM(t, C_{ik}, c_4^{(i)}) = 1.$$

При припущенні, що СХМОбч достатньо велика для розміщення усіх запитів/завдань та для кожного типу ресурсів необхідно обов'язкове розміщення та сума усіх зарезервованих ресурсів не повинна перевищувати ємність сервера по заданому ресурсу, маємо що для процесорного ресурсу  $\alpha_i(t)$  умова така:

$$\sum_{i \in Act\_VM(t)} Condition\_Pl\_VM(t, C_{ik}, c_4^{(i)}) \cdot c_4^{(i)}(t) \leq C_{ik}, j = \overline{1, s} \quad (2.1)$$

Це фактично побудована оцінка ресурсів сервера, яка дає можливість надалі побудувати оцінку ресурсів хмарної системи.

Припустимо, що  $Condition\_Pl\_VM(t, C_{ik}, c_4^{(i)})$  це відсотковий еквівалент зайнятості вузлів ХА (ймовірність заповнення ресурсів сервера), яке може бути визначено в інший спосіб:

$$Condition\_Pl\_VM(t, C_{ik}, c_4^{(i)}) = \frac{C_{ik}}{c_4^{(i)}(t)}, i \in Pl\_VM(t) \quad (2.2)$$

Для розгортання додаткових серверних потужностей зазвичай застосовують класичний метод *Resize()*. Інтеграція цього підходу для масштабування кількості серверів передбачає встановлення граничного порогу для ініціації зазначеної процедури. Одним із критеріїв може служити сформульована вище умова контролю місткості (2.1). Застосування стандартної функції *Resize()* доцільним за припущення, що граничне значення для початку реконфігурації (зменшення або збільшення кількості пристроїв) становить 75%

$$(Condition\_Pl\_VM(t, C_{ik}, c_4^{(i)}) > 0.75).$$

Додавання порогу 0.75 перетворює математичну модель на систему прийняття рішень: якщо утилізація перевищує цей рівень, система ініціює створення нових віртуальних вузлів або розширення кластера.

Ефективність двох запропонованих підходів до контролю ресурсів під час моделювання істотно залежить від обраних структур даних, специфікації

їхніх типів та обраного середовища програмування для реалізації моделі хмарної системи. В цьому напрямі проведено багато наукових досліджень, опрацювання яких не входить в рамки цієї роботи.

Якщо розглянути зворотну задачу, де питання стоїть чи можна масштабувати ресурси  $c_4^{(j)} = (R_{41}^{(j)}, R_{42}^{(j)})$  для обробки запитів/завдань з кожного серверу  $S_i$ ,  $i = 1, 2, \dots, N_{11}$ :

$$Condition\_Add\_VM(t, c_{1k}, c_4^{(j)}) = \frac{c_4^{(j)}(t)}{f(C_i)}, \quad (2.3)$$

де  $k = 2, 3, \dots, 5$ ;  $j \in Act\_VM(t)$  і

$$f(C_i) = \sum_{k=1}^{N_{11}} \left( c_{1k}^{(i)} \cdot (c_{12}^{(i)} + c_{13}^{(i)} + c_{14}^{(i)} + c_{15}^{(i)}) \right).$$

Ця умова є умовою включення вимкнених та підготовлених ресурсів  $Rem\_VM(t)$  четвертого рівня, що надалі при повторному використанні цієї умови на множині ресурсів  $Pl\_VM(t) \cup Rem\_VM(t)$  отримати додаткові розгорнуті ресурс  $Add\_VM(t)$ .

Поєднання умов (2.2) та (2.3) з моделлю Байєса з детермінованим порогом (0.75) дозволяє створити адаптивний поріг. Замість жорсткого правила, система прийматиме рішення на основі ймовірності того, що поточний стан призведе до відмови (клас  $y_2$  - критичний). Це може означати, що навіть якщо навантаження 80%, але результати моделі Байєса повідомляють про короткочасний шум (стабільний стан), - тоді система не витратить кошти на нові  $VM$ . Модель з запропонованими умовами враховує не лише "скільки" ресурсів спожито, а й "як швидко" вони споживаються. При додаванні модулю звітності можна кількісно оцінити ефективність розробленої архітектури за трьома стратегіями: прямолінійне масштабування (поріг 0.75), масштабування лише за ймовірністю (модель класифікації Байєса) та інтегрована модель з функцією вартості.

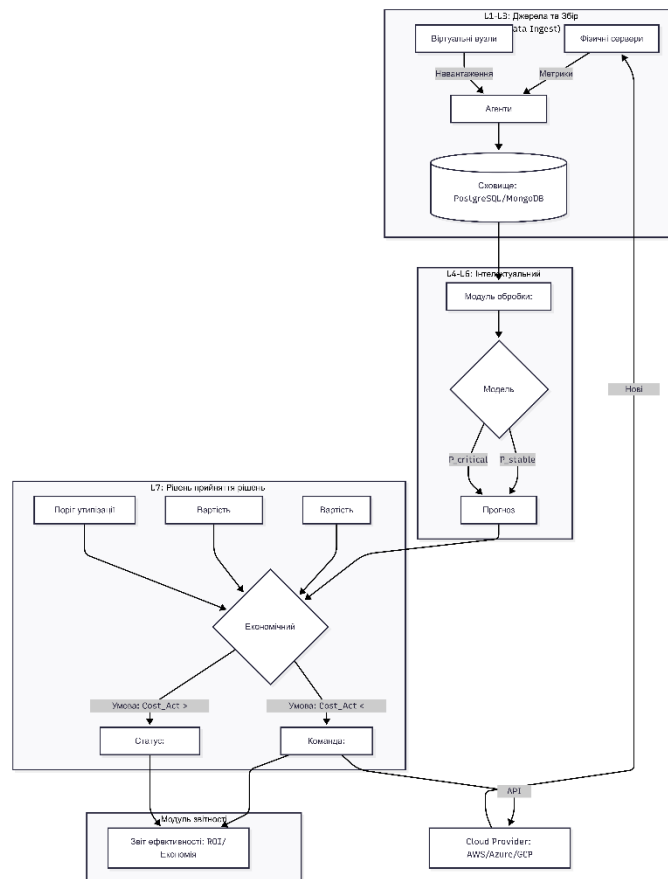


Рисунок 2.3 - Логічна схема архітектури при агрегації стратегій

Завдяки поєднанню цих стратегій отримано систему (рис. 2.3), яка працює на рівні L7 (додатки), але розуміє обмеження L1 (фізичні сервери), використовує алгоритми машинного навчання для фільтрації шумів, приймає рішення на основі бізнес-метрик, а не лише технічних показників.

## 2.2.2 Оцінка ресурсів хмари

Під час проектування хмарної інфраструктури в умовах жорсткого обмеження ресурсів застосовують методи прогнозування динаміки навантаження системи.

При невизначеній кількості користувачів проведений аналіз методів прогнозної оцінки, які реалізуються на основі отриманих даних хмарної системи.

### **2.2.3 Задача оптимального розподілу послуг між ресурсами рівнів хмарної архітектури**

Для визначення наявності достатнього обсягу ресурсів побудуємо модель для обробки запитів/завдань на обслуговування зовнішнього споживача СХМОбч (рис. 1.13(a)).

Для кожної послуги обсяг ресурсів може бути прописаний у відповідній угоді про рівень послуг між постачальником і споживачем про рівень послуг (Service-level agreement, SLA). Зазначена угода регламентує параметри доступності сервісів, технічної підтримки користувачів, а також нормативний час усунення несправностей. Ця інформація визначає кількісні та якісні характеристики надаваних послуг. Сформульовані в SLA умови зазвичай застосовуються всередині організації для оптимізації взаємодії між підрозділами. Водночас угода виступає базовим інструментом безперервного моніторингу й контролю якості сервісів. Наявні домовленості слугують основою для конфігурування апаратного й програмного забезпечення з метою максимізації пропускної здатності системи.

Ця угода регламентує метрики доступності, рівень підтримки користувачів, час усунення збоїв тощо. Зазначена інформація визначає кількісні та якісні характеристики надаваних послуг. Фіксовані в SLA умови зазвичай використовуються в організації для координації внутрішніх підрозділів. Проте документ є головним механізмом постійного моніторингу й контролю якості сервісів. Ці регламенти визначають параметри конфігурації програмно-апаратних засобів для забезпечення максимальної обчислювальної спроможності.

Базова статична модель оптимального розподілу запитів/завдань багаторівневої архітектури для семи рівнів СХМОбч представлено у роботі [50]. Розглянемо запропонований підхід для архітектури СХМОбч, яка має

довільну кількість рівнів  $l \in \{1, 2, \dots\}$ . Для визначених вище обмежень щодо кількості ресурсів додамо поняття швидкодії роботи цих ресурсів.

Загальний обсяг ресурсів СХМОбч дорівнює сумі обсягів ресурсів кожного рівня хмарної системи.

Нехай

$$X = (x_1, x_2, \dots, x_l),$$

де  $x_j$  — кількість запитів/завдань на кожному з рівнів побудованої архітектури системи ( $j = 1, 2, \dots, l$ ).

Визначимо доступні обсяги ресурсів СХМОбч через змінну першого рівня раніше визначеної архітектури інструментарію  $C_{ik} = c_{1k}^{(i)}$  ( $i = 1, 2, \dots, N_{11}, k = 2, 3, \dots, 5$ ):

$$Capacity_j(t) = \sum_{i=1}^{N_{11}} \left( c_{11}^{(i)} \cdot \left( c_{12}^{(i)} + c_{13}^{(i)} + c_{14}^{(i)} + c_{15}^{(i)} \right) \right) - \text{обсяги ресурсів,}$$

які використовуються;

$v_j(t) = \sum_{i=1}^{N_{11}} v_{ik}(t)$  - швидкість мережевої взаємодії для  $j$ -го рівня та його складових ( $i = 1, 2, \dots, N_{11}, k = 2, 3, \dots, 5$ ).

Обсяг споживання ресурсів окремого  $s$  - запиту/завдання визначимо як

$$Amount_{js} = \{Capacity_{js}(t), v_{js}(t)\},$$

де  $j$  - номер відповідного рівня,  $s$  - номер запиту/завдання на відповідному рівні.

Отже, обсяг ресурсів, які є задіяними послугами на запитів/завдань на всіх рівнях архітектури:

$$Capacity(t) = \sum_{i=1}^l f(C_i),$$

$$v(t) = \sum_{i=1}^l \sum_{s=1}^{\sum_{k=1}^l x_k} v_{js}(t),$$

$$Amount = \{Capacity(t), v(t)\}.$$

У кожен момент часу поточний стан системи є окремим випадком, а сама конфігурація динамічно змінюється протягом певного часового інтервалу.

Для кожного запиту споживача потрібно послідовно виконувати перевірку наявності ресурсів, що приводить до збільшення обчислювальної складності.

Позитивним вважається запит, за якого сумарний обсяг ресурсів, необхідний для функціонування нових та наявних сервісів, не перевищує обсяг ресурсів платформи хмарних обчислень:

$$\begin{cases} Capacity_s(t) \leq Capacity(t) \\ v_s(t) \leq v(t) \end{cases} \quad (2.4)$$

При умові невиконання умови генерується відповідь про відмову замовлення послуги. При визначенні попередньої умови можна не відмовляти у послугі, а надати технологію масштабування рівнів споживання та розміщення (рис. 2.4).

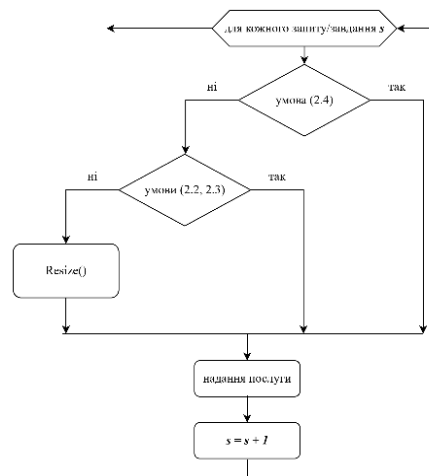


Рисунок 2.4 - Блок-схема спрацьовування умов розміщення та споживання послуг

Система при такій модифікації умов перевіряє кількість ресурсів для стабільної роботи, а й оцінює ймовірність критичного збою та економічну доцільність запуску нового запиту/завдання.

## 2.3 Моделі хмарних систем

### 2.3.1 Хмарна система як система масового обслуговування

Стохастичні моделі, до яких належать марківські моделі та моделі систем масового обслуговування визначені ймовірнісними відношеннями з нескінченною множиною станів:

$$P(Z_s = Z_{sj} | Z_m = Z_{mj}) \leq 1$$

$$P(Z_m = Z_{mj} | Z_s = Z_{sj}) \leq 1,$$

де  $Z_{sj}$ ,  $Z_{mj}$  – дискретно визначені стани системи та її моделі, задані на скінченній множині можливих значень  $i, j$ .

*Система масового обслуговування* – це одне або декілька обслуговуючих пристроїв з чергою.

*Мережа масового обслуговування (МО)* – це сукупність взаємозалежних систем масового обслуговування (СМО), у якій об'єкти, що обслуговуються, з визначеною ймовірністю переходять з однієї СМО в іншу.

У незамкненій мережі МО вимоги надходять ззовні мережі і після обробки залишають її.

У замкненій мережі МО деяка кількість вимог весь час знаходиться в ній, переходячи з однієї СМО до іншої, але ніколи не залишаючи мережу МО.

Для позначення моделей СМО широко застосовують нотацію, запропоновану Кендаллом:

$$X/Y/Z,$$

де  $X$  - розподіл часу прибуття надходження вимог,

$Y$  - розподіл часу обслуговування,

$Z$  - кількість пристроїв для обслуговування.

Якщо інформації про систему недостатньо, її опис подають у вигляді моделі типу  $G/G/m$  - моделі з довільними розподілами ймовірностей випадкових величин та  $m$  пристроями. У теорії масового обслуговування

аналітичні результати отримано лише для окремих класів таких моделей  $D/D/1$ ,  $M/M/1$  і  $M/G/1$ , де  $M$  позначає, що процеси розподілу часу є марковськими (час має експоненціальний розподіл).

У класичній теорії масового обслуговування, як правило, розглядається пуассоновський (найпростіший) потік вимог, в якому кількість вимог  $k$  для будь-якого проміжку часу  $t$  має розподіл Пуассона:

$$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}, k \geq 0, t \geq 0,$$

де  $\lambda$  - інтенсивність потоку вимог (кількість вимог, які надійшли до системи за одиницю часу).

Основними характеристиками найпростішого потоку вимог є такі:

1) Стационарність – ймовірність надходження певної кількості вимог протягом інтервалу тривалості  $\tau$  залежить лише від довжини цього інтервалу та інтенсивності  $\lambda$ , і не залежить від його розташування на часовій осі. При цьому часові відрізки, для яких розглядаються ймовірності, не повинні перетинатися.

2) Відсутність післядії – надходження вимог у поточному інтервалі часу не залежить від кількості вимог, що надійшли раніше; тобто потік не має пам'яті й не визначається передісторією процесу.

3) Ординарність – у будь-який момент часу ймовірність появи двох або більше вимог одночасно дорівнює нулю, тобто вимоги надходять поодиночі.

Спочатку було розглянуто хмарну систему як СМО з припущеннями стаціонарності, відсутності післядій та властивістю ординарності.

При визначенні

$request_{mean}$  – середня кількість запитів/завдань, які знаходяться в системі,

$\lambda$  – інтенсивність надходження запиту/завдання (середня кількість запитів, що надходять в СМО за одиницю часу,  $\lambda = 1/t_{krok}$ ),

$T$  – середній час перебування запитів/завдань (вимог) в системі,

$\mu$  - інтенсивність потоку обслуговування (інтенсивність вихідного потоку запитів/завдань,  $\mu = 1/t_{обсл}$ ),

$\rho$  – завантаження (середня кількість запитів/завдань, що приходять за середній час обслуговування одного запиту/завдання,  $\rho = \lambda/\mu$ ),

закон збереження стаціонарної черги (формула Литтла) обчислює середню кількість запитів/завдань (вимог), що знаходяться в системі:

$$request_{mean} = \lambda T_{mean}.$$

Якщо розглядати хмарну систему, як СМО з стаціонарним потоком, тоді  $\lambda$  постійна ( $t_{крок} = const, t_{обсл} = const$ ).

Використання опису та процесів СМО (рис. 2.3) для хмарної системи може надати можливість зосередитися часі очікування запитів/завдань в черзі (при її наявності або про можливості її створення) [87-89]. Наприклад, якщо обирати задачу кібербезпеки, тоді побудова моделі зводиться до аналізу часу між інцидентами безпеки.

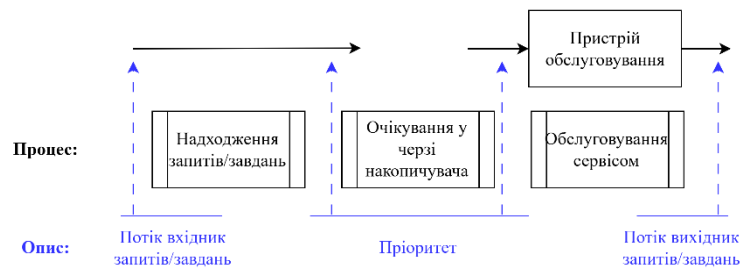


Рисунок 2.5 - Узагальнена схема процесів СМО

Розподіл Пуассона описує **кількість подій** за фіксований проміжок часу, тоді як експоненціальний розподіл описує **час між послідовними** цими подіями. Ці розподіли є пов'язаними, якщо події відбуваються за пуассонівським процесом з інтенсивністю  $\lambda$ , то час між ними підпорядковується експоненціальному розподілу з параметром  $\lambda$ .

Стаціонарний ординарний потік без наслідків є потоком з розподілом Пуассона:

$$P_n(h) = \frac{(\lambda h)^n e^{-\lambda h}}{n!} - \text{ймовірність появи } n \text{ подій на проміжку часу } h \text{ з}$$

інтенсивністю  $\lambda$ , щільність розподілу  $f(t) = \lambda e^{-\lambda t}$ , математичне сподівання

$M(h) = \lambda h$ , дисперсія  $D(h) = \lambda h$ . Розподіл Пуассона моделює кількість подій, що відбуваються протягом певного часу або в певній області, коли ці події відбуваються незалежно і з постійною середньою частотою, де параметр  $\lambda$  є середньою кількістю подій за одиницю часу/об'єму.

Проміжки часу мають експоненціальний закон розподілу:  $F(t) = 1 - e^{-\lambda t}$ , де щільність розподілу обчислюється як  $f(t) = \lambda e^{-\lambda t}$  при  $t > 0$ . Експоненціальний розподіл моделює час, що минув між двома послідовними подіями в пуассонівському процесі, або час до настання першої події, де параметр  $\lambda$  є інтенсивністю процесу.

Розподіли Пуассона та експоненціальний є «дзеркальним відображенням» одного й того ж процесу: Пуассон рахує події, експонента — інтервали між ними.

Розподіл Пуассона застосовують для опису дискретної кількості подій: за достатньо тривалий період спостережень можна стверджувати, що у середньому надходить 10 запитів/завдань за хвилину, отже інтенсивність потоку становить  $\lambda = \frac{10}{60}$  зап./хв. Та середня довжина інтервалу часу між окремими запитами/завданнями становить  $\mu = \frac{1}{\lambda} = 6$  хв.).

Експоненціальний розклад можна використовувати для неперервного часу: середній інтервал часу між послідовним надходженням подій становить 6 хвилин та інтенсивність потоку становить  $\lambda = \frac{10}{60}$  зап./хв.

Ймовірність того, що за час  $T_{interval} = h = 2$  хв. на СМО надійде  $X = \{0, 1, \dots\}$  запитів/завдань з інтенсивністю  $\lambda = \frac{10}{60}$  зап./хв. (потік є простим):

$$\lambda h = \lambda T_{interval} = \frac{10}{60} \cdot 2 = \frac{1}{3}$$

$$P_0(h) = P(X = 0) = \frac{\left(\frac{1}{3}\right)^0 e^{-\frac{1}{3}}}{0!} \approx 0.718$$

$$P_1(h) = P(X = 1) = \frac{\left(\frac{1}{3}\right)^1 e^{-\frac{1}{3}}}{1!} \approx 0.239$$

Розрізняють замкнені та розімкнуті СМО, де множина запитів/завдань, що може вироблятися генератором при моделюванні, є нескінченною та скінченною відповідно.

Генерацію запитів зазвичай розглядають як потік однорідних подій, що з'являються через випадкові інтервали часу за певними законами розподілу. Далі ми дослідимо простий потік замовлень, де часові інтервали між подіями підпорядковані експоненціальному закону розподілу:

$$f(t) = \lambda e^{-\lambda t}, \quad t = t_{i+1} - t_i$$

Появу кількості  $x$  запитів/завдань за одиницю часу можна описати ймовірністю розподілу Пуасона з інтенсивністю  $\lambda$  (середньою кількістю замовлень за одиницю часу):

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Розробленню модулів симулятора хмарної системи для порівняння стратегій масштабування, присвячено багато наукових публікацій [90-92] та технічних рішень [93]. Для розробки зазначеного симулятора необхідно формалізувати критерії оцінювання продуктивності хмарної системи, що дозволить розрахувати такі параметри, як середній час обробки мережевих запитів і коефіцієнт завантаження процесорних та ресурсів пам'яті.

При загальному підході імітації роботи хмарної системи генерація запитів/завдань в загальному випадку може бути розглянута як потік однорідних подій, що виникають через випадкові проміжки часу і підпорядковані різним законам розподілу.

Надалі розглянемо простий потік запитів/завдань, у якому часові інтервали між подіями описані експоненціальним законом розподілу.

У роботі [94] запропоноване архітектурне рішення хмарної системи з трьома рівнями: кінцеві пристрої (DL), граничні сервери (EL) та хмарний центр (CL).

### 2.3.2 Моделі чотирирівневої СХМОбч на прикладах найпростішого потоку

Для більш реалістичного представлення приділена увага побудові моделей СХМОбч з доданим рівнем туману (FL), ідея якої представлена у роботі [95].

Метою наступного кроку дослідження є проведення аналізу параметрів моделі хмарної системи, де визначені ймовірності переходів запитів/завдань між підсистемами, що приводить багатофазову систему до стохастичної мережі.

Розглянемо формалізовану модель СХМОбч (спрощений варіант представлений на рис. 2.5), у якої чотири архітектурні рівні DL, EL, FL та CL, на яких визначено кількість кінцевих користувачів  $U = \{u_1, u_2, \dots, u_N\}$ , граничних серверів  $S = \{s_1, s_2, \dots, s_{ES}\}$ , туманних вузлів  $R = \{r_1, r_2, \dots, r_{FR}\}$  та віртуальних машин  $VM = \{VM_1, VM_2, \dots, VM_V\}$ .

Припустимо, що визначені ймовірності переходів запитів/завдань між підсистемами СМО, що приводить багатофазову систему до стохастичної мережі, де наявні можливості вбудовування блоків для вибору альтернативи клієнтом (надалі можуть бути розглянуті для дослідження навантаження на систему).

Для вирішення задачі горизонтального масштабування в термінології СМО можна розглянути питання щодо мінімальної кількості пристроїв та каналів обслуговування на кожному рівні, при якому можливий сталий режим незамкненої мережі масового обслуговування:

$$k_i > \lambda_0 \cdot \frac{e_i}{\mu_i}, \quad i = 1, \dots, n$$

де  $\lambda_0$  – вхідний потік запитів/завдань,  $e_i$  - коефіцієнт передачі (доля вхідного потоку запитів/завдань до СМО<sub>*i*</sub>),  $k_i$  – кількість каналів обслуговування.

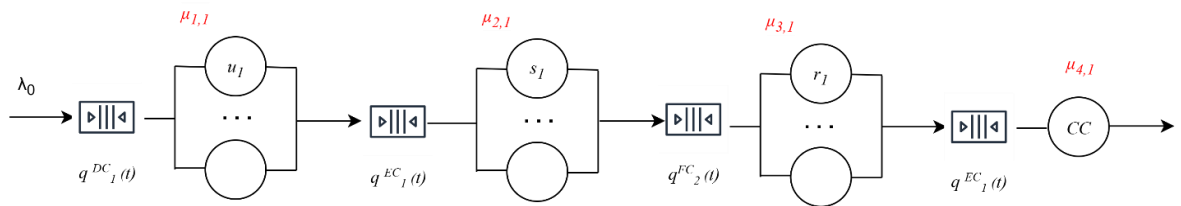


Рисунок 2.6 - Загальна схема СХМОбч в термінології найпростішого потоку

Число  $e_i$  є середньою кількістю разів, коли вимога надійшла та залишила мережу:

$$e_i = p_{0i} + \sum_{j=1}^n p_{ji} \cdot \lambda_j, \quad i = 1, \dots, n$$

$$e_1 = 1,$$

$$e_2 = e_1 = 1$$

$$e_3 = e_2 = 1$$

$$e_4 = e_3 = 1$$

Кількість каналів на кожному рівні СМО<sub>*i*</sub> системи залежить від інтенсивностей вхідного  $\lambda_0$  та вихідних  $\mu_i$  потоків запитів/завдань:

$$k_i > \frac{\lambda_0}{\mu_i}$$

Отже, горизонтальне масштабування суттєво залежить від властивостей пристроїв, які є задіяними в побудові архітектури хмарної системи.

Для візуалізації формалізованої моделі обираємо  $N = 1, ES = 2, FR = 3, V = 1$  (рис. 2.5).

Організація вибору: Розімкнена СМО

Характер утворення черги: без очікування (з очікуванням: обмеження черги на довжину/на перебування в черзі; з/без пріоритету за правилом відбору та характеристикою пріоритету відповідно).

Кількість каналів: багатоканальна.

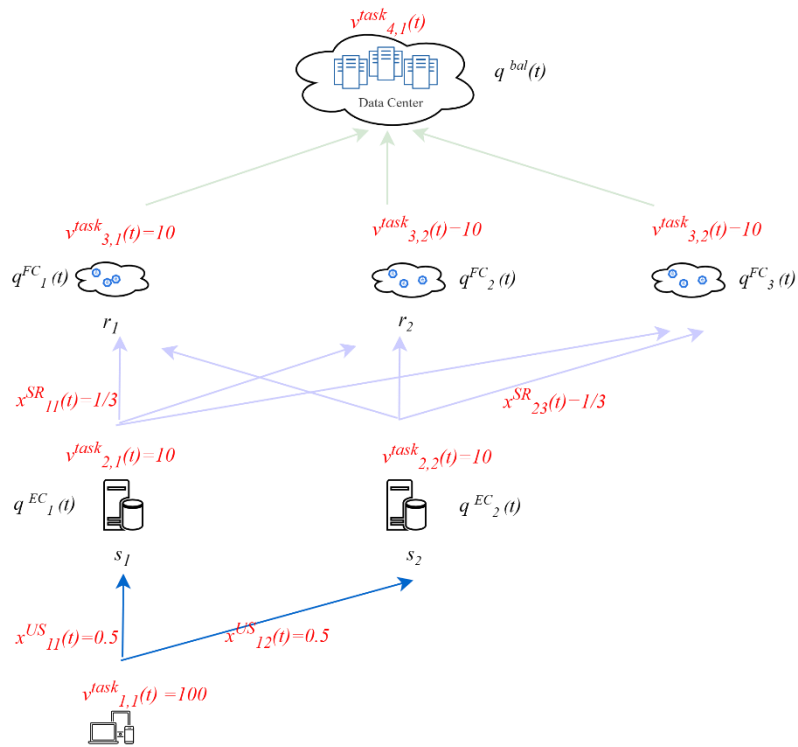


Рисунок 2.7 - Формалізована модель хмарної системи СХМОбч ( $N = 1, ES = 2, FR = 3, V = 1$ )

Метою моделювання є визначення коефіцієнта завантаження обчислювальних вузлів кожного архітектурного рівня, середньої довжини черг заявок, а також ймовірності відмови в обробці запиту/завдання.

### Модель системи: СХМОбч 1

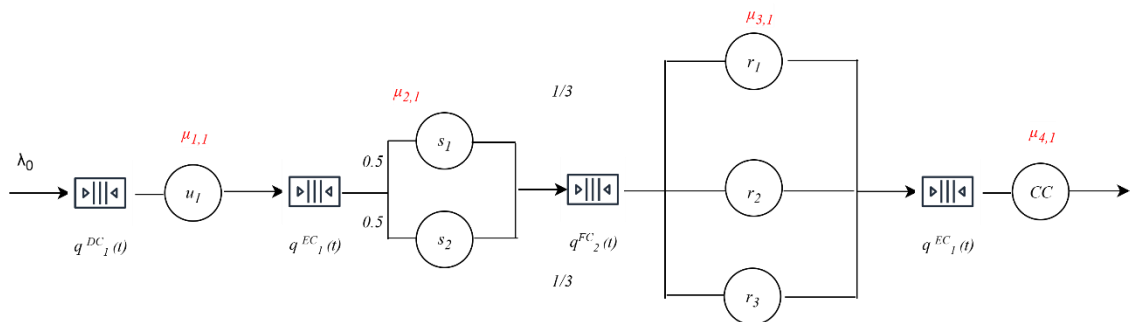


Рисунок 2.8 - Модель системи СХМОбч 1

Припустимо, що умова сталого режиму для системи зберігається:

$$\lambda_0 < k_1 \cdot \mu_{1,1} \text{ (при } k_1 = 1 \text{ виконується нерівність } \lambda_0 < \mu_{1,1})$$

$$\lambda_0 < k_2 \cdot \mu_{2,1} \text{ (при } k_2 = 2 \text{ виконується нерівність } \lambda_0 < 2\mu_{2,1}\text{)}$$

$$\lambda_0 < k_3 \cdot \mu_{3,1} \text{ (при } k_3 = 3 \text{ виконується нерівність } \lambda_0 < 3\mu_{3,1}\text{)}$$

$$\lambda_0 < k_4 \cdot \mu_{4,1} \text{ (при } k_4 = 1 \text{ виконується нерівність } \lambda_0 < \mu_{4,1}\text{)}$$

Середня кількість зайнятих пристроїв у кожній підсистемі при такій архітектурі для будь-якого  $i$  дорівнює:

$$R_i = \frac{e_i \cdot \lambda_0}{\mu_{i,1}} < \frac{1 \cdot \mu_{i,1} \cdot k_i}{\mu_{i,1}} = k_i,$$

$$R_i < k_i$$

що підтверджує нормальну роботу систему без перенавантаження.

