Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем
(повна назва інституту/факультету)

Кафедра телекомунікацій
(повна назва кафедри)

«На правах рукопису»
УДК _____

До захисту допущено

Завідувач кафедри
_____ Сергій Кравчук
(підпис) (ініціали, прізвище)
“___”_____2020 р.

Магістерська дисертація
на здобуття освітнього ступеня «магістр»

Спеціальність 172 Телекомунікації та радіотехніка
(код і назва спеціальності)

За освітньо-професійною програмою Інженерія та програмування інфокомунікацій

на тему: «Методи підвищення ефективності алгоритмів та методів балансування навантаження у хмарних середовищах інфокомунікаційних систем»

Виконав: студент 2 курсу, групи ТЗ-91мп
(шифр групи)

Боровський Богдан Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доцент кафедри ТК Явіся В.С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент . _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 р.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем
(повна назва)

Кафедра телекомунікацій
(повна назва)

Спеціальність 172 Телекомунікації та радіотехніка
(код і назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою Інженерія та програмування інфокомунікацій.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Явіся В.С.
(підпис) (ініціали, прізвище)
« 20 » січня 2020 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Боровському Богдану Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Методи підвищення ефективності алгоритмів та методів балансування навантаження у хмарних середовищах інфокомунікаційних систем»
науковий керівник дисертації к.т.н., доц. каф. ТК Явіся Валерій Сергійович ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «03» листопада 2020 р. №3208-с

2. Строк подання студентом дисертації 11.12.2020

3. Об'єкт дослідження: процес балансування навантаження в "хмарних" середовищах

4. Предмет дослідження: методи та алгоритми розподілу навантаження.

5. Перелік завдань, які потрібно розробити:
   1. Аналіз особливостей функціонування та масштабування веб-додатків у хмарних середовищах
   2. Розглянути методи та алгоритми розподілу навантаження, сформувати рекомендації по вибору кінцевих рішень

3. Дослідити шляхи підвищення ефективності існуючих способів балансування навантаження

6. Орієнтовний перелік ілюстративного матеріалу: презентація по темі «Методи підвищення ефективності алгоритмів та методів балансування навантаження у хмарних середовищах інфокомунікаційних систем»: Особливості проектування та роботи хмарних веб-сервісів; Масштабування систем на базі хмарної інфраструктури; Загальна концепція балансування навантаження хмарних додатків (проекція на рівні моделі OSI, методи, конфігурації та алгоритми); Порівняння способів балансування навантаження; Запропоновані шляхи підвищення ефективності

7. Орієнтовний перелік публікацій:

1) Bohdan Borovskyi. A Comparison between Open-Source Platforms for Cloud Computing / XIII International Scientific Conference "Modern Challenges in Telecommunications" MCT-2019. / Conference proceedings. Kyiv. Igor Sikorsky Kyiv Polytechnic Institute, 2019 – p. 377. / ISSN (print) 2663-502X

2) Bohdan Borovskyi. Load Balancing Algorithms and challenges in cloud computing. / 7th International Scientific and Practical Internet Conference “Modern Movement of Science” June 6-7, Dnipro / International Electronic Scientific and Practical Journal "WayScience" – 2019 – №1.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 28.10.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів роботи	Примітка
1	Збір інформації про технології хмарних обчислень в інфокомунікаційних системах	01.11.2019 - 01.12.2019	
2	Вивчення особливостей розгортання та масштабування хмарних веб-додатків	02.12.2020 – 01.02.2020	
3	Аналіз методів та алгоритмів розподілу навантаження в хмарі	02.02.2019 – 23.05.2020	
4	Розробка рекомендації по вибору оптимального механізму балансування навантаження	23.05.2020 - 01.07.2020	

5	Пошук методів підвищення ефективності розподілу навантаження в хмарному середовищі	02.07.2020 - 01.10.2020	
6	Обробка та систематизація одержаних даних, написання загальних висновків	02.10.2020 - 21.11.2020	
5	Підведення підсумків, оформлення текстової і графічної частини магістерської дисертації	22.11.2020 - 01.12.2020	

Студент _____ Боровський Б.М.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Явіся В.С.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Методи підвищення ефективності алгоритмів та методів балансування навантаження у хмарних середовищах інфокомунікаційних систем.

Робота виконана на 128 сторінках, містить 23 ілюстрації, 17 таблиць та 11 формул . При підготовці використовувалася література з 66 різних джерел.

Мета роботи: комплексний аналіз алгоритмів та способів балансування навантаження інфокомунікаційних систем, розроблених з використанням "хмарних" технологій, дослідження особливостей розгортання, функціонування та масштабування таких систем, пошук методів підвищення ефективності розподілу навантаження в хмарному середовищі.

Об’єкт дослідження: процес балансування навантаження в "хмарних" середовищах.

Предмет дослідження: методи та алгоритми розподілу навантаження.

В роботі проведено загальний огляд технології хмарних обчислень та визначені перспективи застосування хмарних середовищ для розгортання інфокомунікаційних систем. Було розглянуто існуючі способи та алгоритми балансування навантаження, проаналізовано їх недоліки та переваги, що є передумовами для створення нових рішень в області систем розподілу навантаження. Розроблено рекомендації по вибору оптимального механізму балансування навантаження в залежності від критеріїв, встановлених для проектованого веб-сервісу. Запропоновано методи підвищення ефективності підсистеми розподілу навантаження в хмарному середовищі.

Ключові слова: хмарні технології, хмарні сервіси, балансування навантаження, масштабованість, розподілений веб-додаток, хмарні моделі надання послуг.

ABSTRACT

on master's thesis

on topic: Methods of improving the efficiency of load balancing algorithms and techniques in cloud environments of infocommunication systems.

Work carried out on 128 pages containing 23 figures, 17 tables and 11 formulas. The paper was written with references to 66 different sources.

The purpose of the master's thesis lies in comprehensive analysis of algorithms and methods of load balancing infocommunication systems developed using "cloud" technologies, study of the deployment, operation and scaling of such systems, search for methods to increase the efficiency of load distribution in a cloud environment.

Object of research: the process of load balancing in "cloud" environments.

Subject of research: methods and algorithms of load distribution.

The paper provides a general overview of cloud computing technology and identifies prospects for the use of cloud environments for the deployment of infocommunication systems. The existing methods and algorithms of load balancing were considered, their disadvantages and advantages were analyzed, which are prerequisites for creating new solutions in the field of load distribution systems. Recommendations for choosing the optimal load balancing mechanism depending on the criteria established for the designed web service have been developed. Methods for increasing the efficiency of the load distribution subsystem in a cloud environment are proposed.

Keywords: cloud technologies, cloud services, load balancing, scalability, distributed web application, cloud service delivery models.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DNS – Domain Name System, програма, призначена для відповідей на DNS-запити за відповідним протоколом.1

Віртуальна машина – повністю ізольований програмний контейнер, який працює з власною ОС і додатками, незалежними від фізичного комп'ютера (хоста) на якому цей контейнер запущено.

SaaS – Software as a Service, модель поширення через хмарне середовище програм споживачам, при якій постачальник розробляє веб-програму, розміщує її й управляє нею

PaaS – Platform as a Service, модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ.

IaaS – Infrastructure-as-a-service, модель обслуговування, в межах якої споживачу надається можливість керувати засобами обробки та збереження, комунікаційними мережами, та іншими фундаментальними обчислювальними ресурсами.

API – Application Programming Interface, набір визначень взаємодії різнотипного програмного забезпечення.

SSL – Secure Sockets Layer, протокол, що забезпечує конфіденційність та цілісність даних, котрі пересилаються між веб-серверами і веб-браузерами.

ПЗ – програмне забезпечення.

NAPT – Network Adress and Port Translation, найпоширеніший тип NAT (механізм заміни мережевих адрес), в якому декілька приватних мережевих адрес транслюються в різні порти однієї публічної IP-адреси.

ВСТУП

Сьогодні все більше телекомунікаційних компаній прагнуть створити власні веб-сервіси як для внутрішнього використання, так і доступні широкому загалу Інтернет-користувачів. Все частіше для таких цілей користуються готовими рішеннями на базі приватних і публічних хмарних сервісів. Розробники подібних сервісів характеризують хмарні технології як інноваційне рішення для побудови високонавантажених систем, при якому доступ до динамічно масштабованих обчислювальних ресурси або програмних платформ надається у вигляді сервісу під управлінням постачальника послуг.

Подібні технології мають величезний потенціал, оскільки вони вирішують проблему «мінімальних системних вимог» сучасного програмного забезпечення (вимогливості програм та додатків до ресурсів пристроїв кінцевих користувачів).

Для забезпечення безперебійної роботи вузлів комп'ютерної мережі на стороні хмарного постачальника використовується спеціалізоване проміжне програмне забезпечення, яке контролює стан обладнання та програм, балансування навантаження, надання ресурсів для вирішення проблеми. Одним з головних рішень для згладжування нерівномірного навантаження на послуги є розміщення рівня віртуалізації серверів між рівнем програмних послуг та обладнання. В умовах віртуалізації балансування навантаження може здійснюватися за допомогою програмного розподілу віртуальних серверів на реальних

Еволюція мереж зв'язку неминуче відбивається на кількості та структурі внутрішнього трафіку в інфраструктурі постачальників телекомунікаційних послуг та інтернет-сервісів. Згідно з численними дослідженнями, загальний обсяг даних, переданих через мережу WAN, показує стійке експоненціальне

зростання, незважаючи на те, що ця тенденція збережеться в найближчі роки [1,3].

Зі збільшенням кількості даних поведінка трафіку в сучасній глобальній мережі демонструє таку негативну особливість, як нестабільність навантаження, яка характеризується можливістю виникнення непередбачуваних сплесків інтенсивності передачі. Причин такої нестабільності багато. Прекрасним прикладом може бути збільшення кількості користувачів, спричинене вірусною природою поширення даних в популярних засобах масової інформації. Потужні маркетингові кампанії, лавиноподібний приріст нових користувачів, нові способи взаємодії користувачів між собою, засновані на широкому використанні соціальних послуг, масове онлайн-мовленні - все це змушує говорити про нову природу виникаючих перевантажень.

Дослідники відзначають, що сьогоднішні мережі страждають від браку пропускної здатності. Згідно з дослідженнями, близько 20-30% WAN-каналів проходять через перевантажені ділянки. У той же час спостерігається значна нерівномірність розподілу навантажень між канальними ресурсами, що вказує на те, що процедури управлінням трафіком є неефективними у поточному середовищі [2].

Виходом із цієї ситуації є використання спеціальних методів врівноваження трафіку для ефективного розподілу навантаження відповідно до існуючих ресурсів.

Питання планування ємності ресурсів слід вирішувати на ранніх етапах побудови мережі або інфраструктури проекту. Спочатку проблему недостатньої місткості вузлів через збільшення навантажень можна вирішити шляхом збільшення їх потужностей або оптимізації алгоритмів їх використання, програмного коду тощо [3]. Але рано чи пізно настає момент, коли цих заходів недостатньо. І тоді потрібно використовувати спеціалізовані методи балансування навантаження.

Балансування навантаження реалізується за допомогою апаратних, програмних інструментів або комбінації обох. Раніше було чітке розмежування апаратного та програмного балансування навантаження. Тепер, у зв'язку з розробкою та вдосконаленням апаратного та програмного забезпечення, що виконує задачу балансування навантаження, межі між ними зникають [4]. Припустимо, що якщо використовується концентратор, коммутатор, контролери доставки додатків (Application Delivery Controllers або ADC), то подібну реалізацію можна вважати чисто апаратним рішенням.

Під час використання лише ресурсів сервера (фізичного або віртуального), доцільно говорити, що відбувається балансування завантаження програмними засобами. Апаратні рішення часто використовується на канальному, мережевому та транспортному рівнях моделі OSI. Зазвичай апаратне балансування завантаження значно швидше, ніж програмні рішення, але його головним недолік є вартість.

Програмного рішення на відміну від апаратного балансиру навантаження працюють на основі звичайних операційних систем та стандартних апаратних компонентів (подібних до тих, що використовуються в звичайних персональних комп'ютерах). Програмні рішення працюють на спеціалізованих вузлах балансування навантаження (виділеному обладнанні) або безпосередньо в додатку. Апаратні пристрої для балансування навантаження називаються брокерами мережевих пакетів (Network Packet Broker або NPB) або мережевими моніторинговими комутаторами, мають мережеві інтерфейси з величезною пропускною здатністю (зазвичай - 100GbE) і працюють на 2 та 3 рівнях мережевої моделі OSI [5].

NPB вбудований у мережевий пристрій серверної стійки. Він отримує та агрегує трафік з портів мережевого пристрою , яким він маніпулює в майбутньому. Основна і найважливіша функція NPB - балансування навантаження. Для балансування навантаження на 3-7 рівнях моделі OSI використовують контролери доставки додатків (ADC). Процедура

балансування навантаження здійснюється за допомогою складних алгоритмів та методів, які відповідають наступним рівням мережевої моделі OSI: канальному; мережевому; транспортному та рівню додатків [Бажин, 2010].

При побудові власних сервісів на основі хмарних технологій особливе пильно потрібно контролювати такі аспекти системи як масштабованість та балансування навантаження. Масштабованість – важливий характеристика будь-якої електронної системи (програмного комплексу, кластеру бази даних, мережі і т. П). В системі, що погано масштабується додавання ресурсів перестає бути ефективним рішенням (дає незначне підвищення продуктивності), а після певної межі– стає взагалі неефективним, шо в свою чергу впливає на безперебійність надання послуг. В той же час оптимальне балансування навантаження має позитивний вплив на ефективність роботи "хмарної" системи та сприяє підвищенню її відмовостійкості.

Значимість роботи полягає у вдосконаленні вже існуючих підходів до розв’язання однієї з основних проблем сучасних розподілених систем, а саме ефективного розподілу навантаження між вузлами подібної системи.

Метою магістерської дисертації є комплексний аналіз алгоритмів та способів балансування навантаження інфокомунікаційних систем, розроблених з використанням "хмарних" технологій, дослідження особливостей розгортання, функціонування та масштабування таких систем, пошук методів підвищення ефективності розподілу навантаження в хмарному середовищі.

Задачі, на вирішення яких спрямовано роботу:

1. Аналіз особливостей функціонування та масштабування веб-додатків у хмарних середовищах.
2. Систематизація та порівняння методів та алгоритмів розподілу навантаження для “хмарних” систем, формування рекомендацій по їх вибору.

3. Дослідження шляхів оптимізації ефективності існуючих методів балансування навантаження.

Об'єктом дослідження є процес балансування навантаження в "хмарних" середовищах, а предметом дослідження – методи та алгоритми розподілу навантаження, критерії їх порівняння та обґрунтування вибору способів балансування навантаження відповідно до вимог сучасних інфокомунікаційних систем.

РОЗДІЛ 1. ОПИС ТЕХНОЛОГІЇ ХМАРНИХ ОБЧИСЛЕНЬ

Хмарні обчислення (англ. cloud computing) – це модель забезпечення повсюдного і зручного мережевого доступу на вимогу до загального пулу (англ. pool) зконфігурованих обчислювальних ресурсів (наприклад, мереж передачі даних, пристроїв зберігання, додатків і сервісів – як разом, так і окремо), які можуть бути отримані та звільнені з мінімальними експлуатаційними витратами[1]. Діаграма, що показує можливі варіанти використання «хмарних» обчислень подана на рис. 1.1.

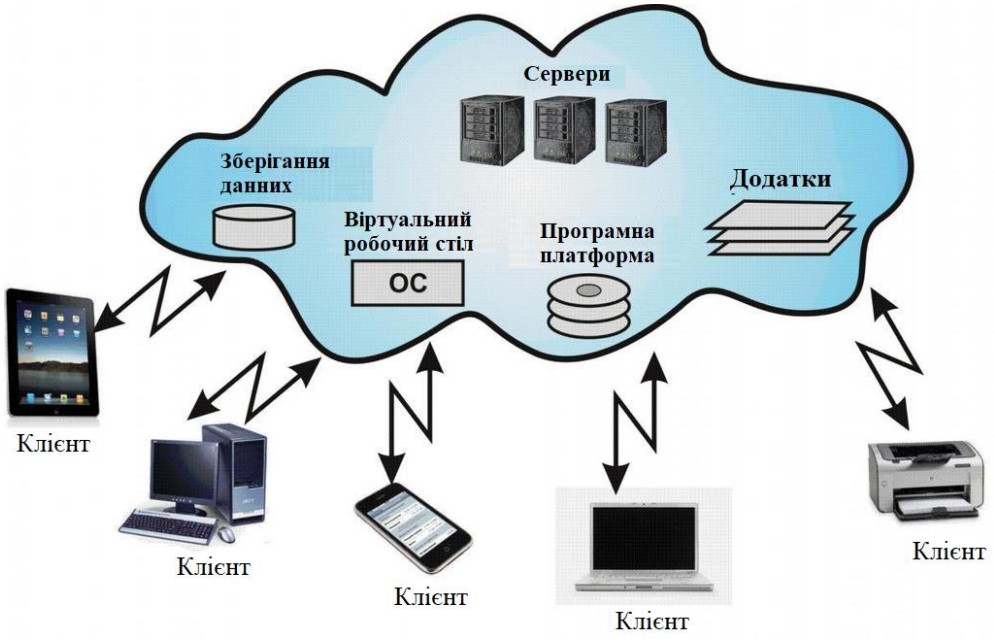

Рисунок 1.1. Різноманітні «хмарні» сервіси

При використанні технології «хмарних» обчислень комп'ютерні ресурси і служби стають доступні користувачеві в якості звичайного веб-сервісу. Користувач може мати доступ до власних даних, але може не мати можливості керувати або підтримувати інфраструктуру, операційну систему та програмне забезпечення, з якими він працює. Теоретично "хмарою" може бути будь-яка розподілена інфраструктура, до якої вона забезпечує доступ через глобальний Інтернет (або локальні мережі, якщо це закрита інфраструктура так званої "приватної хмари"). Згідно з документом IEEE (Institute of Electrical and Electronics Engineers), опублікованим у 2008 році,

«Хмарні обчислення — це парадигма, в рамках якої інформація постійно зберігається на серверах у мережі Інтернет і тимчасово кешується на клієнтській стороні, наприклад на персональних комп'ютерах, ігрових приставках, ноутбуках, смартфонах тощо» [2].