Сумуючи вимоги сталого режиму для кожної підсистеми маємо:

$$\lambda_0 < \frac{k_1 \cdot \mu_{1,1} + k_2 \cdot \mu_{2,1} + k_3 \cdot \mu_{3,1} + k_4 \cdot \mu_{4,1}}{4},$$

для наведеного прикладу:

$$\lambda_0 < \frac{1}{4} (\mu_{1,1} + 2\mu_{2,1} + 3\mu_{3,1} + \mu_{4,1}).$$

### Модель системи: СХМОбч 2

Розглянемо хмарну систему з трьома різними підсистемами на третьому рівні:

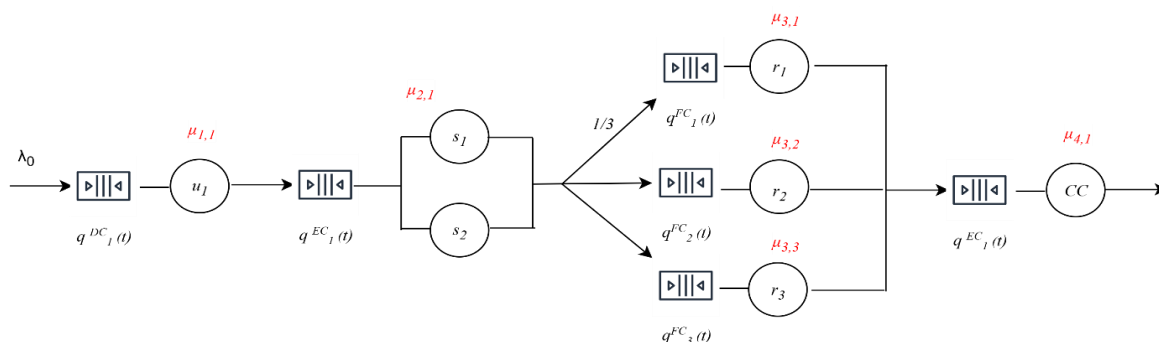


Рисунок 2.9 - Модель системи СХМОбч 2

Коефіцієнти передачі  $e_i$  є розв'язками системи рівнянь:

$$\begin{cases} e_1 = 1 \\ e_2 = e_1 = 1 \\ e_{3,1} = 1/3 \\ e_{3,2} = 1/3 \\ e_{3,4} = 1/3 \\ e_4 = 1 \end{cases}$$

При припущенні того, що умова сталого режиму для побудованої системи зберігається:

$$\lambda_0 < k_1 \cdot \mu_{1,1} \text{ (при } k_1 = 1 \text{ виконується нерівність } \lambda_0 < \mu_{1,1})$$

$$\lambda_0 < k_2 \cdot \mu_{2,1} \text{ (при } k_2 = 2 \text{ виконується нерівність } \lambda_0 < 2\mu_{2,1})$$

$$\lambda_0 < \frac{k_{3,l} \cdot \mu_{3,l}}{e_{3,l}} \text{ (при } k_{3,l} = 1 \text{ виконується нерівність } \lambda_0 < \mu_{3,l}), j = 1, 2, 3$$

$$\lambda_0 < k_4 \cdot \mu_{4,1} \text{ (при } k_4 = 1 \text{ виконується нерівність } \lambda_0 < \mu_{4,1})$$

Середня кількість зайнятих пристроїв у кожній підсистемі для будь-якого  $m$  дорівнює відповідно:

$$R_m = \frac{e_m \cdot \lambda_0}{\mu_{m,l}} < \frac{e_m \cdot \mu_{m,l} \cdot k_m}{\mu_{m,l}} = e_m \cdot k_m.$$

Сумуючи вимоги сталого режиму для кожної підсистеми маємо:

$$\lambda_0 < \frac{k_1 \cdot \mu_{1,1} + k_2 \cdot \mu_{2,1} + \sum_{l=1}^3 \frac{k_{3,l} \cdot \mu_{3,l}}{e_{3,l}} + k_4 \cdot \mu_{4,1}}{1 + 1 + (1 + 1 + 1) + 1}.$$

### Модель системи: СХмОбч 3

Для хмарної системи з двома та трьома різними підсистемами на другому та третьому рівнях відповідно

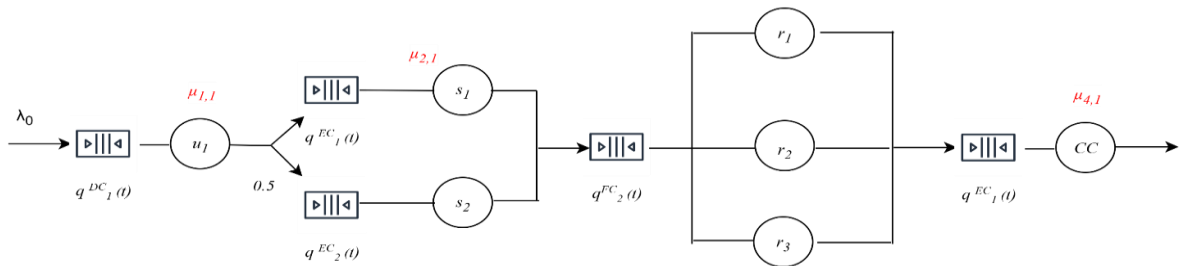


Рисунок 2.10 - Модель системи СХмОбч 3

маємо коефіцієнти передачі

$$\begin{cases} e_1 = 1 \\ e_{2,1} = 1/2 \\ e_{2,2} = 1/2 \\ e_3 = 1 \\ e_4 = 1 \end{cases}$$

$$\lambda_0 < k_1 \cdot \mu_{1,1} \text{ (при } k_1 = 1 \text{ виконується нерівність } \lambda_0 < \mu_{1,1}\text{)}$$

$$\lambda_0 < \frac{k_{2,j} \cdot \mu_{2,j}}{e_{2,j}} \text{ (при } k_{2,j} = 1 \text{ виконується нерівність } \lambda_0 < \mu_{2,j}\text{), } j = 1,2$$

$$\lambda_0 < k_3 \cdot \mu_{3,1} \text{ (при } k_{3,1} = 1 \text{ виконується нерівність } \lambda_0 < 3\mu_{3,1}\text{),}$$

$$\lambda_0 < k_4 \cdot \mu_{4,1} \text{ (при } k_4 = 1 \text{ виконується нерівність } \lambda_0 < \mu_{4,1}\text{)}$$

та умову на вхідний потік:

$$\lambda_0 < \frac{k_1 \cdot \mu_{1,1} + \sum_{j=1}^2 \frac{k_{2,j} \cdot \mu_{2,j}}{e_{2,j}} + \sum_{l=1}^3 \frac{k_{3,l} \cdot \mu_{3,l}}{e_{3,l}} + k_4 \cdot \mu_{4,1}}{1 + (1 + 1) + 3 + 1}$$

### Модель системи: СХМОбч 4

Для хмарної системи, яка представлена у вигляді СМО з перерозподілом потоків запитів/завдань

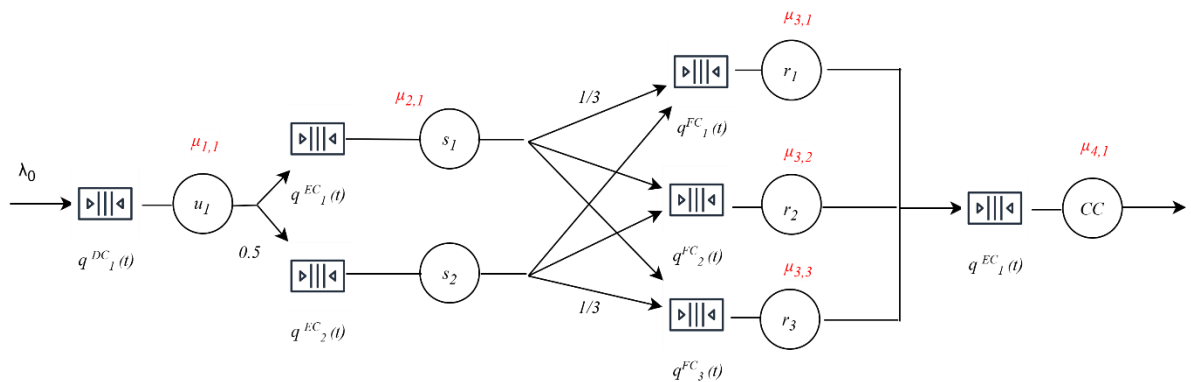


Рисунок 2.11 - Модель системи СХМОбч 4

коефіцієнти передачі розраховуються з системи (на рисунку перерозподіл представлений для третього рівня  $l = 3$ , представлення якого може бути узагальненим на інші рівні хмарної системи):

$$\left\{ \begin{array}{l} e_{1i} = p_{0i} \\ e_{2j} = \sum_{i=1}^N p_{1ij} \cdot e_{1i} \\ e_{3l} = \sum_{j=1}^{ES} p_{2jl} \cdot e_{2j} \\ e_4 = 1 \\ i = 1, \dots, N; j = 1, \dots, ES; l = 1, \dots, FR \end{array} \right.$$

Тоді обмеження на вхідний потік має вигляд:

$$\lambda_0 < \frac{\sum_{i=1}^N \frac{k_{1,i} \cdot \mu_{1,i}}{p_{0i}} + \sum_{j=1}^{ES} \frac{k_{2,j} \cdot \mu_{2,j}}{\sum_{i=1}^N p_{1ij} \cdot e_{1i}} + \sum_{l=1}^{FR} \frac{k_{3,l} \cdot \mu_{3,l}}{\sum_{j=1}^{ES} p_{2jl} \cdot e_{2j}} + k_4 \cdot \mu_{4,1}}{N + ES + FR + 1}$$

### **Модель системи: СХмОбч 5**

Для хмарної системи з чотирма рівнями, де на кожному знаходяться різні за характеристиками підсистеми обслуговування запитів/завдань виконується умова (можна довести за методом математичної індукції):

$$\lambda_0 < \frac{\sum_{i=1}^N \frac{k_{1,i} \cdot \mu_{1,i}}{e_{1,i}} + \sum_{j=1}^{ES} \frac{k_{2,j} \cdot \mu_{2,j}}{e_{2,j}} + \sum_{l=1}^{FR} \frac{k_{3,l} \cdot \mu_{3,l}}{e_{3,l}} + k_4 \cdot \mu_{4,1}}{N + ES + FR + 1},$$

(2.5)

де коефіцієнти передачі обчислюються з системи рівнянь:

$$\left\{ \begin{array}{l} e_{1i} = p_{0i}, i = 1, \dots, N \\ e_{2j} = p_{1j}, j = 1, \dots, ES \\ e_{3l} = p_{2l}, l = 1, \dots, FR \\ e_4 = 1 \end{array} \right.$$

Отже, при збільшенні вхідного потоку для збереження сталого режиму обов'язковим є динамічне збільшення кількості каналів обслуговування та/або заміна підсистем на більш потужні за інтенсивністю опрацювання запитів/завдань на кожному рівні хмарної системи, що призводить до задачі

горизонтального масштабування. Для підтримки стаціонарного режиму роботи системи доцільно застосувати вертикальне масштабування, що передбачає впровадження допоміжних програмних компонентів з метою деескалації навантаження на хмарну інфраструктуру.

## **2.4 Методи дослідження складності системи**

Для оптимізації продуктивності локальної хмарної інфраструктури IaaS передбачено можливість вертикального масштабування компонентів окремого вузла, інтеграцію допоміжних рівнів координації, а також релокацію мережевих сховищ даних. Процес взаємодії в системі базується на низхідному проходженні користувачьких запитів та команд крізь архітектурні шари для їх виконання, після чого результати та сервісні звіти повертаються споживачеві у вихідну точку.

Особливості функціонування сучасних хмарних систем в аспектах менеджменту ресурсів та утилізації потужностей визначають необхідність формування узагальненої структурно-функціональної схеми управління..

Стан побудованої системи можна виміряти за великою кількістю параметрів, наприклад, за вимірами ресурсів, навантажень, динаміки навантажень та інше [96].

У термінології теорії масового обслуговування об'єкти, що обслуговуються і з певною ймовірністю мігрують між вузлами, визначаються як вимоги (запити/завдання). Для адаптації до предметної області дослідження поняття «вимога» та «послуга» інтерпретуються як синоніми, а представлену логічну архітектуру (рис. 1.13) відображено у вигляді мережі масового обслуговування з лімітованими ресурсами та можливістю формування черг.

Ця інфраструктура моделюється як розімкнена стохастична мережа, де сервісні запити перерозподіляються між внутрішніми ресурсами або виходять за її межі.

Під час послідовного проходження елементів мережі кожна вхідна вимога спрямовується на один із вільних паралельних каналів обслуговування. У разі повної утилізації всіх ресурсів поточного вузла запити утворюють чергу. За такого сценарію існує ймовірність відмови в обслуговуванні, внаслідок чого запит залишає мережу необробленим.

При проходженні по порядку усіх СМО при вході на кожну знаходить один Вимоги з паралельно функціонуючих ресурсів, за умови незанятості їх. При зайнятості усіх ресурсів формування послуги може створитися черга запитів. При такому підході послуга може бути не обробленою та може покинути мережу.

Для обчислення основних показників ефективності функціонування незамкненої мережі масового обслуговування використовуємо умову незалежності підсистем та формули роботи [97] для обчислення ймовірності

$$p(x_{11}, x_{12}, \dots, x_{1N}, x_{21}, x_{22}, \dots, x_{2,ES}, x_{31}, x_{32}, \dots, x_{3,FR}, x_{41}) = \prod_{m=1}^{N+ES+FR+1} p_m(x_m)$$

$$p_m(x) = \left(\frac{e_m \lambda_0}{\mu_m}\right)^x \cdot C_m \cdot \begin{cases} \frac{1}{x!} & \text{при } x \leq k_m \\ \frac{1}{k_m! k_m^{x-k_m}} & \text{при } x > k_m \end{cases}$$

та нормуючого множника

$$C_m = \left( \left(\frac{e_m \lambda_0}{\mu_m}\right)^{k_m} \cdot \frac{1}{k_m! \left(1 - \frac{e_m \lambda_0}{\mu_m k_m}\right)} + \sum_{n=0}^{k_m-1} \left(\frac{e_m \lambda_0}{\mu_m}\right)^n \cdot \frac{1}{k_m!} \right)^{-1}$$

де  $x_m$  – кількість вимог, що знаходиться в підсистемі СМО<sub>*i*</sub>,

$$e_m = \{e_{1i}, e_{2j}, e_{3l}, i = 1, \dots, N; j = 1, \dots, ES; l = 1, \dots, FR\},$$

$$\mu_m = \{\mu_{1i}, \mu_{2j}, \mu_{3l}, i = 1, \dots, N; j = 1, \dots, ES; l = 1, \dots, FR\},$$

$$k_m = \{k_{1i}, k_{2j}, k_{3l}, i = 1, \dots, N; j = 1, \dots, ES; l = 1, \dots, FR\}.$$

Середня довжина черги запитів/завдань підсистем систем СХМОбч N ( $N = \overline{1,5}$ ) для будь-якого  $m$  обчислюється, як

$$L_m = \sum_{n=k_m+1}^{\infty} (n - k_m) p_m(n)$$

Середня кількість запитів/завдань, що перебувають у підсистемах систем СХМОбч N ( $N = \overline{1,5}$ ) для будь-якого  $m$  є величиною:

$$M_m = L_m + R_m$$

Середній час очікування в підсистемах дорівнює

$$Q_m = \frac{L_m}{e_m \cdot \lambda_0}$$

Середній час перебування запитів/завдань в підсистемах дорівнює

$$T_m = \frac{M_m}{e_m \cdot \lambda_0}$$

Середній час перебування запитів/завдань у мережі масового обслуговування хмарної системи дорівнює:

$$T = \sum_{m=1}^{N+ES+FR} e_m \cdot T_m$$

## 2.5 Показник залежності від часу

Але в реальності хмарна система не має постійної інтенсивності і можна розглянути цей параметр, наприклад як показник залежності від часу:

$$\lambda = \lambda(t)$$

Цікавим є питання визначення закону розподілу  $F_{identif}(t)$  проміжків часу на реальних даних навантаження системи, де щільність розподілу обчислюється як  $f_{identif}(t)$  при  $t > 0$ . Ідентифікований розподіл зможе

моделювати час, що минув між двома послідовними подіями в імітаційному процесі, або час до настання першої події, де параметр  $\lambda(t)$  є інтенсивністю потоку та  $\mu(t)$  є середньою довжиною інтервалу часу між окремими запитами/завданнями.

Якщо припустити, що розподіл проміжків часу на реальних даних періодів навантаження хмарної системи на кожному рівні є однаковим і визначається як  $F_{identif}(t)$  та є подібним до експоненціального розподілу  $F(t) = 1 - e^{-\lambda t}$ , тоді

$$\begin{aligned} F_{identif}(t) &= F(t) \\ F_{identif}(t) &= 1 - e^{-\lambda t} \\ \lambda(t) &= -\frac{1}{t} \ln(1 - F_{identif}(t)) \end{aligned} \quad (2.6)$$

Надалі роботу хмарної системи можна розглянути у різних випадках:

- при  $\lambda_0 = 1$  в момент часу  $t$ ;
- при  $\lambda_0 = \lambda(t)$  в момент часу  $t$  (при імітації в момент часу  $t$  приходить різна кількість запитів/завдань), що може привести до суттєвого збільшення обчислювальної складності.

Динамічний характер навантаження на хмарну інфраструктуру, інтенсифікація використання CPU й RAM та необхідність підтримки заданого рівня доступності сервісів актуалізують завдання раціонального розподілу ресурсів.

Перспективним напрямом розвитку цієї інфраструктури є інтеграція інтелектуальних агентів, які здійснюють безперервний моніторинг, керування та адаптацію ресурсного потенціалу хмарної системи в режимі реального часу. Функціонування таких агентів базується на взаємодії між собою та з  $N$  центральними оркестраторами, що гарантує автономне масштабування та самовідновлення (self-healing) систем. Синергія багатоагентних технологій та методів машинного навчання (Machine Learning, ML) дозволяє розробляти предиктивні моделі для проактивного менеджменту — тобто прогнозування

цільового попиту на обчислювальні потужності та їх попереднього оптимального розподілу [98].

## **Висновки до розділу 2**

В даному розділі проведено ґрунтовне дослідження архітектури системи хмарних обчислень на основі запропонованої у розділі 1 масштабованої ієрархії хмарної системи з включенням рівнів Edge/Fog для розгляду надійності хмарної системи та її хмарної оптимізації взаємозв'язку.

Детально розглянуто модель Байєса для семирівневої хмарної системи, яка побудована на основі використаних інструментарію/технологій для вирішення задачі вибору базової архітектури хмарної системи для оптимізації її інфраструктури. Нові комбінації інструментарію/технологій можуть бути віднесені до рекомендованого/нерекондованого набору без прив'язки до фінансової сторони використання, що є відмінністю від калькуляторів вибору інструментів створення хмарної системи від відомих провайдерів.

Додатково розглянуто підхід дослідження моделі багаторівневої архітектури хмарної системи для вирішення задачі масштабування (зокрема, горизонтального), що формує умову розміщення усіх запитів/завдань для чотирирівневої хмарної системи у вигляді порівняння суми усіх зарезервованих ресурсів та ємності їх серверів. Визначено параметр відсоткового еквіваленту зайнятості вузлів хмарної архітектури для розгортання додаткових серверів, що сформувало додаткову умову включення вимкнутих ресурсів або необхідність його формування.

Зокрема, запропоновано підхід для хмарної архітектури з довільною кількістю рівнів, який поєднує в собі умову розміщення усіх запитів/завдань на ресурсах та умови споживання (швидкодії роботи використаних ресурсів).

Розглянуто визначення стохастичних моделей та їх основні властивості. Формалізація вимог для модулів симулятора надало можливість розглянути

хмарну систему, як простий потік запитів/завдань між підсистемами, що приводить багатофазову систему до стохастичної мережі.

Побудовано формалізовану модель хмарної системи з чотирма архітектурними рівнями з кількостями кінцевих користувачів, граничних серверів, туманних вузлів та віртуальних машин.

Розглянуто моделі хмарних систем в термінології мереж масового обслуговування та сформовано умову сталого режиму, що підтвердило використання стратегії горизонтального масштабування при збільшенні вхідного потоку за рахунок збільшення кількості каналів обслуговування та/або заміна підсистем на більш потужні за інтенсивністю опрацювання запитів/завдань на кожному рівні хмарної системи.

Наведено наявні проблеми вибору найоптимальнішого рішення при простому збільшенні кількості каналів обслуговування та показано можливість їх вирішення при підвищенні інтенсивності обслуговування на існуючих ресурсах.

У проведеному дослідженні використано умову незалежності підсистем, що може бути змінена за рахунок вибору сценарію розгортання та стратегії масштабування. При припущенні, що рівні хмарної системи є незалежними, обчислено основні показники ефективності її функціонування в термінології систем масового обслуговування.

Сформовано підхід щодо існування хмарної системи з змінною інтенсивністю  $\lambda = \lambda(t)$ , який можна використати при формуванні моделі хмарної системи у режимі реального часу з проактивною оптимізацією (агентні технології, методи/алгоритм/підходи машинного навчання).

Результати розділу можуть бути використані для формування умов контролю складності архітектурного рішення, горизонтального масштабування та зберігання запитів/завдань за рахунок створення додаткових сховищ інформації та/або створення черг розвантаження.

## **РОЗДІЛ 3. МОДЕЛЬ СПІЛЬНОГО ДИНАМІЧНОГО РОЗВАНТАЖЕННЯ ХМАРНОЇ АРХІТЕКТУРИ**

### **3.1 Про моделі розвантаження хмарної архітектури**

Для вирішення задачі оброблення великого обсягу запитів/завдань різного типу з низькою затримкою останніми роками запропоновано великий спектр парадигм обчислень. Прикладом однієї з яких є гібридна парадигма (cloud-edge-end, CEE), яка використовує принципи побудови системи на ієрархічній архітектурі, що може використовувати ресурси хмарного центру, численних периферійних серверів з обмеженими можливостями та величезну кількість кінцевих пристроїв [99-100].

В останні роки проводяться велика кількість досліджень щодо вирішення проблеми розвантаження системи хмарних обчислень (СХМОбч) при використанні різних підходів та методологій. Наприклад, авторами роботи [101] це питання досліджується в рамках ієрархічної системи граничного туману та розроблено механізм стимулювання для зміщення переваг егоїстичних користувачів з граничного шару на шар туману відповідно до толерантності користувача до затримки. У роботі [102] запропонована спрощена структура прогнозування та розвантаження мобільності з використанням методу машинного навчання, спрямована на спільне вирішення проблем розвантаження обчислень та управління мобільністю в граничному рівні EC-Layer. Ідея автора роботи [94] щодо запропонованої стратегії динамічного спільного розвантаження хмарних граничних пристроїв з урахуванням балансування навантаження використана в наведеному дослідженні для більш складної структури СХМОбч.

### 3.2 Модель чотирирівневої хмарної архітектури з балансуванням рівнів

Результати роботи [103] містять результати побудови моделі спільного розвантаження хмарної архітектури з балансуванням рівнів.

В запропонованій термінології роботи, - це рівні *DL*, *EL* та *CL*.

При додаванні рівня *FL* типовий робочий процес системи можна описати наступним чином.

Спочатку користувачі створюють завдання та на цьому етапі може також використовуватися горизонтальне масштабування для покращення роботи окремих рівнів (виконання умов 2.3-2.5).

Потім кінцеві пристрої рівня *DL* розподіляють отримані завдання на граничні сервери рівня *EL* через основну мережу під керуванням схеми розвантаження рівня *DL*.

Граничні сервери розміщують отримані завдання у черзі.

На кожному граничному сервері декотра частина завдань оброблюється локально, а інша частина може переноситися на інші граничні сервери для балансування навантаження на рівень *EL*.

Паралельно завдання можуть перенаправлятися за рахунок роботи схеми керування щодо вибору маршрутизаторів підключення рівня *FL*.

На цьому рівні частина завдань може бути також перерозподілена за рахунок роботи хмарного центру ( $DCC = 1$ ) для віддаленої допомоги під керуванням схеми розвантаження хмарної системи.

Після оброблення завдання надходять на кінцеві пристрої до користувачів з граничних серверів, туманних вузлів або з хмарного центру (рис. 3.1).

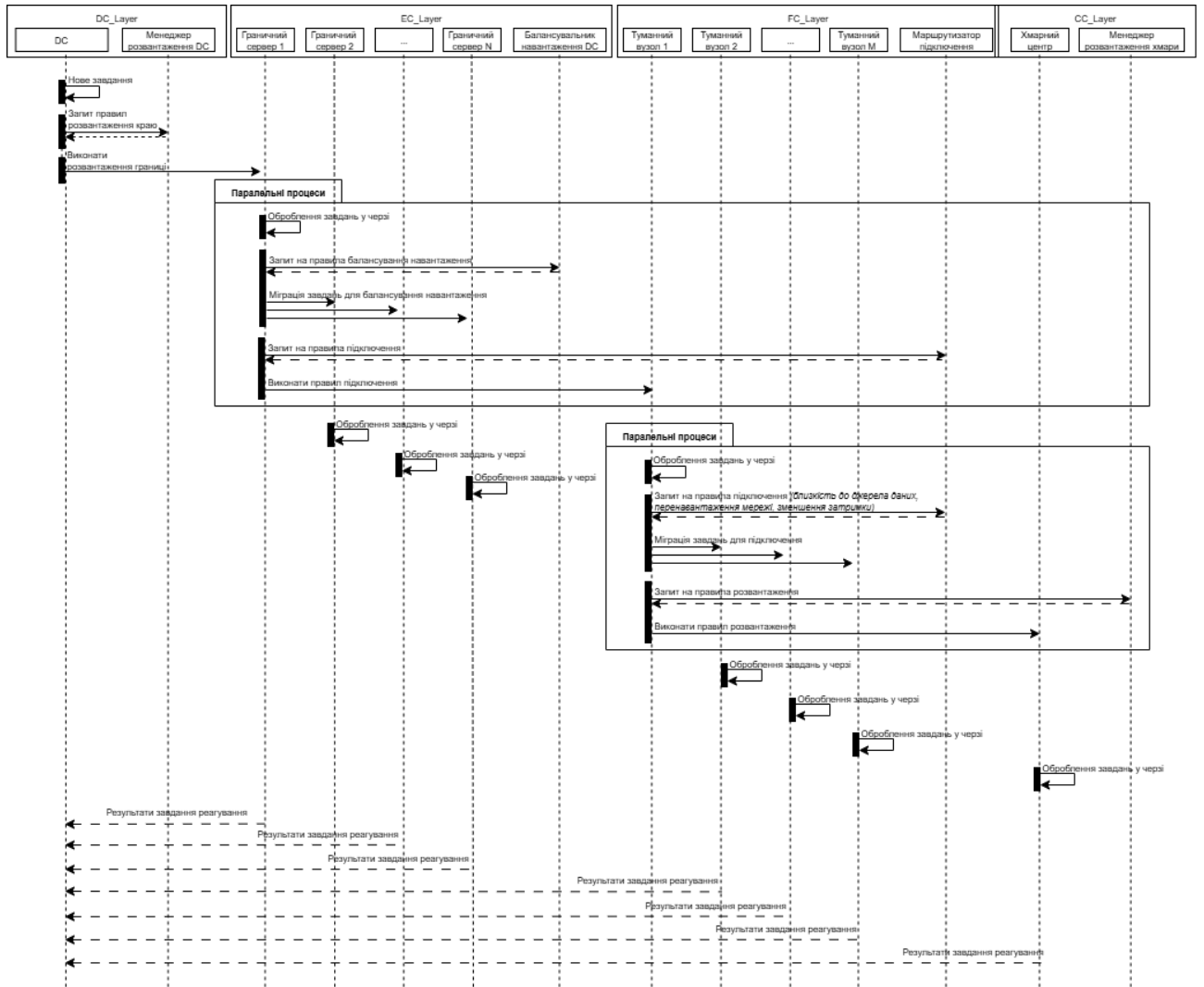


Рисунок 3.1 - Схема послідовності процесів СХМОбч

В термінології розділу 2 запропоновану систему можна представити як систему масового обслуговування  $G/G/4$ , але з додаванням зв'язків між вузлами хмарної системи з функціями балансування.

**Система 1.** Розглянемо гібридну систему, яка включає в себе типову систему граничних, туманних та хмарних обчислень для дослідження питання оптимізації навантаження за рахунок перенесення частини запитів/завдань *послідовно між рівнями* хмарної системи *без* використання алгоритмів балансування.

При такому визначенні послідовності процесів в СХМОбч надалі можна розглянути модифікації моделей з різними комбінаціями зв'язків між вузлами хмарної системи:

**Система 2.** Розглянемо гібридну систему, яка включає в себе типову систему граничних, туманних та хмарних обчислень для дослідження питання оптимізації навантаження за рахунок перенесення частини запитів/завдань *послідовно між рівнями* хмарної системи з використанням алгоритмів балансування *окремо на кожному рівні системи*.

**Система 3.** Розглянемо гібридну систему, яка включає в себе типову систему граничних, туманних, хмарних обчислень та Load-Layer для дослідження питання оптимізації навантаження за рахунок перенесення частини запитів/завдань *послідовно між рівнями* хмарної системи з використанням алгоритмів балансування *на створеному додатково додатку-агенті між/на рівнях системи*.

Запропонована модель поєднує принципи проектування граничних, туманних та хмарних обчислень. Зокрема, кінцеві пристрої користувачів, граничні сервери та туманні вузли разом з хмарним центром можуть утворювати СХМОбч. При перевантаженості граничних серверів частина завдань може бути частково перенесена до маршрутизатора підключення на FC-Layer, а при перезавантаженості туманних вузлів – частково перенесена до хмарного центру.

Подальші дослідження показано на прикладі **Системи 2** при врахуванні структури **Системи 1**.

Частина **Системи 1** при включенні першого та останнього рівнів з архітектури (граничний та хмарний) представляє собою СЕЕ, що забезпечує послуги з низькою затримкою для кінцевих користувачів через граничні сервери [104, 105].

Припустимо, що швидкість виконання завдань кожною колекцією вузлів рівнів відома або передбачувана.

Припустимо, що в системі доступне балансування навантаження.

Для обмеженого часу потрібно визначити швидкість розвантаження системи та пропорції розвантаження складових системи для мінімізації загальної затримки завдань.

### 3.3 Математичне формулювання стратегії балансування

Розглянемо СХМОбч, у складі якої є  $N$  кінцевих користувачів,  $ES$  граничних серверів та  $FR$  маршрутизаторів навантаження (рис. 3.2).

Нехай множина кінцевих користувачів

$$U = \{u_1, u_2, \dots, u_N\},$$

множина граничних серверів

$$S = \{s_1, s_2, \dots, s_{ES}\},$$

множина туманних вузлів (маршрутизаторів підключення)

$$R = \{r_1, r_2, \dots, r_{FR}\}.$$

Розглянемо скінчений часовий проміжок  $[0, T]$ .

В будь-який момент часу  $t \in [0, T]$  швидкість виконання запитів/завдань кінцевими пристроями  $u_i$  рівня DC позначимо як  $v_{1,i}^{task}(t)$ , швидкість виконання завдань граничними серверами  $s_j$  рівня ЕС позначимо як  $v_{2,j}^{task}(t)$  та швидкість виконання завдань туманними вузлами  $r_k$  рівня FC позначимо як  $v_{3,k}^{task}(t)$ .