Основою «хмарних» обчислень є декілька підходів до надання послуг [14]:

1. Доступність сервісів через Інтернет. Кінцевий користувач взаємодіє з хмарною інфраструктурою як зі звичайним веб-сервером, оскільки зазвичай хмара має єдину точку входу.
2. Віртуалізація. Віртуалізація полегшує масштабування. Завдяки віртуалізації всі користувачі можуть отримати необхідні ресурси та мають можливості динамічно збільшувати чи зменшувати їх в майбутньому. Усі службові процеси, при цьому, непомітні для кінцевого користувача. Фізичні ресурси для роботи необхідної системи можуть виділяться як в межах одного серверу так і на різних серверах, що знаходяться в різних дата центрах.
3. Простота і стандартність. Для опису та взаємодії з хмарними ресурсами непотрібно створювати складні конфігураційні файли. Все, що існує всередині хмарної інфраструктури доступно через стандартні API-запити і відкриті протоколи (наприклад, протокол REST, який дозволяє всі операції над даними виконувати за допомогою HTTP-запитів).

1.1.1 Принцип роботи «хмар»

Майже всі хмарні технології застовуються на принципах віртуалізації. Віртуалізація – це вид подання обчислювальних ресурсів або їх логічного об'єднання, абстрагований від апаратної реалізації, що забезпечує логічну ізоляцію один від одного обчислювальних процесів, які виконуються на одному фізичному ресурсі. В рамках віртуалізової архітектури розгядаються не конкретні апаратні обчислювальні блоки, а віртуальні сервери, кожен з

яких емулює роботу фізичного комп’ютера із власною операційною системою.

Віртуальні сервери використовують ресурси, виділені фізичною машиною, і їх програмне забезпечення повністю ізольоване. Фізичний хост використовує спеціальний рівень програмного забезпечення, який називається гіпервізором, для встановлення та управління такими серверами та надання ресурсів, що знаходяться під його контролем. Термін "гіпервізор" зазвичай використовується для позначення фактичного фізичного хоста, на якому встановлені гіпервізори (і віртуальних серверів, що знаходяться під їх контролем). Кожен віртуальний сервер має операційну систему, встановлену гіпервізором, яка є основою для встановлення інших програм. Модель хмарної інфраструктури IaaS працює на рівні віртуалізації (гіпервізор).

Для масштабуваня хмари кількість віртуальних серверів в її складі збільшується. Коли ви додаєте новий віртуальний сервер до існуючої конфігурації, хост відповідає за розподіл пам’яті, ядра процесора та підключення до мережі до сервера, а гіпервізор планує процеси між віртуальним та фізичним ядрами, оскільки одне і те ж фізичне ядро може використовуватися декількома віртуальними серверами. Насправді, завдяки спільному використання фізичних ресурсів, продуктивність віртуального сервера, як правило, нижча, ніж у ідентичного фізичного сервера.

Якщо взяти групу віртуальних серверів, що працюють під гіпервізором, і помножити їх у фізичній мережі із загальним доступом до пристрою зберігання та організувати управління цією інфраструктурою, забезпечити кешування та фільтрацію, ми отримаємо віртуальну інфраструктуру, яка є хмарою.

Оскільки роботу віртуальних серверів можна динамічно розподіляти між вузлами залежно від їх індивідуальної завантаженості, хмарна інфраструктура має досить високу ступінь гнічкості. Залежно від потреб, можна підняти чи опустити планку використовуваних хмарою ресурсів, за якої буде потрібно додавання нових віртуальних серверів. Так наприклад,

можна зменшувати запущених віртуальних серверів, коли їх обчислювальні потужності не використовуються сповна (стають надлишковими).

1.2 Основні характеристики хмарного середовища

Постачальники хмарних рішень дозволяють орендувати обчислювальну потужність та дисковий простір через Інтернет. Перевагами цього підходу є доступність (користувач платить лише за необхідні ресурси) та можливість гнучкого масштабування. Клієнти позбавляються потреби створювати та підтримувати власну обчислювальну інфраструктуру.

На думку експертів, використання «хмарних» технологій у багатьох випадках може зменшити витрати у два-три рази порівняно із підтримкою власної вдосконаленої IT-структури [2].

Cloud відкриває новий підхід до обчислень, в якому ні апаратне, ні програмне забезпечення не належать компанії, натомість постачальник послуг надає клієнту готове рішення.

Однією з перших доступних "хмарних" Інтернет-служб була електронна пошта на основі Інтернету. У цьому випадку всі дані зберігаються на віддалених серверах, і користувач отримує доступ до своїх електронних листів з будь-якого комп’ютера або досить потужного мобільного пристрою через браузер.

Національним інститутом стандартів і технологій США встановлені такі обов'язкові характеристики «хмарних» обчислень [1]:

- Самообслуговування на вимогу (англ. self service on demand) - споживачі самостійно визначають і змінюють обсяг збережених даних, обчислювальні вимоги, такі як час сервера, швидкість доступу та обробки даних, не звертаючись до представника постачальника послуг.;
- Універсальний доступ через мережу - послуги доступні споживачам через мережу передачі даних (Інтернет) незалежно від ткінцевого пристрою;

- Об'єднання ресурсів в пул (англ. resource pooling) - постачальники послуг об'єднують ресурси для обслуговування багатьох споживачів як єдиний пул, динамічно перерозподіляючи потужність серед споживачів на основі постійно мінливих потреб у потужності. У цьому випадку споживач контролює лише основні параметри послуги (наприклад, обсяг даних, швидкість доступу), але фактичний розподіл ресурсів, що надаються провайдером споживачеві (в деяких випадках споживач може контролювати деякі фізичні параметри перерозподілу, наприклад: бажаний центр обробки даних спираючись на географічну близькість);
- Гнучкість - послуги можуть бути надані, а їх обсяг розширено або зменшено в будь-який момент часу, без додаткових витрат на взаємодію з постачальником, як правило, в автоматичному режимі;
- Облік споиживання – хмарний провайдер автоматично розраховує спожиті ресурси на певному рівні абстракції (наприклад, обсяг збережених даних, пропускна спроможність, кількість користувачів, кількість транзакцій) та оцінює обсяг наданої послуги, виходячи з них.

З точки зору постачальника, через мінливий характер агрегування ресурсів та споживання споживачів, хмарні обчислення використовують менше апаратних ресурсів, ніж розподіл апаратної потужності окремим споживачам та автоматизацію процесу, щоб суттєво змінити розподіл ресурсів при одночасному зменшенні абонентських витрат.

З точки зору споживача, ці функції забезпечують високий рівень доступності послуг (HA, англ. high availability) і низький ризик відмов, забезпечуючи гнучкість для швидкого розширення комп'ютерних систем без необхідності їх створення, обслуговування та оновлення самостійно [17].

Зручність і універсальність доступу забезпечується підтримкою різного класу термінальних пристроїв (персональних комп'ютерів, мобільних телефонів та інших пристроїв з доступом в мережу Інтернет).

1.3 Моделі обслуговування та існуючі рішення

Виділяють наступні моделі надання послуг за допомогою хмари [23]:

– Software-as-a-Service (SaaS, програмне забезпечення як послуга);
– Platform-as-a-Service (PaaS, платформа як послуга);
– Infrastructure-as-a-Service (IaaS, інфраструктура як послуга).

Короткий опис частин хмарної інфраструктури, підконтрольних користувачеві в рамках кожної моделі представлення хмарних послуг, наведено на рис.1.2.

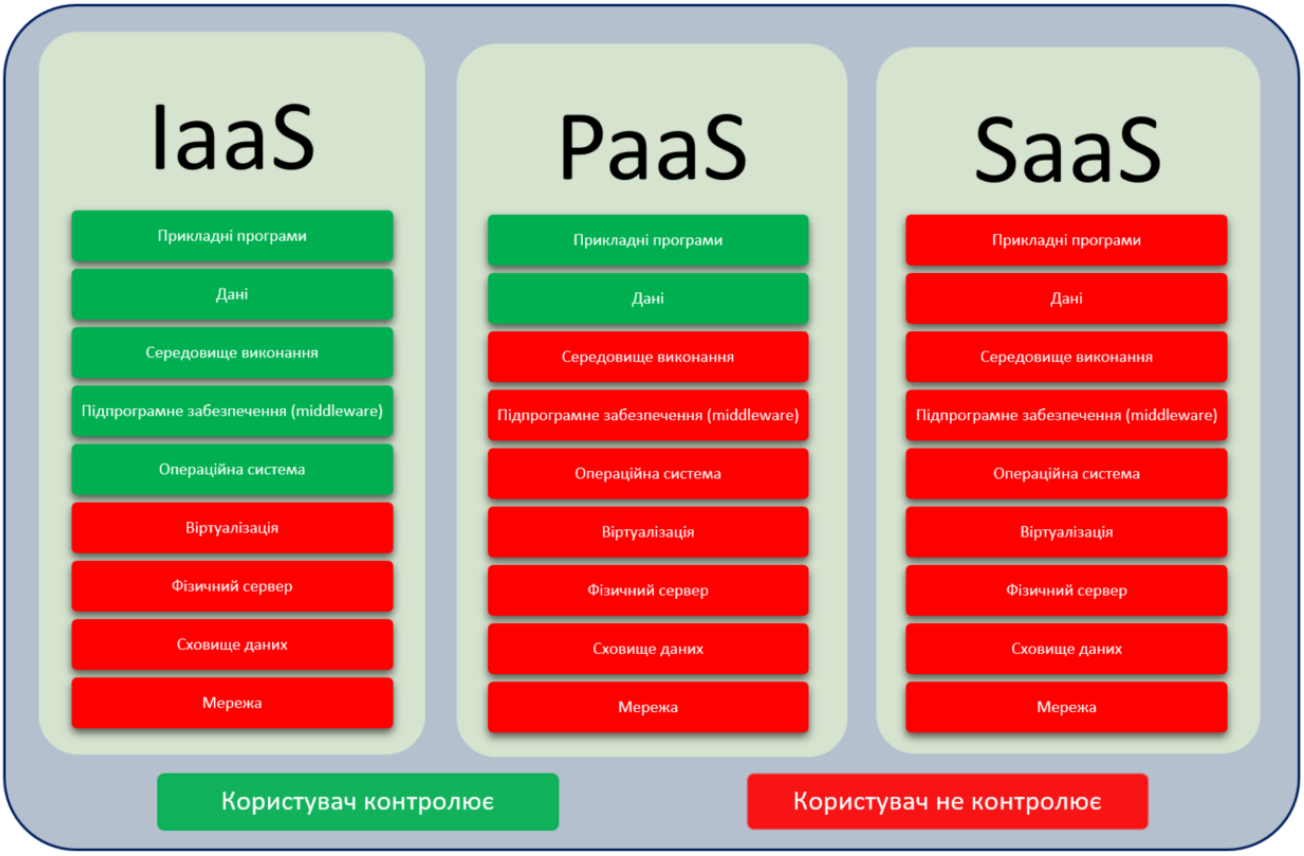

Рис. 1.2. Можливості користувача згідно з моделями хмарних послуг

Програмне забезпечення як послуга (SaaS) – модель, в якій споживачеві надається можливість використання прикладного програмного забезпечення провайдера, що працює в хмарній інфраструктурі і доступне через веб-інтерфейс або API вендора [1]. Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, в тому числі мережею, серверами, операційними системами, зберіганням, або навіть індивідуальними можливостями додатків (за винятком обмеженого набору налаштувань,

доступних користовачу) здійснюється хмарним провайдером. Прикладами SaaS, можуть бути сервіси Dropbox та Google Docs.

У рамках моделі SaaS замовники платять за користування додатками, що працюють на серверах SaaS-провайдера. Таким чином замовнику не потрібно інвестувати значні кошти в придбання ПЗ та апаратної платформи для його розгортання, а потім підтримувати його працездатність. Схема періодичної оплати передбачає, що в разі відсутності потреби в даному програмному забезпеченні замовник може призупинити його використання і заморозити виплати його провайдеру.

Для SaaS-моделі можна виділити наступні ключові ознаки:

- програмне забезпечення підтримується та оновлюється централізовано з боку постачальника програм.
- програмне забезпечення надається за абонентською моделлю (звичайна оренда), і ця сума включає вартість негайної технічної підтримки.
- програмне забезпечення встановлюється в центрі обробки даних у вигляді єдиного програмного ядра, з яким працюють усі клієнти постачальника хмарних послуг.
- доступ до програмного забезпечення зазвичай надається віддалено через мережевий канал через веб-інтерфейс або API.

Платформа як послуга (PaaS) – модель, в якій споживачеві надається можливість використання хмарної інфраструктури для розміщення базового програмного забезпечення для подальшого розміщення на ньому нових або існуючих додатків (власних, розроблених на замовлення або придбаних по ліцензії додатків). До складу таких платформ входять інструменти для розробки, тестування та розміщення додатків на їх основі [1]. Контроль та управління основною фізичною і віртуальною інфраструктурою хмари, в тому числі мережею, серверами, операційними системами, системами зберігання здійснюється хмарним провайдером, за винятком розроблених або встановлених додатків, а також, параметрів конфігурації середовища

(платформи). Наприклад, Heroku дозволяє користувачам розгортати та виконувати додатки на базі їх хмарних сервісів, хоча всі ці сервіси розміщені і використовують платоформу EC2 від Amazon.

Інфраструктура як послуга (IaaS) позиціонується як можливість використання хмарної інфраструктури провайдера для самостійного управління ресурсами обробки, зберігання, комунікаійними мережами та іншими фундаментальними обчислювальними ресурсами [1]. Таким чином споживач може встановлювати і запускати довільне програмне забезпечення, тобто розміщати власні платформи та додатки в хмарному середовищі постачальника. Споживач контролює операційні системи, віртуальні системи зберігання даних і встановлені програми, а також має обмежений контроль доступу до сервісів (наприклад, за допомогою конфігурації міжмережевого екрану). Підтримка і управління базовою фізичною та віртуальною інфраструктурою хмари здійснюється постачальником.

IaaS складається з трьох основних компонентів: апаратних засобів (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання); операційних систем та системного ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами); зв'язуючого ПЗ (наприклад, для управління системами).

За допомогою технології віртуалізації ви можете додати своє обладнання та розділити обчислювальну потужність на частини, які відповідають поточним вимогам сервісу, щоб збільшити використання наявної потужності. Як результат, придбання, управління та амортизація апаратного забезпечення перетворюється в придбання лише процесорного часу, простору на диску та пропускної здатність мережі, необхідних для запуску програми. Крім того, віртуалізація дозволяє повністю управляти різними типами обладнання за допомогою єдиної платформи.

Infrastructure as a Service (IaaS) позбавляє підприємства від необхідності підтримки складних інфраструктур центрів обробки даних, клієнтських і

мережевих інфраструктур, а також дозволяє зменшити пов'язані з цим капітальні витрати та поточні витрати.

Найбільшими гравцями на ринку подібних послуг є Amazon, Microsoft, VMWare, Rackspace та Red Hat. Деякі з них пропонують не просто інфраструктуру, а ще й пов’язані з її використанням сервіси.

Обчислювальна «хмара» може бути розгорнута як: приватна, публічна, громадська або гібридна.

Приватна хмара - це "хмарна" інфраструктура, призначена для використання однією організацією з декількома користувачами (наприклад, класами). Приватна "хмара" може належати, експлуатуватися та експлуатуватися самою організацією та третіми сторонами (або їх комбінаціями) [1].

Публічна хмара - це «хмарна» інфраструктура, призначена для безкоштовного використання широкою громадськістю. Публічна "хмара" може належати, експлуатуватися та експлуатуватися комерційними, науковими (освітніми та академічними) чи державними установами (або їх поєднанням) [1].

Хмара спільноти (англ. community cloud) - це «хмарна» інфраструктура, призначена для використання споживчим співтовариством організації, яка має спільні цілі (наприклад, місію, вимоги до безпеки, політику та відповідність різним вимогам). Публічна хмара може належати та експлуатуватися однією або кількома громадськими організаціями або третіми сторонами (або їх комбінацією) [1].

Гібридна хмара - це "хмарна" інфраструктура, що складається з двох або більше різних "хмарних" інфраструктур (приватна, публічна чи загальнодоступна), які залишаються окремими об'єктами, але використовують стандартизовані технології або технології перенесення даних та додатків (наприклад, через загальнодоступні хмарні ресурси) [1].

1.4 Переваги та недоліки технології

Технології «хмарних» обчислень мають свої переваги та недоліки. Спочатку розглянемо переваги «хмарних» обчислень.

По-перше до них можна віднести доступність. Користувачі можуть легко отримати доступ до своїх даних з будь-якої точки на планеті та економити час і ресурси. Це дозволяє користувачам (або компаніям) заощаджувати на закупівлі високопродуктивних, дорогих комп'ютерів.

Наступна перевага – це низька вартість. Основні фактори, що знизили вартість використання хмар, наступні:

- Зменшення витрат на обслуговування віртуальної інфраструктури, спричинене розвитком технологій віртуалізації (потрібен менший штат співробітників для обслуговування всієї IT-інфраструктури підприємства);
- За допомогою хмари в оренду користувачі можуть зменшити витрати на придбання дорогого обладнання та зосередитись на інвестиціях, спрямованих на побудову бізнес-процесу підприємства, полегшуючи розвиток та розширення бізнесу.
- Оплата фактичного використання ресурсів. Користувач хмари платить лише за фактичне використання обчислювальних потужностей, що дозволяє йому ефективно розподіляти свої грошові ресурси.

Крім того, однією з переваг є необмежена гнучкість обчислювальних ресурсів (пам’ять, процесори, дисководи). Завдяки використанню систем віртуалізації, процес масштабування та управління "хмарою" стає дуже простим завданням, оскільки він може забезпечити необхідні ресурси за необхідності.

Ще одна з переваг – надійність та безпека. Надійність «хмарного» середовища, особливо такого, що функціонує у спеціально обладнаних ЦОД, дуже висока. Такі центри обробки даних мають резервні джерела живлення, регулярне резервування та шифрування даних, високу пропускну здатність

мережевого каналу, високу стійкість до DDoS-атак і таким чином забезпечують безперервність послуг для користувача або бізнесу.