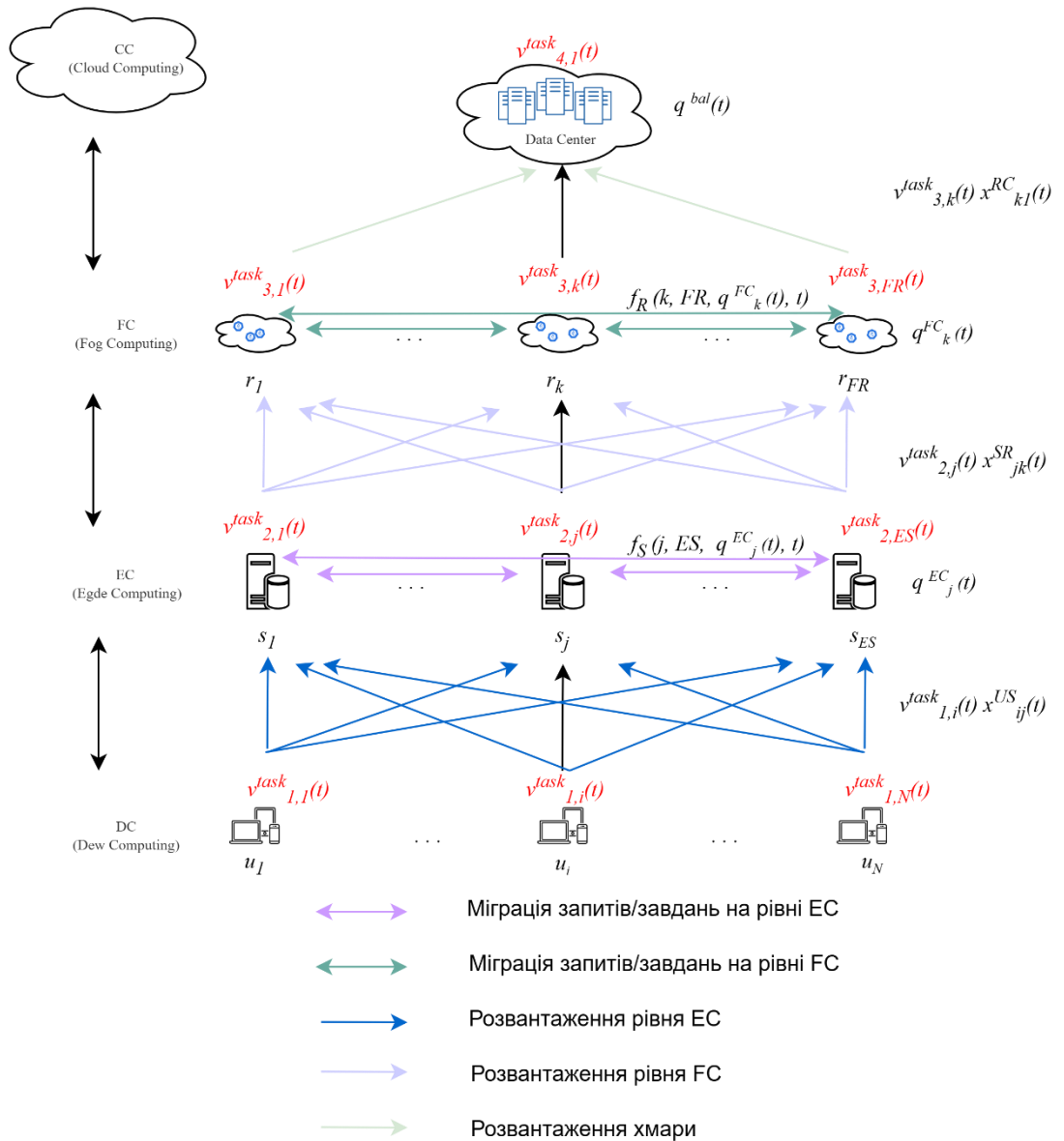


Рисунок 3.2 - Схема зв'язків чотирирівневої ХА

Частка запитів/завдань, які є вивантаженими у момент часу  $t \in [0, T]$  з кінцевого пристрою  $u_i$  на граничний сервер  $s_j$ , з граничного серверу  $s_j$  на туманний вузол  $r_k$  та з туманного вузлу до хмарного центру позначимо відповідно  $x_{ij}^{US}(t)$ ,  $x_{jk}^{SR}(t)$  та  $x_{k1}^{RC}(t)$ . За визначенням цих величин для всіх  $i$  та  $t$  виконується умова

$$\sum_{j=1}^{ES} x_{ij}^{US}(t) = 1$$

$$\sum_{k=1}^{FR} x_{jk}^{SR}(t) = 1$$

$$x_{k1}^{RC}(t) = 1$$

Функції  $x^{US}(t) = \{x_{ij}^{US}(t)\}_{N \times ES}$  та  $x^{SR}(t) = \{x_{jk}^{SR}(t)\}_{ES \times FR}$  для  $t \in [0, T]$

є стратегіями розвантаження рівнів ЕС та FC відповідно.

Функція

$$x(t) = \{x_{ik}(t)\}_{N \times FR} \text{ для } t \in [0, T]$$

є стратегією розвантаження рівнів DC, ЕС та FC.

Навантаження (довжина черги запитів/завдань) граничних серверів  $s_j$  та туманних вузлів  $r_k$  позначено як  $q_j^{EC}(t)$  та  $q_k^{FC}(t)$  відповідно. Навантаження хмарного центру позначимо функцією  $q^{bal}(t)$ .

Прослідкувати за траєкторією навантаження можна за допомогою функції

$$Balance(t) = \{q_1^{EC}(t), \dots, q_{ES}^{EC}(t), q_1^{FC}(t), \dots, q_{FR}^{FC}(t), q^{bal}(t)\}$$

Після надходження запитів/завдань до черг граничного сервера/туманного вузла виконується часткове їх локальне оброблення або перенесення на інші граничні сервери/туманні вузли за допомогою схем розвантаження/підключення з подальшим вивантаженням на наступні рівні.

Припустимо, що швидкості виконання запитів/завдань на граничних серверах та туманних вузлах є відносно стабільною та прийняти до роботи їх середні показники. Швидкість опрацювання запитів/завдань на кінцевому пристрої  $u_i$ , граничному сервері  $s_j$  та туманному вузлі  $r_k$  позначимо  $v_{1,i}^{task}(t)$ ,  $v_{2,j}^{task}(t)$  та  $v_{3,k}^{task}(t)$  відповідно. Швидкість хмарного центру позначимо як  $v_{4,1}^{task}(t)$ .

Визначимо швидкості перенесення власних запитів/завдань з  $s_n$  на інший  $s_m$  для контролю балансування навантаження рівня ЕС через функцію:

$$f_S(n, m, q_n^{EC}(t), t) = \begin{cases} A, & \text{якщо } n \neq m \\ 0, & \text{якщо } n = m \end{cases}$$

та балансування підключення рівня FC через функцію:

$$f_R(n, m, q_n^{FC}(t), t) = \begin{cases} B, & \text{якщо } n \neq m \\ 0, & \text{якщо } n = m \end{cases}$$

При  $n = m$  об'єктам побудованої схеми немає потреби переносити власні запити/завдання на себе.

Стратегію розвантаження хмари визначає множина

$$y(t) = \{y_k(t)\}_{k=1}^{FR} = \{v_{3,k}^{task}(t)x_{k1}^{RC}(t)\}_{k=1}^{FR}, \quad t \in [0; T]$$

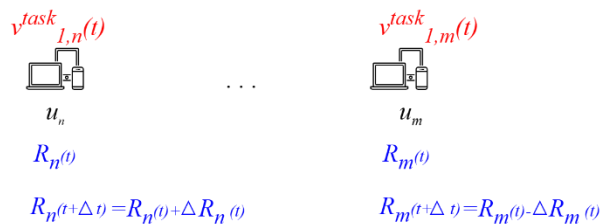
### 3.4 Математичний вираз стратегій розвантаження СХМОбч

Розглянемо набір функцій  $x(t)$  та  $y(t)$ , які є стратегіями розвантаження рівнів СХМОбч (у визначеному прикладі, це рівні DC, ES та FC) та хмарного рівня для  $t \in [0; T]$ .

Множина стратегій розвантаження СХМОбч може бути представлена як:

$$\Phi = \{(x, y): x \in \mathbb{R}^{N \times ES} \times \mathbb{R}^{ES \times FR}, y \in \mathbb{R}^{FR} | \\ x_{ij}(t) \in [0; 1], x_{jk}(t) \in [0; 1], \sum_{j=1}^{ES} x_{ij}^{US}(t) = 1, \sum_{k=1}^{FR} x_{jk}^{SR}(t) = 1, \\ y_k(t) \in [0; y_{max}], i = 1, \dots, N, j = 1, \dots, ES, k = 1, \dots, FR, t \in [0; T] \}$$

Розглянемо можливість збільшення/зменшення навантаження на рівень DC, яке пов'язано з змінами кількості запитів від кожного користувача  $\Delta R_i(t) = R_i(t + \Delta t) - R_i(t)$  протягом інтервалу часу  $[t; t + \Delta t]$ :



Якщо позначити через  $\overline{v_{1,l}^{task}}$  середню швидкість виконання

запитів/завдань у будь-який момент часу, тоді швидкості виконання завдання в моменти часу  $t$  та  $t + \Delta t$  будуть

$$v_{1,i}^{task}(t) = R_i(t) \cdot \overline{v_{1,i}^{task}}$$

$$v_{1,i}^{task}(t + \Delta t) = R_i(t + \Delta t) \cdot \overline{v_{1,i}^{task}}$$

Отже,

$$v_{1,i}^{task}(t + \Delta t) = (R_i(t) + \Delta R_i(t)) \cdot \overline{v_{1,i}^{task}}$$

### 3.5 Дослідження зв'язку між стратегією та функції траєкторією навантаження

Припустимо, що кожний об'єкт побудованої схеми СХМОбч має чергу запитів/завдань нескінченної довжини (без пріоритетів).

Навантаження на об'єкти  $s_j$  та  $r_k$  формує черги, для яких виконуються властивості:

- навантаження може зростати, якщо буде отримуватися запити/завдання від об'єктів нижчого рівня,
- через схему балансування навантаження/підключення об'єкти будуть переносити власні запити/завдання на інші об'єкти того ж рівня або отримуватиме запити/завдання від інших об'єктів.

При врахуванні твердження роботи [94] сформуємо динамічну систему для характеристики динаміки навантаження рівнів DC, EC, FC та CC.

Позначимо початковий розподіл навантаження СХМОбч через  $Q_0$ .

Нехай

$$\delta(x) = \begin{cases} 0, & \text{якщо } x = 0 \\ 1, & \text{якщо } x > 0 \end{cases}$$

Розглянемо рівень CC в часовому інтервалі  $[t; t + \Delta t]$  та використаємо визначення напрямку навантаження:

$$\begin{aligned} (q^{bal}(t))' &= \lim_{\Delta t \rightarrow 0+0} \frac{q^{bal}(t + \Delta t) - q^{bal}(t)}{\Delta t} \\ q^{bal}(t + \Delta t) &= q^{bal}(t) + (q^{bal}(t))' \Delta t, t \in [0; T] \end{aligned}$$

На рівні СС в момент часу  $t + \Delta t$  навантаження  $q^{bal}(t)$  через розвантаження хмари

1) збільшиться на суму добутоків величини про наявність черги, швидкості опрацювання запитів/завдань на туманному вузлі  $r_k$  та частки запитів/завдань, які є вивантаженими з  $r_k$  на хмарний центр

$$\sum_{k=1}^{FR} \delta(q_k^{FC}(t)) y_k(t) = \sum_{k=1}^{FR} \delta(q_k^{FC}(t)) v_{3,k}^{task}(t) x_{k1}^{RC}(t)$$

2) зменшиться через обробку запиту/завдання на добуток величини про наявність черги та швидкості опрацювання запитів/завдань на хмарному центрі

$$\delta(q^{bal}(t)) v_{4,1}^{task}(t)$$

Отже, навантаження рівня СС в момент часу  $t + \Delta t$  буде дорівнювати:

$$\begin{aligned} q^{bal}(t + \Delta t) &= \\ &= q^{bal}(t) + \Delta t \sum_{k=1}^{FR} \delta(q_k^{FC}(t)) v_{3,k}^{task}(t) x_{k1}^{RC}(t) \\ &\quad - \Delta t \delta(q^{bal}(t)) v_{4,1}^{task}(t) \end{aligned}$$

На рівні FC в момент часу  $t + \Delta t$  навантаження  $q_k^{EC}(t)$  через розвантаження рівня

1) збільшиться на суму добутоків швидкості виконання завдань кінцевими пристроями  $s_i$  рівня ЕС та частки запитів/завдань, які є вивантаженими з  $s_i$  на туманний вузол  $r_k$

$$\sum_{j=1}^{ES} v_{2,j}^{task}(t) x_{jk}^{SR}(t)$$

2) збільшиться на суму швидкостей перенесення запитів/завдань з  $r_n$  на інший  $r_k$  для контролю балансування навантаження рівня FC

$$\sum_{k=1}^{FR} f_S(n, k, q_n^{FC}(t))$$

3) зменшиться через обробку запиту/завдання на добуток величини про наявність черги та швидкості опрацювання запитів/завдань на туманному вузлі  $r_k$

$$\delta(q_k^{FC}(t)) v_{3,k}^{task}(t)$$

4) зменшиться через обробку запиту/завдання на добуток величини про наявність черги, швидкості опрацювання завдань на граничному сервері  $r_k$  та частки запитів/завдань, які є вивантаженими з  $r_k$  на хмарний центр

$$\delta(q_k^{FC}(t)) v_{3,k}^{task}(t) x_{k1}^{RC}(t)$$

5) зменшиться через обробку запиту/завдання на суму швидкостей перенесення запитів/завдань з  $r_k$  на інший  $r_n$  для контролю балансування навантаження рівня FC

$$\sum_{n=1}^{FR} f_S(k, n, q_k^{FC}(t))$$

Отже, навантаження рівня FC в момент часу  $t + \Delta t$  буде дорівнювати:

$$q_k^{FC}(t + \Delta t) =$$

$$= q_k^{FC}(t) + \Delta t \sum_{j=1}^{ES} v_{2,j}^{task}(t) x_{jk}^{SR}(t)$$

$$+ \Delta t \left( \sum_{k=1}^{FR} f_S(n, k, q_n^{FC}(t)) - \sum_{n=1}^{FR} f_S(k, n, q_k^{FC}(t)) \right) -$$

$$- \Delta t \delta(q_k^{FC}(t)) v_{3,k}^{task}(t) - \Delta t \delta(q_k^{FC}(t)) v_{3,k}^{task}(t) x_{k1}^{RC}(t)$$

На рівні ЕС в момент часу  $t + \Delta t$  навантаження  $q_j^{EC}(t)$  через розвантаження рівня збільшиться на

1) збільшиться на суму добутоків швидкості виконання завдань кінцевими пристроями  $u_i$  рівня DC та частки запитів/завдань, які є вивантаженими з пристрою  $u_i$  на граничний сервер  $s_j$

$$\sum_{i=1}^N v_{1,i}^{task}(t) x_{ij}^{US}(t)$$

2) збільшиться на суму швидкостей перенесення запитів/завдань з  $s_n$  на інший  $s_j$  для контролю балансування навантаження рівня ЕС

$$\sum_{n=1}^{ES} f_s(n, j, q_n^{ES}(t))$$

3) зменшиться через обробку запиту/завдання на добуток величини про наявність черги та швидкості опрацювання запитів/завдань на граничному сервері  $s_j$

$$\delta(q_j^{ES}(t)) v_{2,j}^{task}(t)$$

4) зменшиться через обробку запиту/завдання на суму добутоків величин про наявність черги, швидкості опрацювання запитів/завдань на граничному сервері  $s_j$  та частки запитів/завдань, які є вивантаженими з  $s_j$  на туманний вузол  $r_k$

$$\sum_{k=1}^{FR} \delta(q_j^{ES}(t)) v_{2,j}^{task}(t) x_{jk}^{SR}(t)$$

5) зменшиться через обробку запиту/завдання на суму швидкостей перенесення запитів/завдань з  $s_j$  на інший  $s_n$  для контролю балансування навантаження рівня ЕС

$$\sum_{n=1}^{ES} f_s(j, n, q_j^{ES}(t))$$

Отже, навантаження рівня ЕС в момент часу  $t + \Delta t$  буде дорівнювати:

$$\begin{aligned}
q_j^{EC}(t + \Delta t) &= \\
&= q_j^{EC}(t) + \Delta t \sum_{i=1}^N v_{1,i}^{task}(t) x_{ij}^{US}(t) \\
&+ \Delta t \left( \sum_{\substack{n=1, \\ n \neq j}}^{ES} f_S(n, j, q_n^{ES}(t)) - \sum_{\substack{n=1, \\ n \neq j}}^{ES} f_S(j, n, q_j^{ES}(t)) \right) - \\
&\quad - \Delta t \delta(q_j^{ES}(t)) v_{2,j}^{task}(t) - \sum_{k=1}^{FR} \delta(q_j^{ES}(t)) v_{2,j}^{task}(t) x_{jk}^{SR}(t)
\end{aligned}$$

Траєкторія навантаження функції  $Balance(t)$  залежить від множини стратегій розвантаження  $\Phi(x, y)$  динамічної системи:

$$q^{bal}(t) = Q_0, t \in [0, T]$$

$$(q^{bal}(t))' = \lim_{\Delta t \rightarrow 0} (q^{bal}(t + \Delta t) - q^{bal}(t)) / \Delta t$$

$$(q_k^{FC}(t))' = \lim_{\Delta t \rightarrow 0} (q_k^{FC}(t + \Delta t) - q_k^{FC}(t)) / \Delta t \quad (3.1)$$

$$(q_j^{EC}(t))' = \lim_{\Delta t \rightarrow 0} (q_j^{EC}(t + \Delta t) - q_j^{EC}(t)) / \Delta t$$

де функції навантаження на кожному рівні в момент часу  $t + \Delta t$  визначені наступним чином:

$$q^{bal}(t + \Delta t) =$$

$$\begin{aligned}
&= q^{bal}(t) + \Delta t \sum_{k=1}^{FR} \delta(q_k^{FC}(t)) v_{3,k}^{task}(t) x_{k1}^{RC}(t) \\
&\quad - \Delta t \delta(q^{bal}(t)) v_{4,1}^{task}(t)
\end{aligned}$$

$$q_k^{FC}(t + \Delta t) =$$

$$\begin{aligned}
&= q_k^{FC}(t) + \Delta t \sum_{j=1}^{ES} v_{2,j}^{task}(t) x_{jk}^{SR}(t) \\
&\quad + \Delta t \left( \sum_{\substack{n=1, \\ n \neq k}}^{FR} f_R(n, k, q_n^{FC}(t)) - \sum_{\substack{n=1, \\ n \neq k}}^{FR} f_R(k, n, q_k^{FC}(t)) \right)
\end{aligned}$$

$$\begin{aligned}
& - \Delta t \delta \left( q_k^{FC}(t) \right) v_{3,k}^{task}(t) - \Delta t \delta \left( q_k^{SR}(t) \right) v_{3,k}^{task}(t) x_{k1}^{RC}(t) \\
q_j^{EC}(t + \Delta t) = & \\
= q_j^{EC}(t) + \Delta t \sum_{i=1}^N v_{1,i}^{task}(t) x_{ij}^{US}(t) & \\
+ \Delta t \left( \sum_{\substack{n=1, \\ n \neq j}}^{ES} f_s(n, j, q_n^{ES}(t)) - \sum_{\substack{n=1, \\ n \neq j}}^{ES} f_s(j, n, q_j^{ES}(t)) \right) - & \\
- \Delta t \delta \left( q_j^{ES}(t) \right) v_{2,j}^{task}(t) - \sum_{k=1}^{FR} \delta \left( q_j^{ES}(t) \right) v_{2,j}^{task}(t) x_{jk}^{SR}(t) &
\end{aligned}$$

Отже, умова (3.1) може бути записана так:

$$\begin{aligned}
q^{bal}(t) = Q_0, t \in [0, T] & \\
\left( q^{bal}(t) \right)' = \sum_{k=1}^{FR} \delta \left( q_k^{FC}(t) \right) v_{3,k}^{task}(t) x_{k1}^{RC}(t) - \delta \left( q^{bal}(t) \right) v_{4,1}^{task}(t) & \\
\left( q_k^{FC}(t) \right)' = \sum_{j=1}^{ES} v_{2,j}^{task}(t) x_{jk}^{SR}(t) & \\
+ \left( \sum_{\substack{n=1, \\ n \neq k}}^{FR} f_R(n, k, q_n^{FC}(t)) - \sum_{\substack{n=1, \\ n \neq k}}^{FR} f_R(k, n, q_k^{FC}(t)) \right) & \\
- \delta \left( q_k^{FC}(t) \right) v_{3,k}^{task}(t) - \delta \left( q_k^{FC}(t) \right) v_{3,k}^{task}(t) x_{k1}^{RC}(t), & \\
k = 1, 2, \dots, FR, t \in [0, T] &
\end{aligned}$$

$$\begin{aligned}
\left( q_j^{EC}(t) \right)' = \sum_{i=1}^N v_{1,i}^{task}(t) x_{ij}^{US}(t) & \\
+ \left( \sum_{\substack{n=1, \\ n \neq j}}^{ES} f_s(n, j, q_n^{ES}(t)) - \sum_{\substack{n=1, \\ n \neq j}}^{ES} f_s(j, n, q_j^{ES}(t)) \right) - &
\end{aligned}$$

$$- \delta \left( q_j^{ES}(t) \right) v_{2,j}^{task}(t) - \sum_{k=1}^{FR} \delta \left( q_j^{ES}(t) \right) v_{2,j}^{task}(t) x_{jk}^{SR}(t),$$

$$j = 1, 2, \dots, ES, t \in [0, T]$$

Кількість запитів/завдань в момент часу  $t$  на об'єкті  $j$  рівня  $level$  з кількістю  $L$  пристроїв визначимо як

$$q_j^{level}(t) = \sum_{i=1}^L v_{level-1,i}^{task}(t) x_{i,j}^{name}(t) \quad (3.2)$$

Розглянемо СХмОбч, яка складається з хмарного центру, двох кінцевих пристроїв, двох граничних серверів, двох туманних вузлів (рис. 3.3).

Припустимо, що одиницею часу є секунда і  $\Delta t = 1$ .

Швидкості виконання запитів/завдань (кількість запитів/завдань, що система може обробити за одну секунду, англ. *tps*) на рівнях системи визначені так:

$$v_{1,i}^{task}(t) = \{100, 50\}, i = 1, 2$$

$$v_{2,j}^{task}(t) = \{10, 20\}, j = 1, 2$$

$$v_{3,k}^{task}(t) = \{16, 10\}, k = 1, 2$$

Стратегії розвантаження рівнів визначені, як

$$x_{1j}^{US}(t) = \{0.4, 0.6\}, j = 1, 2$$

$$x_{2j}^{US}(t) = \{0.6, 0.4\}, j = 1, 2$$

$$x_{1k}^{SR}(t) = \{0.8, 0.2\}, k = 1, 2$$

$$x_{2k}^{SR}(t) = \{0.2, 0.8\}, k = 1, 2$$

$$x_{r1}^{RC}(t) = \{0.5, 0.5\}, r = 1, 2$$

Нехай швидкості перенесення запитів/завдань (припустимо, що кількість запитів/завдань, які мігрують дорівнює  $f_{level,max}$ , де  $level = 2, 3$ ) з одного об'єкта на інший системи на кожному рівні представлена функціями

$$f_{level}(j, n, q_j^{level}(t)) = \begin{cases} 0, & \text{якщо } q_j^{level}(t) \leq q_n^{level}(t) \\ f_{level,max}(j, n), & \text{якщо } q_j^{level}(t) > q_n^{level}(t) \end{cases}$$

при  $j, n = 1, 2$ .

СХМОбч при заданих умовах на початку роботи (при  $t = 0$ ) має вигляд:

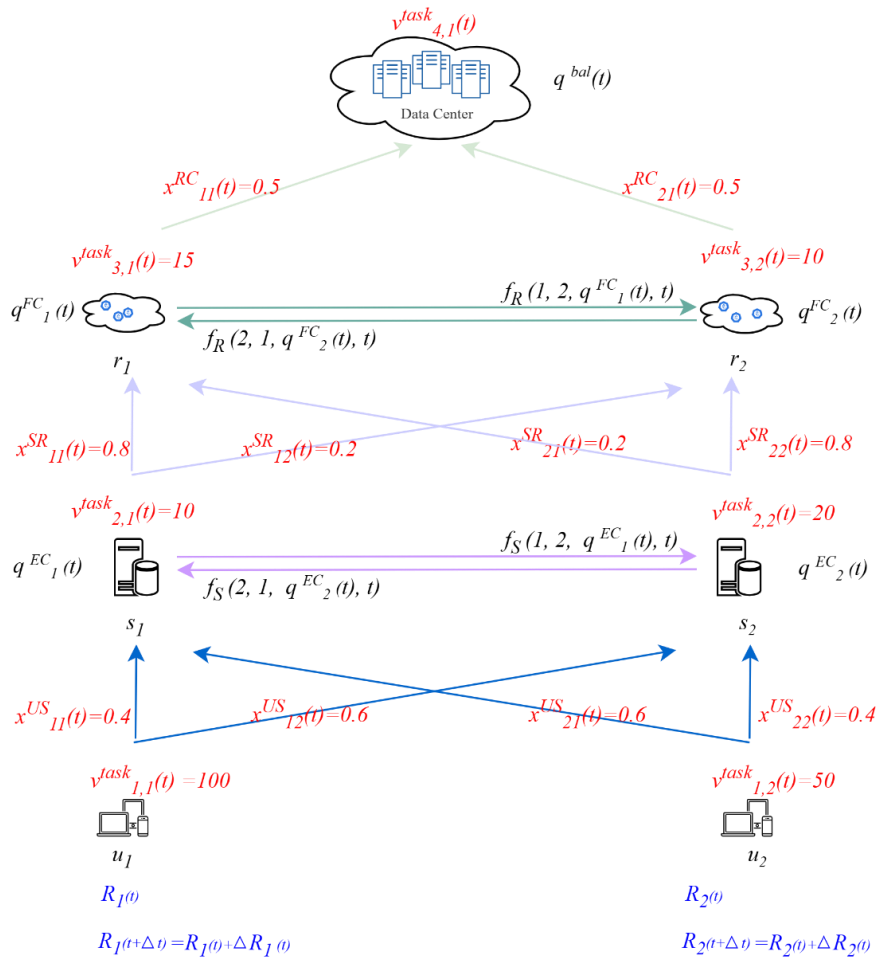


Рисунок 3.3 - Схема зв'язків прикладу ( $DCC = 1, N = 2, ES = 2, SR = 2$ )

Розглянемо інтервал часу  $t \in [0; 2], \Delta t = 1$ .

Ітерація 1.  $t = 0, t + \Delta t = 1$

З пристрою  $u_1$  на граничні сервери  $s_j (j = 1, 2)$  було перенесено відповідно запитів/завдань:

$$v_{1,1}^{task}(t)x_{11}^{US}(t) = 100 \cdot 0.4 = 40$$

$$v_{1,1}^{task}(t)x_{12}^{US}(t) = 100 \cdot 0.6 = 60$$

З пристрою  $u_2$  на граничні сервери  $s_j (j = 1, 2)$  було перенесено відповідно запитів/завдань:

$$v_{1,2}^{task}(t)x_{21}^{US}(t) = 50 \cdot 0.6 = 30$$

$$v_{1,2}^{task}(t)x_{22}^{US}(t) = 50 \cdot 0.4 = 20$$

Нехай при  $j, n = 1, 2$

$$f_{level,max}(j, n) = \lceil |q_j^{level}(t) - q_n^{level}(t)| / 2 \rceil,$$

де  $\lceil \cdot \rceil$  - ціла частина числа.

Кількість запитів/завдань на граничних серверах  $s_1$  складає

$$q_1^{EC}(t) = v_{1,1}^{task}(t)x_{11}^{US}(t) + v_{1,2}^{task}(t)x_{21}^{US}(t) = 40 + 30 = 70$$

та  $s_2$  складає

$$q_2^{EC}(t) = v_{1,1}^{task}(t)x_{12}^{US}(t) + v_{1,2}^{task}(t)x_{22}^{US}(t) = 60 + 20 = 80,$$

що показує більше навантаження на другому граничному сервері. Отже, для балансування навантаження було перенесено 5 запитів/завдань з другого серверу на перший:

$$f_s(1, 1, q_1^{EC}(t)) = 0$$

$$f_s(1, 2, q_1^{EC}(t)) = 0$$

$$f_s(2, 1, q_2^{EC}(t)) = \frac{|q_2^{EC}(t) - q_1^{EC}(t)|}{2} = \frac{|80 - 70|}{2} = 5$$

$$f_s(2, 2, q_2^{EC}(t)) = 0$$

Після перерозподілу запитів/завдань на рівні ЕС на граничних серверах  $s_1$  складає  $q_1^{EC}(t) = 70 + 5 = 75$  та  $s_2$  складає  $q_2^{EC}(t) = 80 - 5 = 75$ .

З пристрою  $s_1$  на граничні сервери  $r_k (k = 1, 2)$  перенесено відповідно запитів/завдань:

$$v_{2,1}^{task}(t)x_{11}^{SR}(t) = 10 \cdot 0.8 = 8$$

$$v_{2,2}^{task}(t)x_{21}^{SR}(t) = 20 \cdot 0.2 = 4$$

З пристрою  $s_2$  на граничні сервери  $r_k (k = 1, 2)$  було перенесено відповідно запитів/завдань:

$$v_{2,1}^{task}(t)x_{12}^{SR}(t) = 10 \cdot 0.2 = 2$$

$$v_{2,2}^{task}(t)x_{22}^{SR}(t) = 20 \cdot 0.8 = 16$$

Протягом часу  $[0; 1]$  навантаження на граничний сервер  $s_1$  зміниться на

$$\Delta q_1^{EC} = q_1^{EC}(1) - q_1^{EC}(0) = 55,$$

де

$$\begin{aligned}q_1^{EC}(1) &= \left( v_{1,1}^{task}(1)x_{11}^{US}(1) + v_{1,2}^{task}(1)x_{21}^{US}(1) \right) \\ &\quad + \left( f_S(2,1, q_2^{ES}(1)) - f_S(1,2, q_1^{ES}(1)) \right) q_1^{EC}(0) \\ &= \delta(q_1^{ES}(0)) v_{2,1}^{task}(0) \\ &\quad + \left( \delta(q_1^{ES}(0)) v_{2,1}^{task}(0)x_{11}^{SR}(0) + \delta(q_1^{ES}(0)) v_{2,1}^{task}(0)x_{12}^{SR}(0) \right)\end{aligned}$$

$$q_1^{EC}(1) = (100 \cdot 0.4 + 50 \cdot 0.6) + (5 - 0) = (40 + 30) + 5 = 75$$

$$q_1^{EC}(0) = 1 \cdot 10 + (1 \cdot 10 \cdot 0.8 + 1 \cdot 10 \cdot 0.2) = 10 + (8 + 2) = 20$$

Протягом часу  $[0; 1]$  навантаження на граничний сервер  $s_2$  зміниться на

$$\Delta q_2^{EC} = q_2^{EC}(1) - q_2^{EC}(0) = 45,$$

де

$$\begin{aligned}q_2^{EC}(1) &= \left( v_{1,1}^{task}(1)x_{12}^{US}(1) + v_{1,2}^{task}(1)x_{22}^{US}(1) \right) \\ &\quad + \left( f_S(1,2, q_1^{ES}(1)) - f_S(2,1, q_2^{ES}(1)) \right) \\ q_2^{EC}(0) &= \delta(q_2^{ES}(0)) v_{2,2}^{task}(1) \\ &\quad + \left( \delta(q_2^{ES}(0)) v_{2,2}^{task}(0)x_{2,1}^{SR}(0) + \delta(q_2^{ES}(0)) v_{2,2}^{task}(0)x_{22}^{SR}(0) \right)\end{aligned}$$

$$q_2^{EC}(1) = (100 \cdot 0.6 + 50 \cdot 0.4) + (0 - 5) = (60 + 20) - 5 = 85$$

$$q_2^{EC}(0) = 1 \cdot 20 + (1 \cdot 20 \cdot 0.2 + 1 \cdot 20 \cdot 0.8) = 20 + (4 + 16) = 40$$

Протягом часу  $\Delta t = 1$  змінилася кількість вузлів на об'єктах рівнів DC, EC та FC системи.

Кількість запитів/завдань, що залишилися не обробленими на граничних серверах

$$q_1^{EC}(t) = 75 - 10 = 65$$

$$q_2^{EC}(t) = 85 - 20 = 65.$$

Кількості запитів/завдань в черзі на туманних вузлах  $r_1$  та  $r_2$  відповідно складає

$$q_1^{FC}(t) = v_{2,1}^{task}(t)x_{11}^{SR}(t) + v_{2,2}^{task}(t)x_{21}^{SR}(t) = 10 \cdot 0.8 + 20 \cdot 0.2 = 8 + 4 = 12$$

$$q_2^{FC}(t) = v_{2,1}^{task}(t)x_{12}^{SR}(t) + v_{2,2}^{task}(t)x_{22}^{SR}(t) = 10 \cdot 0.2 + 20 \cdot 0.8 = 2 + 16 = 18,$$

що показує більше навантаження на другому туманному вузлі. Отже, для балансування навантаження було перенесено 3 завдання з другого вузла на перший:

$$f_R(1,1, q_1^{FC}(t)) = 0$$

$$f_R(1,2, q_1^{FC}(t)) = 0$$

$$f_R(2,1, q_2^{FC}(t)) = \frac{|q_2^{FC}(t) - q_1^{FC}(t)|}{2} = \frac{|18 - 12|}{2} = 3$$

$$f_R(2,2, q_2^{FC}(t)) = 0$$

Після перерозподілу завдань на рівні FC (при спрацьовуванні функції балансування  $f_R$ ) на туманних вузлах  $r_1$  черга складає  $q_1^{FC}(t) = 14$  та  $r_2$  складає  $q_2^{FC}(t) = 14$ .