Найважливішою перевагою є велика обчислювальна потужність. Користувачі "хмарної" системи можуть використовувати всі обчислювальні ресурси, платячи лише за фактичний час використання. Компанії можуть скористатися цією можливістю, щоб проаналізувати великі обсяги даних за короткий проміжок часу.

Незважаючи на всі перераховані вище переваги, "хмари" мають і недоліки.

По-перше, це вимагає постійного підключення до мережі. Щоб отримати доступ до «хмарної» служби, потрібно стабільне та постійне з’єднання з Інтернетом. Однак сьогодні це незначний недолік, особливо в світлі поширення LTE та 5G технологій.

Наступним недоліком є конфіденційність даних. Більшість експертів сходяться в тому, що дані, збережені на публічних «хмарах» є вразливими до витоків та оприлюднення, оскільки на даний момент немає технології яка б гарантувала 100% конфіденційність для них і захищала від корпоративного шпіонажу та кібер-атак.

Що стосується надійності зберігання інформації в "хмарі", то з упевненістю можна сказати що у випадку апаратних збоїв або проблем на стороні хмарного провайдера – її втрата буде незворотною. Хоча різноманітні системи резервного копіювання (як апаратні так і програмні) частково вирішують цю проблему.

Іншим вразливим місцес є безпека. Сама «хмара» є досить потужною системою, але якщо зловмисник увірветься, він може отримати доступ до величезного сховища даних. Іншим недоліком є те, що хтось можете скористатися вразливостей в ядрах стандартних операційних систем (таких як Linux та Windows) що використовуються в якості гіпервізор.

Якщо розглянути економічні показники використання «хмарних» технологій, то можна виділити ще одну проблему. Вартість обладнання для

створення власної хмари дуже висока. У цьому випадку підприємству доводиться видти значні матеріальні ресурси, що є невигідним для нових установ та невелиих компаній. В результаті, більшість із них використовують модель PaaS.

Отож, які проблеми створюють «хмарні» технології? Основним недоліком є необхідність постійно бути он-лайн, оскільки відсутність мережевого з'єднання унеможливлює нормальне функціонування робочих процесів або бізнесу вцілому. Також не варто забувати про питання безпеки і ймовірність безповоротно втратити всі дані у випадку прострочення щомісячного платежу.

Які ж складнощі допомагаю вирішувати «хмарні» технології? По-перше, користувачеві не потрібно піклуватися про операційні витрати і процеси пов'язані з підтримкою інфраструктури, такі як виділення додаткових потужностей, реплікація даних та оновлення обладнання або базового ПЗ. з хмарними технологіями це питання вирішується постачальником «хмарних» послуг.

По-друге, користувачеві не потрібно витрачатися на весь потрібний йому продукт. Він платить тільки за необхідні послуги, надані можливості або окремі функції того чи іншого програмного комплексу .

По-третє, «хмарні» технології допомагають бізнесу в сезонному сенсі. Оскільки компанія більше не зобов'язана нести величезні витрати на обслуговування інфраструктури, яка потрібна їй лише в пікові періоди підвищеного попиту.

1.5. Використання хмарних технологій в телекомунікаційних системах

Одним з перспективних напрямків використання хмарних обчислень є застосування їх для побудови сучасних мобільних телекомунікаційних мереж. На сьогоднішній день до хмарної обчислювальної системи отримують доступ все більше користувачів, які мають смартфони та

планшети. Саме хмара в мобільному середовищі породила новий термін «мобільні хмарні обчислення» МСС (mobile cloud computing) [15], які використовують потужну платформу хмарних обчислень для ефективного рішення завдань по зберіганню даних, керуванню та синхронізації, тощо.

З іншого боку, такі потужні технології можуть бути корисними для мереж радіо доступу RAN (radio access networks), що призвело до появи нової концепції хмарних мереж радіодоступу C-RAN (cloud radio access networks) [16].Хмарна мережа радіодоступу передбачає перехід від розподіленої до централізованої інфраструктури для обробки інформації та керування.

У порівнянні з базовими станціями (BS) традиційних стільникових мереж, BS в C-RAN спрощені, тому що обробка сигналів і прийняття рішень в більшості випадків відбувається в хмарі безпроводової мережі (рис. 1.3.). BS підключені до хмари безпроводової мережі за допомогою опорних транспортних мереж. Для забезпечення передачі даних в умовах значних затримок використовується спліт-TCP (split-TCP) [31].

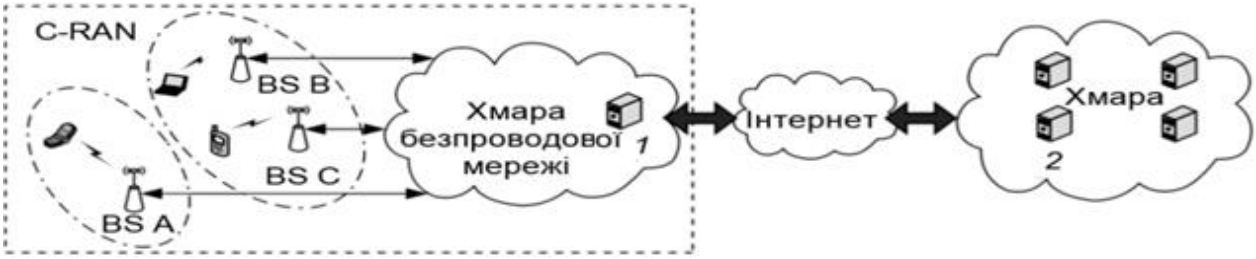

Рис. 1.3. Хмарна мережа радіодоступу в середовищі МСС:
1 – спліт-TCP проксі (split-TCP proxy); 2 – внутрішній сервер.

Клієнт встановлює TCP-з'єднання з найближчим спліт-TCP проксі сервером, а TCP проксі, в свою чергу, підтримує постійне з'єднання. TCP-потоки передаються з мобільних пристроїв до внутрішніх серверів в хмарі. Хмара безпровідної мережі динамічно управляє безпроводовою мережею за допомогою TCP-проксі, і розбиває з'єднання з кінця в кінець між користувачем і внутрішнім сервером на два з'єднання, а також підтримує постійний зв'язок між собою і внутрішнім сервером. Управління включає в себе розподіл швидкості передачі даних і конфігурацію топології, за допомогою якої можлива кооперація декількох BS [15]. Наприклад, на рис.

1.3. базові станції B і C утворюють кластер для спільного обслуговування мобільних користувачів, а BS A обслуговує інший кластер самостійно. У середині кластера сигнали обробляються спільно, так що інтерференція радіосигналів відсутня.

Таким чином, технології хмарних обчислень можна застосувавати для побудови сучасних високопродуктивних безпровідних телекомунікаційних систем. Використання подібних технологічних рішень дозволяє зменшити витрати на розгортання та підтримку таких систем для операторів зв’язку. В той же час, питання автомасштабування ресурсів та балансування навантаження хмарних сервісів залишається актуальним та набуває ще більшого значення в умовах надання телекомунікаційних послуг, що потребують забезпечення безперебійного доступу до динамічно масштабованої високонавантаженої системи.

1.6. Поточний стан інтеграції хмарних обчислень телеком-операторами

Хмарні обчислення в довгостроковій перспективі для операторів зв'язку є найбільш сприятливим способом розвитку та введення повного спектру послуг та сервісів для користувача. З глобальним доступом до мереж LTE та впровадженням 5G, оператори можуть побудувати високоінтегрованну систему на основі хмарних обчислень, а наявна база користувачів та існуюча інфраструктура стають очевидними перевагами для подальшого безперервного розвитку подібної системи. При безперервній інтеграції телекомунікацій та Інтернету, хмарні обчислення будуть мати істотний вплив на традиційну індустрію телекомунікацій. Традиційні IDC (Центри обробки даних) не в змозі задовольнити вимоги розвитку та темпи росту індустрії потокових сервісів. Щоб осідлати цю нову хвилю обчислень, як довгострокове рішення, телекомунікаційні оператори починають будувати мультимережеві інтеграції, тож тенденції впровадження хмарних обчислень стають очевидними.

Для прикладу розглянемо стадії розвитку хмарних інфокомунікаційних систем в Китаї, як одній з країн-лідерів ринку телекомунікаційних послуг. Індустрія хмарних обчислень Китаю розділена на три етапи: період підготовки ринку, період зльоту і етап зрілості. В даний час індустрія хмарних обчислень перебуває в стадії стрімкого розвитку [28].

Підготовчий етап (2007-2010 роки): основна увага приділяється технічним резервам і етапам концептуалізації, рішення і бізнес-моделі все ще знаходяться в стадії апробації. Користувачі все ще в меншій мірі інформовані про "хмарні" обчисленнях, а успішних прикладів використання технології небагато. Початковий розвиток заснований на створенні державної публічної хмари [25].

Етап зльоту (2010-2017): стрімкий розвиток промисловості, екологічне будівництво інфраструктури і побудова бізнес-моделей стали ключовими словами цього періоду, починається "золота епоха" індустрії хмарних обчислень. За цей період кількість успішних кейсів поступово збільшувалася, а розуміння і визнання з боку користувачів постійно зростало. Все більше і більше провайдерів послуг хманих обчислень, багато прикладних рішень, перші приклади використання хмари для створення власного бізнесу. Створюються публічні хмари, приватні хмари та гібридні рішення.

Зріла стадія (2015-триває): індустрія хмарних обчислень та екосистема в основному стабільні; рішення від хмарних провайдерів більш зрілі та стабільні, забезпечуючи багатство продуктів XaaS (Anything-as-a-Service) [28]. Користувацькі програми на основі хмарних обчислень є високопродуктивними, і стають невід'ємною частиною IT-систем, хмарні обчислення стали закінченим інфраструктурним рішенням.

У зарубіжних країнах, крім надання послуг мережевого доступу (ISP) для населення, Orange, O2 та інші великі телекомунікаційні компанії також виступають в якості провайдерів послуг "хмарних обчислень". Ці компанії надають послуги з оренди обладнання IDC і SaaS-продукти для різних галузей промисловості. За допомогою своїх інноваційних продуктів вони покращують

якість послуг і забезпечують потужний імпульс швидкому розвитку і росту публічних хмар [25]. Таким чином, в майбутньому, домашні телекомунікаційні компанії стануть одним з основних бенефіціарів галузі хмарних обчислень, отримуючи великі прибутки від різних видів платних послуг, досягаючи зростання прибутку, створення власного бренду для системи хмарних послуг через аналіз потреб користувачів різноманітних галузей вітчизняної промисловості та дослідження в якості постачальника послуг хмарних обчислень.

В області телекомунікаційних додатків хмарні обчислення можуть бути "ще більш потужними". Відновлена традиційна телекомунікаційна індустрія; оператори, які отримали ліцензії на LTE та 5G, мають можливість розвивати інтегровані інформаційні послуги [44]. Закордонні великі оператори зв'язку вже приступили до будівництва хмарних сервісів і добилися перших успіхів.

Таблиця 1.1 Стратегія розвитку хмарних обчислень іноземними операторами зв'язку

Провайдер	Стратегія	План впровадження
AT&T	– Підтримує хмарні обчислення як керований сервіс. – Взаємодія з ISV для надання корпоративних додатків, що використовують моделі хмарних обчислень. – Працює з постачальниками додатків для розробки їх хмарних додатків для мобільних пристроїв. – Співпраця по моделі SaaS. – План по створенню сервісів PaaS на базі IBM WebSphere.	1. 2008 - запущений перший сервіс хмарних обчислень Synaptic Hosting. 2. Травень 2009 року - запущено Synaptic Storage-as-a-Service. 3. Листопад 2009 року - запущено Synaptic Compute-as-a-Service. Оплата згідно з користуванням, без фіксованих і мінімальних платежів за використання.

	– Позиціонування в якості корпоративного постачальника послуг хмарних обчислень [25].	4. 2011 - інвестовано 1 мільярд доларів США в забезпечення хмарних рішень для корпоратівних користувачів на базі глобальних мереж [25].
Verizon	– Бачать хмарні обчислення частиною еволюції IT-відділу в напрямку "Все як послуга". – Згодні з "хмарними" обчисленнями як природною еволюцією їх бізнесу керованого хостингу. – Подкреслює, що хмарні обчислення - це "движок доставки" сервісу. – Позиціонує себе як великого глобального постачальника керованих послуг безпеки	1. 2009 - запущений перший сервіс хмарних обчислень. 2. Травень 2009 року - запущено Synaptic Storage as a Service. 3. Листопад 2011 року - запущено Synaptic Compute в якості сервісу. Оплата згідно з користуванням, без вказівки причини і мінімальний платіж за використання.
BT	– Хмарні обчислення як ключова рушійна сила корпоративної продуктової стратегії. – Хмарні рішення поділені на три області: IaaS, SaaS, CaaS (Communication-as-a-Service) – Розглядають CaaS як хмарний сервіс, побудований на комунікаціях в реальному	1. Угода 2009 року з Microsoft про запуск онлайн-бізнесу Microsoft для бізнес-клієнтів, надаючи послуги хмарних обчислень і спільної комунікації для клієнтів. 2. Надання керованих IP-АТС і послуг об'єднаних комунікацій в Європі і

	часі (таких як голос, відео, веб-конференції, обмін миттєвими повідомленнями і т.д.) [26]. – Мережа як ключовий чинник в наданні послуг хмарних обчислень корпоративного класу. – Надання послуг хмарних обчислень корпоративного класу для наскрізної продуктивності і доступності.	США з використанням розрахованої на багато користувачів платформи Cisco HUCS.
Orange	– Позіціонує себе як IT-оператор (як розширення мережевого оператора). – Пакування мережевих і хмарних обчислювальних послуг у вигляді наскрізних хостингових послуг, що забезпечують єдиний портал надання обчислювальних ресурсів, єдину довідкову службу і єдину наскрізну угоду про рівень обслуговування (SLA) [25]. – Розділ хмара на чотири компоненти: IaaS, SaaS, "Collaboration-as-a-Service", "Security-as-a-Service". – План для інтеграції голосу, відео	1. Запуск SaaS-рішення в 2008 році для ринку малого та середнього бізнесу заснованого на наступних принципах 2. Щомісячна плата $3k на користувача в місяць. Рішення надає додатки для підвищення продуктивності, поштових програм та бізнес-додатків (SAP, Sage і т.д.). 3. У 2009 р були запущені рішення IaaS для гнучких обчислень, модернізовані в 2010 р, плюс можливості порталу самообслуговування і управління клієнтами, які

		оплачуються за використання.
T-Systems	– Наскрізна доступність в якості основної компетенції хмарних обчислень, побудова себе в якості довіреного постачальника послуг хмарних обчислень. стандартизувати різні компоненти хмарного обчислювального центру обробки даних (сервер, сховище, мережеве обладнання та програмне забезпечення), автоматизувати роботу і управління хмарним обчислювальним центром обробки даних, оптимізувати структуру витрат хмарного обчислювального центру обробки даних, а також поступово збагачувати функції платформи хмарних обчислень [26]. – Співпраця з лідерами галузі для створення галузевих хмарних рішень і активну участь в галузевих стандартах хмарних обчислень. – Скорочення IT-витрат і динамічне управління IT-	1. Почали свою діяльність в 2004 році IaaS хмарна обчислювальна платформа AppCom, що надає віртуальні сервери, сховища, загальні брандмауери і віртуальні локальні мережі і т.д. 2. Хостинг баз даних, проміжного програмного забезпечення, SAP та інших стандартних додатків для корпоративних клієнтів на власній платформі хмарних обчислень AppCom. 3. Запуск сервісу керованих верстатів на базі хмарних обчислювальних центрів обробки даних для місіонерських працівників, фахівців в галузі знань і мобільних працівників для надання керованих робочих станцій. 4. Інтенсивне використання комунікаційних послуг, заснованих на хмарні

	ресурсами за допомогою хмарних обчислень. – Побудова енергоефективних зелених центрів обробки даних за допомогою технології хмарних обчислень. – Створення незалежного від постачальника рішення для хмарних обчислень.	обчислення, включаючи такі послуги, як голосові і відеоконференції, обмін миттєвими повідомленнями. 5. Консалтингові послуги в області хмарних обчислень, включаючи оцінку готовності до хмарних обчислень, міграцію традиційних IT-моделей на моделі хмарних обчислень і системну інтеграцію, оптимізацію моделей хмарних обчислень. 6. Більше дюжини центрів хмарних обчислень в Європі, Північній Америці і на півдні.

Таблиця 1.2 Стратегія розвитку хмарних обчислень китайськими операторами зв'язку

Провайдер	Стратегія	План впровадження
China Mobile	– Створення платформи хмарних обчислень на основі віртуалізації, централізованого управління, автоматичного планування і розподілених обчислень, забезпечення	1. У 2009 році запустилась програма "Велике Хмара", в 2010 році запустили систему "Велике Хмара" 1.0.

	розумного розподілу внутрішніх ресурсів і надання хмарних IT-послуг [29]. – Масштабна обробка даних і паралельні розрахунки на основі хмарних обчислень. – Створення хмарної платформи управління для підтримки внутрішніх та зовнішніх потреб бізнесу	2. Великомасштабне просування десктопної хмари 3. У 2010 році був запущений 3-річний офіційний план розширення дата-центр. 4. У 2011 році спільно з компанією Xinhua запустили пошук Pangu на основі "великий хмарної" платформи [27]. 5. В кінці 2011 року в Харбіні був побудований Китайський мобільний національний центр хмарних обчислень.
China Unicom	Використовує базові, безпечні і надійні експлуатаційні можливості мережі зв'язку загального користування для розвитку інформаційних послуг, виробництва інформації, інформаційних послуг, послуг "хмарних обчислень" на основі розвитку технологій широкосмугового зв'язку та "хмарних обчислень"[29].	1. Проведено пілотне випробування VDC в офісах в Пекіні, Шанхаї, Гуанчжоу, Хубей і інших філіях. 2. Створена розподілена інтелектуальна відкрита систему DIOS, що охоплює віртуальний центр обробки даних VDC, архітектуру BI System Cloud , Хмарні обчислення OSS, Колл-центр по оновленню

		хмарних терміналів, дослідження мереж і хмарних обчислень майбутнього [27]. 6. Інвестиції $ 500 млн. в Циндао в 2011 році на будівництво центру обробки даних в області хмарних обчислень 7. В кінці 2011 року була запущена стратегія "Вояж".
China Telecom	– Керуючись потребами бізнесу, в поєднанні з перевагами телекомунікаційних ресурсів, відповідно до ринкової і технологічної зрілості для активного просування послуг хмарних обчислень – Для задоволення потреб великого числа малих і середніх підприємств в гнучких IT-ресурсах, внутрішніх IT-систем для поліпшення ефективності роботи і зниження витрат.	1. 2010 - запустили програму "Nebula", в Гуанчжоу, Шанхаї, Ченду і Наньчане. Чотири міста провели польові експерименти з хмарними обчисленнями, включаючи IDC будівництво, бізнес-платформу та можливості створення платформи та IT-додатків[28] 2. У 2010 році в Гуандуні був створений Науково-дослідний інститут хмарних обчислень. 3. Цзянсу Телеком стала IDC, що надає хмарне обслуговування дата-центрів.[28]

Аналіз переваг розвитку хмарних обчислень для операторів зв'язку:

a) Маючи такі ресурси, як доступ в Інтернет, IDC (Internet Data Center), мережі мобільного зв'язку і точки доступу WiFi, ці ресурси є найбільш базовим ресурсом для розвитку хмарних обчислень, на основі яких оператори зв'язку можуть створювати більш привабливий портфель послуг ICT (Information Communication Technology).

b) Скориставшись всіма перевагами своєї глобальної магістральної мережі для хмарних обчислень, можна надавати безпечні і надійні послуги доступу до мережі, які забезпечують наскрізний рівень сервісу (SLA).

c) Можливий досвід надання великомасштабних комунікаційних послуг десяткам мільйонів користувачів. Білінг також відбувається лише за фактичне використання. Цей досвід може бути безпосередньо використаний для надання послуг хмарних обчислень, щоб забезпечити надійність і продуктивність послуг хмарних обчислень [43].

d) Традиційний бізнес IDC (Internet Data Center) розвивається десятиліттями, накопичивши велику базу корпоративних клієнтів і завоювавши їх довіру. Ці клієнти віддають перевагу хмарним обчисленям, наданим операторами зв'язку.

Аналіз недоліків операторів зв'язку в розвитку хмарних обчислень:

a) Технологій і професіоналів в області хмарних обчислень менше в порівнянні з такими інтернет-компаніями, як Google і Amazon, та IT-компаніями, такими як IBM і Microsoft. Бракує нових досліджень і розробок.

b) Замість того, щоб бути основним бізнесом, "хмарні" обчислення виступають в якості суб-бізнесу традиційного телекомунікаційного бізнесу, що призводить до браку ресурсів [43].

c) Стиль роботи орієнтований на надійність і консервативність, менш конкурентоспроможний в швидкому розвитку сфери хмарних обчислень.

d) Складність існуючою комплексної IT-інфраструктури, що включає в себе різних постачальників, різні типи серверів, обладнання для зберігання

даних та мережеве обладнання, підвищила складність побудови платформи хмарних обчислень на цій інфраструктурі [44].

Для традиційних телекомунікаційних компаній зазвичай сфера діяльності обмежується бізнесом, пов'язаним з телефонним зв'язком, але з розвитком Інтернету і стріким прогресом в сфері портативної електроніки та смартфонів, користувачі більше не обмежені домом з мережевим кабелем, підключеним до Інтернету. Більшість людей володіють смартфоном або планшетним комп'ютером з підтримкою 3G/LTE для виходу в Інтернет. У зв'язку зі змінами в портативності і тенденціями бездротового і високошвидкісного використання, традиційний бізнес вже давно не в змозі задовольнити потреби користувачів. За допомогою хмарних обчислень бізнес може розширюватися свої сфери проникнення до потокових сервісів, онлайн-розваг, покупок через Інтернет, офісних додатків і так далі.

Висновки до розділу 1

На основі проведеного аналітичного огляду технології хмарних обчислень показано, що користувачі даної технології можуть значно зменшити інфраструктурні витрати (в короткостроковому і середньостроковому планах) і швидко адаптуватись до значних змін обчислювальних потреб.

Було наведено принципи роботи хмарного середовища, проаналізовано особливості найпоширеніших моделей хмарного розміщення, таких як SaaS (програмне забезпечення як послуга), PaaS (платформа як послуга) та IaaS (інфраструктура як послуга). Виділено переваги та недоліки хмарних технологій, які показують, що завдяки можливості швидкого масштабування ресурсів при непостійному характері споживання з боку користувачів можна досягти значної економії, використовуючи менші апаратні ресурси, ніж при резервуванні апаратних потужностей під прогнозовані всплески навантаження, а завдяки автоматизації виділення ресурсів значно знизити витрати на обслуговування інфраструктури вцілому. Було обгрунтовано

можливість практичного застосування хмарних технологій в телекомунікаційній сфері та актуальність проблеми балансування навантаження хмарних додатків. Наведено приклади стратегій розвитку та імплементації хмарних технологій провідними міжнародними телекомунікаційними провайдерами.

РОЗДІЛ 2. МАСШТАБУВАННЯ ТА БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНИХ СЕРЕДОВИЩАХ

Давайте розглянемо принципову модель базової архітектури сучасного хмарного веб-сервісу та його інфраструктурні компоненти. Найпримітивніша архітектура бекенд-частини веб-додатків може складатися з CDN, сховища даних, системи балансування навантаження (зазвичай, програмної), власне серверів де запущено веб-додаток та певної бази даних.

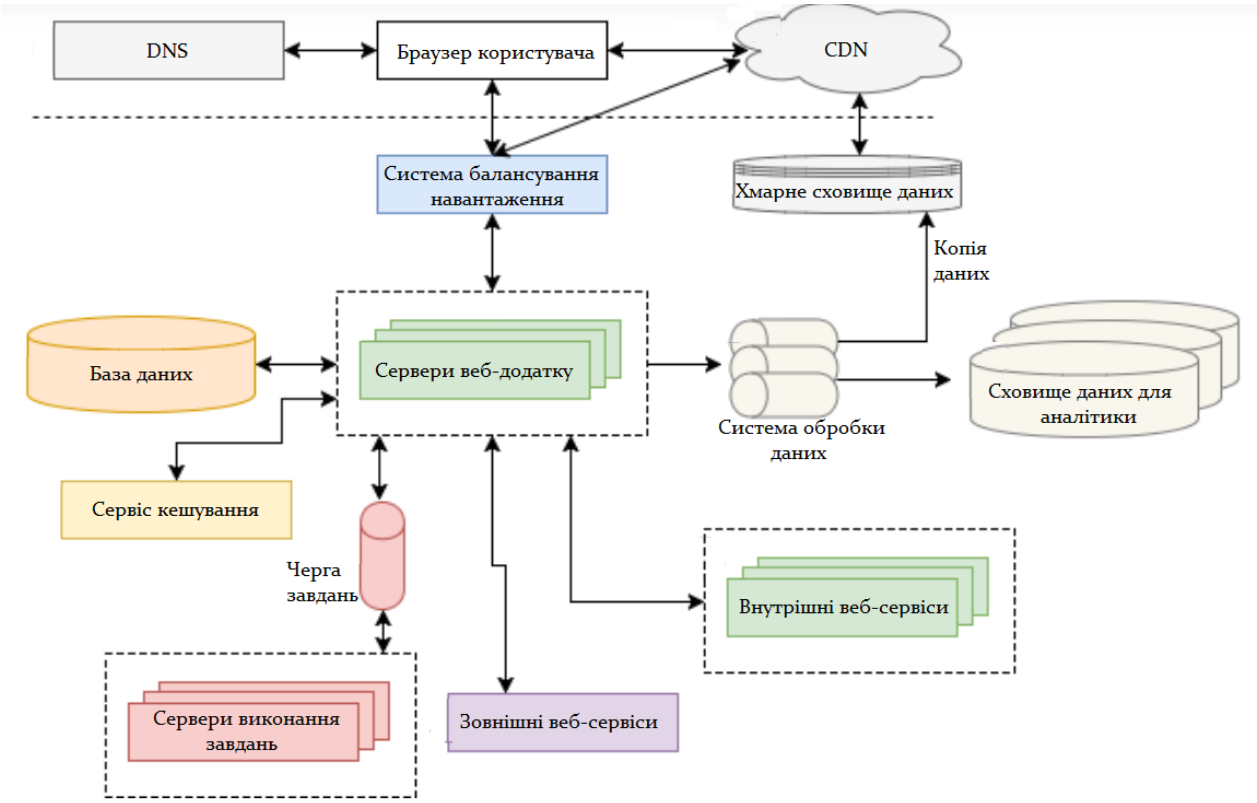

Рисунок 2.1. Базова архітектура хмарного веб-додатку.

В рамках даного дослідження буде роглядатися лише процес взаємодії клієнтської частина, балансувальника та, безпосередньо, обчислювальних серверів.

2.1. Масштабування хмарних сервісів

Масштабування серверного пулу стає дуже важливим коли вам потрібно швидко збільшити доступні ресурси, але наявні сервери не можуть забезпечити необхідну потужність, яку ще й потрібно безперешкодно

розвивати. Якщо вам постійно потрібно більше ресурсів, найкращим способом досягти цього є масштабування вашого сервера.

Масштабованість - у галузі електроніки та інформатики це означає здатність системи, мережі або процесу справлятися зі збільшеним навантаженням після додавання ресурсів (як правило, апаратних засобів). Системи, що підвищують продуктивність за рахунок додавання обладнання пропорційно доданим ресурсам, називаються масштабованими [4]. Для повністю масштабованої системи підвищення продуктивності є кратним збільшенню ресурсів (1:1). Масштабованість іноді визначається як здатність системи створювати додаткові ресурси без структурних змін центрального вузла системи.

Автоматичне масштабування - один з найпопулярніших способів застосування віртуальних серверів. Система може прогнозувати обсяг необхідних ресурсів та автоматично масштабувати сервери для запобігання перевантаження поточних обчислювальних блоків. Необхідна кількість серверів може бути як додана для обробки напливу запитів, так і відключена якщо навантаження на систему нижче мінімального порогового значення для запобігання «недовантаженості» та економії.

Масштабованість є бажаною характеристикою будь-якого програмного комплексу, електронної систем або мережі, оскільки зазвичай їх експлуатація пов’язана з великим навантаженням, що іноді має піковий характер. Хоча, як правило, її важко визначити як загальну властивість системи, тому зазвичай визначають конкретні вимоги до параметрів системи, які вважаються важливими для її масштабованості. В галузі телекомунікацій і програмного забезпечення, такими параметрами є здатність системи обробити більшу кількість запитів або бути легко розширеною за необхідності. В контексті телекомунікаційних систем масштабованість означає здатність системи до збільшення загальної пропускної спроможності відповідно до підвищеного навантаження, коли додано апаратні ресурси. Потрібно враховувати, що у випадку нехтування даним критерієм при проектуванні системи можна

зіштовхнутися з фатальними наслідками, коли система буде не здатна коректно відпрацьовувати підвищене навантаження. Додатковим негативним моментом, буде фінансова невигідність процесу масштабування такої системи, оскільки після додавання ресурсів отримаємо лише незначний приріст продуктивності, а після деякого «порогового» значення додавання ресурсів не матиме сенсу взагалі (продуктивність системи майже не змінюватиметься).

Дуже часто масштабованість хмарної системи поєднують з гнучкістю. Обидві ці характеристики описують здатність системи до адаптації, проте описують різні аспекти адаптивності. Гнучкість передбачає можливості гіпервізора створювати віртуальні машини або контейнери з ресурсами, достатними для задоволення підвищеного попиту на обчислювальні потужності, а масштабованість – здатність системи обробити цей попит або її потенціал збільшуватись відповідно до його обсягу. Хмарні обчислення вимагають від системи (або програми) і гнучкості, і масштабованості.

Виділяють два види масштабування – горизонтальне та вертикальне [4].

2.1.1 Горизонтальне масштабування

Горизонтальне масштабування – це процес розбиття системи на дрібні структурні компоненти з їх подальшим розподілом по окремим фізичним або віртуальним машинам, або збільшення кількості ідентичних серверів, які паралельно виконують одну й ту ж функцію. В данному випадку масштабованість означає можливість додавати до системи апаратні вузли з більшими обчислювальними потужностями, переміщення додатків на більш продуктивну віртуальну машину або апгейд запущеного віртуального сервера (як фізичний, так і програмний) для збільшення загальної продуктивності системи (як зображено на рисунку 2.1). Цей вид масштабування може вимагати адаптації «хмарних» додатків до збільшення ресурсів, щоб вони коректно опрацьовували подібний процес і могли повною мірою використовувати збільшені ресурси [5]. Здатність системи до горизонтального

мастабування значно збільшує її пропускну здатність, загальну надійність та відмовостійкість. Так, наприклад, у випадку апаратних збоїв або виходу із ладу частини серверів, система автоматично підлаштується та перерозподілить навантаження між нормально функціонуючими серверами.

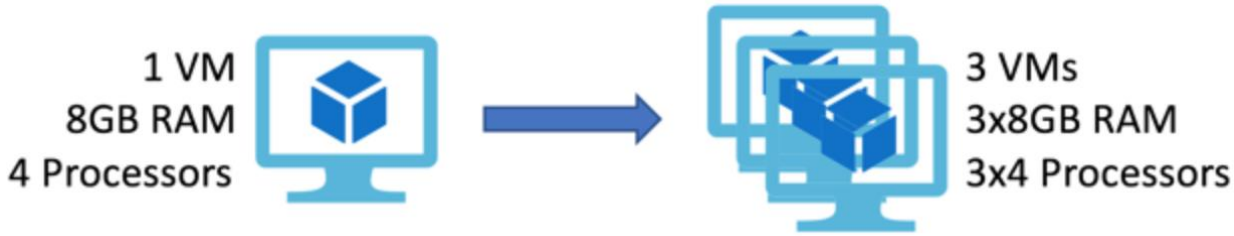

Рисунок 2.2. Принцип горизонтального масштабування веб-додатку у хмарі.

Також горизонтальне масштабування значно спрощує управління зростаючою IT-інфраструктурою, оскільки система та всі бізнес-процеси, що з нею пов’язані, заздалегідь адаптовані до апаратних змін та динамічної переконфігурації інфраструктури.

При горизонтальному масштабуванні приріст доступних обчислювальних ресурсів відбувається за рахунок збільшення кількості серверів, на яких розміщено програму, або примірники додатку (якщо мова йде про PaaS-модель). Концептуально даний процес можна описати наступним чином: якщо раніше додаток використовував лише один сервер, який в певний момент перестав вчасно опрацьовувати вхідний потік запитів, до системи можна додати ще один (ідентичний) сервер, перемістити на нього копію додатку і розподілити навантаження між обома примірниками додатку (частина запитів до додатку буде оброблюватись на першому сервері, а частина - на другому).

Подібний принцип лежить в основі горизонтального масштабування «хмарних» додатків, але замість реальних фізичних серверів використовуються віртуальні машини. Ключовим вузлом подібної системи є брокер навантаження (load balancer), окремий хардверний вузол або програмний компонент, основна функція якого - визначити на який сервер послати запит.

2.1.2. Вертикальне масштабування

Вертикальне масштабування передбачає підвищення загальної продуктивності системи за рахунок збільшення продуктивності кожного окремо взятого її компонента. Масштабованість в цьому випадку означає можливість при потребі замінювати в існуючій обчислювальній системі компоненти більш потужними і швидкими (оновлення апаратної частини), як зображено на рисунку 2.2. Вертикальне масштабування передбачає збільшення продуктивності додатка при додаванні ресурсів (процесора, пам'яті, диска) в рамках одного вузла (хоста) [6].

Рисунок 2.3. Принцип вертикального масштабування веб-додатку у хмарі.

Це найпростіший спосіб масштабування, оскільки він не потребує змінювати або будь-яким чином адаптувати прикладні програми, що працюють на таких системах. Однак, вертикальне масштабування дуже швидко досягає межі ефективності в зв’язку з обмеженими можливостями сучасних апаратних комплексів та фундаментальними обмеженнями технологій (наприклад, темпами розвитку науково-технічного прогресу, далекими від технологічної сингулярності, або законом Амдала), після чого невідворотною стає необхідність горизонтального масштабування системи.

Всі віртуалізовані середовища дуже добре масштабуються горизонтально, але в них існують певні нюанси з вертикальним масштабуванням.

Сильною стороною хмарної інфраструктури в питанні вертикального масштабування є простота, з якою можна тестувати різні по продуктивності конфігурації і оцінювати потреби системи в додаткових ресурсах або більш спеціалізованих конфігураціях.

Вертикальне динамічне масштабування складніше, ніж горизонтальне. Частково це пояснюється тим, що сам процес динамічного вертикального

масштабування частково є окремим випадком горизонтального. Оскільки застосовується наступна послідовність дії:

1. В хмарне середовище додаються нові обчислювальні вузли, використовуючи точно такий же підхід, як при горизонтальному масштабуванні. Єдина відмінність в тому, що використовуються більш потужні екземпляри машин замість повних копій існуючих.
2. Перевірка цих вузлів на працездатність (“health-check”) та часткове перенаправлення запитів на нові екземпляри додатку.
3. Видалення з поточної конфігурації системи одного або декількох (всіх) менш потужних примірників та повне перенаправлення запитів.

При збільшенні навантаження або кількості запитів до веб-сервісу, рано чи пізно вертикальне масштабування (тобто, збільшення фізичних ресурсів сервера) впирається в певну межу і не дає відчутного приросту потужності. У такому випадку подальше масштабування системи можна здійснювати лише горизонтально. Якщо ж комбінувати обидва ці види масштабування можна отримати інфраструктуру, що найбільш ефективним чином споживає доступні обчислювальні ресурси.

2.2. Підходи до масштабування

Автомасштабування або автоматичне масштабування - це метод, який зокрема знайшов застосування в хмарних обчисленнях. При цьому підході обсяг обчислювальних ресурсів у фермі серверів, який зазвичай вимірюється як кількість активних серверів, автоматично коригується залежно від навантаження на ферму. Цей підхід тісно пов'язаний з концепцією балансування навантаження та ґрунтується на ній [18].