Протягом часу  $[0; 1]$  підключення на туманний вузли  $r_1$  та  $r_2$  зміниться на

$$\Delta q_1^{FC} = q_1^{FC}(1) - q_1^{FC}(0) = -8,$$

$$q_2^{FC} = q_2^{FC}(1) - q_2^{FC}(0) = -1,$$

що може говорити про відсутність черги в той момент часу на рівні FC.

Ітерація 2.  $t = 1, t + \Delta t = 2$

Процес перерозподілу запитів/завдань проходить за подібною схемою, яка наведена вище. На цьому кроці моделювання на рівні хмари при заданих початкових умовах функція  $q^{bal}(t)$  може показувати відсутність черги.

Наступні ітерації роботи алгоритму побудови функції навантаження системи надають можливість не тільки подивитися на обчислювальну

складність моделі, яка зростає з горизонтальним масштабуванням, а і можливі наявні проблеми архітектурного рішення структури хмарної системи.

### 3.6 Використання агентів-додатків в хмарній архітектурі

У роботі [106] розглянуто хмарну систему IaaS, яка складається з кількох центрів обробки даних з ресурсами зберігання та обчислення, що представлені віртуальними машинами (система MOWS). Авторами запропоновано планування з урахуванням безпеки, яке досліджує вплив взаємодії завдань на ризик безпеки хмари.

Визначимо набір незалежних одне від одного запитів/завдань

$$T^{(i)} = \{T_1^{(i)}, T_2^{(i)}, \dots, T_{k(i)}^{(i)}\},$$

де

$i$  – кількість додатків-агентів, які можуть бути вбудовані на будь-якому рівні архітектури системи та можуть утворювати ансамблі управління,

$k(i)$  – кількість запитів/завдань в кожному наборі  $T^{(i)}$  в залежності від додатку-агенту  $ag\_W^{(i)}$ .

Внесення подібної зміни в структуру хмарної системи трансформує її з централізованої моделі в агенто-орієнтовану децентралізовану систему, де управління ресурсами здійснюється не одним алгоритмом, а ансамблем агентів  $ag\_W^{(i)}$ , кожен з яких відповідає за свій набір запитів/завдань  $T^{(i)}$  (рис. 3.4). Таке перетворення на кожному рівні архітектури СХМОбч приводить до розгляду модифікації моделі **Системи 3**, структура якої є актуальною за рахунок збільшення великої кількості різних додатків-агентів для контролю навантаження на різних рівнях/підрівнях хмарної системи.

Наприклад, агент-додаток на рівні інфраструктури  $L4$  (рис. 1.14) може керувати тільки віртуалізацією, а на рівні застосунків  $L7$  (рис. 1.14) керувати тільки бізнес-додатками.

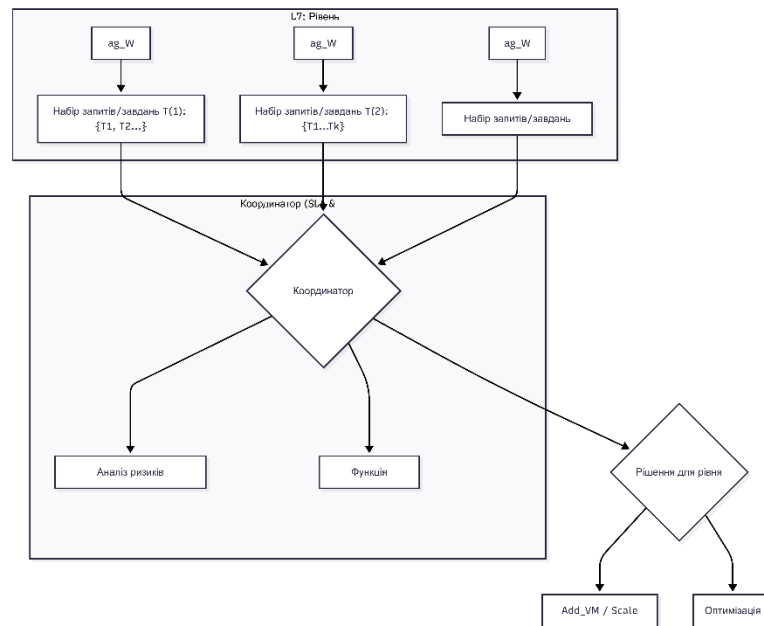


Рисунок 3.4 - Використання ансамблю агентів-додатків на рівні застосунку  $L7$  (рис. 1.14)

Додавання нового агента-додатка  $ag_W^{(i)}$  на один із рівнів чотирирівневої системи створює ефект «каскадної адаптації» [107]. Оскільки рівні взаємозалежні, поява агента змінює баланс ресурсів  $Capacity(t)$  та швидкості  $v(t)$ .

Розглянемо вплив одного агента-додатку на кожен з рівнів рисунку 1.14.

Додавання агента-додатку на фізичний рівень ( $L1$  - Hardware), який відслідковує параметри  $c_{11}, \dots, c_{15}$ , збільшує точність розрахунку  $f(C_i)$  та створює мінімальне додаткове навантаження на CPU, але дозволяє швидше ідентифікувати деградацію дисків або перегрів, запобігаючи помилковим командам  $Add\_VM$ .

Додавання агента-додатку на рівень віртуалізації ( $L2$  - IaaS), який керує розміщенням вузлів  $R_{41}, R_{42}$ , оптимізує умову  $Condition\_Pl\_VM$  та може перерозподілити віртуальні машини між серверами так, щоб вивільнити цілий

фізичний вузол, що безпосередньо впливає на зменшення функції вартості  $Cost_{VM}$  за рахунок консолідації.

Додавання агента-додатку на рівень платформи (L3 - PaaS), який працює з чергами та проміжного програмного забезпечення, керує параметром швидкістю взаємодії  $v_j(t)$  та може впровадити «розумне» чергування запитів, що згладжує піки навантаження (через цю модель класифікації рідше видає високу ймовірність  $P_{critical}$ , навіть якщо фізичний поріг 0.75 тимчасово перевищено).

Додавання агента-додатку на рівень додатків (L4 - SaaS/App), який фокусується на завданнях користувача  $T^{(i)}$ , дозволяє системі «бачити» об'єм даних  $Amount_s$  та їх пріоритетність (система починає працювати вибірково: VIP-завдання отримують миттєвий ресурс, а фонові завдання очікують, що стабілізує загальну ємність платформи).

Для запропонованої чотирирівневої структури СХМОбч додавання агентів-додатків  $ag_W^{(i)}$  перетворює статичну ієрархію на динамічну екосистему, де кожен рівень отримує власну «автономну логіку». На рівні пристроїв (DL) агент-додаток фільтрує надлишкові запити безпосередньо в джерелі, що радикально може знижувати параметр навантаження. На граничному рівні (EL) агент-додаток швидко класифікує запити/завдання для подальшого перенаправлення. На рівні туману (FL) агент-додаток координує локальні ансамблі, утворюючи ансамблі управління. На рівні хмари (CL) в функції агента-додатку покладено задачі глобальної оптимізації.

У таблиці 3.1 наведена інформація щодо впливу агента-додатку між визначеними рівнями.

При додаванні одного агента-додатку на рівень СХМОбч має математичні локальні та глобальні наслідки, що визначаються зростанням складності управління на рівні (через додаткові обчислення агента-додатку) та зменшенням загальної ентропії системи (формула  $Amount =$

$\{Capacity(t), v(t)\}$  стає більш прогнозованою, оскільки агент «відсікає» неефективне споживання ресурсів на своєму рівні).

Таблиця 3.1. Вплив агента-додатку на СХМОбч при вбудовуванні між рівнями

Взаємодія між рівнями	Роль агента-додатка (посередника)	Результат для системи
Dev ↔ Edge	Стиснення даних	Мінімізація затримок
Edge ↔ Fog	Оптимізація ресурсів	Перерозподіл між вузлами
Fog ↔ Cloud	Синхронізація станів	Оновлення моделей прогнозування

Отже, один агент-додаток робить вхідний потік запитів/завдань  $T^{(i)}$  впорядкованим, що зменшує сумарні витрати на масштабування.

Порівняння агентного підходу (на рівнях та між ними) з алгоритмом перерозподілу (всередині вузлів) демонструє різницю між глобальною стратегією та локальною тактикою. Алгоритм перерозподілу має найвищу швидкість реакції (мікросекунди) і є ефективним для миттєвого балансування черги, але агенти-додатки з більшою швидкістю реакції забезпечують вищу загальну пропускну здатність системи. Щодо ефективності використання ресурсів запропонований алгоритм перерозподілу оптимізує вузли кожного рівня, що порівнює цей підхід для агентів -додатків між рівнями як перенаправлення потоку запитів/завдань між рівнями. Нажаль, алгоритм перерозподілу погано адаптується до зміни топології мережі, що найкраще демонструють агенти-додатки між рівнями. Для запропонованого алгоритму перерозподілу запропоновано створення надлишковості за рахунок динамічного збільшення/зменшення вузлів рівня за рахунок виконання умов (2.2)-(2.4). Для максимальної ефективності їх потрібно використовувати разом: алгоритми контролюють стабільність вузлів, а агенти підтримують баланс між рівнями.

### Висновки до розділу 3

В даному розділі розглянуто принципи побудови системи на ієрархічній архітектурі та досліджено методологію розвантаження хмарної системи, яка складається з рівнів кінцевих користувачів, граничних серверів, туманних вузлів та хмарного центру.

Наведено схему послідовності процесів хмарної системи.

Для дослідження питання оптимізації представлено визначення трьох модифікацій чотирирівневих хмарних систем з різними комбінаціями зв'язків між вузлами системи та відповідно різними правилами перенесення запитів/завдань між рівнями.

Розроблено математичну модель навантаження чотирьох рівнів хмарної системи для контролю стану хмарної системи. Зокрема, запропоновано  $q^{bal}(t)$ ,  $q^{FC}(t)$ ,  $q^{EC}(t)$  навантаження рівнів хмарної системи.

Проведено ґрунтовне дослідження однієї модифікації чотирирівневої хмарної системи для обмеженого часу  $[0, T]$  з припущенням, що в системі доступне балансування  $q^{bal}(t)$  та швидкість  $v_j^{task}(t)$  ( $j = 1, 2, 3, 4$ ) виконання запитів/завдань кожною колекцією вузлів рівнів відома або передбачувана.

Приділена увага питанню побудові архітектурного рішення при моделюванні хмарної системи для підвищення ефективності її роботи при використанні агентно-орієнтованого підходу та технологій аналізу даних.

## РОЗДІЛ 4. ПРОГРАМНІ ЕКСПЕРИМЕНТИ

### 4.1 Формування множини моделей хмарних систем

#### 4.1.1 Параметри генерування моделей хмарних систем

Основними детермінантами для формування множини моделей хмарних систем є такі базові характеристики: середня кількість запитів у системі ( $N$ ); інтенсивність надходження вхідного потоку заявки ( $\lambda$ ); середній час перебування вимоги в системі ( $T$ ); інтенсивність обслуговування (або інтенсивність вихідного потоку запитів ( $\mu$ ); а також коефіцієнт завантаження системи ( $\rho$ ), який визначається як середня кількість запитів, що надходять протягом середнього часу обслуговування однієї вимоги.

Для побудови формалізованої моделі обираємо  $N = 1, ES = 2, FR = 3, V = 1$  (рис.2.5).

Методику вибору архітектурних рішень проілюстровано на прикладі розімкненої багатоканальної системи масового обслуговування з відмовами (без очікування).

Для формування простору можливих комбінацій без повторень як базовий критерій обрано параметр  $N = \{1, 2, 3\}$  для створення пристроїв на кожному рівні з різними кількостями каналів  $k = \{1, 2\}$  та інтенсивностями  $\mu = \{60, 70, 80, 90, 100\}$  обслуговування при вхідному потоці запитів/завдань  $\lambda_0 = 100$ . Вибір такого набору параметрів обумовлений потужністю обчислювального пристрою (оперативна пам'ять 16 Gb), який формує 810000 комбінацій системи.

З метою розширення простору можливих станів було збільшено значення параметрів системи: кількість пристроїв  $N$  та кількість каналів  $k$  до 5 одиниць включно. Таке варіювання параметрів дозволило згенерувати по 125 комбінацій на кожному з чотирьох архітектурних рівнів, що сумарно

сформувало 244140625 комбінаторних варіантів архітектури системи (обчислення реалізовано за обсягу оперативної пам'яті 64 ГБ).

#### 4.1.2 Програмна реалізація

Програмні експерименти реалізовано мовою Python. Розроблені модулі містять комплекс інженерних рішень, спрямованих на оптимізацію обчислювальної складності та забезпечення масштабованості системи.

У результаті було згенеровано повну множину можливих конфігураційних варіантів  $R\_system$  для чотирьох підсистем, кожен запис якого представляє одну унікальну модель системи. Кожна комбінація містить номер моделі, а потім послідовність значень (тип пристрою, кількість каналів, інтенсивність обслуговування) для кожної з чотирьох підсистем.

Для кожної з 810000 (244140625) моделей створено дві важливі структури:

- словник  $Model\_external\_arrivals\_to\_queues$ , де ключом є номер моделі, а значенням є список зовнішніх швидкостей надходження до кожної черги в цій моделі;

- словник  $Model\_routing\_matrix$ , де ключом є номер моделі, а значенням є матриця маршрутизації для цієї моделі, що визначає ймовірності переходу потоку між чергами.

Для кожної з 810000 (244140625) згенерованих моделей за допомогою функції  $calculate\_e\_values$  було розв'язано систему лінійних рівнянь. Це дозволило визначити ефективні інтенсивності надходження (коефіцієнтів передачі  $e\_values_i$ ) для кожної  $i$  черги в досліджуваній мережі.

Знайдені коефіцієнти передачі  $e\_values$  є базовими параметрами для розрахунку всіх подальших показників продуктивності мережі. Без їх точних значень неможливе коректне обчислення таких системних метрик, як

коефіцієнт завантаження серверів, ймовірність відмови в обслуговуванні, середня довжина черги запитів та середній час очікування.

Визначення сукупного обсягу завдань, що надходять до конкретної черги, дає об'єктивну оцінку навантаження на кожну підсистему.

На основі коефіцієнтів передачі та матриці маршрутизації було розраховано інтенсивності внутрішніх вхідних потоків для кожної черги (*internal\_arrival\_rates\_per\_queue*), що унаочнює характер взаємодії між структурними елементами системи.

### **4.1.3 Експериментальні дослідження складності побудованих моделей**

Під час комп'ютерного моделювання значна частина початкових конфігурацій моделей не задовольняла умову існування стаціонарного (сталого) режиму функціонування. Для стабілізації системи було апробовано підхід, що передбачав двократне збільшення кількості каналів обслуговування у вихідних моделях, що продемонструвало позитивну динаміку. Водночас стратегія динамічного нарощування кількості каналів на кожному обчислювальному пристрої чотирирівневої хмарної системи не забезпечує системного розв'язання проблеми перевантаження інфраструктури.

У межах подальшого експериментального дослідження було реалізовано альтернативний підхід двократного збільшення інтенсивності обслуговування вимог за умови повернення кількості каналів до базових значень. Отримані емпіричні дані засвідчили вищу ефективність такого рішення. Це дозволяє сформулювати наукове припущення щодо доцільності вибору стратегії розвантаження на користь підвищення пропускної здатності каналів (вертикальне масштабування), а не екстенсивного збільшення кількості паралельних каналів обслуговування (горизонтальне масштабування) у межах наявних пристроїв.

Із масштабуванням обчислювальних ресурсів, а також за потенційного зростання інтенсивності зовнішнього вхідного потоку  $\lambda_0$  або коефіцієнтів передачі  $e$ , дотримання умови існування стаціонарного режиму набуває визначального значення. Створене програмне рішення дозволяє оперативно ідентифікувати порушення критерію стабільності мережі та розраховувати мінімально необхідну кількість каналів обслуговування для гарантування стабільності системи. Ефективне функціонування високонавантажених систем передбачає забезпечення ємності каналів на всіх рівнях ієрархії для мінімізації ризиків дестабілізації. Водночас у великомасштабних мережах зі значною кількістю вузлів процедура розрахунку коефіцієнтів передачі характеризується високою обчислювальною складністю, що призводить до суттєвого зростання часових та апаратних витрат.

Для верифікації умов розподілу ресурсів та керування потоками даних у хмарній системі (2.2–2.5) використано різні підмножини з масиву 810000 сформованих комбінацій. Обмеження на число досліджуваних характеристик системи під час моделювання безпосередньо зумовлені обчислювальною потужністю використовуваного програмного середовища. Попри фокус на відносно малих значеннях кількості вузлів та каналів обслуговування, отримані закономірності зберігають свою теоретичну та практичну цінність, що обґрунтовано структурно-функціональним аналізом чотирирівневої хмарної системи.

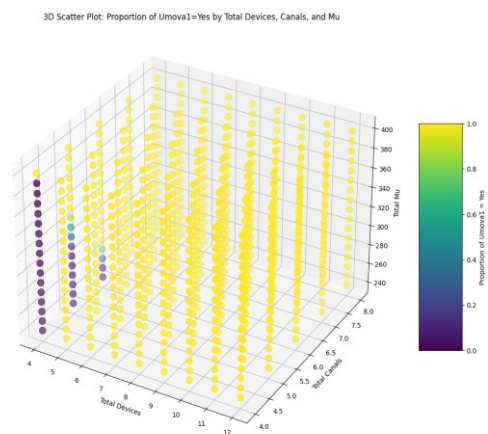


Рисунок 4.1 - 3D-графік розсіювання виконання умови сталого режиму

Наприклад, для перших 83621 моделей, в яких наявні різні архітектурні схеми, розглянуто роботу алгоритмів перевірки умов стабільності (рис. 4.1). Підсистема 1 постійно порушує умову «Складність ( $R > r$ )» (30 000 випадків), що свідчить про її роботу в нестабільному або перевантаженому стані, яке вимагає збільшення потужності. Підсистема 2 демонструє змішану поведінку, хоча наявні 13 800 випадків нестабільності («Так») і набагато більша кількість випадків (43 200) вказує на стабільність («Ні»). Загальна кількість (57 000) свідчить про те, що підсистема 2 часто складається з кількох черг, а її загальна стабільність залежить від конкретної конфігурації моделі. Підсистема 3 демонструє ще більшу стабільність порівняно з Підсистемою 2, маючи лише 4920 випадків «Так» проти 54 780 випадків «Ні». Це вказує на те, що ця підсистема загалом є більш стійкою або менш схильною до перевантаження у згенерованих моделях. Підсистема 4 є найстабільнішою з усіх, маючи лише 1120 випадків «Так» та велику кількість випадків «Ні» (58 880). Це свідчить про те, що кінцева підсистема зазвичай добре налаштована або ефективніше обробляє трафік.

Отже, підсистема 1 є критичним "вузьким місцем" і це означає, що її поточна потужність (кількість каналів  $r$ ) майже завжди недостатня для робочого навантаження, яке вона отримує (інтенсивність руху  $R$ ).

Для оптимізації підсистеми 1 можна розглянути такі стратегії:

- збільшення кількості каналів обслуговування ( $r$ ):
- збільшення інтенсивності обслуговування ( $\mu$ ):
- перерозподіл робочого навантаження або визначення пріоритетів трафіку (можна розглянути перерозподіл вхідного робочого навантаження на інші, менш використовувані частини рівня системи, або впровадження правил формування/пріоритету трафіку для зменшення ефективної швидкості надходження ( $\lambda_0 \cdot e$ ) до першої підсистеми у години пік, що вимагає змін у загальній архітектурній схемі хмарної системи).

Проведений аналіз свідчить, що перша підсистема виступає ключовим обмежувальним фактором, систематично порушуючи критерій стаціонарності, тоді як подальші компоненти демонструють значно вищий рівень стабільності. Отримані результати є визначальними для оптимізації мережевих проєктів.

#### 4.1.4 Експериментальні дослідження продуктивності побудованих моделей

В наступній серії експериментів проведений комплексний аналіз продуктивності системи, зосередившись на виявленні "вузьких місць" та впливу різних параметрів швидкості обслуговування ( $S_{\mu}$ ) на загальну пропускну здатність, а також впливу ймовірностей маршрутизації. Для цього розроблено універсальну функцію *calculate\_throughput\_for\_model*, яка дозволяє обчислювати загальну пропускну здатність для будь-якої моделі, а також гнучко перевизначати значення  $S_{\mu}$  для окремих рівнів та використовувати модифіковані матриці маршрутизації.

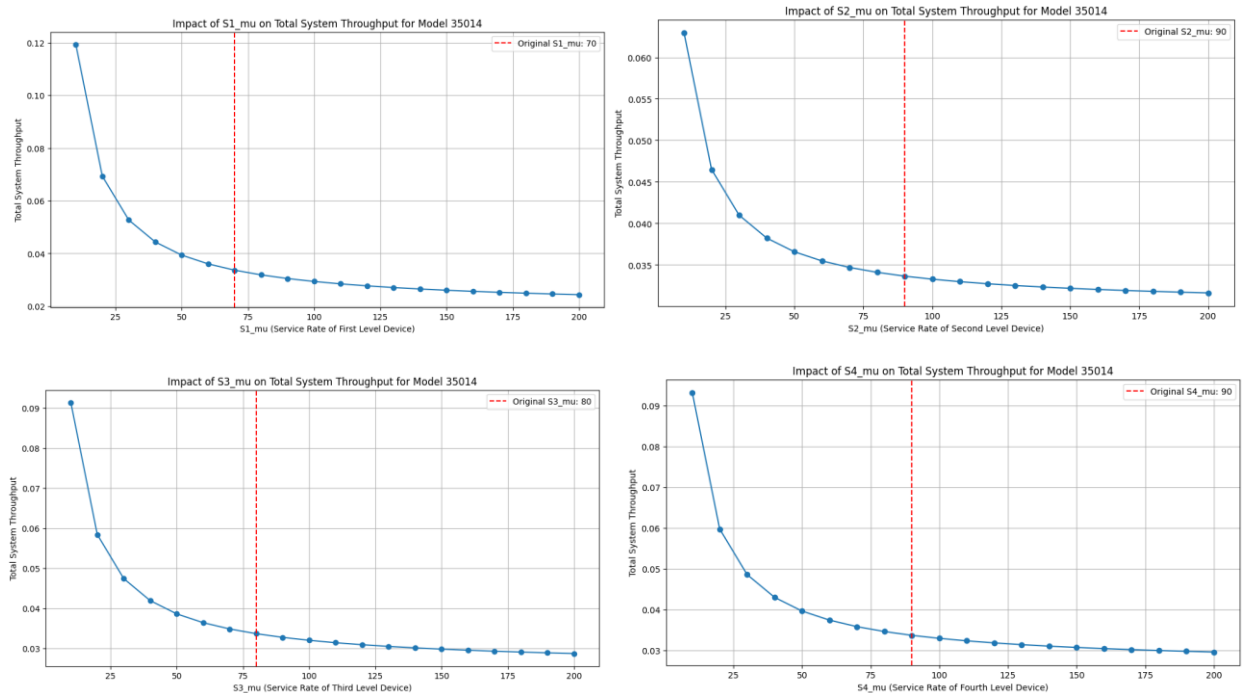


Рисунок 4.2 - Пропускна здатність за рівнями ХА для моделі 35014

Для прикладу сформовані результати для моделі 35014 з набору 810000 (рис. 4.2). Перший графік для рівня 1 показує, що зі збільшенням  $S1\_mu$  (швидкості обслуговування пристроїв першого рівня) загальна пропускна здатність системи також зростає, але зі зменшенням віддачі. Спочатку збільшення  $S1\_mu$  призводить до значного стрибка пропускної здатності. Однак, після певної точки крива вирівнюється, що вказує на те, що інші "вузькі місця" в системі починають обмежувати загальну пропускну здатність.

Отже, збільшення швидкості обробки пристрою першого рівня ( $S1\_mu$ ) загалом покращує загальну пропускну здатність системи. Однак існує точка зменшення віддачі, коли подальше збільшення до  $S1\_mu$  не призведе до суттєвого покращення загальної продуктивності системи, оскільки інші частини системи стають новими "вузькими місцями". Такі спостереження також можна бачити на інших графіках по кожному рівню системи.

За допомогою функції `calculate_throughput_for_model` проаналізовано вибірку моделей та ідентифіковано найкращі та найгірші моделі за показником загальної пропускної здатності.

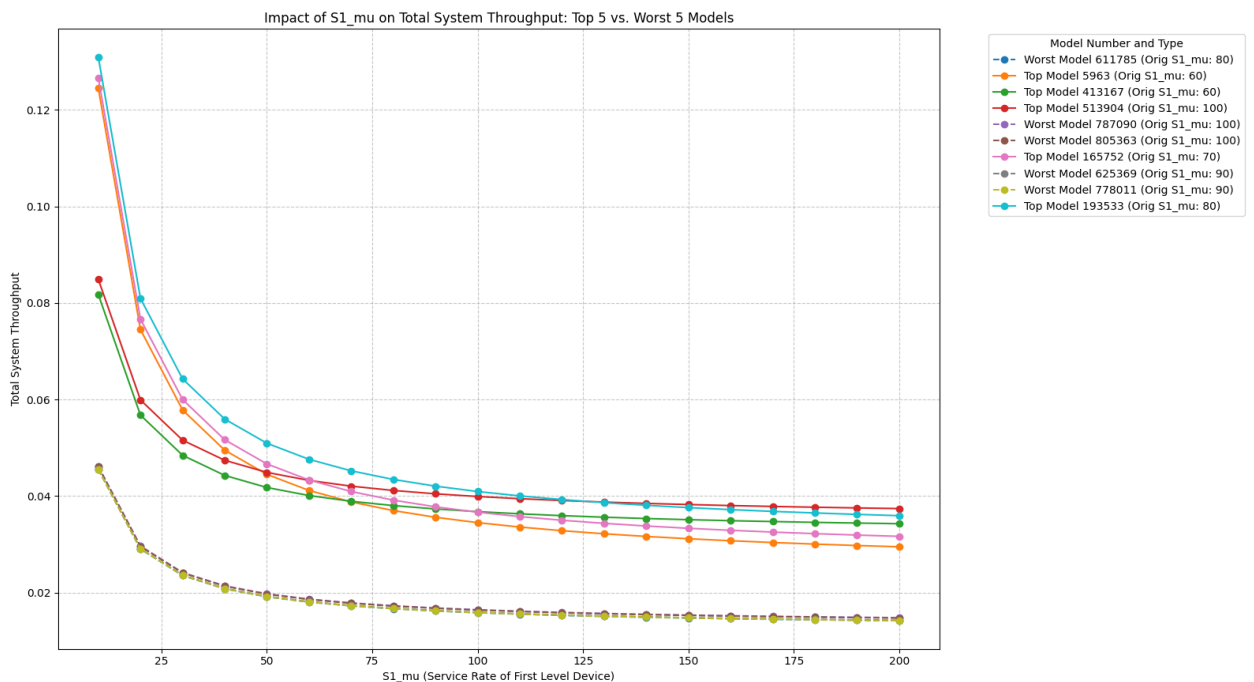


Рисунок 4.3 - Вплив  $S1\_mu$  на загальну пропускну здатність системи

На рисунку 4.3 показано вплив  $S1\_mu$  на загальну пропускну здатність системи як для 5 найкращих, так і для 5 найгірших моделей, які були виявлені при випадковому виборі обраного набору моделей. Для всіх моделей існує точка зменшення віддачі зі збільшенням  $S1\_mu$ , де інші "вузькі місця" в системі стають обмежувальним фактором. Різниця між «Найкращими» та «Найгіршими» моделями говорить про те, що крім  $S1\_mu$ , існують інші структурні або параметричні відмінності (наприклад,  $mu$  інших рівнів, кількість пристроїв, канали або логіка маршрутизації), які впливають на загальну продуктивність і створюють "вузькі місця" в менш ефективних моделях.

Отже, "вузькі місця" пропускну здатності можуть бути динамічними, зміна одного параметра (наприклад, збільшення  $S1\_mu$ ) може перенести "вузьке місце" на інший компонент системи, для оптимізації загальної пропускну здатності необхідно ідентифікувати та усунути найкритичніші "вузькі місця", можливо, шляхом збільшення потужності або ефективності інших рівнів, пристроїв або каналів.

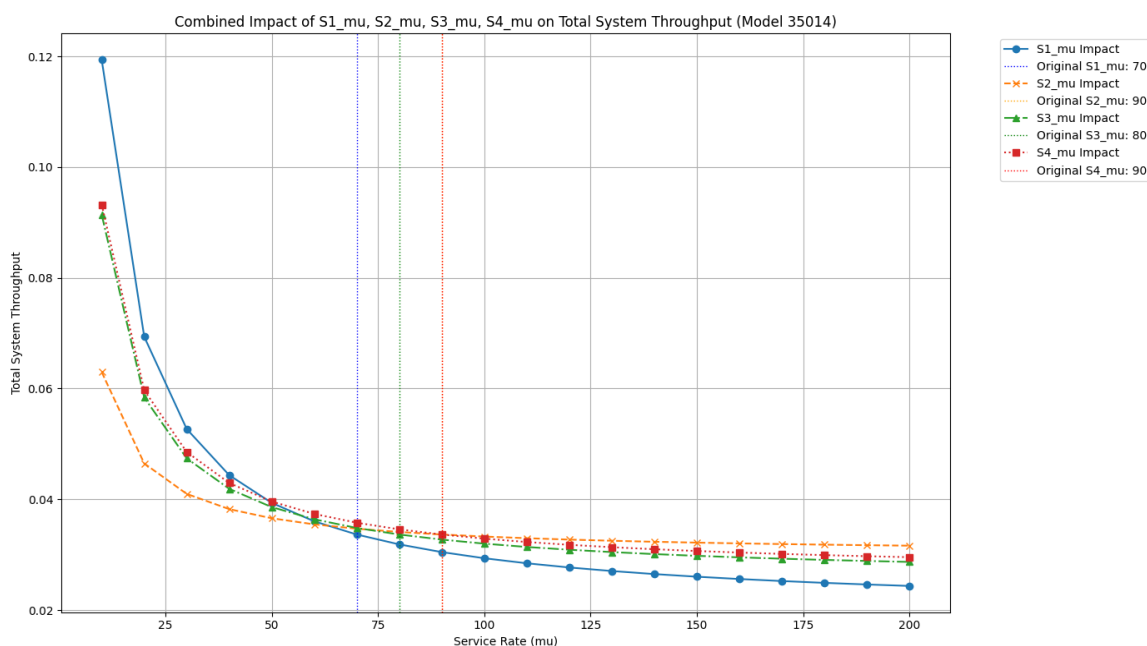


Рисунок 4.4 - Комбінований графік чотирирівневої архітектури хмарної системи (модель 35014)

На комбінованому графіку (рис. 4.4.) спостерігається спільна тенденція зменшення відгуку. Для всіх рівнів ХА спостерігається чітка закономірність того, що після досягнення певного порогу подальше збільшення потужності на цьому конкретному рівні не дає значного приросту загальної продуктивності, оскільки "вузьке місце" переміщується на іншій компонент системи. Наявні відмінності в початковій чутливості щодо сили впливу кожного рівня на пропускну здатність при низьких значеннях  $\mu$ . Наприклад,  $S1_{\mu}$  та  $S4_{\mu}$  можуть демонструвати більш різкий підйом пропускну здатності на початку порівняно з  $S2_{\mu}$  та  $S3_{\mu}$ , що може свідчити про те, що вони частіше є початковими "вузькими місцями". Пунктирні вертикальні лінії (рис. 4.4) показують оригінальні значення  $\mu$  для кожного рівня. Можна помітити, що для деяких рівнів оригінальне  $\mu$  знаходиться в ділянці, де крива вже вирівнюється, тоді як для інших (особливо  $S1_{\mu}$ , де крива продовжує зростати після оригінального значення) є потенціал для подальшого покращення пропускну здатності за рахунок збільшення  $\mu$  цього рівня.