Автомасштабування відкриває наступні переваги:

- Компанії, що працюють над власною інфраструктурою веб-серверів, зазвичай вкладають в автоматичне масштабування процес переходу деяких сервери в енергоефективний режим ("сплячий режим") при низькому навантаженні для економії енергетичних витрат (та витрат на

вентиляцію та воду, що використовується для механічного охолодження серверних стійок) [19].

- Для компаній з хмарною інфраструктурою автомасштабування може означати нижчі тарифи, оскільки більшість постачальників хмарних послуг бере плату на основі використання, а не максимальної потужності.
- Навіть для компаній, які не в змозі зменшити використовувану загальну обчислювальну потужність, за яку вони платять, автоматичне масштабування може працювати з меншими затримками часу в системах, з автоматичним масштабуванням при меншому трафіку.
- Автоматизовані рішення для масштабування, такі як Amazon Web Services Autoscaling Groups, можуть обробляти невдалі екземпляри віртуальних серверів, забезпечуючи частковий захист від збоїв обладнання, мережі та додатків.
- Автоматичне масштабування може забезпечити більший час роботи та доступність, коли виробничі навантаження є змінними та непередбачуваними.

Автомасштабування відрізняється від звичайних щоденних, тижневих та річних циклів використання серверів, оскільки реагує на фактичні схеми використання, зменшуючи потенційний витрати за рахунок наявності занадто великої кількості або недостатньої кількості серверів для поточного трафіку. Наприклад, якщо трафік до веб-сервісу зазвичай нижче опівночі, то рішення статичного масштабування може запланувати відключення деяких серверів, хоча це може призвести до перевантаження (або повної відмови системи) вночі, коли люди користуються Інтернетом частіше (наприклад, через вірусні новини). Таким чином, автомасштабування може краще опрацьовувати неочікувані скачки трафіку та згладжувати сплески навантаження [19][20].

Проте автомасштабування – це реактивний спосіб боротьби зі змінами трафіку, оскільки масштабування відбувається лише у відповідь на зміни певних параметрів в реальному часі. У випадках коли зміни відбуваються

дуже швидко реактивний підхід до автомасштабування є недостатнім. Тому використовують ще два інших види масштабування, описані нижче.

Планове масштабування – це підхід до масштабування, коли зміни мінімально або максимально допустимої кількості запущених серверів та бажаного розміру обчислювальних ресурсів здійснюються у певний час доби. Масштабування за розкладом доцільно коли в певний час доби спостерігається гарантовне збільшення чи зменшення навантаження на систему, але зміна занадто раптова для реакції автомасштабування.

Прогнозне масштабування – підхід до масштабування, що використовує прогнозні аналітичні моделі, створені штучним інтелектом. Ідея полягає в кореляції поточниї трендів з історичними даними, а також іншими видами даних для прогнозування навантаження в майбутньому та масштабування на основі цих прогнозів. Так, наприклад, Netflix виявив, що Scryer (їх двигун аналітичного прогнозування), надав кращі результати щодо масштабування для частини інфраструктури та специфічних робочих навантажень, ніж автомасштабування Amazon. Зокрема, він був кращим для: [21]

- Ідентифікації величезних сплесків попиту в найближчому майбутньому і отриманню необхідних потужностей завчасно;
- Роботи з глобальними відключеннями та проблемами з доступністю декількох зон та регіонів хмарної інфраструктури;
- Роботи зі змінними шаблонами трафіку, забезпечуючи більшу гнучкість в швидкості масштабування порівняно з типовим рівнем та швидкістю зміни попитт.

Варто зазначити, що рішення автомасштабування від Amazon Web Services, яке є невід’ємною частиною Elastic Compute Cloud (EC2, веб-сервіс керування обчислювальними ресурсами, віртуальними машинами та примірниками додатків в рамках хмарної інфраструктури AWS), дозволяють використовувати обидва типи масштабування з вже існуючими групами серверів (примірників додатку), не зважаючи на те, що вони концептуально дещо відрізняються від автомасштабування.

2.3. Концепція балансування навантаження

У термінології обчислювальних систем, балансування навантаження (англ. Load balancing) - метод розподілу завдань (запитів) між декількома обчислювальними мережевими пристроями (наприклад, серверами, кластерами, мережевими з’єднаннями, процесорними блоками або накопичувачами данних) з метою оптимізації використання ресурсів, збільшення пропускної здатності, скорочення середнього часу обслуговування запитів та горизонтального масштабування кластера (динамічне додавання/видалення примірників), а також резервування (забезпечення відмовостійкості) [7]. Використання декількох обчислювальних вузлів з балансуванням навантаження замість одного може підвищити надійність і доступність системи за допомогою надлишковості.

Балансування навантаження дозволяє продовжувати роботу навіть в умовах, коли декілька обчислювальних серверів вийшли з ладу. Завдяки цьому зростає відмовостійкість системи і з'являється можливість динамічно регулювати використовувані обчислювальні ресурси, за рахунок швидкого додавання/видалення примірників в кластер [24].

Балансування навантаження передбачає рівномірний розподіл навантаження між усіма доступними обчислювальними вузлами. При появі нових запитів (завдань) балансувальник має прийняти рішення про те, де (на якому обчислювальному вузлі) слід виконувати обчислення, пов'язані з цим новим завданням. Крім того, балансування передбачає міграцію частини обчислень з найбільш завантажених вузлів на менш завантажені. [24]

Балансування може відбуватися як між конкретними віртуальними серверами, так і між групами серверів, сформованими за певним принципом (однакова обчислювальна потужність, оптимізація під певний тип запитів і т.д.). На рис. 2.1. наведена найпростіша схема балансування навантаження хмарного веб-додатку запущено на декількох групах віртуальних серверів. На цій схемі балансувач навантаження здійснює розподіл вхідних запитів між трьома групами серверів, в кожній із яких запущено декілька примірників веб-

додатку. Кожна з трьох черг запитів є незалежною. Таким чином вони можуть містити в собі різну кількість запитів та мати різні показники продуктивності (середню тривалість обробки запиту, розміри запитів, тривалість перебування запиту в черзі). Балансувач навантаження відслідковує ці показники і приймає рішення про доцільність балансування навантаження в залежності від їх значень.

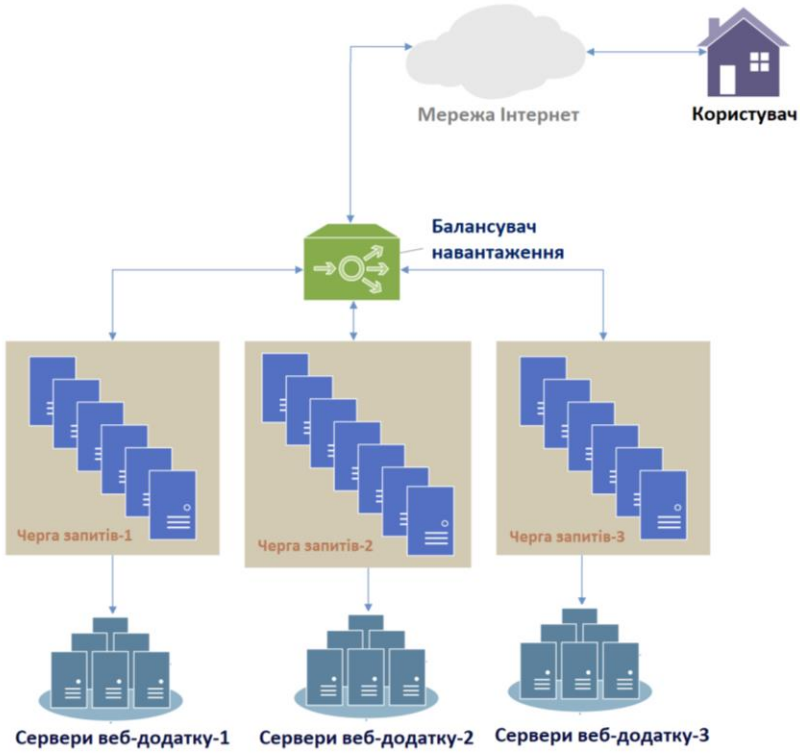

Рисунок 2.4. Принципова схема балансування навантаження веб-додатку у хмарі.

Проблема балансування навантаження виникає з тієї причини, що: [22]

- структурно компоненти розподіленого додатку є неоднорідними, різні логічні процеси вимагають різних обчислювальних потужностей;
- структура самого обчислювального комплексу (кластера), також неоднорідна, так різні обчислювальні вузли мають різну продуктивність;
- структура взаємодії між компонентами неоднорідна, оскільки лінії зв'язку між різними вузлами можуть мати різні характеристики (наприклад, різну пропускну здатність).

2.4. Види балансування навантаження

Балансування навантаження є однією з центральних проблем в області хмарних обчислень. Концептуально схеми балансування навантаження можна розділити на два типи: статичні та динамічні [24].

Статичне балансування виконується до початку виконання розподіленого додатка. Дуже часто при розподілі логічних процесів по фізичним процесорам або віртуальним машинам використовується досвід попередніх виконань, застосовуються генетичні алгоритми[8]. Однак попереднє розміщення логічних процесів є неефективним.

На це впливають як мінімум два фактори:

- відсутність відмовостійкості, обчислювальне середовище, в якому відбувається виконання програми може змінитись, а сам вузол може вийти з ладу.
- обчислювальний вузол, де запущено розподілений додаток, зайнятий ще й іншими обчисленнями, навантаження яких з часом може зрости.

З часом процес попереднього розподілу логічних процесів між комп'ютерами з метою їх паралельної обробки стає неефективним. В тому випадку застосовують динамічне балансування навантаження. Воно передбачає динамічний перерозподіл обчислювального навантаження між вузлами протягом усього часу роботи системи згідно з поточними показниками навантаження, що визначаються для кожної конкретної системи або додатку окремо (в залежності від бізнес-потреб або угоди про рівень обслуговування – SLA). Алгоритми динамічного навантаження не передбачають використання жодних попередніх знань про особливості робочого процесу або глобальний стан системи. Вони базуються виключно на існуючому або поточному стані системи. Програмне забезпечення, що реалізує цей тип балансування, враховує:

- поточне навантаження на обчислювальних вузлах;
- пропускну здатність ліній зв'язку між ними;

– швидкість та частоту обмінів повідомленнями між компонентами розподіленого додатка та ін.

Задачу оптимального розподілу навантаження можна сформулювати наступним чином: при збільшенні потоку запитів, кількість запитів, оброблюваних на кожному з вузлів, повинна збільшитися пропорційно його продуктивності (за умови попереднього рівномірного розподілу навантаження), щоб не допустити появи черги з необроблених запити. Саме це дозволить досягти максимального значення числа оброблених системою запитів за одиницю часу. Нажаль, вирішення подібного типу задач зазвичай невід'ємно пов'язано з їх принциповою складність. Оскільки дана задача є аналогом задачі «про упаковку в контейнери» недетермінованої поліноміальної складності, суть якої полягає в розміщенні змінного набору предметів у мінімальній кількості мішків з певного їх набору. Дана проблема додатково ускладнюється двома обставинами:

1. розподіл навантаження має враховувати відразу декілька параметрів віртуального сервера (наприклад, продуктивності процесора, обсяг оперативної пам'яті, пропускну здатність мережевого з'єднання), внаслідок цього задача стає «багатовимірною задачею про упаковку в контейнери»;
2. навіть у випадку вдалого прогнозування вхідного навантаження хмарні сервіси безперервно змінюють свої апаратні потреби, тобто попередня конфігурація балансування стає неактуальною вже за декілька годин.

Одже, найважливіше завдань при балансуванні – це визначення оптимальних пропорцій розподілу запитів і їх подальше динамічне корегування при зміні умов функціонування системи (зміна числа вузлів, складу встановленого і запущених додатків, модернізація апаратної платформи окремих вузлів і т.д.). Оптимальні пропорції в даному випадку - це таке співвідношення числа оброблених запитів різними вузлами, при якому рівні завантаження цих вузлів приблизно однакові.

2.5. Етапи процесу балансування навантаження

Рішення задачі балансування навантаження зазвичай складається з чотирьох етапів [10]:

- Оцінка завантаження обчислювальних вузлів.
- Ініціація балансування навантаження.
- Прийняття рішень про ре-балансування системи.
- Міграція завдань.

Оцінка завантаженності обчислювальних вузлів

На цьому етапі здійснюється приблизна оцінка завантаження кожного процесора віртуального серверу. Отримана інформація про завантаження використовується в якості бази даних для процесу балансування, по-перше, для визначення виникнення дисбалансу. По-друге, для визначення нового розподілу об'єктів імітаційної моделі шляхом обчислення обсягу робіт, необхідного для переміщення об'єктів. Тобто, якість балансування безпосередньо залежить від точності і повноти інформації в подібній базі даних.

Зазвичай така база даних містить дані двох типів:

- дані про роботу обчислювального вузла, а саме: завантаження процесора, час простою процесора, фонове завантаження, швидкість передачі інформації по лініях зв'язку, обсяг використаної оперативної пам’яті і т.д.
- дані про роботу розподіленого додатка, що включають час виконання окремого завдання, час знаходження в черзі, інтенсивність обміну інформацією і т.д.

Ініціалізація балансування

Занадто часте виконання балансування навантаження може призвести до того, що виконання імітаційної моделі тільки сповільниться. Витрати на сам процес балансування можуть перевищити можливу вигоду від його

проведення. Отже, для продуктивності балансування необхідно якимось чином визначати момент його ініціалізації.

Для цього слід:

- Визначити момент виникнення дисбалансу завантаження.
- Визначити ступінь необхідності балансування шляхом порівняння можливої користі від його проведення та витрат на нього.

Прийняття рішень в процесі балансування

Більшість стратегій динамічного балансування завантаженості можна віднести до класу централізованих або до класу повністю розподілених.

При централізованої стратегії виділений сервер збирає необхідну інформацію про поточний стан всієї обчислювальної системи і приймає рішення про переміщення завдань окремо по кожному з віртуальних серверів.

При повністю розподіленій стратегії на кожному вузлі виконується алгоритм балансування завантаження, який обмінюється інформацією про стан з іншими вузлами. Переміщення відбувається лише між сусідніми вузлами.

Міграція завдань

Після прийняття рішень про балансування відбувається міграція завдань між обчислювальними вузлам для досягнення балансу. При міграції повинна забезпечуватися цілісність стану завдання. Для переміщення даних зазвичай використовуються допоміжні функції програми, особливо коли мова йде про складні об’єктні структури даних.

Для "хмарних" обчислень останній крок замінюється на редірект запитів, що надходять до системи на вибраний примірник додатку.

Висновки до розділу 2

У даному розділі було наведено загальні принципи масштабування та балансування навантаження в хмарних середовищах. Описано основні види

масштабування систем (вертикальне та горизонтальне масштабування) та підходи до масштабування (автомасштабування, планове та прогнозне масштабування).

Також була проаналізована загальна концепція балансування навантаження, визначено найпоширеніші види її реалізації та їх відмінності. Була сформульована задача оптимального розподілу навантаження, що лежить в основі процесу балансування, та описані основні етапи цього процесу.

РОЗДІЛ 3. АНАЛІЗ АЛГОРИТМІВ ТА МЕТОДІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНИХ СИСТЕМАХ

3.1 Балансування на різних рівнях моделі OSI

Балансування навантаження в хмарному середовищі відрізняється від традиційної моделі архітектури балансування навантаження та реалізації серверами для того, щоб виконати балансування навантаження, оскільки важко передбачити кількість запитів, які будуть надіслані на сервер. Балансування навантаження саме по собі є ключовим питанням хмарних обчислень. Це пояснюється тим, що динамічне локальне навантаження має бути рівномірно розподілене між усіма хмарними вузлами, щоб уникнути ситуацій, коли одні вузли перевантажені, а інші простоюють. Це сприяє збільшенню кількості користувачів та використання ресурсів, тим самим збільшуючи загальну ефективність та економічні переваги системи [66].

Операції веб-служб, як правило, не обмежуються одним запитом. Отже, при розробці важливо розуміти, чи можна запити клієнтів обробляти послідовно на різних серверах, чи клієнти повинні бути прив’язані до одного сервера під час роботи на сайті. Це особливо важливо, якщо веб-версія зберігає тимчасову інформацію про сеанси користувачів (у цьому випадку можливе безкоштовне розповсюдження, але сеанс потрібно зберігати у спільному сховищі на всіх серверах). Ви можете "прив'язати" відвідувача до певного сервера за IP-адресою (але може бути змінено) або за допомогою файлів cookie (ідентифікатор сервера попередньо записаний) або просто перенаправити їх у вибраний вами домен.

Процедура балансування виконується з використанням набору алгоритмів та методів, що відповідають наступним рівням моделі OSI [11].

- мережевий;
- транспортний;
- прикладний.

Для балансування на рівні мережі, потрібно вирішити наступну проблему: потрібно переконатися, щоб конкретна IP-адреса сервера відповідала різним фізичним машинам. Балансиру присвоюється IP-адреса веб-служби. Коли до нього виконується запит, застосовується Destination NAT (підміна IP-адреси отримувача). Отже, поточна IP-адреса сервера змінюється на обрану IP-адресу за необхідним алгоритмом, який обробляє запит. Цей підхід показано на рис. 3.1.

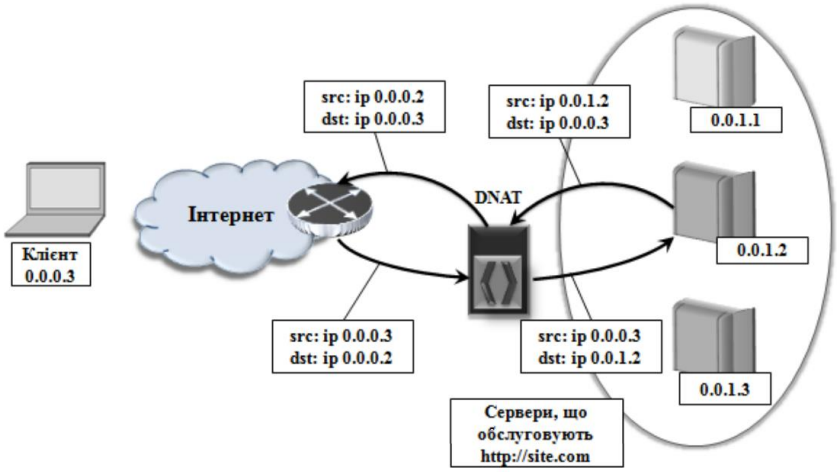

Рисунок 3.1 – Балансування на мережевому рівні моделі OSI.

Можна виділити декілька реалізацій даного підходу.

- DNS-балансування. На одне доменне ім'я виділяється кілька IP-адрес, а обчислювальний сервер визначається за допомогою алгоритму Round Robin (детальніше про даний алгоритм в підпункті 3.3).
- Network Load Balancer кластер. При використанні цього способу сервери об'єднуються в кластер, що складається з вхідних і обчислювальних вузлів.
- Балансування за IP клієнта з використанням маршрутизатора.
- Балансування за територіальною ознакою шляхом розміщення сервісів під однаковими адресами в географічно різних регіонах.

Переваги балансування на третьому рівні [31]:

- незалежність від протоколу вищого рівня;
- абсолютна прозорість для серверів;
- незалежність від мережевого розташування серверів.

Загальні недоліки всіх способів балансування на мережевому рівні:

- весь трафік серверів проходить через балансувальник (єдина точка відмови);
- статичні правила розподілу, без аналізу доступності і поточної завантаженості серверів;
- не підтримують довгострокові сесії між користувачем і сервером.

Балансування на транспортному рівні полягає в тому, що клієнт надсилає запит до балансувальника, а той перенаправляє запит до одного із серверів, що і буде виконувати його обробку. Даний процес зображено на рис.3.2. Вибір сервера для обробки запиту може здійснюватися шляхом простого кругового перебору, шляхом вибору найменш завантаженого сервера з пулу або інщими алгоритмами. На транспортному рівні розподіл відбувається на основі діапазону IP-адреси і порту (наприклад, якщо запит надійшов на http://site.com, то трафік буде перенаправлено бекенду, що працює з 80 портом цього домену).

Відмінність між балансуванням навантаження на транспортному рівні та на мережевому полягає в будові сесії між клієнтом та сервером. До мережевого рівня відносять рішення, що не термінують на собі користувальницькі сесії та TLS [31] (вони просто перенаправляють трафік). На мережевому рівні балансувальник просто вирішує, на який сервер передавати пакети.

На транспортному рівні спілкування з клієнтом замикається на балансувальник, який працює в режимі проксі. Він взаємодіє з серверами від свого імені, передаючи інформацію про клієнта в додаткових даних і заголовках. Таким чином працює популярний програмний балансувальник HAProxy.

Перевагою балансування на транспортному рівні є підтримка різних типів прикладного протоколів до будь-яких TCP-сервісів: HTTPS, SMTP, IMAP, SSH, FTP, SQL і т.д.

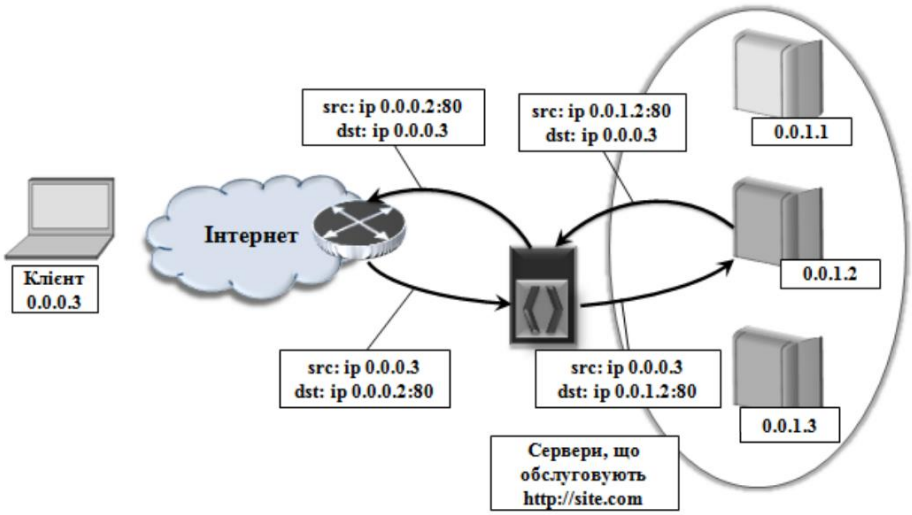

Рисунок 3.2 – Балансування на транспортному рівні моделі OSI.

При балансуванні на прикладному рівні балансувальник працює в режимі «розумного проксі». Він аналізує клієнтські запити і перенаправляє їх на різні сервери залежно від характеру запитуваного контенту. В такому разі у проксі-сервера може виникнути потреба в перетворенні імен ресурсів доступних зовні в імена доступні у внутрішній мережі. Якщо HTTP протоколу треба знати IP-адресу клієнта або інші параметри запиту, можна додавати власні заголовки, наприклад, X-Real-IP [31]. Так працює, наприклад, веб-сервер Nginx, розподіляючи запити фронтенда і бекенда за допомогою модулю Upstream.

Користувач сучасного веб-сайт протягом сесії може виконувати запити як статичного (зображення або відео), так і динамічного контенту (новини, інтерактивні форми, транзакційна інформація, як то банківські виписки або статус замовлення). Балансування навантаження на прикладному рівні дозволяє маршрутизувати ці запити на основі, наприклад, запитуваного змісту. Таким чином запити зображень або відео будуть направлені до серверів, де вони зберігаються, і які, зокрема, можуть бути додатково оптимізовані для обробки мультимедійного контенту. Запити транзакційнної інформації в свою чергу будуть направлені на сервер відповідних додатків і

бізнес-логіки. Також балансування навантаження на прикладному рівня дозволяє збільшити ефективність інфраструктури додатків, тому що різні типи контенту мають різноманітні вимоги до центрального процесора, пропускної здатності тощо.

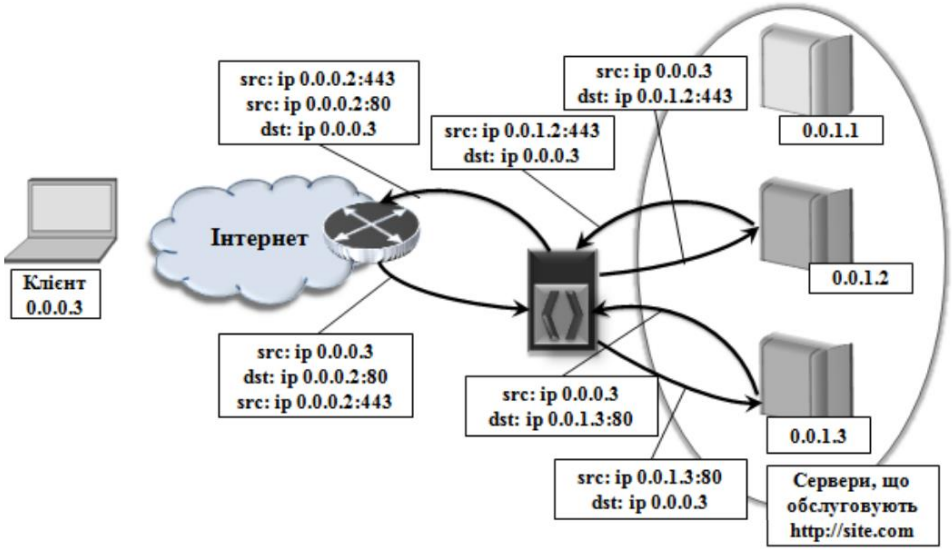

Рисунок 3.3 – Балансування на прикладному рівні моделі OSI.

У параметрах конфігурації можна задавати ваги серверам, допустимий час відгуку і допустиму кількість неуспішних запитів. Сервер може підтримувати довгострокові сесії клієнтів з одним і тим самим сервером на основі їх IP-адрес або значень cookie. HTTP-проксінг застосовується в модулі mod_backhand для сервера Apache, в сервері nginx та HAProxy. На його базі випускаються також спеціалізовані апаратно-програмні платформи для балансування навантаження [31].

Список переваг та недоліків рішень, що працюють на прикладному рівні наведено в таблиці 3.1.

Таблиця 3.1. Переваги та недоліки балансування на прикладному рівні

Переваги	Недоліки
• Маніпуляція запитами і відповідями, додавання заголовків, що можуть використовуватися, наприклад, для підвищення безпеки програми або для перекодування.	• Найбільше споживання ресурсів з усіх розглянутих методів, оскільки робота йде з протоколами більш високого рівня.

• Підтримується прив'язка клієнта до сервера для зберігання його довготривалої сесії. • Аналіз змісту запитів дозволяє розподіляти їх в залежності від типу по різних серверах (наприклад, до статичних сторінок, до динамічних сторінок і т.д.), що прискорює час відгуку вцілому. • Можливість стискати дані і кешувати відповіді на проксі-сервері, знижуючи загальне навантаження обслуговуючого сервера. • У ряді випадків є можливість перемістити обробку SSL з обслуговуючих серверів на проксі; • Аналіз непрямих показників завантаження и автоматична перевірка працездатності серверів додатків.	• Для кожного прикладного протоколу повинен бути свій тип проксі. Не для всіх протоколів проксінг можна реалізувати таким чином, щоб він вирішував поставлену задачу

3.2 Типи систем балансування навантаження.

Можна виділити наступні класи систем рожподілу навантаження (рис 3.4):

- за допомогою єдиного пристрою балансування (рис. 3.4, а);
- балансування засобами кластера (рис. 3.4, б);
- без єдиного пристрою балансування або балансування на клієнтській стороні (рис. 3.4, в) [9].

Розглянемо ці конфігурації та методи, що в них працюють, докладніше.

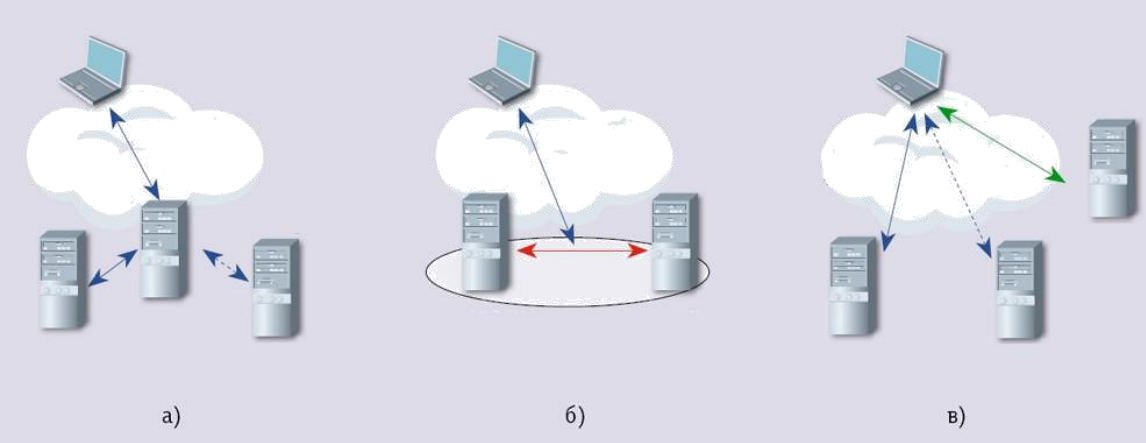

Рисунок 3.4 – Можливі конфігурації систем балансування навантаження.

3.2.1 Балансування з пропуском трафіку через один пристрій

Клієнт шле запит на один фіксований, відомий йому сервер, а той вже перенаправляє запит на один з робочих серверів. Типовий приклад - система з одним frontend і декількома backend-серверами, на які відправляються запити. Однак «клієнт» може знаходитися і всередині нашої системи - наприклад, скрипт може слати запит до проксі-сервера бази даних, який передасть запит одному з серверів СУБД [32]. Перевагою цього підходу є те що клієнту нічого не треба знати про внутрішній устрій системи, все це знає тільки балансувальник. Основний недолік в тому, що балансувач стає єдиною точкою відмови в системи - якщо він вийде з ладу, вся система стане непрацездатною. Тому такий підхід є доцільним лише якщо сам балансувач є кластеризованим або є дуже потужним сам по собі для того щоб забезпечити потрібний рівень резервування пропускнох здатності системи.

Система балансування що використовує NAT-метод, направляє отриманий від клієнта пакет до призначеного сервера лише після того, як замінить в пакеті адресу одержувача IP-адресою призначеного сервера і змінить IP-адресу відправника на свою [32]. Це дозволяє приховувати від клієнтів IP-адреси веб-серверів, тобто для них можливо використовувати приватні IP-адреси. Даний метод доцільний тільки якщо не потрібно створення тривалих сесій між користувачем і сервером, оскільки для кожного

користувача потрібно виділити індивідуальну IP-адресу на внутрішньому інтерфейсі системи балансування на весь час сесії.

Відмінністю методу NAPT (Network Address Port Translation) є те, що система балансування змінює в заголовку вже не тільки IP-адреси, як в NAT, але і номери портів транспортного рівня. Це дозволяє для кожного нового надходження запиту від нового унікального користувача вже не виділяти індивідуальну IP-адресу на внутрішньому інтерфейсі системи балансування для сесії, а виділяти унікальний номер порту транспортного рівня [32]. Даний механізм балансування можна реалізувати на універсальних апаратних платформах, на програмних компонентах він входить в стандартні поставки сучасних операційних систем (natd в BSD, iptables у складі дистрибутивів Linux, RRAS у складі серверних збірок Windows і т.д).

Прикладами продуктів, у яких реалізовано підхід NAPT, є пристрої XTM Firewall Appliance компанії WatchGuard, універсальні маршрутизатори Cisco 1900, міжмережевий екран D-Link DFL-860 і т.п. Всі вони не аналізують можливості серверів, час відгуку, доступність серверів і т.п. Вони здатні виконувати балансування за алгоритмами послідовного перебору або на підставі кількості відкритих сесій TCP на кожен з серверів.

Балансування з використанням трансляції заголовків на рівнях вище транспортного також називають проксінг. Проксі-сервер, міняючи поля заголовків транспортного та рівня додатків, перенаправляє запит на сервери, на яких розташовані запитувані ресурси. У ряді випадків він може надати запитуваний ресурс взагалі без підключення до якого-небудь сервера. При цьому проксі-сервер «кешує» відповіді віддалених серверів і надає збережену інформацію у відповідь на всі наступні запити. Проксі-сервер може розподіляти навантаження між декількома веб-серверами, які можуть обслуговувати різні галузі застосування. Проксінг застосовується в модулі mod_backhand для сервера Apache і в сервері nginx. При надходженні HTTP-запитів проксі перенаправляє їх на заздалегідь сконфігуровані сервери з

однаковими ресурсами. У параметрах конфігурації можна задавати ваги серверам, допустимий час відгуку і допустиму кількість неуспішних запитів. Прикладом програмного рішення, що використовує методологію проксінгу запитів TCP/HTTP з більш просунутими функціями управління перенаправлення запитів, є HAProxy, що дозволяє також виконувати балансування додатків. HAProxy працює на платформах Linux, Free BSD і Solaris. На його базі випускаються також апаратно-програмні платформи, призначені тільки для балансування навантаження.

Для HAProxy була продемонстрована можливість функціонування на швидкостях до 10 Гбіт/с і при 40 тис. одночасних сесій (без контролю системою балансування часу обробки запитів серверами). На базі HAProxy розробляються сервіси балансування навантаження Loadbalancer.org, які використовуються спільно з хмарними сервісами Amazon.

Прикладами спеціалізованих апаратно-програмних рішень, що виконують балансування з використанням трансляції заголовків мережевого, транспортного та рівня додатків, є Brocade Server Iron ADX, xBalancer компанії Net Optics, Cisco ACE 4700 Application Control Engine, Equalizer E650GX компанії Coyote Point і ряд інших продуктів. Додатково на ці пристрої можливе винесення функцій формування захищеного з'єднання SSL і стиснення даних, що дозволяє значно знизити навантаження на сервери додатків. Такі рішення мають більш високу продуктивність, зокрема, для Server Iron ADX заявлена наскрізна продуктивність може досягати 70 Гбіт/с та 16 млн сесій в секунду. При цьому зберігається вся функціональність з контролю за встановленими сесіями на окремі сервери. Більшість хмарних платформ, що надають сервіси SaaS, використовують подібні апаратно-програмні рішення для балансування навантаження всередині ЦОД. Зокрема, Brocade Server Iron ADX використовується в хмарних сервісах Google, Apple, Yahoo, в програмних продуктах корпоративного рівня, додатках Microsoft, HP, IBM і т.д.

Серед основних недоліків рішень, що використовують виділений балансувач, слід відзначити наявність єдиної точки відмови, та обмежені фугкціональні можливості при використанні програмних продуктів на базі універсальних апаратних платформ, та високу вартість рішень на базі спеціалізованих апаратно-програмних комплексів (що зумовлена вузькою сферою застосування і високими вимогами по продуктивності і надійності).

3.2.2. Балансування засобами кластера

Кластеризація дозволяє управляти групою незалежних серверів як єдиною системою, що підвищує відмовостійкість, спрощує управління і дозволяє домогтися більшої масштабованості. Сервіс балансування в кластері забезпечує розподіл потоку IP-даних між вузлами, за замовчуванням навантаження буде рівномірно розподілятися між всіма доступними вузлами. Для програм з великою кількістю клієнтів і породжуваних ними коротких запитів, таких як веб-сервіси, подібний механізм розподілу навантаження забезпечує ефективне балансування навантаження і швидку реакцію на зміну складу вузлів кластера.

Кожен сервер кластера під управлінням сервісу балансування навантаження періодично розсилає широкомовні або групові повідомлення іншим вузлам і приймає такі ж повідомлення від інших членів кластеру. При збої сервера решта вузлів перерозподіляють навантаження, забезпечуючи безперервне обслуговування. Подібним чином працюють механізми балансування мережного навантаження в кластерах Oracle Solaris Cluster, Windows Server 2008 R2, Linux Virtual Server і Open HA Cluster.

До недоліків таких рішення можна віднести необхідність розташування серверних платформ в одному сегменті мережі та необхідність використання на кожному вузлі спеціалізованого ПЗ, в завдання якого входитиме контроль стану суміжних вузлів в кластері і диспетчеризація запитів, що надходять.