Цей комбінований графік наочно демонструє, що оптимізація пропускну здатності системи є багатофакторною задачею. Збільшення потужності одного компонента понад певну точку може виявитися неефективним, якщо інші компоненти стануть новими обмежувальними факторами. Це підкреслює важливість комплексного аналізу та балансування ресурсів між різними рівнями системи.

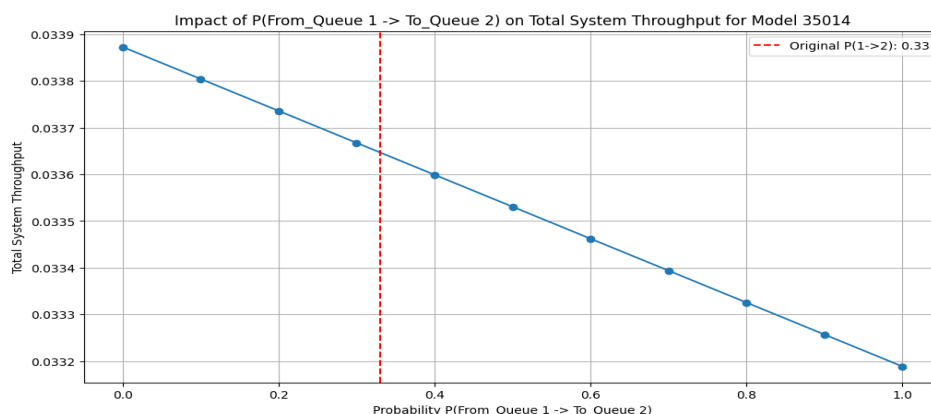


Рисунок 4.5 - Критичний рівень (підсистема 1) моделі 35014

Для моделі 35014, найбільш критичним рівнем є Рівень 1 (рис. 4.5). Цей аналіз підкреслює важливість не лише швидкості обслуговування окремих компонентів ( $S_{mu}$ ), а й ефективності маршрутизації запитів/завдань між ними. Неоптимальна маршрутизація може створити "вузькі місця", навіть якщо окремі рівні мають високу швидкість обробки. Вирішенню питання перерозподілу запитів/завдань на пристроях кожного рівня запропоновано у розподіленій моделі побудови стратегії розвантаження (п.3.4).

Збільшення ймовірності  $P(From\_Queue\ 1 \rightarrow To\_Queue\ 2)$  призвело до зменшення загальної пропускної здатності системи. Це свідчить про те, що  $S2_{mu}$  рівень для моделі 35014 є відносно слабким місцем або має обмежену потужність. Направлення більшої кількості запитів/завдань до цього рівня створює надмірне навантаження і обмежує загальну продуктивність.

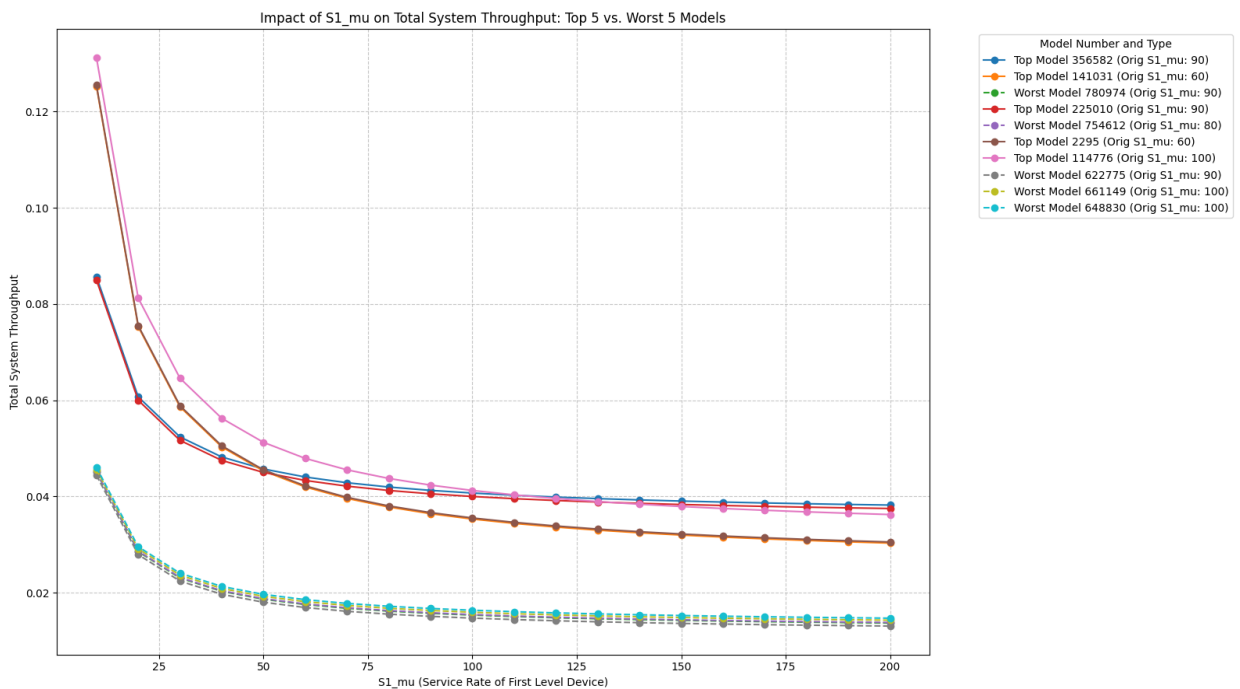


Рисунок 4.6 - Вплив  $S1_{mu}$  на загальну пропускну здатність системи

На рисунках 4.6 та 4.7 зображено поведінку найкращих та найгірших моделей, які були виявлені при випадковому виборі з 810000 варіантів. Для моделі 810000 параметри  $S1_{mu}$ , так і  $S4_{mu}$  демонстрували більш різкий

початковий підйом пропускної здатності, що вказує на їхню роль як потенційних початкових "вузьких місць".

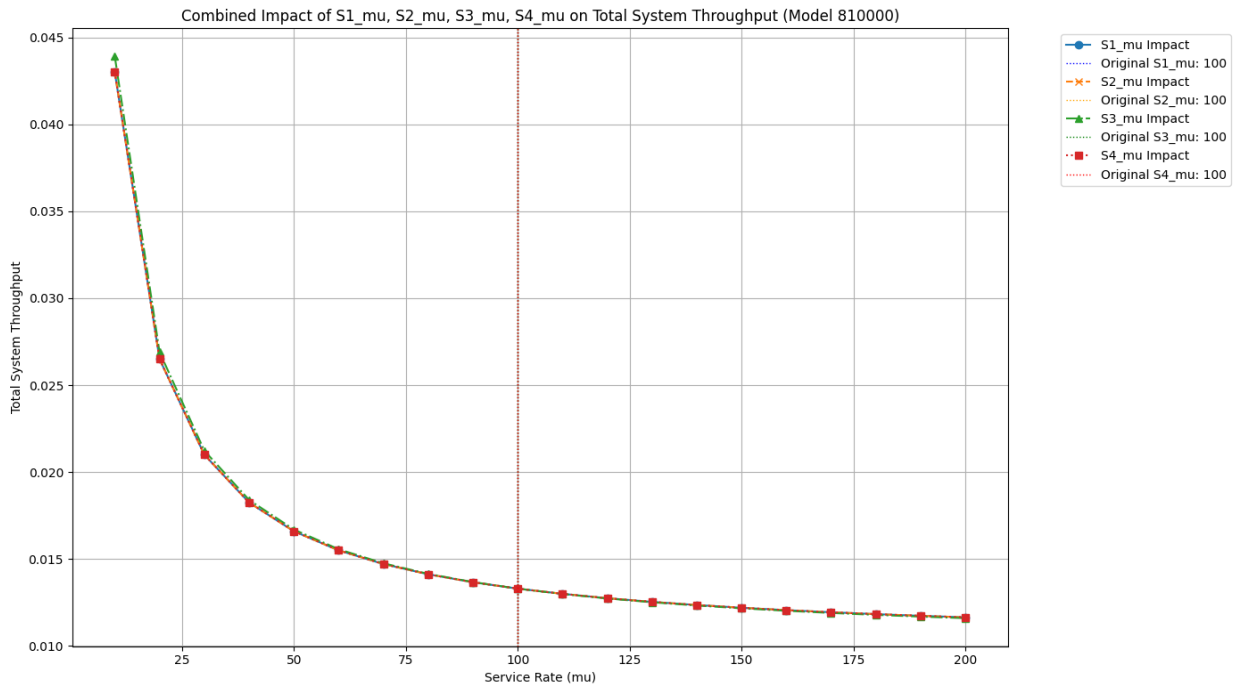


Рисунок 4.7 - Комбінований графік чотирирівневої архітектури хмарної системи (модель 810000)

На початку дослідження була висунута гіпотеза щодо існування порогу (який приблизно дорівнює 50%) зв'язку між вхідним потоком ( $\lambda_0 = 100$ ) запитів/завдань та швидкістю обробки (10-100), що і підтверджують надані результати експериментів.

Проведені дослідження в більшій мірі підтвердили нашу гіпотезу щодо існування "вузького місця" на першому рівні хмарної системи. Для всіх рівнів спостерігався ефект зменшення віддачі. Збільшення швидкості обслуговування ( $\mu$ ) спочатку різко покращувало пропускну здатність, але після певного порогу приріст ставав незначним, що вказує на переміщення "вузького місця" до інших компонентів системи.

## 4.2 Імітаційні моделі блоків СХмОбч

Для ефективного керування потоками запитів, забезпечення взаємодії компонентів (Service), виділення обчислювальних ресурсів (Pod) та еластичного розподілу вхідного трафіку (Application Load Balancer) було розроблено узагальнену структурну схему хмарної архітектури (рис. 3.2). Для моделювання практичних експериментів розглянуто архітектуру з інгрес-контролерами (Ingress) [108], на базі яких сформовано масиви аналітичних даних для різних сценаріїв навантаження. З метою поглибленого дослідження ефективності зазначених інженерних рішень було застосовано методи імітаційного моделювання [109].

Результатами роботи [40] є оцінка доцільності вибору ХА та ефективності використання блоків на різних наборах вхідних даних. На цьому етапі дослідження побудовано імітаційні моделі роботи вузлів хмарної системи (як окремих програмних модулів хмарної архітектури), які використовуються для вирішення різних типів задач.

### 4.2.1 Програмні експерименти роботи першого блоку

Перший програмний модуль реалізує функцію генерації вхідного потоку запитів із заданою інтенсивністю протягом визначеного часового інтервалу від  $Start\_work\_node$  до  $(Start\_work\_node + h)$  з врахуванням інтервалів між їх послідовним створенням  $t\_krok$ , та тривалості їх оброблення  $Processing\_time\_request$ .

Досліджено сценарії, за яких початок обробки запиту на обчислювальному вузлі фіксується з моменту  $Start\_receive\_request$  (наприклад, використання стратегії попереднього запису).

Конфігурація віртуального вузла може передбачати фіксовану кількість  $Number\_Node$  вузлів та кількості додаткових вузлів  $Additional\_node$ , які

здійснюють обробку запитів відповідно до заданого закону розподілу ймовірностей.

Алгоритм моделювання передбачає, що у разі неможливості реалізації цільової стратегії вибору  $i$ -го вузла, запит класифікується як загальний і перенаправляється до вузла з максимальним значенням доступного (вільного) часу. Для верифікації моделі генерація елементарних подій здійснюється за рівномірним або експоненціальним законами розподілу. Функціонал першого програмного модуля дозволяє конструювати сценарії функціонування віртуальних вузлів з метою оцінювання таких метрик, як середня й максимальна довжина черги, часові характеристики очікування та обслуговування, а також обсяг втрачених (відхилених) запитів. Також можна додатково визначити кількість запитів на моменти часу ( $Start\_work\_node + h$ ) і  $Start\_receive\_request$  та середній час роботи віртуального вузла на різних наборах вхідних даних (табл. 4.1).

Таблиця 4.1 - Результати проведення експериментів імітації роботи першого блоку

Кількість вузлів (Number_Node)	Кількість додаткових вузлів(Additional_node)	Час початку роботи вузла (Start_work_node)	Час початку оброблення запитів (Start_receive_request)	Інтервал часу для оброблення запитів (Processing time request)	Ймовірності активності клієнтів	Ймовірності появи інтервалів між запитами	Середня довжина черги	Максимальна довжина черги	Середній час очікування	Максимальний час очікування	Мінімальний / максимальний час оброблення запиту	Кількість оброблених запитів	Кількість запитів (всього)	Ознак перевагання	
1	0	0	0	0	{0.6262, 0.2347, 0.1391}	{0.5179, 0.1832, 0.1201, 0.0726, 0.1062}	496.19	989	1007.08	1991.00	4.52 / 6.00	796	1785	+	
1	1	0	0	0			490.14	1002	990.83	1985.00	4.5 / 6.00	799	1801	+	
1	1	0	0	0.5			487.90	973	1004.43	1958.00	4.43 / 6.00	812	1785	+	
1	1	0	0	1			490.36	971	999.15	1997.00	4.47 / 6.00	805	1776	+	
2	1	0	0	0			100.42	204	198.08	404.00	4.5 / 6.00	1597	1796	+	
2	1	0	0	0.5			88.52	172	179.52	371.00	4.51 / 6.00	1592	1763	+	
2	1	0	0	1			86.54	171	175.08	347.00	4.48 / 6.00	1605	1776	+	
0	1	1	0	0				876				877	1753	+	
1	1	1	0	0				480.38	952	975.51	1960	4.46 / 6.00	807	1759	+
1	1	1	0	1				477.71	950	974.98	1960	4.49 / 6.00	800	1750	+

При моделюванні використана черга (без пріоритетів).

На основі результатів дослідження першого блоку системи проведено порівняльний аналіз декількох конфігурацій із різною кількістю основних та додаткових обчислювальних вузлів.

Під час комп'ютерного моделювання за умов функціонування 10 каналів обслуговування та надходження 1000 запитів протягом 300 секунд (5 хвилин) при синхронізації моментів прибуття вимог та початку їх обробки, система продемонструвала високу ефективність. Середній інтервал між надходженнями, що становить 0,3 с, забезпечує стабільне оброблення вхідного трафіка. Середній час очікування в черзі варіюється в межах 23–24 с, тоді як максимальна тривалість перебування в черзі не перевищує 47 с. Отримані значення свідчать про достатню пропускну здатність архітектури для обслуговування вхідного потоку, попри наявність прогнозованих затримок. Порівняльний аналіз рівномірного та експоненціального генераторів подій виявив подібні тенденції. Застосування експоненціального закону розподілу зумовлює незначне зростання середнього та пікового часу очікування, проте загальна динаміка системи залишається стаціонарною. Обидва підходи адекватно відтворюють стохастичний процес надходження запитів, підтверджуючи роботу системи в межах допустимого навантаження

За умови моделювання аналогічних початкових параметрів, але за відсутності часової синхронізації надходження запитів, кумулятивне накопичення черги фіксується вже на початковому етапі функціонування. Цей експеримент підтверджує неефективність ініціації прийому вимог задовго до початку їх фактичного обслуговування, що призводить до значних часових затримок, попри високу швидкодію обчислювальних вузлів після їх безпосереднього запуску.

Результати проведених експериментів підтверджують, що побудована імітаційна модель вузла першого типу адекватно відтворює динаміку багатоканальної мережі масового обслуговування з необмеженою чергою. Застосований підхід перенаправлення запитів до каналу з максимальним значенням доступного часу забезпечує раціональний розподіл навантаження, що підтверджується рівномірною завантаженістю каналів на графіках (рис. 4.8). Відсутність істотних розбіжностей у ключових метриках між

рівномірним та експоненціальним законами розподілу підтверджує стійкість та передбачуваність досліджуваної системи.

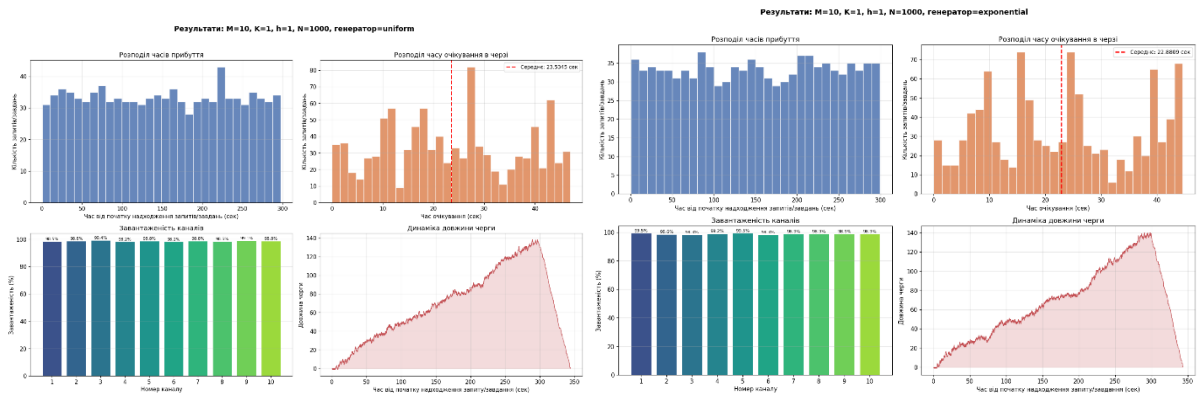


Рисунок 4.8 - Візуалізація результатів роботи вузла першого типу з рівномірним/експоненціальним генераторами

Але залишається відкритим питання часу початку використання додаткових вузлів та тривалості обслуговування ними пікового навантаження.

#### 4.2.2 Програмні експерименти роботи другого блоку

Другий блок програмного додатку формує імітаційну модель випадкової появи користувачів з різною кількістю запитів/завдань до віртуального вузла в режимі реального часу (при умові, що  $Start\_receive\_request = Start\_work\_node$ ) при використанні черги з пріоритетами (як додаткового вузла) обмеженого розміру  $q$ , пріоритети якої формуються за рахунок мінімальної кількості запитів/завдань від користувача).

При моделюванні при  $m \geq 10$  використана залежність кількості оброблених запитів/завдань кожного користувача  $k$  від кількості користувачів  $m$  ( $k = 2^m, q = 3^m$ ).

При моделюванні часу оброблення запиту/завдання використаний рівномірний розподіл з можливими значеннями 1, 2, 3 і 4 секунд. Інтервали між запитами/завданнями для перевірки адекватності моделі визначені  $\{0, 3, 5\}$  секунд з розподілом ймовірностей  $\{0.4, 0.3, 0.3\}$  відповідно.

Таблиця 4.2 Результати проведення експериментів імітації роботи другого блоку

Кількість вузлів	Кількість користувачів ( $m$ )	Запланований розмір / Максимальний розмір черги ( $q$ )	Кількість / Максимальна кількість оброблених запитів від кожного користувача	Середня кількість запитів на користувача	Загальний час імітації моделі (сек.)	Середній час очікування (сек.)	Кількість відхилених запитів	Кількість запитів (всього)	Ознаки перевантаження
1	10	3	$2^{10}$	1024	7425	6.83	7472	10240	+
1	10	3	$2^{10}$	1024	7193	6.84	7563	10240	+
1	12	$3^{12} / 15$	$2^{12}$	4096	34836	39.02	36177	49152	+
1	15	$3^{15} / 12$	$2^{15}$	32768	347062	30.96	362080	491520	+
1	15	10 / 10	10	32768	344965	25.6	362080	491520	+

Високий відсоток відхилення запитів, який у середньому становить близько 70 %, безпосередньо свідчить про стан перевантаження системи та середній час очікування досить значний в порівнянні з часом оброблення запитів. Довжина черги повністю пропорційна кількості користувачів, які надсилають запити/завдання ( $m \approx q$ ).

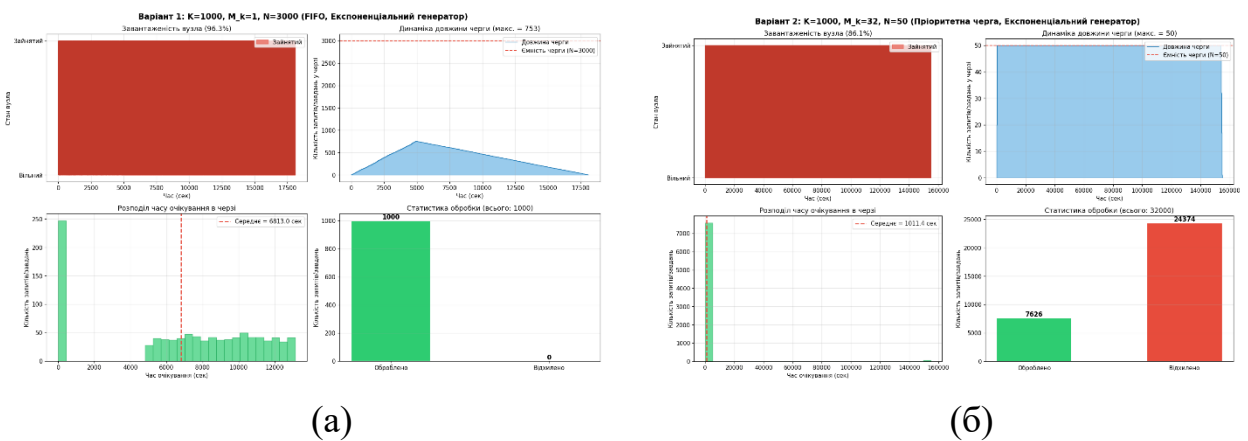


Рисунок 4.9 - Візуалізація результатів проведення експериментів:

(a)  $M_k = 1$ , (б)  $M_k = 32$

При детальнішому розгляді експерименту моделювання роботи вузла другого типу, де 1000 користувачів генерують по 1 запиту, а ємність черги

становить 3000 запитів/завдань, - видно сильну перенавантаженість системи (рис. 4.9 - а). Середній час надходження запиту становить 4.8 сек, тоді як середній час його обробки — 17.5 сек. Інтенсивність надходження приблизно в 4 рази перевищує інтенсивність обслуговування. Вузол працює практично безперервно, його завантаженість становить близько 87-88%. Рівномірний та експоненціальний генератори показали майже ідентичні результати щодо всіх метрик, це доводить, що при такому високому рівні завантаження вузла тип базового розподілу не відіграє вирішальної ролі.

При детальному розгляді експерименту, де моделюється система з 1000 користувачів, кожен з яких генерує по 32 запити/завдання, розглянуто чергу ємністю 50 з використанням пріоритету щодо мінімальної кількості запитів/завдань (рис. 4.9 – б). Через високе навантаження на вузол, відсоток відхилених запитів залишається високим (близько 76% для обох генераторів), що свідчить про те, що система все ще є сильно перевантаженою. На відміну від попереднього експерименту ( $M_k = 1$ ) і пріоритет не мав значення, у цьому варіанті ( $M_k = 32$ ) механізм пріоритетної черги (заснований на мінімальній кількості запитів від користувача) починає працювати. Хоча це не зменшує загальний відсоток втрат, він впливає на те, які запити та від яких користувачів обробляються, намагаючись забезпечити більш рівномірне обслуговування між користувачами, які мають менше вже згенерованих запитів. Як і в попередньому порівнянні експериментів першого блоку, рівномірний та експоненціальний генератори показують дуже схожі результати щодо загальних метрик (відсоток відхилень, завантаженість вузла, середній час очікування), це підтверджує, що при такому рівні перевантаження, тип розподілу не є вирішальним фактором.

Усі проведені експерименти яскраво демонструють класичну проблему СМО — якщо інтенсивність вхідного потоку  $\lambda$  (1 запит на 4.8 сек) значно перевищує інтенсивність обслуговування  $\mu$  (1 запит на 17.5 сек), система неминуче деградує. Обмежена черга може тимчасово затримувати запити, але

при її переповненні ( $M_k = 32$ ) це призводить до значної втрати запитів (понад 67% при  $M_k = 32$ ). Визначимо це, як проблему "вузького місця" для подальшого дослідження структури хмарної системи. Використання пріоритетної черги в експерименті ( $K = 1000, M_k = 32, N = 3000$ ) довело свою ефективність у задачах балансування ресурсів. Хоча пріоритет не вирішив проблему загальних втрат (це неможливо фізично без збільшення швидкості обробки), він забезпечив більш справедливий розподіл процесорного часу між різними користувачами, не даючи жодному з них повністю "захопити" вузол, рівномірно відхиляючи та обробляючи запити (наскільки це можливо за умов 67-68% відмов) від усіх тисячі користувачів.

Розроблені модулі імітаційної моделі дозволяють моделювати поведінку системи оброблення запитів/завдань та надає статистику для подальшого аналізу навантаження на систему. Рекомендаціями щодо нормального функціонування вузла другого типу (щоб відсоток відхилень був близьким до 0) є або збільшення швидкості обробки запитів мінімум у 4 рази (оптимізація ПЗ/сервера), або розпаралелювання вузла, додавши ще щонайменше 3 канали обслуговування.

### 4.2.3 Тактичне планування

Тактичне планування (запуск моделі як мінімум  $n = 100$  разів з різними *seed*) є критично важливим етапом експерименту. Оскільки імітаційна модель є стохастичною (випадковою), кожен її прогін дає дещо відмінний результат. Запуск визначеної кількості прогонів дозволяє отримати більш точну та надійну оцінку відгуку.

Використовуючи метод усереднення для блоку другого типу, отримано середню ймовірність відмови 0.6757 зі стандартним відхиленням 0.0081 і це підтверджує, що 100 прогонів є достатньою кількістю для отримання

статистично достовірних результатів у кожній точці факторного плану, усуваючи вплив випадкових коливань окремих прогонів.

Розраховані коефіцієнти лінійної регресії дозволили побудувати математичну модель у кодованих змінних ( $x_1$  – інтервал появи запиту/завдання,  $x_2$  – час обслуговування запиту/завдання):

$$\hat{y} = 0.6642 - 0.0569 \cdot x_1 + 0.0569 \cdot x_2 + 0.0113 \cdot x_1 x_2$$

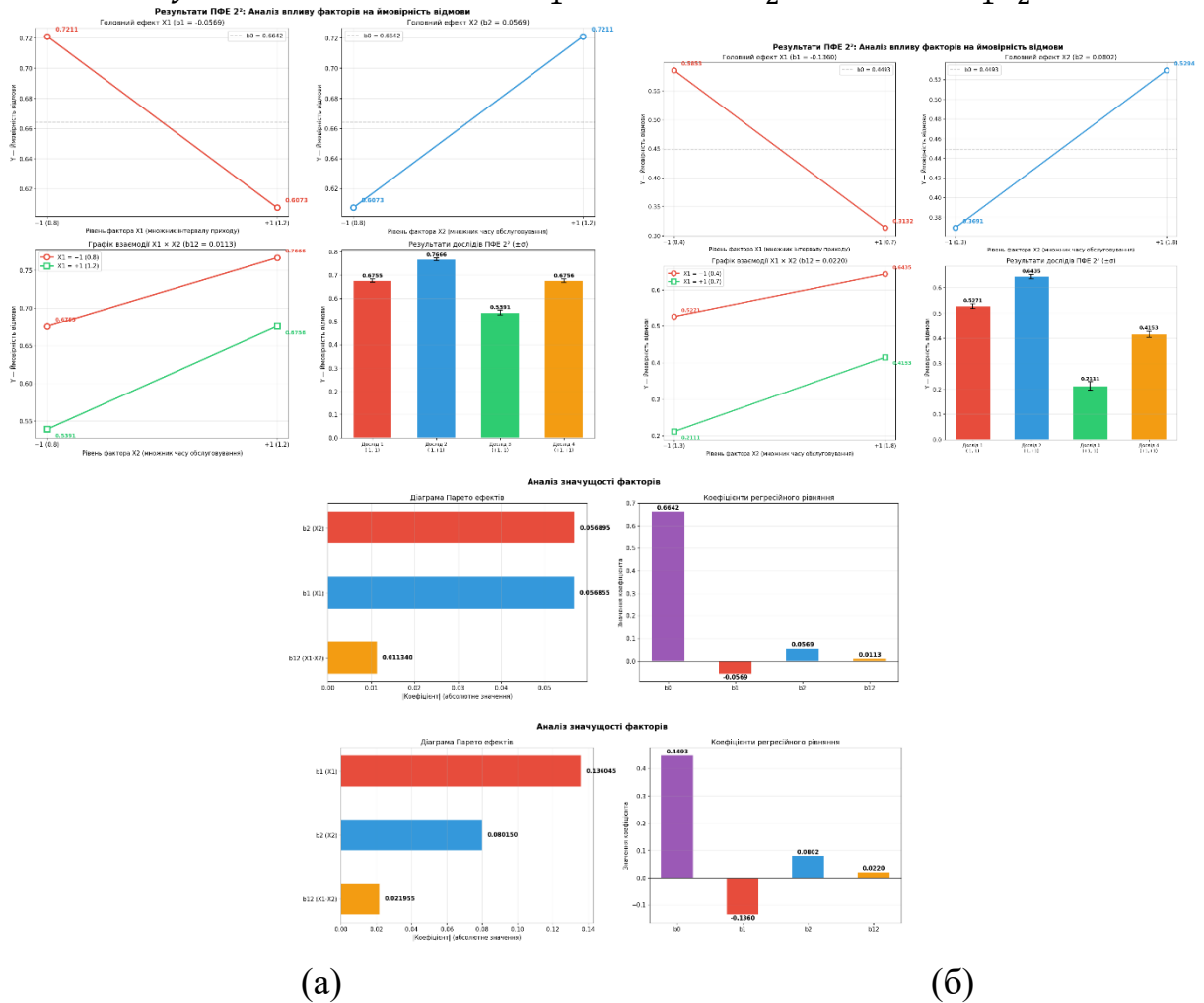


Рисунок 4.10 - Результати повного факторного експерименту для блоку другого типу: (а) при  $M_k = 1$ , (б) при  $M_k = 5$ .

Перевірка адекватності показала абсолютний збіг між експериментальними значеннями  $y$  та змодельованими  $\hat{y}$  ( $\Delta = 0$ ,  $R^2 = 1.0$ ), що є класичною властивістю повного факторного експерименту  $2^k$ , де кількість ступенів свободи моделі (4 коефіцієнти) дорівнює кількості дослідів

(4 точки). Математична модель ідеально інтерполює відгук у межах обраного факторного простору (рис. 4.10 - а).

Коефіцієнт  $b_0 = 0.6642$  побудованого регресійного рівняння свідчить про те, що система знаходиться у стані глибокого перенавантаження (у центрі плану система відхиляє понад 66% запитів). Коефіцієнт  $b_1 = -0.0569$  фактору  $x_1$  означає, що при зростанні інтервалу появи (клієнти з'являються рідше), ймовірність відмови зменшується приблизно на 5.7%. Коефіцієнт  $b_2 = 0.0569$  фактору  $x_2$  є додатнім, тому збільшення часу обробки одного запиту/завдання погіршує пропускну здатність вузла і збільшує ймовірність відмови на 5.7%. Ефект синергії між двома факторами присутній ( $b_{12} = 0.0113$ ), але він приблизно у 5 разів слабший за вплив головних факторів (рис. 4.4).

Отже, для покращення якості обслуговування (зменшення відмов) оптимізація швидкості вузла дасть такий самий ефект, як і штучне обмеження інтенсивності вхідного потоку (наприклад, введення лімітів на кількість звернень від користувачів в секунду). Однак, враховуючи величезний базовий відсоток відхилень ( $> 66\%$ ), зміна факторів у межах  $\pm 20\%$  (діапазон експерименту) не здатна повністю вирішити проблему і вузлу потрібна фундаментальна модернізація (багатоканальність).

Тактичне планування було проведено для блоку другого типу з обмеженою чергою та кількістю каналів обслуговування  $M_k = 5$  (рис. 4.10 – б) та підтвердило, що зі збільшенням кількості каналів ймовірність відмови різко зменшується і починаючи з певної кількості каналів (для останнього наведеного експеримента – це 17 каналів), ймовірність відмови стабілізується на нульовому або дуже низькому рівні, що вказує на достатню пропускну здатність системи для заданого навантаження.