3.2.3. Балансування без єдиного пристрою

Якщо ми хочемо уникнути єдиної точки відмови, існує альтернативний варіант - доручити вибір сервера самому клієнту. Безсумнівним плюсом є відсутність точки відмови - при відмові одного з серверів клієнт зможе звернутися до інших. Однак платою за це є ускладнення логіки клієнта і менша гнучкість балансування.

Даний варіант розподілу навантаження організовується шляхом відправки первісного запиту від користувача (наприклад, запиту IP-адреси для встановлення з'єднання або першого запиту на встановлення HTTP-сесії) на виділений сервер-балансувальник. Після отримання запиту балансувальник вказує, з яким сервером додатків з інфраструктури хмари продовжувати роботу (повертаючи відповідну IP-адресу або перенаправляючи його засобами HTTP Redirect). Подальша взаємодія відбувається без участі балансувальника.

Найбільш відомий метод, який реалізує даний підхід, - використання кругових DNS або Round robin DNS. Це методів розподілу навантаження за рахунок управління відповідями DNS-сервера відповідно до якоїсь статистичної моделі. Він зазвичай застосовується для веб-серверів та FTP-серверів[12]. Більшість сучасних DNS-серверів підтримують опцію кругових DNS. Дуже часто цей метод використовується для розподілу навантаження територіально розподілених веб-серверів.

У найпростішому випадку Round robin DNS відповідає на запити не однією IP-адресою, а списком з декількох адрес серверів, що мають ідентичні можливості. Порядок, в якому повертаються IP-адреси зі списку, заснований на алгоритмі Round-Robin (детальніше в пункті 3.3). З кожною відповіддю послідовність ip-адрес змінюється. Проте не існує стандартної процедури для визначення того, які адреси будуть використовуватися - деякі сервери намагаються змінити порядок списку, віддаючи приорітет більш «близьким» мережам.

Хоча його легко реалізувати, все ж цей метод має декілька недоліків, пов'язаних з кешуванням записів в ієрархії, а також з кешуванням на стороні клієнта виданої адреси та її повторного використання, поєднанням яких важко керувати. Також цей метод не перевіряє і не гаратує доступність серверів, не враховуються відповідность IP-адреси користувача і його географічного розташування, час виконання, навантаження на сервер, перевантаження мережі і т. д. Кругова система DNS найкраще підходить для послуг з великою кількістю рівномірно розподілених з'єднань з серверами еквівалентної потужності

Існують способи, щоб подолати такі обмеження. Наприклад, модифіковані DNS-сервера (наприклад, lbnamed), що регулярно опитують дзеркала серверів для перевірки їх доступності та завантаженості. Якщо сервер не відповідає, то він може бути тимчасово вилучений з пулу DNS, поки не стане відомо, що він знову працює у відповідності зі специфікацією.

Рідше використовується балансування засобами HTTP Redirect. Рішення, що використовують даний підхід, дозволяють задавати в конфігурації сервера, що виконує балансування, групу IP-адрес серверів додатків, для яких може бути задано параметр ваги, що дозволяє визначити частку HTTP-запитів, які відправляються на кожен з серверів додатків. Прикладом рішень з механізмами HTTP Redirect, є модуль mod_backhand для серверів Apache та ADC Citrix NetScaler.

З точки зору простоти, передбачуваності адміністрування і масштабування рішення без єдиного балансувача можна вважати практично ідеальними, оскільки необхідно лише один раз внести в базу DNS кілька записів. Оскільки пристрій балансування при цьому фактично обробляє тільки початкові запити користувачів, а не весь їх трафік, то межі масштабування пристроїв досить високі. Відмовостійкість самих DNS-серверів або серверів, що виконують HTTP Redirect, можливо забезпечити традиційними засобами резервування DNS.

Разом з тим надійність розподіленої програми, створеної за допомогою кругового DNS або засобів HTTP Redirect, досить низька: якщо один або більше серверів, на яких міститься копія додатку, виходить з ладу або зупиняється на профілактику, то частина запитів «відправляється в порожнечу». Відсутність зворотного зв'язку не дозволяє виключити перевантажений сервер додатків зі списку, куди перенаправляються користувачі.

Найбільш придатними до використання при побудові SaaS-систем виглядають рішення без пропуску трафіку через один балансувальник, проте відсутність механізмів зворотного зв'язку між серверними платформами, на яких розташовується додаток, і пристроєм, що виконує балансування, не дозволяє повною мірою скористатися їх перевагами.

3.3. Порівняння способів балансування навантаження

Методи балансування зазвичай жорстко прив’язані до певного рівня моделі OSI і характеризуються певною конфігурацією системи розподілу запитів. Базуючись на цьому твердженні, проведемо порівняння не конкретних методів, а проаналізуємо кожну з розглянутих в пункті 3.2. конфігурацій та загальні особливості методів що відносяться до кожного з описаних рівнів моделі OSI.

Балансування на мережевому рівні здійснюється у вигляді DNS-балансування, NLB-кластеру, балансування за IP-адресою клієнта засобами вхідного шлюзу або засобами маршрутизаторів глобальної мережі. Метод NAPT (Network Address Port Translation) та використання проксі-серверів відносяться до транспортного рівня. До методів прикладного рівня належать проксінг та HTTP Redirect.

У таблицях 3.2 та 3.3 наведено порівняльну характеристику конфігурацій та методів балансування навантаження на кожному рівні моделі OSI за основними критеріями, що характеризують потреби інфокомунікаційних систем та висонавантажених веб-сервісів. Критерії, що розглядаються:

- Ефективність – рівномірність розподілу навантаження при різних умовах (однотипні запити, різні запити, сесії, т.д.);
- Показники завантаження вузлів – прямими показниками є об’єм використаного процесорного часу, оперативної пам’яті, диску і т.д.); непрямі - час відгуку (latency), кількість активних з’єднань до вузла;
- Врахування гео-положення – можливість зниження часу відгуку рахунок врахування показників завантаженості ліній зв’язку;
- Маштабовність – можливість ефективного масштабування системи в процесі балансування;
- Контентно залежний аналіз, кешування – кешування відповідей та балансування вхідних запитів в залежності від запитуваного контенту;
- Забезпечення довгострокових сесій – декількох повторних послідовних запитів від клієнта будуть опрацьовані одним і тим самим вузлом в межах поточної сесії;
- Різні платформи обчислювальних вузлів – можливість розподіляти запити між неоднорідними обчислювальними платформами;
- Незалежність від типу мережі – вплив мережі, де знаходяться обслуговуючі сервери (локальна чи глобальна) на процес балансування;
- Автоматичне визначення і видалення непрацюючих вузлів – наявність моніторингу стану здоров'я обслуговуючих серверів, автоматичне видалення непрацездатних вузлів з загального пулу з подальшим перерозподілом навантаження що на них надходило;
- Незалежність від прикладного протоколу – вплив протоколу прикладного рівня, що використовується в системі на балансування;
- Затрати ресурсів на балансування – обсяг фізичних ресурсів обслуговуючих серверів затрачених лише на процес розподілу навантаження як такий.

Розшифровка позначень:

+ – є (значний);

+ (лише єдин.) – проявляється лише в конфігураціях з єдиним балансиром;

- – немає (незначний);

-/+ – частіше немає, чим є (переважно незначний);

+/- – частіше є, чим немає (переважно значний).

Таблиця 3.2. Порівняння різних конфігурацій балансування навантаження

	Через єдиний пристрій балансування	NLB-кластери	Без єдиного пристрою балансування
Ефективність розподілу навантаження	+	+	-
Показники завантаження вузлів (прямі,непрямі)	прямі, непрямі	прямі, непрямі	-, або непрямі
Врахування геоположення вузлів	+	лише деякі реалізації	-/+
Можливості маштабування	-	+	+
Контентно залежний аналіз, кешування	+	-	-
Забезпечення довгострокових сесій	+	+	-
Різні платформи обчислювальних вузлів	+	+/-	+
Незалежність від типу мережі	+	-/+	+
Автоматичне визначення і видалення непрацюючих вузлів	+	-/+ (з HA)	-/+
Незалежність від прикладного протоколу	+/-	+/-	+
Затрати ресурсів на балансування	+	+	+

Таблиця 3.3. Порівняння методів балансування на різних рівнях моделі OSI

	3-й рівень OSI	Проксі 4-го рівень OSI	7-й рівень OSI	
			Проксінг	HTTP Redirect
Ефективність розподілу навантаження	+ (лише єдин)	+	+	+
Показники завантаження вузлів (прямі,непрямі)	-, рідко непрямі	прямі, непрямі	непрямі	-
Врахування геоположення вузлів	-/+	-	-	-
Можливості маштабування	+ (лише без един.)	-	-	++
Контентно залежний аналіз, кешування	-	+	+	+ (але без кешування)
Забезпечення довгострокових сесій	+ (лише един.)	+	+	-
Різні платформи обчислювальних вузлів	+	+	+	+
Незалежність від типу мережі	+	-	-	-
Автоматичне визначення і видалення непрацюючих вузлів	-/+ (рідко)	+	+	-
Незалежність від прикладного протоколу	+	+	-	-
Затрати ресурсів на балансування	+	+	- -	++

Як бачимо з таблиці 3.2 оптимальними для хмарних сервісів є конфігурації з єдиним пристроєм балансування (лоад-балансером), проте разом з усіма перевагами вони мають декілька очевидних критичних недоліків (єдина точка

відмови та обмежені можливості масштабування), які можна частково вирішити за допомогою постійного моніторингу стану віртуальних серверів та резервування. Кластерні рішення виглядають перспективними для високонавантажених систем з великою кількістю одночасних користувацьких сеансів середньої тривалості з однотипними запитами, але вони вимагають розміщення серверної платформи в одному сегменті мережі та унеможливлюють побудову системи, що складається з груп обчислювальних вузлів різної потужності. Власне ці недоліки не дозволяють в повній мірі використовувати кластерні конфігурації в хмарній інфраструктурі. Найбільш придатними до використання при побудові хмарних SaaS-систем виглядають рішення без єдиного пристро. балансування з огляду на простоту, масштабованість та найвищу відмовостікість подібних рішень. Але в подібних конфігураціях потрібно враховувати відсутність механізмів зворотного зв'язку, ускладнення клієнтської частини додатку та неможливість адаптуватись до швидких змін навантаження та апаратних збоїв.

Що стосується вибору рівня на якому буде здійснюватись розподіл навантаження, то однозначну відповідь можна отримати лише для кожної окремо взятого випадку. Зокрема, методи мережевого рівня будуть ідеальними для систем з прогнозованим характером навантаження (більш менш рівномірний трафік, зміну якого в часі можна описати певним шаблоном), оскільки дозволяють виконувати статичне балансування навантаження (плановий розподіл запитів). Методи транспортного рівня дозволяють виконувати ефективний розподіл навантаження веб-сервісів, що повинні опрацьовувати різні типи вхідних запитів одразу декількох TCP-протоколів. Для потреб хмарних сервісів найбільш доцільно використовувати методи прикладного рівня з огляду на їх ефективність та адаптованість до особливостей роботи хмарних додатків.

3.4 Алгоритми розподілення навантаження

Алгоритми балансування навантаження власне вирішують якому серверу буде віддано наступне завдання або запит. Рішення визначає, якому серверу буде передана наступна порція навантаження. Подібні алгоритми класифікуються на статичні та динамічні.

3.4.1. Статичні алгоритми

Ці алгоритми найкраще підходять для систем, що мають меншу різницю навантажень в пікові та звичайні періоди (трафік зазвичай стабільний або сильно детермінований та його поведінка піддається прогнозу). У цих видах алгоритмів навантаження рівномірно розподіляється між кількома серверами. Як показано на рис. 3.5, ці алгоритми вимагають попередніх знань та інформації про системні ресурси [45]. На початку виконання завдань продуктивність процесорів визначена заздалегідь, оскільки завдання розподіляються на сервери під час компіляції. Переміщення навантаження посеред виконання на інший сервер для збалансування навантаження неможливо апріорі. Це також можна розглядати як загальний мінус алгоритмів статичного балансування навантаження, оскільки це негативно впливає на продуктивність системи, що може призвести до коливань навантаження.

Рис. 3.5. Принцип роботи статичного алгоритму

Нижче перераховані алгоритми балансування навантаження, які вимагають попередніх знань та інформації про систему або є статичними в своїй поведінці:

Round Robin

Це найпростіший і зрозумілий алгоритм балансування навантаження. У Round Robin кожному серверу надається часовий квант, протягом якого він виконує завдання. При цьому запити передаються на кожен сервер один за одним і після того, як запит передається останньому серверу - наступний запит знову призначається першому серверу [46]. Тобто цей алгоритм в основному пересилає запити на сервери в циклічній формі. Наприклад - якщо три сервери є A, B і C, а шість клієнтських запитів є 1,2,3,4,5 і 6, то цей алгоритм відправляє перший запит на сервер A, другий запит на сервер B, третій запит до сервера C, а потім він знову переміщується назад до сервера A і відправляє четвертий запит серверу A. Цей процес триває до тих пір, поки всі запити не будуть перенаправлені на сервери. Цей алгоритм передбачає, що всі сервери мають однакову обчислювальну здатність і циклічно пересилає запити. Але це може бути також недоліком цього алгоритму, оскільки може існувати можливість, коли всі сервери недостатньо схожі для управління та обробки однакового навантаження. Навіть якщо будь-який сервер має більше обчислювальних можливостей, таких як оперативна пам'ять або дисковий простір, алгоритм не має можливості переадресувати більше запитів на цей сервер порівняно з іншими (що мають меншу обчислювальну потужність). Це може призвести до перевантаження серверів з меншими обчислювальними можливостями, тоді як потужніші сервери простоюють.

Цю проблему вирішує зваженого алгоритм (*Weighted Round Robin*), оскільки він дозволяє призначати вагу серверам на основі критеріїв того, яку кількість трафіку може обробити кожен сервер.

Opportunistic Load Balancing (OLB)

Будучи статичним алгоритмом, OLB не дбає про поточне навантаження на кожному вузлі. Основна мета OLB - тримати кожен сервер на вузлі зайнятим. Незалежно від поточного навантаження на вузол OLB відправляє незавершені завдання на сервери випадковим чином, навіть не обчислюючи час виконання вузла [47]. Подібний план балансування навантаження має незначні

результатів. Обробка завдань відбувається повільно, оскільки обчислення часу виконання вузлів не виконується цим методом, що іноді спричиняє вузькі місця, хоча деякі вузли простоюють [48].

Min-Min Algorithm

У цьому алгоритмі процедура починається з розрахунку мінімального часу виконання всіх завдань. Потім із часу завершення вибирається мінімальне значення і запит перенаправляєтьсядо будь-якої машини відповідно до мінімального часу [49]. Потім усі інші терміни виконання завдань оновлюються на цій конкретній машині, а завдання потім скидається з машини. Цей процес триває доти, доки всі завдання не будуть розподілені поміж ресурсами. У цьому методі завдання з великим часом виконання повинні чекати довше, ніж інші. Цей алгоритм дає найкращі результати, коли кількість завдань з меншим часом більше, ніж великих [50]. Недоліком *Min-Min* алгоритму є голод.

Max-Min Algorithm

У цьому алгоритмі максимальне значення вибирається серед мінімальних часів виконання всіх завдань і саме ця машина отримує запит [51]. Потім усі інші терміни виконання завдань оновлюються на цій конкретній машині, а завдання потім скидається з машини [52]. Оскільки великі завдання розподіляються на повільніші ресурси, це мінімізує час очікування коротких завдань, призначаючи їх до більш швидких ресурсів. Це надає найвищий пріоритет завданням з максимальним часом виконання. Max-Min покращує одночасну обробку завдань на ресурсах.

3.4.2. Динамічні алгоритми

Для динамічних алгоритмів попередні знання про систему не потрібні, а продуктивність залежить від поточного або теперішнього стану системи, як показано на рис. 3.6. В динамічному середовищі здійснюється пошук найлегшого сервера серед всіх доступних серверів, і цей сервер обирається для обробки завдань/запитів. Ці алгоритми легко адаптуються до змін

сумарного навантаженням під час виконання завдань [53, 54]. Динамічне середовище вважається складним і комплексним для застосування, але в ньому розподіл навантаження відбувається більш ефективно.

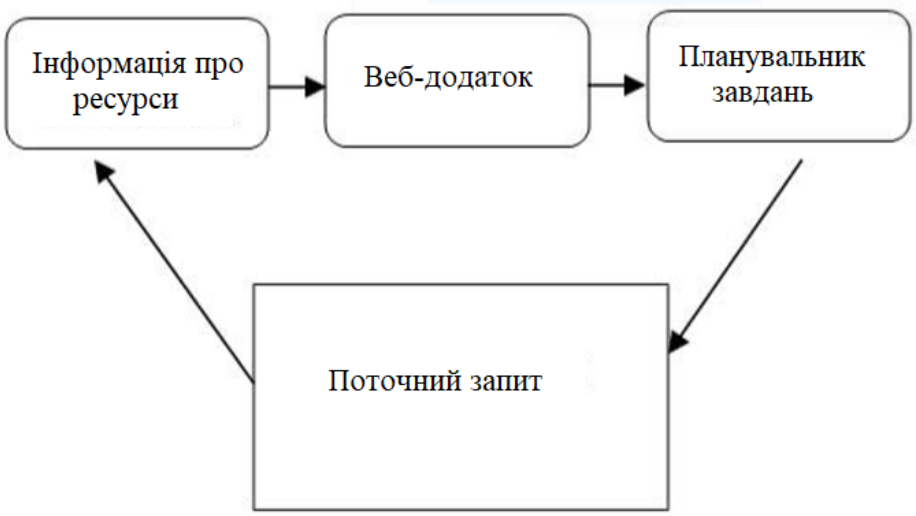

Рис. 3.6. Принцип роботи динамічного алгоритму

Ant Colony Optimization Algorithm (ACO)

ACO натхненний поведінкою справжніх мурах при пошуку нових джерел їжі та поверненні їжі до гнізда за допомогою існуючих джерел їжі. Дослідники знаходять найкоротший шлях, використовуючи феромоновий шлях, якщо знаходять якусь перешкоду на шляху. Інтенсивність феромону продовжує зростати, якщо все більше і більше мурашок йде з одного шляху. Шлях з найбільшою інтенсивністю феромонів вважається найкоротшим шляхом. Подібна поведінка застосовується для збалансування навантаження між вузлами в вищезгаданому алгоритмі [55]. Якщо мураха знайде цільовий вузол, то він покінчить життя самогубством, щоб уникнути непотрібного руху назад. Мураха генерує два типи феромонів під час подорожі [56]:

1. Феромон для добування їжі (FP): В алгоритмі феромон для добування генерується після того, як мураха знаходить недовантажений вузол і шукає перевантажений вузол [57]. Отже, коли виявляється недовантажений вузол, мураха знаходить наступний вузол за допомогою FP.