Проведено 81 окремий ПФЕ  $2^2$  експеримент, кожен з яких дав унікальний набір коефіцієнтів регресії та діапазон ймовірності відмови. Комбінації з низькими значеннями  $x_1$  та високими  $x_2$  (наприклад,  $x_1 \in$

$\{0.2, 0.4\}$ ,  $x_2 \in \{1.6, 1.9\}$ ) призводять до значного перевантаження системи та високої кількості відмов у.

Побудовано програмний модуль користувацького сценарію для оцінки впливу кількості каналів на ймовірність відмови вузла від обраних інтервалів надходження та часу обслуговування. Ці результати є ключовими для оптимізації системи, дозволяючи визначити оптимальні конфігурації для підтримки бажаного рівня обслуговування при різних навантаженнях, а також демонструють, наскільки критично важливим є баланс між навантаженням та ресурсами системи.

Отримані результати підтверджують необхідність ретельного проєктування інфраструктури, що вимагає досягнення балансу між ефективністю обробки запитів, часом очікування на допоміжному вузлі (інтеграція якого становить предмет подальших досліджень) та пропускнуою здатністю основних вузлів системи. Зокрема, практична реалізація цього модуля дозволяє конструювати сценарії функціонування вузла для визначення доцільності структурної реконфігурації загальної архітектури хмарного середовища.

### **4.3 Підготовка даних для імітації хмарного навантаження**

Для побудови точної моделі прогнозування навантаження на архітектурні рівні хмарної системи необхідно здійснити збір та обробку різних типів даних. Ключові масиви інформації для подальшого аналізу було відібрано з відкритих джерел, які містять колекції наборів даних DevOps [110] і спрямовані на підтримку досліджень у сфері штучного інтелекту для DevOps. Зокрема, використано дані трасування робочого навантаження кластерів Google, Alibaba та Microsoft Azure, логи та траси для детекції аномалій у мікросервісах, а також контрольні показники навантаження.

### 4.3.1 Набір даних UNSW-NB15

Для оцінки навантаження на систему обрано датасет UNSW-NB15, призначений для вдосконалення систем виявлення вторгнень (IDS). Цей набір даних, створений у 2015 році в лабораторії Cyber Range Lab (Австралійський центр кібербезпеки), ключовою особливістю якого є гібридний склад, що охоплює як сучасну повсякденну мережеву активність, так і штучно створені вектори атак [111, 112]. Датасет UNSW-NB15 позиціонується як актуальна альтернатива класичним наборам даних, таким як KDD99 та NSL-KDD, оскільки він релевантніше відображає динаміку сучасного мережевого трафіку та специфіку новітніх кіберзагроз. Завдяки високій репрезентативності цей масив даних є базою для апробації алгоритмів машинного та глибокого навчання (зокрема Random Forest, SVM та нейронних мереж) у задачах забезпечення інформаційної безпеки.

Набір даних UNSW-NB15 налічує 2540047 записів (близько 100 ГБ необроблених даних), який характеризується 49 ознаками (наприклад, тривалість з'єднання, протокол, кількість байтів тощо) та доступний у вигляді сирих пакетів (Pcap), файлів логів (Bro, Argus) та готових таблиць (CSV). Для зрозуміння структури даних та їх властивостей узагальнено інформацію та описову статистику для числових і категоріальних змінних. Для забезпечення коректної роботи з даними виконано перетворення типів даних. Колонка *Stime* (час початку) була визначена як змінна часу, перетворена в тип *datetime* та виведена описова статистика. Обчислено інтервали прибуття між послідовними записами *Stime*. Ці інтервали були згруповані в 100 бінів, і були створені масиви *ARRIVAL\_INTERVALS* (середні точки бінів) та *ARRIVAL\_PROBS* (нормалізовані частоти). Додатково проведено аналіз та візуалізація розподілу інтервалів прибуття, а також фільтрація інтервалів у діапазоні від 0 до 0.3 секунд з подальшим перерахунком *ARRIVAL\_INTERVALS* та *ARRIVAL\_PROBS* для цього діапазону. Масиви *ARRIVAL\_INTERVALS* та

*ARRIVAL\_PROBS* були успішно збережені у файли для подальшого використання.

### 4.3.2 Набір даних CICIoT2023

Набір даних CICIoT2023 (або CICIoT23) був розроблений фахівцями Канадського інституту кібербезпеки (CIC), що діє при Університеті Нью-Брансвік (UNB). Датасет CICIoT2023 характеризується високим рівнем деталізації та масштабності, охоплюючи понад 46,6 мільйона записів мережевих подій [113]. Структура даних включає 47 інформативних ознак, отриманих шляхом аналізу трафіку зі 105 фізичних IoT-пристроїв. Класифікаційна модель набору охоплює 33 різновиди мережевих загроз, систематизованих у сім ключових категорій (DDoS, DoS, Reconnaissance, Web-based, Brute Force, Spoofing та Mirai). Важливою особливістю є наявність як необроблених pcap-файлів, так і структурованих CSV-таблиць, що дозволяє використовувати набір для глибокого навчання нейромереж.

До основних груп змінних входять мережеві характеристики пакетів (наприклад, *Header\_Length*, *Protocol\_Type*, *Duration*, *Rate*), прапорці TCP (*fin\_flag\_number*, *syn\_flag\_number* тощо), протоколи прикладного та мережевого рівнів (такі як HTTP, TCP, UDP), а також статистичні показники потоку (*Min*, *Max*, *AVG*, *Std*).

У датасеті CICIoT23 відсутня класична змінна "*Timestamp*" (точна дата та час). Для аналізу часових характеристик або інтервалів прибуття запитів, замість цього використовуються змінні *Duration* (тривалість потоку) та *Rate* (швидкість передачі пакетів). Для моделювання двох типів інтервалів прибуття (*IAT*) використано:

- обчислення  $1 / Rate$ ;
- експоненційний розподіл, використовуючи  $1/Rate$  як параметр  $\lambda$ .

Використання логарифмічних бінів (англ. logarithmic binning) зазвичай потрібне для візуалізації розподілів з «важкими хвостами» (наприклад, коли багато дрібних значень і дуже мало величезних, як у мережевому трафіку). Заміна лінійних бінів на логарифмічні дозволяє побачити структуру в «хвості» (у датасетах аномалії часто ховаються в екстремальних значеннях, які при звичайних бінах просто злипаються в одну лінію) та вирівняти щільність. Отже, на основі розподілів з використанням логарифмічних бінів (англ. logarithmic binning) сформовано масиви *ARRIVAL\_INTERVALS* (центри бінів) та *ARRIVAL\_PROBS* (нормовані частоти). Для обох типів *IAT* було обрано 20 найменших інтервалів прибуття з ненульовими ймовірностями для подальшого збереження у файлах та їх використання при моделюванні.

Для ефективної обробки великих файлів без перевищення обсягу оперативної пам'яті було використано читання файлів частинами - розділення обробки великих файлів на менші частини (англ. Chunking).

#### 4.4 Моделювання роботи хмарної системи

Для імітації роботи чотирирівневої хмарної системи розроблено програмні модулі з підтримкою сформованих комбінацій (810000) архітектурних рішень з блоками першого/другого типу без/з перенаправленням запитів/завдань між вузлами відповідного рівня при врахуванні запропонованого рішення (3.1).

Для моделювання всієї хмарної системи, яка складається з кількох рівнів і пристроїв на кожному рівні, створено клас *SystemSimulation*, який об'єднує окремі вузли-пристрої (*ConsultingSimulation*) та керувати потоком запитів/завдань між ними з урахуванням маршрутизації. Черги в цій імітаційній моделі є необмеженими (рис. 4.11). Показник *Tasks/requests in the system (active)* відображає кількість запитів/завдань, які

були згенеровані, але ще не завершили свій повний цикл обробки та не покинули систему до моменту закінчення симуляції.

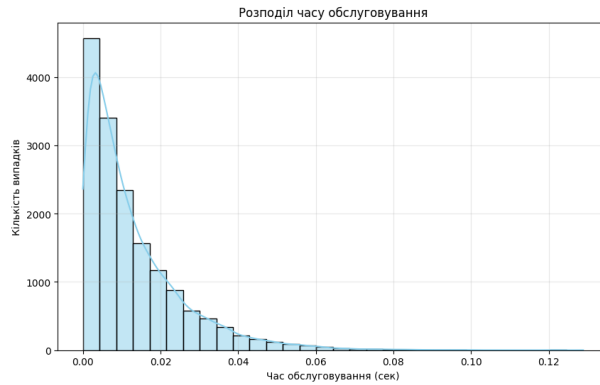


Рисунок 4.11 - Час обслуговування запитів/завдань в системі (без балансування, з необмеженими чергами)

На рисунку 4.12 наведено діаграму подій, яка відображає основні процеси та взаємодії між компонентами у моделі *SystemSimulation* та *ConsultingSimulation*.

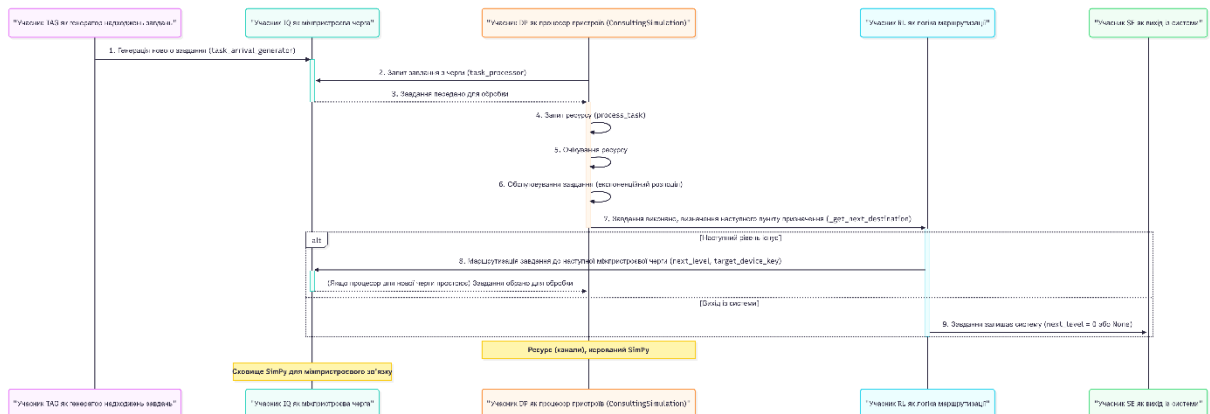


Рисунок 4.12 - Діаграма подій відображення основних процесів та взаємодій між компонентами у моделі *SystemSimulation* та *ConsultingSimulation*

Для наочності обрано модель 35014. Для зменшення обчислювальної складності зменшено інтервали для створення черг

$$ARRIVAL\_INTERVALS = [0.001, 0.002, 0.003, 0.004, 0.005]$$

$$ARRIVAL\_PROBS = [0.20, 0.20, 0.20, 0.20, 0.20]$$

Згенеровано велику кількість завдань (33 242), але лише частина (6 911) була оброблена протягом часових рамок моделювання (100 секунд), що свідчить сильну завантаженість системи,

## **4.5 Моделювання хмарного навантаження з різними сценаріями розгортання**

### **4.5.1 Експерименти розгортання хмарних сервісів**

Розглянуто моделі розгортання платформи хмарних обчислень та продемонстровано принципи організації хмарних технологій за способом їх взаємодії з користувачем (наприклад, SaaS, IaaS та ін.) на узагальненому прикладі. Робота зосереджена на емпіричному аналізі того, як вибір моделі впливає на використання запропонованих алгоритмів провайдерами та швидкість оброблення інформації (зокрема, запитів). Експерименти проведені на ряді сучасних пристроїв, включаючи ноутбуки та смартфони Apple для досягнення максимальної продуктивності на різних платформах.

З метою емпіричної оцінки ефективності моделей розгортання хмарних сервісів проведено моделювання експериментального сценарію з використанням симульованого тестування. Для цього було створено REST API-застосунок, який уявно розгортається в трьох основних хмарних середовищах: AWS, Microsoft Azure та GCP.

Оскільки реальні розгортання у трьох провайдерах вимагають великих витрат ресурсів, оцінювання проводилося з використанням навантажувального тестування API-застосунку через Postman Collection Runner та Apache JMeter у режимі імітації трафіку.

Метриками оцінювання обрані такі характеристики: час відповіді (response time), час відгуку під навантаженням (latency under load),

завантаженість CPU / Memory під час роботи, середня доступність сервісів (% uptime).

Проведення тестових експериментів з наступними параметрами:

- Тип запиту: GET /status;
- Кількість паралельних запитів: 100 / 1000 / 5000;
- Час відповіді: середній, мс;
- Відсоток помилок (HTTP 500, 502, 503).

Як умовне середовище обрано: Local (Docker) – базовий еталон, Simulated AWS, Microsoft Azure, GCP – моделювання на базі JMeter.

Результати моделювання хмарного навантаження демонструють, як система поводить себе при різних рівнях використання ресурсів. Це дозволяє оцінити продуктивність, масштабованість та стабільність хмарної інфраструктури, а також виявити потенційні "вузькі місця". Моделювання допомагає оптимізувати використання ресурсів, забезпечити надійність та знизити витрати.

*Таблиця 4.3 Результати проведення імітаційних експериментів у спеціально створених умовах при використанні інструменту JMeter*

Середовище	100 запитів	1000 запитів	5000 запитів	Помилки (%)
Docker local	45 мс	60 мс	120 мс	0%
AWS (simulation)	110 мс	140 мс	200 мс	1,5%
Microsoft Azure (simulation)	130 мс	160 мс	190 мс	1,0%
GCP (simulation)	100 мс	125 мс	160 мс	0,6%

При використанні методів імітаційного (симуляція з використанням спеціальних інструментів) та натурального моделювання (тестування системи на реальних навантаженнях у спеціально створених умовах) та використанні інструменту JMeter проведено оцінку хмарної структури при різних навантаженнях, визначено ресурси фізичної інфраструктури та

проаналізовано необхідність щодо додаткової масштабованості для забезпечення роботи застосунку (табл. 4.3).

Найкраща продуктивність за умовами навантаження була у GCP, який мав найнижчу середню затримку та найменший відсоток помилок. Microsoft Azure продемонстрував стабільну продуктивність, хоча час відповіді трохи вищий. AWS мав найбільше помилок при високому навантаженні, ймовірно через обмеження у конфігурації модулю балансування навантаження, яке використовує параметри налагоджування за умовчанням, що формує питання для подальшого розгляду модулю балансування та аналізу його алгоритмів. Локальний Docker працював швидко, однак не масштабований та не моделює реальне хмарне середовище.

#### **4.5.2 Про інтеграцію агентно-орієнтованих систем DevOps-підходами**

Використання моделей глибинного навчання (зокрема, LSTM-модуль для запам'ятовування значення як на короткі, так і довгі проміжки часу) та агенти штучного інтелекту надали змогу ефективно оцінювати переваги ресурсів і сховищ, що сприяє зменшенню затримок, оптимізації енергоспоживання та вартості обслуговування. Розглянуто алгоритм з методом розподілу ресурсів на основі агентної системи в хмарному середовищі з використанням методів навчання з підкріпленням [114], що демонструє покращення корисності ресурсів, доступності ресурсів та масштабованості ресурсів порівняно зі статичним розподілом ресурсів.

Експерименти проводилися з модельованими даними при використанні навантажувального тестування API-застосунку через Postman Collection Runner та Apache JMeter у режимі імітації трафіку при різних параметрах тестування (тип запиту, кількість паралельних запитів, час відповіді, відсоток

помилки). Метриками оцінювання моделі обрано час відповіді (response time), час відгуку під навантаженням (latency under load), завантаженість CPU/Memory під час роботи, середня доступність сервісів (% uptime). У якості умовного середовища обрано базовий еталон Local (Docker), моделювання на базі JMeter: Simulated AWS, Microsoft Azure, GCP.

Інтеграція агентно-орієнтованих систем DevOps-підходами (наприклад, автоматичним CI/CD у Kubernetes-кластерах) дозволяє реалізувати безперервну адаптацію інфраструктури до змін у навантаженні користувачів або сервісів [79, 86, 115-117]. Сучасні методи оптимізації хмарних інфраструктур, що базуються на агентному управлінні, аналізі даних і прогнозуванні, забезпечують підвищення продуктивності, зниження витрат і поліпшення якості сервісів. При вирішенні питання щодо оптимізації навантаження на інфраструктуру особливу увагу приділяють практичним результатам, які спрямовані на розвиток колективного навчання агентів (multi-agent reinforcement learning) для повної автономізації управління хмарними середовищами.

#### **Висновки до розділу 4**

У четвертому розділі в результаті проведеного дослідження проведено обґрунтоване розроблення програмного забезпечення при врахуванні результатів порівняння з існуючими програмними рішеннями.

Запропонована реалізація програмних модулів дозволяє комплексно оцінити ефективність розгортання хмарних систем, а також визначити шляхи для подальшої оптимізації та поліпшення архітектурного рішення.

Побудовано формалізовану модель *R\_system* чотирирівневої хмарної системи. Для аналізу впливу умов (2.1) - (2.5) на хмарну систему була обрана стратегія її представлення у двох виглядах (рис. 2.7 та рис. 3.2).

Сформовано достатньо великі набори різних комбінацій чотирирівневих хмарних систем без повторення для імітації створення пристроїв на кожному рівні з різними кількостями каналів та інтенсивностями обслуговування при визначеному потоці запитів/завдань, кількість даних яких обумовлено було потужностями обчислювальних пристроїв, які приймали участь в експериментах.

Проведено програмні експерименти щодо формування для кожної раніше побудованої конфігурації хмарної системи *R\_system* коефіцієнтів передачі, матриці маршрутизації та внутрішні швидкості надходження (*internal\_arrival\_rates\_per\_queue*) запитів/завдань, що допомагає краще зрозуміти взаємодію між компонентами системи.

Проведено програмні експерименти щодо перевірки умови 2.2-2.3 розділу 2 спрацьовування умов розміщення ресурсів на конфігураціях хмарної системи *R\_system*.

Розроблено програмне забезпечення щодо контролю порушення умови сталого режиму та обчисленням основних показників ефективності роботи хмарної системи (умова 2.4 споживання послуг в термінології систем масового обслуговування) з формуванням пропозицій щодо вибору мінімальної кількості каналів, необхідних для стабільної роботи.

Використано імітаційний підхід до дослідження архітектурних рішень системи.

Сформовано програмні модулі роботи вузлів моделі чотирирівневої хмарної системи. Зокрема, за результатами програмних експериментів при використанні побудованих умов розділу 2 досліджено поведінку хмарної системи, як випадкового процесу у дискретному просторі, на основі станів системи.

За побудованими у розділі 3 умовами та схемою проведено програмні експерименти, що показали динаміку навантаження чотирьох рівнів хмарної системи, яка сформована на взаємодії рівнів ієрархії хмарної архітектури.

Проведено огляд існуючих наборів даних (зокрема, UNSW-NB15 та CICIoT2023) для подальшої апробації моделі чотирирівневої хмарної системи з умовами розміщення ресурсів та споживання послуг. Обговорюється питання аналізу даних з відкритих джерел та формування власного набору даних для виявлення порушень умов розміщення ресурсів та споживання послуг при імітаційному моделюванні.

Проведено серію експериментів з різними сценаріями розгортання хмарної платформи. Виконано аналіз різних моделей розгортання хмарної платформи та принципи організації хмарних технологій за способом їх взаємодії з користувачем. Побудовано емпіричну оцінку запропонованих алгоритмів провайдером та швидкість оброблення запитів на різних апаратних платформах.

## ВИСНОВКИ

Сукупність отриманих в дисертаційному дослідженні результатів вирішує актуальне науково-технічне завдання розроблення методів побудови хмарної системи.

Зазначене наукове завдання має істотне значення для розвитку архітектурних рішень побудови хмарної системи задачі відстеження складності побудови та контролю динаміки навантаження, зокрема у галузі управління хмарними системами.

Враховуючи стрімкий розвиток інформаційних технологій, постійне оновлення вже існуючих рішень та обмеженість доступу до інформаційних ресурсів наукових публікацій (іноді за рахунок великої швидкості оновлення програмних рішень та періодичності відповідних наукових публікацій щодо цих рішень), можна визначати результати досліджень такими, що мають високу значущість.

В дисертаційному дослідженні отримані наступні основні результати:

1. У роботі проведено ґрунтовний аналіз сучасних концепцій, моделей, підходів управління та інформаційних технологій, що лежать в основі проєктування й функціонування хмарних обчислювальних систем. Визначено архітектурне рішення побудови хмарної системи, у межах якого формалізовано послідовність проходження запиту чи обчислювального завдання через багаторівневу структуру представлення взаємозв'язків. Найбільша складність полягає у забезпеченні узгодженості між рівнями, зокрема в оптимізації механізмів маршрутизації, синхронізації та контролю цілісності даних у розподіленому середовищі. Запропоновано концептуальну модель організації цих рівнів, що підвищує структурованість, масштабованість та керованість хмарної системи, а також створює підґрунтя для подальшого удосконалення механізмів оброблення запитів/завдань.

2. В роботі проведено аналіз предметної області та визначено особливості проєктування ІТ-інфраструктури хмарної системи із застосуванням підходів розподіленого управління, методів розміщення ресурсів, моделювання споживання послуг та прогнозування навантажень у динамічному хмарному середовищі. Складністю проєктування є забезпечення цілісності та узгодженості цих підходів у межах єдиної архітектурної моделі, здатної адекватно відображати змінність потоків запитів і варіативність ресурсних потреб. На основі проведеного аналізу запропоновано представлення хмарної системи у вигляді чотирирівневої структури, що дає змогу інтерпретувати її функціонування як впорядкований потік запитів/завдань та формалізувати взаємодію між рівнями.

3. Визначено підхід до проєктування хмарної архітектури з довільною кількістю рівнів, який інтегрує умови розміщення всіх запитів/завдань на доступних ресурсах та умови їх споживання в межах розподіленого середовища. Найбільша складність полягає у забезпеченні узгодженості між цими умовами, зокрема у формуванні такої архітектурної моделі, що здатна коректно відображати динаміку потоків запитів і варіативність навантажень на різних рівнях системи. Для чотирирівневої архітектури хмарної системи запропоновано застосування перевірки показників ефективності функціонування системи масового обслуговування як формального інструмента оцінювання її працездатності та стабільності.

4. Запропоновано програмне рішення щодо об'єднання умов сталого режиму та вибору критеріїв структури чотирирівневої хмарної системи, що дають змогу оцінити її здатність підтримувати стабільний потік запитів/завдань та ускладнюються їх узгодженням з процесом вибору архітектурного рішення. Визначено, що об'єднання цих умов незначно збільшує обчислювальну складність роботи запропонованих алгоритмів, що може бути вирішено за рахунок оптимального вибору структур даних.

5. Визначено три модифікації чотирирівневих хмарних систем, що характеризуються значною кількістю можливих комбінацій зв'язків між вузлами та формалізованими правилами перенесення запитів/завдань між рівнями. Складністю цього завдання є забезпечення коректної взаємодії між рівнями за умов високої варіативності структурних конфігурацій та динамічної зміни інтенсивності потоків запитів.
6. Визначено підхід до розроблення програмного рішення, орієнтованого на підтримку роботи вузлів хмарної системи за умов змінного навантаження, що моделюється засобами імітаційного моделювання. Найбільша складність полягає у формалізації поведінки вузлів у динамічному середовищі, що потребує побудови адекватних моделей взаємодії та механізмів адаптації. Запропоновано програмну реалізацію, яка забезпечує відтворення різних сценаріїв навантаження, аналіз реакції вузлів та оцінювання ефективності функціонування хмарної системи в умовах варіативних параметрів.
7. Розроблено програмне рішення для формування наборів різних комбінацій чотирирівневих хмарних систем.
8. Розроблено програмне рішення для побудованої структури хмарної системи та досліджено різні її комбінації (зокрема, побудовано 810000 моделей), що надають можливості аналізу її функціональних зв'язків.
9. Використано набори UNSW-NB15 і CICIoT2023 у загальній кількості 49/47 параметрів та 2540047/46686579 для формування наборів даних з меншою кількістю параметрів та розроблено відповідне програмне рішення.
10. Досліджені особливості вбудовування агента-додатку на рівні хмарної системи, що зумовлює як локальні, так і глобальні математичні наслідки у процесі зростання складності управління та зниження прогнозованості поведінки системи. Найбільша складність полягає у формалізації взаємодії між агентом-додатком та вузлами різних рівнів, оскільки глобальна стратегія перерозподілу навантаження може суперечити локальним тактикам оптимізації, що реалізуються всередині окремих вузлів та окремих рівнів.

11. Проведено експерименти моделювання чотирирівневої хмарної системи та розроблено програмне рішення моделі хмарної системи, яка дає змогу оцінити вплив кожного з розроблених рішень на функціонування системи, порівняти їх ефективність та визначити оптимальні конфігурації для подальшого вдосконалення її архітектури.

Таким чином, усі поставлені завдання у даному науковому дослідженні повністю виконані.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC/IEEE 42010:2011. Systems and software engineering — Architecture description. Geneva: ISO, 2011.
2. Balalaie A., Heydarnoori A., Jamshidi P. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. IEEE Software. 2016. Vol. 33, No. 3. P. 42–52. DOI: <https://doi.org/10.1109/MS.2016.64>
3. Ткаченко О.В., Семенов С.Г. Моделивання багаторівневих ієрархічних структур розподілених обчислювальних систем та хмарних платформ. Системи управління, навігації та зв'язку. 2025. № 2 (80). С. 112–118.
4. Mell P., Grance T. The NIST Definition of Cloud Computing. Gaithersburg, MD: National Institute of Standards and Technology, 2011. 7 p. (NIST Special Publication 800-145).  
URL: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf>
5. Лозінський А. П. Синтез технологій платформ хмарних обчислень. Control Systems and Computers. 2019. № 6. С. 35–45. DOI: <https://doi.org/10.15407/csc.2019.06.035>.
6. Божуха Д. І., Байбуз О. Г. Про формалізацію внутрішніх процесів платформи хмарних обчислень. Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2022): тези доповідей XX Міжнародної науково-практичної конференції (Дніпро, 23–25 листопада 2022 р.). Дніпро: ДНУ, 2022. С. 38. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2022/12/MPZIS-2022-1.pdf>.
7. ISO/IEC 22123-1:2023(en). Information technology — Cloud computing — Part 1: Vocabulary. Geneva: ISO/IEC, 2023. URL: <https://www.iso.org/standard/82758.html>

8. ISO/IEC 22123-2:2023(en). Information technology — Cloud computing — Part 2: Concepts. Geneva: ISO/IEC, 2023. 35 p. URL: <https://www.iso.org/standard/80351.html> .
9. Badger, L., Grance, T., Patt-Corner, R., & Voas, J. Cloud Computing Synopsis and Recommendations. Recommendations of the National Institute of Standards and Technology. NIST, 2012. 81 p. (NIST Special Publication 800-146). URL: <https://csrc.nist.gov/publications/detail/sp/800-146/final>
10. Коваленко А.А., Кулик М.І. Аналіз сценаріїв розгортання гібридних хмарних інфраструктур для критичних бізнес-додатків. Сучасні інформаційні технології у сфері безпеки та оборони. 2023. № 1 (46). С. 45–52.
11. Ruizhe Yang, F. Richard Yu, Pengbo Si, Zhaoxin Yang, Yanhua Zhang. Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges. IEEE Communications Surveys & Tutorials. 2019. Vol. 21, No. 2, P. 1508–1532.
12. Armbrust, M., Stoica, I., Zaharia, M. et al. A View of Cloud Computing. Communications of the ACM, 2010, Vol. 53, No. 4, P. 50–58. DOI: <https://doi.org/10.1145/1721654.1721672>.
13. Krishnaraj N., Daniel A., Saini K., Bellam K. EDGE/FOG computing paradigm: Concept, platforms and toolchains. Advances in Computers. 2022. Vol. 127. P. 413–436. DOI: <https://doi.org/10.1016/bs.adcom.2022.02.012>
14. Волк М., Поповкін М. Методи моделювання масштабованих хмарних ресурсів. Системи управління, навігації та зв'язку. 2024. Т. 3, № 77. С. 97–101. DOI: <https://doi.org/10.26906/SUNZ.2024.3.097>
15. Muhammad Akbar Ibnu Farhan Putra Sujarwo et al. Analysis of Load Balancing Least Connection and Shortest Expected Delay Algorithm for Web Server Using Kube-Proxy on Kubernetes. ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika. 2023. Vol. 11, no. 2. P. 439–452. DOI: <http://dx.doi.org/10.26760/elkomika.v11i2.439>

- 16.** Zaharia M. [et al.]. Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM. 2016. Vol. 59, no. 11. P. 56–65. DOI: <https://doi.org/10.1145/2934664>
- 17.** Божуха Д.І., Байбуз О.Г. Про узагальнену схему складних обчислювальних систем платформи хмарних послуг. Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2023) : тези доповідей XXI Міжнародної науково-практичної конференції (Дніпро, 22–24 листопада 2023 р.). Дніпро, 2023. С. 77. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf>
- 18.** Burns B., Beda J., Hightower K., Evenson L. Kubernetes: Up & Running: Dive into the Future of Infrastructure. 3rd ed. Sebastopol, CA: O'Reilly Media, 2022. 328p.
- 19.** Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol, CA : O'Reilly Media, 2021. 584 p.
- 20.** Ворона Д. О. Дослідження методів оптимізації продуктивності хмарних веб-додатків контейнеризованих у Kubernetes. Радіоелектроніка та молодь у XXI столітті: матеріали 29-го Міжнар. молодіж. форуму (Харків, 16–19 квітня 2025 р.). Харків : ХНУРЕ, 2025. Т. 6. С. 244–246. URL: <https://openarchive.nure.ua/entities/publication/f6be08e0-ccc6-49ff-85c4-b9dbab3351b2>
- 21.** Pahl C., Jamshidi P. Microservices: A Systematic Mapping Study. Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016). Rome, Italy, 2016. P. 137–146. DOI: [10.5220/0005785501370146](https://doi.org/10.5220/0005785501370146)
- 22.** Buyya, R., & Srirama, S. N. (Eds.). Fog and Edge Computing: Principles and Paradigms. John Wiley & Sons, Inc., 2019. 512 p. (Wiley Series on Parallel and Distributed Computing). DOI: <https://doi.org/10.1002/9781119525080>
- 23.** Toosi A. N., Mahmud M. R., Chi Q., Buyya R. Management and Orchestration of Network Slices in 5G, Fog, Edge, and Clouds. Fog and Edge Computing: Principles and Paradigms / ed.: R. Buyya, S. N. Srirama. Hoboken, New Jersey :

John Wiley & Sons, Inc., 2019. P. 79–101. DOI: <https://doi.org/10.1002/9781119525080.ch4>

**24.** RahimiZadeh K., Beheshti A., Javadi B. et al. An integrated queuing and certainty factor theory model for efficient edge computing in remote patient monitoring systems. *Scientific Reports*. 2025. Vol. 15, No. 1. Art. 28703. DOI: <https://doi.org/10.1038/s41598-025-28703-1>.

**25.** Bozhukha D. The object of research is the architecture and system of cloud computing. Тенденції та перспективи розвитку науки і освіти в умовах глобалізації : матеріали 95-ї Міжнародної науково-практичної інтернет-конференції. Переяслав, 2023. Вип. 95. С. 57–59. URL: [https://0a30397da1-clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1](https://0a30397da1.clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1)

**26.** Binayak Kar, Widhi Yahya, Ying-Dar Lin and Asad Ali. Offloading Using Traditional Optimization and Machine Learning in Federated Cloud–Edge–Fog Systems: A Survey. *IEEE Communications Surveys & Tutorials*. 2023. Vol. 25, no. 2. P. 1199–1226. DOI: <https://doi.org/10.1109/COMST.2023.3239579>

**27.** Kuchuk H., Malokhvii E. Integration of IOT with Cloud, Fog, and Edge Computing: A Review. *Advanced Information Systems*. 2024. Vol. 8, no. 2. P. 65–78. DOI: <https://doi.org/10.20998/2522-9052.2024.2.08>

**28.** Liu M., Yu R., Teng Y., Leung V. C. M. Computation Offloading and Content Caching in Wireless Blockchain Networks With Mobile Edge Computing // *IEEE Transactions on Vehicular Technology*. 2018. Vol. 67, No. 11. P. 11008–11021. DOI: <https://doi.org/10.1109/TVT.2018.2866365>.

**29.** Fernando N., Shrestha S., Loke S. W., Lee K. On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture. *Future Internet*. 2025. Vol. 17, No. 1. Art. 22. DOI: <https://doi.org/10.3390/fi17010022>

- 30.** Mao Y., You C., Zhang J., Huang K., Letaief K. B. A Survey on Mobile Edge Computing: The Communication Perspective // IEEE Communications Surveys & Tutorials. 2017. Vol. 19, No. 4. P. 2322–2358. DOI: <https://doi.org/10.1109/COMST.2017.2745201>
- 31.** Singh J., Singh P., Gill S. S. Fog computing: A taxonomy, systematic review, current trends and research challenges. Journal of Parallel and Distributed Computing. 2021. Vol. 157. P. 56–85. DOI: <https://doi.org/10.1016/j.jpdc.2021.06.005>
- 32.** Aslanpour, M. S., Gill, S. S., & Toosi, A. N. (2020). Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. Internet of Things, 12, Article 100273. DOI: <https://doi.org/10.1016/j.iot.2020.100273>
- 33.** Karolj Skala, Davor Davidovic et al. Scalable Distributed Computing Hierarchy: Cloud, Fog and Dew Computing. Open Journal of Cloud Computing (OJCC). 2015. Vol. 2, iss. 1. P. 16–24. DOI: 10.19210/1002.2.1.16.  
URL: [https://www.researchgate.net/publication/289380143\\_Scalable\\_Distributed\\_Computing\\_Hierarchy\\_Cloud\\_Fog\\_and\\_Dew\\_Computing](https://www.researchgate.net/publication/289380143_Scalable_Distributed_Computing_Hierarchy_Cloud_Fog_and_Dew_Computing).
- 34.** Hong C.-H., Varghese B. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms // ACM Computing Surveys. 2019. Vol. 52, No. 5. Art. 97. DOI: <https://doi.org/10.1145/3326066>
- 35.** Sundas Iftikhar et al. AI-Based Fog And Edge Computing: A Systematic Review, Taxonomy And Future Directions. Internet of Things. 2023. Vol. 21. Art. 100674. DOI: <https://doi.org/10.1016/j.iot.2022.100674>
- 36.** Sukhpal Singh Gill et al. Modern computing: Vision and challenges. Telematics and Informatics Reports. 2024. Vol. 13. Art. 100116. DOI: <https://doi.org/10.1016/j.teler.2024.100116>
- 37.** Wang S., Zhang X., Zhang Y., Wang L. A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications. IEEE Access. 2017. Vol. 5. P. 6757–6772. DOI: <https://doi.org/10.1109/ACCESS.2017.2685434>

- 38.** Yousefpour A., Fung C., Nguyen T., Kadiyala K., Jalali F., Niakanlahiji A., Kong J., Jue J. P. All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey. *Journal of Systems Architecture*. 2019. Vol. 98. P. 289–330. DOI: <https://dl.acm.org/doi/10.1016/j.sysarc.2019.02.009>
- 39.** Заблоцький С., Пограничний В., Тарасенко А., Колодій Р. Оптимізація процесу маршрутизації в розподілених мережах з використанням машинного навчання. *Інфокомунікаційні технології та електронна інженерія*. 2025. Т. 5, №1. URL: <https://science.lpnu.ua/sites/default/files/journal-paper/2025/apr/38742/paper6szablockiyvpogranichniytarasenkokolodiy.pdf>
- 40.** Божуха Д.І., Байбуз О.Г. Про імітаційні моделі блоків узагальненої системи хмарних обчислень. *Актуальні проблеми автоматизації та інформаційних технологій*. 2024. Т. 28. С. 81–86. DOI: <https://dx.doi.org/10.15421/432407>
- 41.** Weng Y., Li C.-T., Chen C.-L., Lee C.-C., Deng Y.-Y. A Lightweight Anonymous Authentication and Secure Communication Scheme for Fog Computing Services. *IEEE Access*. 2021. Vol. 9. P. 145522–145537. DOI: <https://doi.org/10.1109/ACCESS.2021.3123234>.
- 42.** Ghazaleh Javadzadeh, Amir Masoud Rahmani, Morteza Saberi Kamarposhti. Mathematical model for the scheduling of real-time applications in IoT using Dew computing. *The Journal of Supercomputing*. 2022. Vol. 78. P. 7464–7488. DOI: <https://doi.org/10.1007/s11227-021-04170-z>
- 43.** You J., Li Y., Guo X., Li C.. Dynamic Resource Aggregation Method Based on Statistical Capacity Distribution in Cloud Computing. *MDPI Electronics*. 2024. Vol. 13, No. 23. Art. 4617. DOI: <https://doi.org/10.3390/electronics13234617>
- 44.** Ghobaei-Arani M., Souri A., Rahmani A. M. Resource Management Approaches in Fog Computing: a Comprehensive Review. *Journal of Grid Computing*. 2020. Vol. 18. No. 1. P. 1–42. DOI: <https://doi.org/10.1007/s10723-019-09491-1>

45. Rajammal K., Chinnadurai M. Dynamic load balancing in cloud computing using predictive graph networks and adaptive neural scheduling. *Scientific Reports*. 2025. Vol. 15. Art. 22181. DOI: <https://doi.org/10.1038/s41598-025-97494-2>
46. Божуха Д.І. Про методи та технології побудови системи. *Ways of Science Development in Modern Crisis Conditions : тези доповідей V Міжнародної науково-практичної інтернет-конференції (Дніпро, 13–14 червня 2024 р.)*. Дніпро, 2024. С. 37–38. URL: <http://www.wayscience.com/wp-content/uploads/2024/06/Conference-Proceedings-June-13-14-2024.pdf>
47. Бойко Н. І. Еволюція побудови архітектур інформаційних систем. Перспективи розвитку “хмарної” архітектури. *Вісник Національного університету «Львівська політехніка»*. Серія: Інформаційні системи та мережі. 2015. № 832. С. 348–368. URL: <https://ena.lpnu.ua/handle/ntb/31595>
48. Іваненко О.А., Марченко О.І. Спосіб уніфікованого опису хмарної інфраструктури різних провайдерів. *Комп’ютерно-інтегровані технології: освіта, наука, виробництво*. 2024. № 54. С. 103–112. URL: <https://cit.lntu.edu.ua/index.php/cit/article/view/530>
49. Honcharenko O., Loutskii H. Methods of effectivization of scalable systems: reiew. *Information, Computing and Intelligent Systems Journal*. 2022. Вип. 3. С. 63–76. DOI: <https://doi.org/10.20535/2708-4930.3.2022.265229>
50. Божуха Д.І., Байбуз О.Г., Машенко Л.В. Про підходи дослідження системи хмарних обчислень. *Актуальні проблеми автоматизації та інформаційних технологій*. 2022. Т. 26. С. 18–30. DOI: <https://dx.doi.org/10.15421/432203>
51. Oboznyi D., Kulakov Y. Method of dynamic reconfiguration of software-defined networks. *Information, Computing and Intelligent systems*. 2024. No. 4. P. 25–36. URL: <https://doi.org/10.20535/2786-8729.4.2024.305095>
52. Al-Dhuraibi Y., Paraiso F., Djarallah N., Merle P. Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing*. 2018. Vol. 13, No. 2. P. 430–447. DOI: <https://doi.org/10.1109/TSC.2017.2711009>

- 53.** Botta A., de Donato W., Persico V., Pescapé A. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*. 2016. Vol. 56. P. 684–700. DOI: <https://doi.org/10.1016/j.future.2015.09.021>
- 54.** Girau R., Anedda M., Floris G., Carboni D., Pinna A., Atzori L. Definition and implementation of the Cloud Infrastructure for the integration of the Human Digital Twin in the Social Internet of Things. *Computer Networks*. 2024. Vol. 253. Art. 110757. DOI: <https://doi.org/10.1016/j.comnet.2024.110632>
- 55.** Martin N., Dogar F. Divided at the Edge: Measuring Performance and the Digital Divide of Cloud Edge Data Centers. *Proceedings of the ACM on Networking*. 2023. Vol. 1, No. 3. Art. 16. P. 1-23. DOI: <https://doi.org/10.1145/3629138>
- 56.** Roumeliotis A. J., Myritzis E., Kosmatos E., Katsaros K. V., Amditis A. J. Multi-Area, Multi-Service and Multi-Tier Edge-Cloud Continuum Planning /. *Sensors*. 2025. Vol. 25, No. 13. Art. 3949. DOI: <https://doi.org/10.3390/s25133949>
- 57.** Beloglazov A., Buyya R. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. *Concurrency and Computation: Practice and Experience*. 2012. Vol.24, No.13. P.1397–1420. DOI:<https://doi.org/10.1002/cpe.1867>
- 58.** Lorido-Botran T., Miguel-Alonso J., Lozano J. A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing*. 2014. Vol. 12, No. 4. P. 559–592. DOI: <https://doi.org/10.1007/s10723-014-9314-7>
- 59.** Raeisi-Varzaneh M., Habbal A. Resource Scheduling in Edge Computing: Architecture, Taxonomy, Open Issues and Future Research Directions. *IEEE Access*. 2023. Vol. 11. P. 39182–39207. DOI: <https://doi.org/10.1109/ACCESS.2023.3256522>
- 60.** Двірна О. А., Набока С. В. Оптимізація продуктивності хмарних сервісів: методи та їх ефективність. *Системи управління, навігації та зв'язку*. 2026. № 1. С. 62–68.

URL:<https://reposit.nupp.edu.ua/files/original/5/74760/a1f44a5a485ac8d5735eb305bf3c3da3be3c6b4d.pdf>

61. Череватенко О., Кулаков Ю. Метод підвищення стабільності передачі даних у програмно конфігурованій мережі з урахуванням метрик якості обслуговування. *Information, Computing and Intelligent systems*. 2024. Вип. 4. С. 37–47. DOI: <https://doi.org/10.20535/2786-8729.4.2024.303467>
62. Lysenko S., Bondaruk O., Bondar O., Koretska L., Sochor T. Mathematical model of the cloud infrastructure life cycle. *CEUR Workshop Proceedings*. 2025. Vol. 4013, paper 19. URL: <https://ceur-ws.org/Vol-4013/paper19.pdf>
63. Taleb I., Guillaume J.-L., Duthil B. A Survey on Services Placement Algorithms in Integrated Cloud-Fog / Edge Computing. *ACM Computing Surveys*. 2025. Vol. 57, No. 11. Art. 271. DOI: <https://doi.org/10.1145/3729214>
64. Khebbeb K., Hameurlain N., Belala F. Formalizing and simulating cross-layer elasticity strategies in Cloud systems. *Cluster Computing*. 2020. Vol. 23, No. 3. P. 2023–2039. DOI: <https://doi.org/10.1007/s10586-020-03080-8>
65. Dolyunnyi O., Nikolskiy S., Kulakov Y. The Method Of Sdn Clustering For Controller Load Balancing. *Information, Computing and Intelligent systems*. 2020. No. 1. P. 43–50. DOI: <https://doi.org/10.20535/2708-4930.1.2020.216056>
66. Ferdman M., Adileh A., Kocberber O. et al. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII) : proceedings of the 17th International Conference (London, March 3–7, 2012)*. London, 2012. P. 37–48. DOI: <https://doi.org/10.1145/2150976.2150982>
67. Божуха Д. Про методи підвищення продуктивності системи. Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем (MEICS-2024): тези доповідей ІХ Всеукраїнської науково-практичної конференції (Дніпро, 27–29 листопада 2024 р.). Дніпро, 2024. С. 35. URL: <http://meics.dnure.dp.ua/files/MEICS-2024.pdf>

- 68.** Binz T., Fehling C., Leymann F., Nowak A., Schumm D. Formalizing the Cloud through Enterprise Topology Graphs. Cloud Computing Technology and Science (CloudCom) : proceedings of the 2012 IEEE 4th International Conference (Taipei, Dec. 3–6, 2012). Taipei, 2012. P. 936–943. DOI: [https://ui.adsabs.harvard.edu/link\\_gateway/2012clou.conf..103B/doi:10.1109/CLOUD.2012.143](https://ui.adsabs.harvard.edu/link_gateway/2012clou.conf..103B/doi:10.1109/CLOUD.2012.143)
- 69.** Ward J. S., Barker A. Observing the Clouds: A Survey and Taxonomy of Cloud Monitoring. Journal of Cloud Computing. 2014. Vol. 3. Art. 24. DOI: <https://doi.org/10.1186/s13677-014-0024-2>
- 70.** Kreutz D., Ramos F. M. V., Verissimo P., Rothenberg C. E., Azodolmolky S., Uhlig S. Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE. 2015. Vol. 103, No. 1. P. 14–76. DOI: <https://doi.org/10.1109/JPROC.2014.2371999>
- 71.** Afzal S., Kavitha G. Load balancing in cloud computing – A hierarchical taxonomical classification. Journal of Cloud Computing. 2019. Vol. 8, No. 1. P. 1–24. DOI: <https://doi.org/10.1186/s13677-019-0146-7>
- 72.** Barbierato E. A Reproducible Monte Carlo Framework for Evaluating Cost–Latency Trade-Offs in Cloud Continuum / E. Barbierato, E. Goldoni, D. Tessera. Electronics. 2026. Vol. 15, iss. 8. Art. 1708. DOI: <https://doi.org/10.3390/electronics15081708>
- 73.** Haidai A., Klymenko I. Method of Load Balancing in Distributed Three-Layer IoT Architecture. Information, Computing and Intelligent systems. 2024. No. 4. P. 60–68. DOI: <https://doi.org/10.20535/2786-8729.4.2024.311595>
- 74.** Возна Н. Я. Теорія та методи побудови моделей руху даних у розподілених КС. Вісник Національного університету «Львівська політехніка». Серія: Комп'ютерні системи та мережі. 2010. № 688. С. 60–64. URL: <https://science.lpnu.ua/sites/default/files/journal-paper/2017/nov/7068/10-60-641.pdf>

- 75.** Deng R., Lu R., Lai C., Luan T. H., Liang H. Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption. *IEEE Internet of Things Journal*. 2016. Vol. 3, No. 6. P. 1171–1181. DOI: <https://doi.org/10.1109/JIOT.2016.2565516>
- 76.** Ren J., Yu G., He Y., Li G. Y. Collaborative Cloud and Edge Computing for Latency Minimization. *IEEE Transactions on Vehicular Technology*. 2019. Vol. 68, No. 5. P. 5031–5044. DOI: <https://doi.org/10.1109/TVT.2019.2904244>
- 77.** Mao H., Alizadeh M., Menache I., Kandula S. Resource Management with Deep Reinforcement Learning. *Proceedings of ACM HotNets*. 2016. DOI: <https://doi.org/10.1145/3005745.3005750>
- 78.** Harjanti T. W. et al. Load Balancing Analysis Using Round-Robin and Least-Connection Algorithms for Server Service Response Time. *Applied Technology and Computing Science Journal*. 2022. Vol. 5, no. 2. P. 40–49. DOI: <https://doi.org/10.33086/atcsj.v5i2.3743>
- 79.** Leite L., Rocha C., Kon F., Milojevic D., Meirelles P. A Survey of DevOps Concepts and Challenges. *ACM Computing Surveys*. 2019. Vol. 52, No. 6. Art. 127. P. 1-35. DOI: <https://doi.org/10.1145/3359981>
- 80.** Бардадим Т., Лефтеров О., Осипенко С. Досвід розгортання тестів OpenStack та порівняння віртуальних і реальних кластерних середовищ. *Кібернетика та комп'ютерні технології*. 2021. № 3. С. 74–85. DOI: <https://doi.org/10.34229/2707-451X.21.3.7>
- 81.** Калькулятор Azure. Microsoft Azure. URL: <https://azure.microsoft.com/en-us/pricing/calculator/>
- 82.** Islam S., Keung J., Lee K., Liu A. Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud. *Future Generation Computer Systems*. 2012. Vol. 28, No. 1. P. 155–162. DOI: <https://doi.org/10.1016/j.future.2011.05.027>
- 83.** Лахно В. та ін. Моделювання мінімальної кількості вузлів кластера віртуалізації приватних університетської хмари. *Кібербезпека: освіта, наука,*

*техніка*. 2023. Т. 1, № 21. С. 177–192. DOI: <https://doi.org/10.28925/2663-4023.2023.21.177192>.

**84.** Андросчук О., Голобородько М., Кондратенко Ю., Литовченко Г. Критерії та рекомендації з оцінювання якості хмарних сервісів для інформаційної інфраструктури / *Сучасні інформаційні технології у сфері безпеки та оборони*. 2024. Т. 51, № 3. С. 60–70. DOI: <https://doi.org/10.33099/2311-7249/2024-51-3-60-70>

**85.** Herbst N. R., Kounev S., Reussner R. Elasticity in Cloud Computing: What It Is, and What It Is Not. *ACM Transactions on Autonomous and Adaptive Systems*. 2013. Vol. 8, No. 4. Art. 18.

URL: [https://www.usenix.org/system/files/conference/icac13/icac13\\_herbst.pdf](https://www.usenix.org/system/files/conference/icac13/icac13_herbst.pdf)

**86.** Sharma U., Shenoy P., Sahu S., Shaikh A. A Cost-Aware Elasticity Provisioning System for the Cloud. *Proceedings of IEEE ICDCS*. 2011. DOI: <https://doi.org/10.1109/ICDCS.2011.59>

**87.** Chaudhary H., Sharma G., Nishad D. K., Khalid S. AI-enhanced modelling of queueing and scheduling systems in cloud computing. *Discover Applied Sciences*. 2025. Vol. 7. Art. 276. URL: <https://doi.org/10.1007/s42452-025-06755-2>

**88.** Adan I., Resing J. *Queueing Theory*. Eindhoven University of Technology, 2015. URL: <https://www.win.tue.nl/~iadan/queueing.pdf>

**89.** Choi D. I., Lim D. E. Analysis of a Markovian Queueing Model with an Alternating Server and Queue-Length-Based Threshold Control. *Mathematics*. 2025. Vol. 13, No. 21. Art. 3555. DOI: <https://doi.org/10.3390/math13213555>.

**90.** Савчук Т. О., Козачук А. В. Development of cloud app simulator. *Technology Audit and Production Reserves*. 2015. Vol. 6, no. 7 (26). P. 4–6. DOI: <https://doi.org/10.15587/2312-8372.2015.54851>

**91.** Савчук Т.О., Козачук А.В. Інформаційна технологія масштабування хмарного додатку зі змінними піками навантаження. *Технологічний аудит і резерви виробництва*. 2015. Т. 5, № 2 (25). С. 4–11. DOI: <https://doi.org/10.15587/2312-8372.2015.51716>

- 92.** Rahman M., Iqbal S., Gao J. Load Balancer as a Service in Cloud Computing. *Proceedings of the 2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)* (Oxford, April 7–11, 2014). IEEE, 2014. P. 204–211. DOI: <https://doi.org/10.1109/SOSE.2014.31>
- 93.** Alankar B., Sharma G., Kaur H., Valverde R., Chang V. Experimental Setup for Investigating the Efficient Load Balancing Algorithms on Virtual Cloud. *Sensors*. 2020. Vol. 20, № 24. Art. 7342. DOI: <https://doi.org/10.3390/s20247342>
- 94.** Fan Y. Load balance-aware dynamic cloud-edge-end collaborative offloading strategy. *PLOS ONE*. 2024. Vol. 19, no. 1. Art. e0296897. DOI: <https://doi.org/10.1371/journal.pone.0296897>
- 95.** Божуха Д.І., Байбуз О.Г. Дослідження моделей хмарних систем на прикладі найпростішого потоку. *Актуальні проблеми автоматизації та інформаційних технологій*. 2025. Т. 29. С. 84–91. DOI: <http://dx.doi.org/10.15421/432508>
- 96.** Жаріков Е.В. Інформаційна технологія управління ІТ-інфраструктурою хмарного центру оброблення даних : дис. ... д-ра техн. наук : 05.13.06. Київ, 2020. 402 с.
- 97.** Стеценко І.В. Моделювання систем : навч. посіб. Черкаси: ЧДТУ, 2010. 392с.
- 98.** Bodra D., Khairnar S. Machine learning-based cloud resource allocation algorithms: a comprehensive comparative review. *Frontiers in Computer Science*. 2025. Vol. 7. Art. 1678976. DOI: <https://doi.org/10.3389/fcomp.2025.1678976>
- 99.** Kai J., Zhou H., Yi Y., Huang W. Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks With Limited Communication Capability. *IEEE Transactions on Cognitive Communications and Networking*. 2021. Vol. 7, № 2. P. 624–634. DOI: <https://doi.org/10.1109/TCCN.2020.3018159>
- 100.** Ding Y., Li K., Liu C., Li K. A Potential Game Approach to Computation Offloading Strategy Optimization in End-Cloud Network Edge Computing. *IEEE*

Transactions on Parallel and Distributed Systems. 2021. Vol. 33, № 6. P. 1503–1519. DOI: <https://doi.org/10.1109/TPDS.2021.3112604>

**101.** Diamanti M., Charatsaris P., Tsiropoulou E. E., Papavassiliou S., Incentive mechanism and resource allocation for edge-fog networks driven by multidimensional contract and game theories, IEEE Open Journal of the Communications Society, 2022, vol. 3, pp. 435–452. DOI: <https://doi.org/10.1109/OJCOMS.2022.3154536>

**102.** Zaman S. K., Jehangiri A. I., Maqsood T., Haq N., Umar A. I., Shuja J., Ahmad Z. LiMPO: lightweight mobility prediction and offloading framework using machine learning for mobile edge computing. Cluster Computing. 2023. Vol. 26, № 1. P. 99–117. DOI: <https://doi.org/10.1007/s10586-021-03518-7>

**103.** Божуха Д., Байбуз О. Модель спільного динамічного розвантаження хмарної архітектури з балансуванням рівнів. Системні технології: Регіональний міжвузівський збірник наукових праць. 2025. Т. 6, № 161. С. 152–157. DOI: <https://doi.org/10.34185/1562-9945-5-161-2025-15>

**104.** Duan S., Wang D., Ren J. et al. Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey IEEE Communications Surveys & Tutorials. 2023. Vol. 25, no. 1. P. 591–624. DOI: <https://doi.org/10.1109/COMST.2022.3218527>

**105.** Wang B., Wang C., Huang W. et al. A Survey and Taxonomy on Task Offloading for Edge-Cloud Computing / IEEE Access. 2020. Vol. 8. P. 186080–186101. DOI: <https://doi.org/10.1109/ACCESS.2020.3029649>

**106.** Abazari F., Analoui M., Takabi H., Fu S. MOWS: Multi-objective workflow scheduling in cloud computing based on heuristic algorithm. Simulation Modelling Practice and Theory. 2019. Vol. 93. P. 119–132. DOI: <https://doi.org/10.1016/j.simpat.2018.10.004>

**107.** Божуха Д.І., Байбуз О.Г. Про оптимізацію навантаження на інфраструктуру методами машинного навчання та агентними технологіями. Математичне та програмне забезпечення інтелектуальних систем : тези

доповідей XXIII Міжнародної науково-практичної конференції (Дніпро, 19–21 листопада 2025 р.). Дніпро, 2025. С. 87–88. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2025.pdf>

**108.** Божуха Д. І., Байбуз О. Г. Про структуру архітектурного рішення. Математичне та програмне забезпечення інтелектуальних систем : тези доповідей XXII Міжнародної науково-практичної конференції (Дніпро, 20–22 листопада 2024 р.). Дніпро, 2024. С. 85. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf>

**109.** Божуха Д.І., Байбуз О.Г. Про імітацію роботи вузлів системи. Автоматика-2024 : тези доповідей XXVII Міжнародної конференції (Дніпро, 20–22 листопада 2024 р.). Дніпро, 2024. С. 76–77. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf>

**110.** DevOpsDataCollection: A collection of DevOps datasets that aim to facilitate research and development to support DevOps intelligence/mooselab. GitHub. URL: <https://github.com/mooselab/DevOpsDataCollection> (дата звернення: 20.11.2024).

**111.** Moustafa N., Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems. MilCIS 2015 : IEEE Military Communications and Information Systems Conference (Canberra, 10–12 November 2015). IEEE, 2015. DOI: <https://doi.org/10.1109/MilCIS.2015.7348942>

**112.** Moustafa N., Slay J. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the

- KDD99 data set. *Information Security Journal: A Global Perspective*. 2016. Vol. 25, no. 1-3. P. 18–31. DOI: <https://doi.org/10.1080/19393555.2015.1125974>
- 113.** Neto E. C., Dadkhah S., Ferreira R., Zohourian A., Lu R., Ghorbani A. A. CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. *Sensors*. 2023. Vol. 23, no. 13. Art. 5941. DOI: <https://doi.org/10.3390/s23135941>
- 114.** Shyam G., Bharti P. Multi-agent Systems for Resource Allocation in Cloud Computing. *IEEE International Conference on Contemporary Computing and Communications (InC4)* (Bangalore, 21–22 April 2023). IEEE, 2023. P. 1–5. DOI: <https://doi.org/10.1109/InC457730.2023.10262945>
- 115.** Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Borg, Omega, and Kubernetes. *Communications of the ACM*. 2016. Vol. 59, No. 5. P. 50–57. DOI: <https://doi.org/10.1145/2890784>
- 116.** Bass L., Weber I., Zhu L. *DevOps: A Software Architect's Perspective*. Boston: Addison-Wesley Professional, 2015. 304 p.
- 117.** Yang Z., Bhatnagar A., Qiu Y., Miao T. Cloud Infrastructure Management in the Age of AI Agents. *ACM SIGOPS Operating Systems Review*, 2025, Vol. 59, No. 2, P. 1–8, DOI: <https://doi.org/10.1145/3759441.3759443>

## ДОДАТКИ

### Додаток А Список праць здобувача за темою дисертації

*Статті у наукових фахових виданнях України:*

1. **Божуха Д.І., Байбуз О.Г., Машенко Л.В.** Про підходи дослідження системи хмарних обчислень. *Актуальні проблеми автоматизації та інформаційних технологій.* 2022. Т.26. с. 18-30. DOI: <http://dx.doi.org/10.15421/432203> (**фахове видання категорії Б**).
2. **Божуха Д.І., Байбуз О.Г.** Про імітаційні моделі блоків узагальненої системи хмарних обчислень. *Актуальні проблеми автоматизації та інформаційних технологій.* 2024. Т.28, с. 81-86. DOI: <http://dx.doi.org/10.15421/432407> (**фахове видання категорії Б**).
3. **Божуха Д., Байбуз О.** Модель спільного динамічного розвантаження хмарної архітектури з балансуванням рівнів. *Системні технології.* 2025. Т. 6 № 161, с. 152-157. DOI: <https://doi.org/10.34185/1562-9945-5-161-2025-15> (**фахове видання категорії Б**).
4. **Божуха Д.І., Байбуз О.Г.** Дослідження моделей хмарних систем на прикладі найпростішого потоку. *Актуальні проблеми автоматизації та інформаційних технологій.* 2025. Т.29, с. 84-91. DOI: <http://dx.doi.org/10.15421/432508> (**фахове видання категорії Б**).

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

5. **Божуха Д.І., Байбуз О.Г.** Про формалізацію внутрішніх процесів платформи хмарних обчислень. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2022): тези доповідей XX міжнародної науково-практичної конференції, Дніпро, 23-25 листопада 2022 р., 2022. С. 38.* URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2022/12/MPZIS-2022-1.pdf>.

6. Bozhukha Daniil. The object of research is the architecture and system of cloud computing. *Міжнародна науково-практична інтернет-конференція: матеріали конференції «Тенденції та перспективи розвитку науки і освіти в умовах глобалізації»*, Переяслав, 2023, Вип. 95. С. 57-59. URL: <https://0a30397da1.clvaw-cdnwnd.com/12ac69b5c0bec343f11779551473023e/200000540-7809b7809d/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%2095-5.pdf?ph=0a30397da1> .

7. Божуха Д.І., Байбуз О.Г. Про узагальнену схему складних обчислювальних систем платформи хмарних послуг. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2023): Тези доповідей XXI Міжнародної науково-практичної конференції*, Дніпро, 22 – 24 листопада 2023 р., 2023. С. 77. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf> .

8. Божуха Д.І. Про методи та технології побудови системи управління. *Ways of Science Development in Modern Crisis Conditions: тези доповідей V Міжнародної науково-практичної інтернет-конференції*, Дніпро, 13-14 червня 2024 р., 2024. С. 37-38. URL: <http://www.wayscience.com/wp-content/uploads/2024/06/Conference-Proceedings-June-13-14-2024.pdf>.

9. Божуха Д.І., Байбуз О.Г. Про структуру архітектурного рішення. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2024): тези доповідей XXII Міжнародної науково-практичної конференції*, Дніпро, 20 – 22 листопада 2024 р., 2024. С. 85. URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2024/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2024-1.pdf> .

10. Божуха Д.І., Байбуз О.Г. Про імітацію роботи вузлів системи. *Міжнародна конференція «Автоматика-2024»*: тези XXVII Міжнародної конференції, Дніпро, 20 – 22 листопада 2024 р., 2024. С. 76-77. URL: <http://mpzis.dnu.dp.ua/wp->

[content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf](http://content/uploads/2025/11/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0-2024-%D1%82%D0%B5%D0%B7%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B5%D0%B9.pdf).

11. Божуха Д. Про методи підвищення продуктивності системи. *Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем (MEICS-2024): тези доповідей IX Всеукраїнської науково-практичної конференції*, Дніпро, 27-29 листопада 2024 р., 2024. С. 35.

URL: <http://meics.dnure.dp.ua/files/MEICS-2024.pdf> .

12. Божуха Д.І., Байбуз О.Г. Про оптимізацію навантаження на інфраструктуру методами машинного навчання та агентними технологіями. *Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2025): Тези доповідей XXIII Міжнародної науково-практичної конференції*, Дніпро, 19 – 21 листопада 2025 р., 2025. С. 87-88.

URL: <http://mpzis.dnu.dp.ua/wp-content/uploads/2025/11/%D0%9C%D0%9F%D0%97%D0%86%D0%A1-2025.pdf>.

*Наукові доповіді, які засвідчують апробацію матеріалів дисертації:*

13. Байбуз О.Г., Божуха Д.І. Інструментарій та технології багаторівневої архітектури платформи хмарних обчислень. Наукова конференція за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара за 2022 рік, 2023.

URL: [https://www.dnu.dp.ua/docs/ndc/2023/Ost\\_var\\_programa.pdf](https://www.dnu.dp.ua/docs/ndc/2023/Ost_var_programa.pdf) .

14. Байбуз О.Г., Божуха Д.І. Про характеристики складних обчислювальних систем платформи хмарних послуг. Наукова конференція за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара за 2023 рік, 2024.

URL: [https://www.dnu.dp.ua/docs/ndc/2024/Pidsumkova\\_za\\_2023.pdf](https://www.dnu.dp.ua/docs/ndc/2024/Pidsumkova_za_2023.pdf) .

15. Байбуз О.Г., Божуха Д.І. Про комбіновані підходи масштабування для управління інфраструктурою. Наукова конференція за підсумками науково-дослідної роботи ДНУ ім. Олеся Гончара за 2024 рік, 2025.

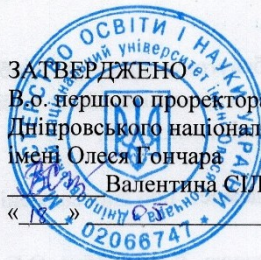
URL: [https://www.dnu.dp.ua/docs/ndc/2025/Pidsumkova\\_za\\_2024\\_zavershena.pdf](https://www.dnu.dp.ua/docs/ndc/2025/Pidsumkova_za_2024_zavershena.pdf)

.

## Додаток Б Акт впровадження результатів роботи в освітній процес

ПОГОДЖЕНО  
Проректор з наукової роботи  
Дніпровського національного університету  
імені Олеся Гончара  
\_\_\_\_\_ Олег МАРЕНКОВ  
« 18 » \_\_\_\_\_ 2026 р.

ЗАТВЕРДЖЕНО  
В.о. першого проректора  
Дніпровського національного університету  
імені Олеся Гончара  
\_\_\_\_\_ Валентина СІЛІЧ-БАЛГАБАЄВА  
« 18 » \_\_\_\_\_ 2026 р.



### АКТ

впровадження результатів роботи **Божухи Данііла Ігоровича**, поданої на здобуття наукового ступеня доктора філософії, на тему  
**«Розроблення методів аналізу та побудови структури хмарних систем»**  
в освітній процес Дніпровського національного університету імені Олеся Гончара

1. Вчена рада факультету прикладної математики та інформаційних технологій у складі 17 осіб заслухала повідомлення аспіранта кафедри інженерії програмного забезпечення та інформаційних технологій Божухи Данііла Ігоровича про результати наукового дослідження та їхнє використання в освітньому процесі галузі інформаційних технологій.

2. Стисла характеристика дослідження:

Данііл Божуха досліджує проблему організації хмарних систем, яка впливає на нестабільність інфраструктури та її показники надійності. Автор пропонує нове архітектурне рішення побудови хмарної системи з визначеною послідовністю руху запиту/завдання, яке враховує узагальнені методи та підходи аналізу її складності та функціональних зв'язків організаційної структури. Також, запропоновано модифікації чотирирівневої хмарної системи та сформовано набори даних їх різних комбінацій для апробації розробленої моделі з перевіркою умов розміщення ресурсів і споживання послуг та обчислення основних показників ефективності.

Методи та підходи щодо побудови хмарної системи реалізовано у вигляді набору програмних рішень мовою Python. При вирішенні задач з достатньо високою обчислювальною складністю використано хмарний сервіс (SaaS) з його онлайн-інструментами. Моделювання на різних комбінаціях хмарних систем з визначеними методами аналізу їх складності підтвердили ефективність запропонованих підходів порівняно з існуючими. Робота має практичне значення на етапі побудови архітектурних рішень хмарної системи для відстеження складності побудови та контролю динаміки навантаження хмарної системи, що дозволяє адаптувати отримані результати до існуючих алгоритмів у галузі управління хмарними системами.

3. Використання в освітньому процесі:

Результати дисертаційних досліджень впроваджено в освітній процес кафедри інженерії програмного забезпечення та інформаційних технологій факультету прикладної математики та інформаційних технологій ДНУ під час викладання освітніх компонент «Інженерія надійності систем» та «Імітаційне моделювання» для здобувачів другого (магістерського) рівня вищої освіти освітньої програми «Інженерія програмного забезпечення» спеціальності 121 Інженерія програмного забезпечення. Окремі теоретичні нароби та результати використано при викладанні освітньої компоненти «Методи теорії

масового обслуговування» для здобувачів другого (магістерського) рівня вищої освіти та при виконанні кваліфікаційних робіт здобувачами факультету прикладної математики та інформаційних технологій.

#### 4. Відомості про впроваджені об'єкти інтелектуальної власності:

1. Божуха Д.І., Байбуз О.Г., Машенко Л.В. Про підходи дослідження системи хмарних обчислень // Актуальні проблеми автоматизації та інформаційних технологій. – Дніпро: ДНУ, 2022. – Т.26. – С. 18-30. DOI: <http://dx.doi.org/10.15421/432203> [Фахове видання України категорії Б]

(особистий внесок: провів аналіз літературних джерел щодо використання методів, технологій, моделей та практичних підходів, які пов'язані з дослідженням хмарної системи; приділив увагу аналізу задачі управління ресурсами IT-інфраструктури; виділив суттєві характеристики ієрархічності IT-інфраструктури; побудував класифікатор для вирішення задачі вибору інструментарію при проектуванні хмарної системи; запропонував для хмарної системи розглянути модель системи масового обслуговування; виділив умови розміщення на ресурсах та умови споживання хмарної системи)

2. Божуха Д.І., Байбуз О.Г. Про імітаційні моделі блоків узагальненої системи хмарних обчислень // Актуальні проблеми автоматизації та інформаційних технологій. – Дніпро: ДНУ, 2024. – Т.28. – С. 81-86. DOI: <http://dx.doi.org/10.15421/432407> [Фахове видання України категорії Б]

(особистий внесок: спроектував два допоміжних типи вузла моделі хмарної системи; розглянув сценарії їх вбудовування в хмарну систему для визначення необхідності внутрішньої реконструкції її архітектури; розробив програмне рішення для імітації роботи спроектованих вузлів хмарної системи; провів експерименти роботи блоків з приділенням уваги до їх навантаження; дослідив "вузькі місця" цих блоків за параметрами надійності та отримав гнучкий інструмент використання розроблених програмних модулів для дослідження поведінки різних структур складної хмарної системи)

3. Божуха Д., Байбуз О. Модель спільного динамічного розвантаження хмарної архітектури з балансуванням рівнів // Системні технології. Регіональний міжвузівський збірник наукових праць. – Випуск 6(161). – Дніпро, 2025. – С. 152-157. DOI: <https://doi.org/10.34185/1562-9945-5-161-2025-15> [Фахове видання України категорії Б]

(особистий внесок: зосередив увагу на взаємодії рівнів ієрархії хмарної архітектури; розробив архітектурне рішення побудови чотирирівневої хмарної системи з визначеною послідовністю руху запиту/завдання та включенням парадигм Edge/Fog; навів опис математичної моделі чотирирівневої хмарної архітектури для аналізу навантаження; сформував набори різних комбінацій чотирирівневих хмарних систем для проведення імітаційних експериментів; запропонував програмне рішення імітації роботи хмарної системи)

4. Божуха Д.І., Байбуз О.Г. Дослідження моделей хмарних систем на прикладі найпростішого потоку // Актуальні проблеми автоматизації та інформаційних технологій. – Дніпро: ДНУ, 2025. – Т.29. – С. 84-91. DOI: <http://dx.doi.org/10.15421/432508> [Фахове видання України категорії Б]

(особистий внесок: підготував набори даних великого обсягу з зменшеною кількістю параметрів; провів експерименти з моделюванням чотирирівневої хмарної системи;

дослідив умову сталого режиму для моделей різних комбінацій чотирирівневих хмарних систем; розробив програмне рішення для аналізу показників продуктивності та візуалізував результати)

5. Пропозиції ради:

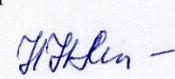
Запропоновано впровадити результати дисертаційної роботи Божухи Даніїла Ігоровича на тему «Розроблення методів аналізу та побудови структури хмарних систем» в освітній процес Дніпровського національного університету імені Олеся Гончара.

Голова вченої ради  
факультету прикладної математики  
та інформаційних технологій



Олена КІСЕЛЬОВА

Секретарка



Наталія ЛИСИЦЯ

## Додаток В Частина програмного коду

```
def K_mu( i, _lambda_0, _N, _number_channel, _number_mu):
    number_channel = _number_channel
    i = _i
    level = {}
    device = {}
    device[i] = {}
    set_k={} # Moved initialization here
    set_m={} # Moved initialization here
    for j in range(1, _N+1):
        device[i, j] = j
        for k in range(1, _number_channel+1):
            set_k[i,j,k]=k
            for m in range(_number_mu, _lambda_0+1, 10):
                set_m[i, j, k, m]=m
                level[i, j, k, m] = (i, device[i,j], set_k[i,j,k], set_m[i,j,k,m]) # Fixed: changed set_m[i,j,m,k] to
    set_m[i,j,k,m]
    return level
```

```
Directory_r_mu_values = {}
Set_Directory_r_mu_values_1={}
i = 1
n = 1
print(f"Підсистема {i}")
found_entries = False
Set_Directory_r_mu_values_1[i] = []
if i in Device_SubSystem:
    for key_tuple, val_tuple in Device_SubSystem[i].items():
        Set_Directory_r_mu_values_1[n]={}
        if key_tuple[0] == i:
            Set_Directory_r_mu_values_1[n]=(i, val_tuple[1], val_tuple[2], val_tuple[3])
            n = n + 1
print(f"\nЗначення підсистеми {i}: \n")
number_komb_1 = n-1
for index in range(1, number_komb_1 +1):
    print(f' {Set_Directory_r_mu_values_1[index]}')
Directory_r_mu_values = {}
Set_Directory_r_mu_values_2={}
i = 2
n = 1
print(f"Підсистема {i}")
found_entries = False
Set_Directory_r_mu_values_2[i] = []
if i in Device_SubSystem:
    for key_tuple, val_tuple in Device_SubSystem[i].items():
        Set_Directory_r_mu_values_2[n]={}
        if key_tuple[0] == i:
            Set_Directory_r_mu_values_2[n]=(i, val_tuple[1], val_tuple[2], val_tuple[3])
            n = n + 1
print(f"\nЗначення підсистеми {i}: \n")
number_komb_2 = n-1
for index in range(1, number_komb_2 +1):
```

```

    print(f'{Set_Directory_r_mu_values_2[index]}')
Directory_r_mu_values = {}
Set_Directory_r_mu_values_3={}
i = 3
n = 1
print(f'Підсистема {i}')
found_entries = False
Set_Directory_r_mu_values_3[i] = []
if i in Device_SubSystem:
    for key_tuple, val_tuple in Device_SubSystem[i].items():
        Set_Directory_r_mu_values_3[n]={}
        if key_tuple[0] == i:
            Set_Directory_r_mu_values_3[n]=(i, val_tuple[1], val_tuple[2], val_tuple[3])
            n = n + 1
print(f'\nЗначення підсистеми {i}: \n')
number_komb_3 = n-1
for index in range(1, number_komb_3 + 1):
    print(f'{Set_Directory_r_mu_values_3[index]}')
Directory_r_mu_values = {}
Set_Directory_r_mu_values_4={}
i = 4
n = 1
print(f'Підсистема {i}')
found_entries = False
Set_Directory_r_mu_values_4[i] = []
if i in Device_SubSystem:
    for key_tuple, val_tuple in Device_SubSystem[i].items():
        Set_Directory_r_mu_values_4[n]={}
        if key_tuple[0] == i:
            Set_Directory_r_mu_values_4[n]=(i, val_tuple[1], val_tuple[2], val_tuple[3])
            n = n + 1
print(f'\nЗначення підсистеми {i}: \n')
number_komb_4 = n-1
for index in range(1, number_komb_4 + 1):
    print(f'{Set_Directory_r_mu_values_4[index]}')

# Формування комбінацій системи
number_komb_1_0 = 1
number_komb_2_0 = 1
number_komb_3_0 = 1
number_komb_4_0 = 1
R_system = []
number_model = 1
header = ['Model_Number',
          'Subsystem1_Device', 'Subsystem1_Channel', 'Subsystem1_Mu',
          'Subsystem2_Device', 'Subsystem2_Channel', 'Subsystem2_Mu',
          'Subsystem3_Device', 'Subsystem3_Channel', 'Subsystem3_Mu',
          'Subsystem4_Device', 'Subsystem4_Channel', 'Subsystem4_Mu']
for i in range(number_komb_1_0, number_komb_1 + 1):
    for j in range(number_komb_2_0, number_komb_2 + 1):
        for k in range(number_komb_3_0, number_komb_3 + 1):
            for l in range(number_komb_4_0, number_komb_4 + 1):
                R_system.append([number_model,
                                Set_Directory_r_mu_values[i][0], Set_Directory_r_mu_values[i][1],

```

```

Set_Directory_r_mu_values[i][2],
                                Set_Directory_r_mu_values[j][0], Set_Directory_r_mu_values[j][1],
Set_Directory_r_mu_values[j][2],
                                Set_Directory_r_mu_values[k][0], Set_Directory_r_mu_values[k][1],
Set_Directory_r_mu_values[k][2],
                                Set_Directory_r_mu_values[l][0], Set_Directory_r_mu_values[l][1],
Set_Directory_r_mu_values[l][2]
    ])
    number_model = number_model + 1

```

```

output_excel_path_device = 'R_system_Device.xlsx'
output_excel_path_device_mu = 'R_system_Device_mu.xlsx'
R_system_Device = []
R_system_Device_mu = []
device_indices = {1, 4, 7, 10}
Model_external_arrivals_to_queues = []
Model_routing_matrix = []
for i_row_idx in range(len(R_system)):
    current_R_system_row = R_system[i_row_idx]
    current_model_number = current_R_system_row[0]
    R_system_Device_row = [current_model_number]
    R_system_Device_mu_row = [current_model_number]
    for index in range(1,11,3):
        device_val = current_R_system_row[index]
        match device_val:
            case 1:
                R_system_Device_row.append(1)
                if index == 1:
                    R_system_Device_mu_row.append(current_R_system_row[3])
                else:
                    if index == 4:
                        R_system_Device_mu_row.append(current_R_system_row[6])
                    else:
                        if index == 7:
                            R_system_Device_mu_row.append(current_R_system_row[9])
                        else:
                            if index == 10:
                                R_system_Device_mu_row.append(current_R_system_row[12])
            case 2:
                R_system_Device_row.extend([1, 1])
                if index == 1:
                    R_system_Device_mu_row.extend([current_R_system_row[3], current_R_system_row[3]])
                else:
                    if index == 4:
                        R_system_Device_mu_row.extend([current_R_system_row[6], current_R_system_row[6]])
                    else:
                        if index == 7:
                            R_system_Device_mu_row.extend([current_R_system_row[9], current_R_system_row[9]])
                        else:
                            if index == 10:
                                R_system_Device_mu_row.extend([current_R_system_row[12], current_R_system_row[12]])
            case 3:
                R_system_Device_row.extend([1, 1, 1])
                if index == 1:

```

```

        R_system_Device_mu_row.extend([current_R_system_row[3], current_R_system_row[3],
current_R_system_row[3]])
    else:
        if index == 4:
            R_system_Device_mu_row.extend([current_R_system_row[6], current_R_system_row[6],
current_R_system_row[6]])
        else:
            if index == 7:
                R_system_Device_mu_row.extend([current_R_system_row[9], current_R_system_row[9],
current_R_system_row[9]])
            else:
                if index == 10:
                    R_system_Device_mu_row.extend([current_R_system_row[12], current_R_system_row[12],
current_R_system_row[12]])
        case _:
            R_system_Device_row.append(device_val)
            R_system_Device_mu_row.append(device_val)
            R_system_Device.append(R_system_Device_row)
            R_system_Device_mu.append(R_system_Device_mu_row)
df_Device = pd.DataFrame(R_system_Device)
df_Device.to_excel(output_excel_path_device, index=False)
df_Device_mu = pd.DataFrame(R_system_Device_mu)
df_Device_mu.to_excel(output_excel_path_device_mu, index=False)
print(f"Згенеровано {len(R_system_Device)} комбінацій та збережено в Ексел-файл за адресою:
{output_excel_path_device}")

```

# Будування матриць маршрутизації

```
import decimal
```

```
import copy
```

```
output_excel_path_Model_external_arrivals_to_queues = 'Model_external_arrivals_to_queues.xlsx'
```

```
output_excel_path_Model_routing_matrix = 'Model_routing_matrix.xlsx'
```

```
N=[]
```

```
ES=[]
```

```
FR=[]
```

```
CL=[]
```

```
S_model = []
```

```
S_subsystem = []
```

```
Right_Part = []
```

```
for i in range(len(all_models_e_values)):
```

```
    N.append(R_system[i][1]) # Appending to list N
```

```
    ES.append(R_system[i][4]) # Appending to list ES
```

```
    FR.append(R_system[i][7]) # Appending to list FR
```

```
    CL.append(R_system[i][10]) # Appending to list CL
```

```
    S_subsystem.append([decimal.Decimal('0'), decimal.Decimal('0'), decimal.Decimal('0'),
decimal.Decimal('0')])
```

```
    # N subsystem
```

```
    for k in range(0, N[i]):
```

```
        divisor_N = df_all_models_e_values.iloc[i][k]
```

```
        if divisor_N is None:
```

```
            divisor_N = decimal.Decimal('1e-9')
```

```

if divisor_N == 0:
    divisor_N = decimal.Decimal('1e-9')
if(R_system[i][1]==1):
    S_subsystem[i][0] = S_subsystem[i][0] +
decimal.Decimal(R_system[i][2])*decimal.Decimal(R_system[i][3])/divisor_N
    elif(R_system[i][1]==2):
    S_subsystem[i][0] = S_subsystem[i][0] +
decimal.Decimal(R_system[i][2])*decimal.Decimal(R_system[i][3])/divisor_N
    S_subsystem[i][0] = S_subsystem[i][0] +
decimal.Decimal(R_system[i][2])*decimal.Decimal(R_system[i][3])/divisor_N
    elif(R_system[i][1]==3):
    S_subsystem[i][0] = S_subsystem[i][0] +
decimal.Decimal(R_system[i][2])*decimal.Decimal(R_system[i][3])/divisor_N
    S_subsystem[i][0] = S_subsystem[i][0] +
decimal.Decimal(R_system[i][2])*decimal.Decimal(R_system[i][3])/divisor_N
    S_subsystem[i][0] = S_subsystem[i][0] +
decimal.Decimal(R_system[i][2])*decimal.Decimal(R_system[i][3])/divisor_N
# ES subsystem
sum_dev = R_system[i][1]
for k in range(0, ES[i]):
    divisor_ES = df_all_models_e_values.iloc[i][sum_dev+k]
    if divisor_ES is None:
        divisor_ES = decimal.Decimal('1e-9')
    if divisor_ES == 0:
        divisor_ES = decimal.Decimal('1e-9')
    if(R_system[i][4]==1):
    S_subsystem[i][1] = S_subsystem[i][1] +
decimal.Decimal(R_system[i][5])*decimal.Decimal(R_system[i][6])/divisor_ES
    elif(R_system[i][4]==2):
    S_subsystem[i][1] = S_subsystem[i][1] +
decimal.Decimal(R_system[i][5])*decimal.Decimal(R_system[i][6])/divisor_ES
    S_subsystem[i][1] = S_subsystem[i][1] +
decimal.Decimal(R_system[i][5])*decimal.Decimal(R_system[i][6])/divisor_ES
    elif(R_system[i][4]==3):
    S_subsystem[i][1] = S_subsystem[i][1] +
decimal.Decimal(R_system[i][5])*decimal.Decimal(R_system[i][6])/divisor_ES
    S_subsystem[i][1] = S_subsystem[i][1] +
decimal.Decimal(R_system[i][5])*decimal.Decimal(R_system[i][6])/divisor_ES
    S_subsystem[i][1] = S_subsystem[i][1] +
decimal.Decimal(R_system[i][5])*decimal.Decimal(R_system[i][6])/divisor_ES
# FR subsystem
sum_dev = R_system[i][1]+ R_system[i][4]
for k in range(0, FR[i]): # Fixed range syntax, starts from 0 to FR[i]-1
    divisor_FR = df_all_models_e_values.iloc[i][sum_dev+k]
    if divisor_FR is None:
        divisor_FR = decimal.Decimal('1e-9')
    if divisor_FR == 0:
        divisor_FR = decimal.Decimal('1e-9')
    if(R_system[i][7]==1):
    S_subsystem[i][2] = S_subsystem[i][2] +
decimal.Decimal(R_system[i][8])*decimal.Decimal(R_system[i][9])/divisor_FR
    elif(R_system[i][7]==2):
    S_subsystem[i][2] = S_subsystem[i][2] +
decimal.Decimal(R_system[i][8])*decimal.Decimal(R_system[i][9])/divisor_FR
    S_subsystem[i][2] = S_subsystem[i][2] +

```

```

decimal.Decimal(R_system[i][8])*decimal.Decimal(R_system[i][9])/divisor_FR
    elif(R_system[i][7]==3):
        S_subsystem[i][2] = S_subsystem[i][2] +
decimal.Decimal(R_system[i][8])*decimal.Decimal(R_system[i][9])/divisor_FR
        S_subsystem[i][2] = S_subsystem[i][2] +
decimal.Decimal(R_system[i][8])*decimal.Decimal(R_system[i][9])/divisor_FR
        S_subsystem[i][2] = S_subsystem[i][2] +
decimal.Decimal(R_system[i][8])*decimal.Decimal(R_system[i][9])/divisor_FR
# CL subsystem
sum_dev = R_system[i][1]+R_system[i][4]+R_system[i][7]
for k in range(0, CL[i]): # Fixed range syntax, starts from 0 to CL[i]-1
    divisor_CL = df_all_models_e_values.iloc[i][sum_dev+k]
    if divisor_CL is None:
        divisor_CL = decimal.Decimal('1e-9') # Treat None as a very small number
    if divisor_CL == 0:
        divisor_CL = decimal.Decimal('1e-9') # Replace 0 with a very small Decimal
    if(R_system[i][10]==1):
        S_subsystem[i][3] = S_subsystem[i][3] +
decimal.Decimal(R_system[i][11])*decimal.Decimal(R_system[i][12])/divisor_CL
    elif(R_system[i][10]==2):
        S_subsystem[i][3] = S_subsystem[i][3] +
decimal.Decimal(R_system[i][11])*decimal.Decimal(R_system[i][12])/divisor_CL
        S_subsystem[i][3] = S_subsystem[i][3] +
decimal.Decimal(R_system[i][11])*decimal.Decimal(R_system[i][12])/divisor_CL
    elif(R_system[i][10]==3):
        S_subsystem[i][3] = S_subsystem[i][3] +
decimal.Decimal(R_system[i][11])*decimal.Decimal(R_system[i][12])/divisor_CL
        S_subsystem[i][3] = S_subsystem[i][3] +
decimal.Decimal(R_system[i][11])*decimal.Decimal(R_system[i][12])/divisor_CL
        S_subsystem[i][3] = S_subsystem[i][3] +
decimal.Decimal(R_system[i][11])*decimal.Decimal(R_system[i][12])/divisor_CL

S_model.append(S_subsystem[i][0]+S_subsystem[i][1]+S_subsystem[i][2]+S_subsystem[i][3])
Right_Part.append(S_model[i]/(decimal.Decimal(N[i])+decimal.Decimal(ES[i])+decimal.Decimal(FR[i
])+decimal.Decimal(CL[i])))

for model_number, levels_data in X_device.items():
    v_task_level_device[model_number] = {}
    for level, devices_data in levels_data.items():
        v_task_level_device[model_number][level] = {}
        for device_index, device_info in devices_data.items():
            device_total_mu = device_info[1] # device_total_mu is the second element in device_info
            if device_total_mu > 0:
                v_task_level_device[model_number][level][device_index] = 1 / device_total_mu
            else:
                v_task_level_device[model_number][level][device_index] = 0

for model, levels_data in list(v_task_level_device.items()):
    print(f" Model {model}:")
    for level, devices_data in list(levels_data.items()):
        print(f" Level {level}:")
        for device_idx, speed in list(devices_data.items()):
            print(f" Device {device_idx}: {speed}")

for model_number, levels_data_R_system in R_system.items():

```

```

fraction_X_device[model_number] = {}

for level, devices_data_R_system in levels_data_R_system.items():
    routing_factor = routing_factors_per_level.loc[(model_number, level)]
    if (model_number, level) in routing_factors_per_level.index else 0
    device_fractions = fraction_X_level[model_number][level]
    fraction_X_device[model_number][level] = {}
    for device_idx, canals_data_R_system in devices_data_R_system.items():
        device_fraction = device_fractions[device_idx - 1]
fraction_X_device[model][level][device_index]
    fraction_X_device[model_number][level][device_idx] = {}
    for canal_idx in canals_data_R_system.keys():
        calculated_fraction = routing_factor * device_fraction
fraction_X_device[model_number][level][device_idx][canal_idx] = calculated_fraction

total_system_throughput = {}

for model_number in R_system.keys():
    model_throughput = 0
    if model_number in fraction_X_device and model_number in v_task_level_device:
        for level, devices_data in fraction_X_device[model_number].items():
            if level in v_task_level_device[model_number]:
                for device_idx, canals_data in devices_data.items():
                    if device_idx in v_task_level_device[model_number][level]:
                        device_speed = v_task_level_device[model_number][level][device_idx]
                        for canal_idx, fraction in canals_data.items():
                            model_throughput += fraction * device_speed
    total_system_throughput[model_number] = model_throughput

#Додаткова функція
def calculate_throughput_for_model(model_number, df_full, routing_matrix_dir, s1_mu_override=None,
s2_mu_override=None, s3_mu_override=None, s4_mu_override=None, routing_matrix_df=None):
    df_R_system = df_full[df_full['Model_Number'] == model_number].copy()
    if df_R_system.empty:
        return 0, {}
    if s1_mu_override is not None:
        df_R_system.loc[:, 'S1_mu'] = s1_mu_override
    if s2_mu_override is not None:
        df_R_system.loc[:, 'S2_mu'] = s2_mu_override
    if s3_mu_override is not None:
        df_R_system.loc[:, 'S3_mu'] = s3_mu_override
    if s4_mu_override is not None:
        df_R_system.loc[:, 'S4_mu'] = s4_mu_override
    R_system = {}
    for index, row in df_R_system.iterrows():
        current_model_number = row['Model_Number']
        R_system[current_model_number] = {}
        for level in range(1, 5):
            s_device_col = f'S{level}_Device'
            s_mu_col = f'S{level}_mu'
            s_canal_col = f'S{level}_Canal'
            if s_device_col in row and s_mu_col in row and s_canal_col in row:
                s_device = int(row[s_device_col])
                s_mu = row[s_mu_col]
                s_canal = int(row[s_canal_col])

```

```

    R_system[current_model_number][level] = {}
    for device_index in range(1, s_device + 1):
        R_system[current_model_number][level][device_index] = {}
        for canal_index in range(1, s_canal + 1):
            R_system[current_model_number][level][device_index][canal_index] = [s_mu, 0]
X_level = {}
X_device = {}
for model_num_R, levels_data in R_system.items():
    X_level[model_num_R] = {}
    X_device[model_num_R] = {}
    for level, devices_data in levels_data.items():
        level_total_devices = 0
        level_total_mu = 0
        X_device[model_num_R][level] = {}

        for device_index, canals_data in devices_data.items():
            level_total_devices += 1
            device_total_canals = 0
            device_total_mu = 0
            for canal_index, mu_data in canals_data.items():
                s_mu = mu_data[0]
                level_total_mu += s_mu
                device_total_mu += s_mu
                device_total_canals += 1
            X_device[model_num_R][level][device_index] = [device_total_canals, device_total_mu]
        X_level[model_num_R][level] = [level_total_devices, level_total_mu]
fraction_X_level = {}
for model_num_X, levels_data in X_level.items():
    fraction_X_level[model_num_X] = {}
    for level, values in levels_data.items():
        total_devices = values[0]
        if total_devices > 0:
            fraction_per_device = 1 / total_devices
            fraction_X_level[model_num_X][level] = [fraction_per_device] * total_devices
        else:
            fraction_X_level[model_num_X][level] = []
if routing_matrix_df is not None:
    local_routing_matrix = routing_matrix_df[routing_matrix_df['Model_Number'] ==
model_number].copy()
else:
    local_routing_matrix_file = None
    for item in os.listdir(routing_matrix_dir):
        if item.startswith('routing_matrix_long_format_') and item.endswith('.xlsx'):
            match = re.search(r'routing_matrix_long_format_(\d+)_to_(\d+)\.xlsx', item)
            if match:
                start_range = int(match.group(1))
                end_range = int(match.group(2))
                if start_range <= model_number <= end_range:
                    local_routing_matrix_file = os.path.join(routing_matrix_dir, item)
                    break

if not local_routing_matrix_file:
    return 0, {} # Changed from return 0

local_routing_matrix = pd.read_excel(local_routing_matrix_file)

```

```

        local_routing_matrix = local_routing_matrix[local_routing_matrix['Model_Number'] ==
model_number]
        routing_factors_to_queue = local_routing_matrix.groupby(['Model_Number',
'To_Queue'])['Probability'].sum()
        all_levels_multiindex = pd.MultiIndex.from_product([[model_number], range(1, 5)],
names=['Model_Number', 'From_Queue']) # Reusing From_Queue name as it's the index name
        routing_factors_per_level = pd.Series(0.0, index=all_levels_multiindex)
        for (model_num_rm, to_queue), prob_sum in routing_factors_to_queue.items():
            if to_queue in range(1, 5): # Only consider levels 1 to 4 from R_system
                routing_factors_per_level.loc[(model_num_rm, to_queue)] = prob_sum
        fraction_X_device = {}
        for model_num_R, levels_data_R_system in R_system.items(): # Should only be one model_num_R
here
            fraction_X_device[model_num_R] = {}
            for level, devices_data_R_system in levels_data_R_system.items():
                routing_factor = routing_factors_per_level.loc[(model_num_R, level)] if (model_num_R, level) in
routing_factors_per_level.index else 0
                if model_num_R in fraction_X_level and level in fraction_X_level[model_num_R]:
                    device_fractions = fraction_X_level[model_num_R][level]
                else:
                    device_fractions = [] # No device fractions for this level
                    fraction_X_device[model_num_R][level] = {} # Initialize for the current level
                for device_idx, canals_data_R_system in devices_data_R_system.items():
                    if (device_idx - 1) < len(device_fractions):
                        device_fraction = device_fractions[device_idx - 1]
                    else:
                        device_fraction = 0 # No corresponding device fraction
                    fraction_X_device[model_num_R][level][device_idx] = {}
                    for canal_idx in canals_data_R_system.keys():
                        calculated_fraction = routing_factor * device_fraction
                        fraction_X_device[model_num_R][level][device_idx][canal_idx] = calculated_fraction
        v_task_level_device = {}
        for model_num_X, levels_data in X_device.items():
            v_task_level_device[model_num_X] = {}
            for level, devices_data in levels_data.items():
                v_task_level_device[model_num_X][level] = {}
                for device_index, device_info in devices_data.items():
                    device_total_mu = device_info[1]
                    if device_total_mu > 0:
                        v_task_level_device[model_num_X][level][device_index] = 1 / device_total_mu
                    else:
                        v_task_level_device[model_num_X][level][device_index] = 0
        total_model_throughput = 0
        level_wise_throughput = {1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0}
        if model_number in fraction_X_device and model_number in v_task_level_device:
            for level, devices_data in fraction_X_device[model_number].items():
                current_level_throughput = 0.0
                if level in v_task_level_device[model_number]:
                    for device_idx, canals_data in devices_data.items():
                        if device_idx in v_task_level_device[model_number][level]:
                            device_speed = v_task_level_device[model_number][level][device_idx]
                            for canal_idx, fraction in canals_data.items():
                                throughput_contribution = fraction * device_speed
                                total_model_throughput += throughput_contribution
                                current_level_throughput += throughput_contribution

```

```

    level_wise_throughput[level] = current_level_throughput
return total_model_throughput, level_wise_throughput

#Модуль пошуку моделей
available_routing_ranges = []
for item in os.listdir(routing_matrix_dir):
    if item.startswith('routing_matrix_long_format_') and item.endswith('.xlsx'):
        match = re.search(r'routing_matrix_long_format_(\d+)_to_(\d+)\.xlsx', item)
        if match:
            start_range = int(match.group(1))
            end_range = int(match.group(2))
            available_routing_ranges.append((start_range, end_range))
unique_models_in_df = df_full['Model_Number'].unique()
models_to_process = []
for model_num in unique_models_in_df:
    for start, end in available_routing_ranges:
        if start <= model_num <= end:
            models_to_process.append(model_num)
            break
if len(models_to_process) > 100:
    models_to_process_sample = random.sample(models_to_process, 100)
else:
    models_to_process_sample = models_to_process
all_models_throughput = {}
for current_model in models_to_process_sample:
    total_throughput_value, _ = calculate_throughput_for_model(current_model, df_full,
routing_matrix_dir)
    if total_throughput_value > 0: # Only store if throughput was successfully calculated and is positive
        all_models_throughput[current_model] = total_throughput_value
sorted_throughput = sorted(all_models_throughput.items(), key=lambda item: item[1], reverse=True)
top_5_models = sorted_throughput[:5]

```