2. Trail Pheromone (TP): TP генерується після того, як мураха знаходить перевантажений вузол і знаходить свій шлях назад до недостатньо завантаженого вузла. Отже, коли мураха знаходить перевантажений вузол, він відслідковуватиметься до ненавантаженого вузла за допомогою TP.

Honey Bee Foraging

Рендлз відкрив цей алгоритм, натхненний поведінкою медоносних бджіл у пошуку їжі [58]. Бджоли бувають двох видів - шукачі та збирачі [59]. Бджоли-шукачі виходять назовні в пошуках їжі, і коли вони повертаються до своїх вуликів, вони сповіщають інших бджіл про якість, кількість та розташування їжі. Потім бджоли-збирачі йдуть до цього джерела і збирають їжу, а коли повертаються, розповідають іншим про кількість залишеної їжі. Використовуючи цю саму поведінку, сервери групуються у віртуальні сервери, і всі ці віртуальні сервери мають власну чергу запитів. Після обробки запиту з черги кожен сервер обчислює наявний прибуток або вигоду. Якщо прибуток виявляється високим, сервер залишається на місці, в іншому випадку він перетворюється в шукача. Недоліком цього алгоритму є те, що вимога наявності різних черг та обчислення прибутку кожним сервером спричиняє додаткові накладні витрати, які не дають жодних покращення пропускної здатності.

Active Clustering Load Balancing

Активна кластеризація - це алгоритм, заснований на концепції кластеризації в хмарних обчисленнях [60]. Принципом активної кластеризації є групування вузлів та робота над цією групою. Завдяки подібному групуванню ресурси можна використовувати ефективніше. Процес кластеризації описується концепцію сватання. Початковий або стартовий вузол спочатку шукає всі сусідні вузли, а потім створює зв’язок із сусіднім вузлом іншого типу (він називається вузлом сватання). Для балансування навантаження цей процес виконується ітеративно для всієї мережі.

Throttled Load Balancing

Цей алгоритм заснований на принципі пошуку відповідної віртуальної машини для запиту [61]. Балансир навантаження відстежує всі віртуальні машини у вигляді таблиці, а їх статуси (зайняті чи доступні) також зберігаються до цієї таблиці. В периметрі хмарної інфраструктури можуть бути різні екземпляри віртуальних машин. Їх можна згрупувати відповідно до запитів, які вони можуть обробити. Щоразу, коли клієнт надсилає запит, балансир шукає в таблиці групу віртуальних машин, що підходить для даного типу запиту, і розподіляє завдання на екземпляр із найменш завантаженною віртуальною машиною. Після призначення завдання віртуальній машині таблиця знову оновлюється зі статусами всіх ВМ. Якщо немає доступного сервера у вільному стані, то завдання зберігається в черзі.

Equally Spread Current Execution Algorithm (ESCE)

ESCE - це алгоритм, який працює над стратегією широкого спектру, в якій балансири розподіляють навантаження між декількома віртуальними машинами. Балансир докладає зусиль, щоб рівномірно розподілити навантаження на декілька серверів. На першому кроці, пріоритет процесів обчислюється за їх розміром та здатністю передавати навантаження на сервер, який зможе опрацювати це навантаження за менший час та з максимальною пропускною здатністю [62]. Потім процес розподіляється на цей сервер. Якщо якась ВМ стає вільною, а інші ВМ хочуть звільнитися від поточного навантження, то балансир передає відсоток навантаження на вільну ВМ, рівномірно розподіляючи навантаження [63]. Подібний розподіл навантаження на різні вузли називається технікою розширення спектру.

Упереджена випадкова вибірка (Biased-Random Sampling)

У цьому алгоритмі використовується метод випадкової вибірки для врівноваження навантаження. Кожен сервер в мережі представлений або розглядається як вузол, а всі вузли представлені у вигляді віртуального графа [64]. Доступність ресурсів представлена ступенем вершини, а ступінь вказує

на кількість вільних ресурсів. Ступінь графа зменшується на одиницю, коли завдання виділяється вузлу, що вказує на зменшення наявності вільних ресурсів, і збільшується на одиницю, коли завдання виконується, що тим самим вказує на наявність вільного ресурсу. Випадкова вибірка використовується для призначення завдання вузлу.

Join-Idle Queue Load Balancing

Цей алгоритм складається з диспетчера, якому процесор повідомляє його час доступності чи простою [65], навіть не беручи до уваги завдання що надходять. Це виключає роботу по балансуванню навантаження з критичного шляху обробки завдань, що тим самим допомагає зменшити навантаження системи. Це захищає систему зв'язку від накладних витрат під час прибуття нових завдань і не дозволяє збільшуватись фактичному часу відгуку. Цей алгоритм в основному застосовується для великомасштабних веб-сервісів.

3.5. Порівняння алгоритмів балансування

Існує кілька заздалегідь визначених метрик, на основі яких ми можемо порівняти всі алгоритми балансування навантаження та перевірити їх ефективність. Ці показники наведені в таблиці 3.4 та детально описані нижче.

- *Пропускна здатність*: використовується для перевірки кількості процесів, виконання яких завершено. Пропускна здатність відображає кількість виконаних завдань за одиницю часу. Чим більше пропускна здатність тим ефективніше працює система в цілому.
- *Відмовостійкість*: передбачає, що якщо в будь-якій частині системі виникає відмова або помилки в роботі, то алгоритм не повинен впливати на інші частини системи, і дозволяє підтримувати безперервність бізнесу.
- *Час відповіді*: час, який витрачається алгоритмом на реакцію на новий запит у системі. Він повинен бути мінімальним для кращої ефективності алгоритму.

- *Накладні витрати (overhead)*: додаткове навантажженя на систему, що відбувається через тривалий процес перенаправлення навантаження з однієї машини до іншої. Його слід звести до мінімуму для ефективної роботи алгоритму.
- *Використання ресурсів*: використання ресурсів у системі для забезпечення процесу балансуваня. Ресурси повинні оптимально використовуватися алгоритмом.
- *Масштабованість*: здатність ефективного масштабування вузлів відповідно до поточних потреб, незалежно від процесу розподілу запитів.
- *Продуктивність*: Це ефективність системи під час балансування навантаження. Продуктивність повинна бути покращена за рахунок скорочення часу відгуку, збільшення пропускної здатності, але за мінімальними витратами.

Таблиця 3.4. Порівняння алгоритмів за ключовими показниками

Параметри / Алгоритми	Пропускна здатність	Відмовостійкість	Час відповіді	Накладні витрати	Використання ресурсів	Масштабованість	Продуктивність
Round Robin	Висока	Низька	Коротко	Ні	Низьке	Так	Висока
OLB	Низька	Низька	Довго	Так	Низьке	Ні	Висока
Min-Min	Висока	Низька	Коротко	Ні	Низьке	Ні	Висока
Max-Min	Висока	Низька	Коротко	Ні	Низьке	Ні	Висока
ALO	Висока	Низька	Довго	Ні	Низьке	Ні	Низька
Honey-Bee Foraging	Висока	Низька	Довго	Так	Низьке	Ні	Низька
Active Clustering	Низька	Низька	Коротко	Ні	Низьке	Ні	Низька
Throttled LB	Низька	Висока	Коротко	Так	Низьке	Так	Висока
ESCE	Низька	Низька	Довго	Так	Низьке	Так	Низька
Biased-Random	Низька	Низька	Довго	Ні	Низьке	Ні	Висока
Join-Idle Queue	Низька	Низька	Коротко	Ні	Значне	Ні	Висока

Як бачимо з таблиці 3.4, найоптимальнішим алгоритмом серед розглянутих з точки зору продуктивності системи та мінімального впливу на її компоненти є *Round Robin*. Він підтримує більшість протоколів прикладного рівня, а балансування ніяк не залежить від навантаження на сервер, тому він допоможе впоратися з будь-яким напливом запитів.

Проте, оскільки *Round Robin* абсолютно не враховує поточну завантаженість та кількість активних з'єднань того чи іншого сервера в пулі, подібна система не дуже відповідає критеріям відмовостійкості. Існує вдосконалена версія цього алгоритму – *Weighted Round Robin*. Суть удосконалень полягає в тому що кожному серверу присвоюється ваговий коефіцієнт відповідно до його потужності та наявних ресурсів. Це допомагає розподіляти навантаження більш гнучко.

Вищеописані алгоритми не враховують певні характеристики інфокомунікаційних систем та не знайшли широко практичного застосування (окрім *Round Robin*, що імплементовано майже в будь-якому балансирі навантаження). Розглянемо алгоритми що використовуються в пропріетарних рішеннях, таких як AWS Elastic Load Balancer та Google Cloud Load Balancer.

У випадку коли вузьким місцем в обчислювальному вузлі виступають не процесорні ресурси чи пам'ять, а відкриті TCP-з'єднання, доцільно використовувати алгоритм *Least Connections* (скорочено - leastconn). Він враховує кількість підключень, підтримуваних серверами в поточний момент часу. Кожен наступний запит передається серверу з найменшою кількістю активних підключень. Для використання в кластерах, що складаються з серверів з різними технічними характеристиками і різною продуктивністю, доцільно використовувати моифікований алгоритм Weighted Least Connections. Він враховує при розподілі навантаження не тільки кількість активних підключень, а й ваговий коефіцієнт кожного з серверів [54].

У випадку необхідності підтримувати довгострокові сесії доцільно використовувати інші модифікації, а саме алгоритми з групи *Locality-Based*

Least Connection Scheduling. В цих алгоритмах кожна клієнтська IP-адреса або певна група IP-адрес закріплюється за окремим очислювальним сервером або цілою групою серверів. Вхідний запит передається найменш завантаженому серверу з групи, а якщо всі сервери з визначеної групи перевантажені, то буде зарезервовано новий сервер, який додасться до групи.

Ще одним варіантом з підтримкою довготривалих сесій є алгоритм *Sticky Sessions*, що використовується в проксі-сервері Nginx. Цей метод гарантує, що запити від одного і того ж клієнта (IP-ареси) будуть передаватися на один і той же обчислювальний сервер, що є однією з ключових вимог сучасних Stateful веб-сервісів (таких що повинні зберігати стан клієнтських запитів протягом всієї сесії).

Для географічно розподіленої інфраструктури зазвичай використовують алгоритми Destination Hash Scheduling та Source Hash Scheduling. У цих алгоритмах сервер, що обробляє запит, вибирається з статичної таблиці за IP-адресою одержувача або відправника відповідно.

Висновки до розділу 3

Важливою частиною процесу проектування та експлуатації "хмарної" інфраструктури є балансування навантаження. Завдяки правильному вибору методів та алгоритмів розподілу навантаження можна досягти оптимізації використання ресурсів, скорочення часу обслуговування запитів, покращення масштабованості веб-додатку, а також забезпечити відмовостійкость та ефективність системи.

Для досягнення рівномірного розподілу обчислювального навантаження необхідно враховувати цілий ряд показників, як апаратних (наприклад, поточне завантаження вузла) так і функціональних (середня тривалість обробки запиту).

З цією метою були розглянуті наступні алгоритми: Round Robin; Opportunistic Load Balancing; Min-Min; Max-Min; Ant Colony Optimization; Honey-Bee Foraging; Active Clustering; Throttled Load Balancing; ESCE; Biased-

Random Sampling; Least Connections (звичайний та його модифікації – Weighted, Locality-Based Scheduling); Destination/Source Hash Scheduling; Sticky Sessions. На основі проведено аналізу було надано рекомендації по вибору алгоритмів в залежності від приорітетності тих чи інших критеріїв та вимог проектованої інфокомунікаційної системи.

Були розглянуті можливі конфігурації систем балансування – з єдиним пристроєм розподілу навантаження, кластерні рішення та конфігурації без єдиного пристрою балансування, проведено їх порівняння та описані переваги та недоліки кожного з рішень.

Також було описано методи балансування навантаження, що працюють на мережевому, транспортному та прикладному рівнях моделі OSI, а в пункті 3.3 виконано їх порівняння. Кожен з цих методів має як переваги, так і недоліки, критичність яких залежить від конкретних потреб окремо взятого веб-додатку. Зважаючи на те, що неіснує ідеального способу балансування навантаження хмарної системи, найкращим варіантом виглядає використання гібридних рішень, що будуть поєднувати переваги статичного та динамічного балансування та уникати недоліки кожного з них. Також доцільно поєднувати одразу декілька методів, використовуючи для різних компонентів хмарної інфраструктури різні підходи до розподілу навантаження. Наприклад, в якості основного методу буде використовуватись проксі-сервер прикладного рівня, що буде розподіляти вхідні запити до веб-сервісу в залежності від їх змісту між декількома групами серверів додатку (серверах в кожної з них будуть оптимізовані під обробку певного типу запитів, так званий мікросервісний підхід). Окремо, в кожній з цих груп буде налаштовано перерозподіл запитів по конкретним обчислювальним вузлам з використанням засобів HTTP Redirect (наприклад, комплексу Citrix NetScaler). Таким чином, ми отримаємо продуктивну, гнучку та відмовостійку багаторівневу систему, що ефективно працює на основі хмарної інфраструктури.

РОЗДІЛ 4. МЕТОДИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОЗПОДІЛУ НАВАНТАЖЕННЯ В ХМАРНОМУ СЕРЕДОВИЩІ

4.1 Метод консолідації віртуальних машин на основі алгоритму променевого пошуку

З метою більш ефективного використання апаратного забезпечення в центрах обробки даних (ЦОД) застосовуються технології віртуалізації. Сервісна модель IaaS дозволяє більш ефективно використовувати апаратне забезпечення фізичного сервера (ФС) за рахунок віртуалізації його локальних ресурсів. При цьому, клієнтові надається частина ресурсів ФС у вигляді віртуальної машини (ВМ). Для реалізації сучасних інформаційних послуг клієнт розгортає одну або декілька ВМ в певній конфігурації, яка визначена провайдером хмарних послуг. Таким чином, виникає комплекс задач, пов'язаних з управлінням віртуальними машинами, фізичними серверами, мережевою взаємодією, сховищами, застосуваннями та іншими системами [66].

Для роботи кожного екземпляру застосування створюється окрема ВМ або контейнер всередині ВМ. Виходячи з певних бізнес-потреб клієнт може динамічно змінювати конфігурацію ВМ або налаштовує механізми балансування навантаження, відмовостійкості та резервного копіювання. В процесі роботи хмарного сервісу клієнти створюють множину ВМ, яка постійно змінюється. Кількість ВМ, розміщених на окремому ФС може постійно змінюватися, тому деякі ФС виявляються незавантаженими до максимально можливого порогу і витрачають зайву енергію [66].

З метою більш ефективного використання апаратних ресурсів ФС засоби віртуалізації надають можливість міграції ВМ з одного ФС на інший. При цьому, вплив на роботу ВМ є мінімальним, і для клієнта міграція ВМ виявляється непомітною. Але навантаження на фізичні сервери, які

обмінюються цією ВМ, зростає.

Таким чином, однією з головних задач при управлінні ресурсами хмарного ЦОД є розміщення віртуальних машин таким чином, щоб задіяти меншу кількість фізичних серверів та зменшити кількість міграцій віртуальних машин. Процес перерозподілу віртуальних машин серед фізичних серверів називають також консолідацією ВМ.

4.1.1 Аналіз підходів до консолідації віртуальних машин

Останнім часом запропоновано багато підходів та алгоритмів для вирішення проблеми консолідації віртуальних машин [33, 34, 35]. Проблема консолідації віртуальних машин розглядається як оптимізаційна проблема з різними цільовими функціями. Особливість цієї проблеми полягає в наявності великої кількості станів середовища та обмежень, що накладаються. Для промислових ЦОД з хмарними інфраструктурами використовуються прості алгоритми управління, такі як first-fit, best-fit та їх модифікації (Eucalyptus, Microsoft [37], Google [36]).

Зазвичай використовуються такі цільові функції, як мінімізація споживання електроенергії, мінімізація порушень угод про якість сервісу (SLA), мінімізація мережевого трафіку, максимізація продуктивності та використання ресурсів. Однією з основних умов для сучасних кластерів ФС є можливість роботи алгоритму в режимі онлайн [34]. Разом з цим допускається застосування методів оптимізації, які спрацьовують при появі певних умов роботи кластеру. Для такого випадку нові завдання розміщуються на ФС, які не задіяні в процесі консолідації.

Крім традиційних евристик та детермінованих алгоритмів для вирішення проблеми консолідації ВМ використовуються і алгоритми локального пошуку, такі як еволюційні алгоритми, алгоритми оптимізації мурашиних колоній, табу пошук, пошук з емуляцією відпалу. На мою думку більш ефективним є використання алгоритму локального пошуку на другій стадії оптимізації, після підготовки відповідного набору станів

детермінованими алгоритмами з метою покращити рішення, знайдене на першій стадії.

Таким чином, пропонується застосувати локальний променевий пошук на підготовленому наборі даних з метою зменшити кількість міграцій ВМ та збільшити кількість вивільнених ФС, отриманих на попередній стадії оптимізації.

4.1.2. Модель системи

Хмарні ЦОД являють собою складні системи, що складаються з серверних підсистем, підсистем зберігання даних, мережевих підсистем та підсистем інженерного забезпечення.

Для побудови кластеру в сучасних ЦОД використовують два підходи компоновки: з використанням гетерогенної конфігурації або гомогенної конфігурації. Обидва підходи мають свої недоліки та переваги, однак уникнути розміщення гетерогенних конфігурацій в масштабі ЦОД, коли конфігурації кожного кластеру відрізняються, в більшості випадків не вдається з причин еволюції елементної бази серверів та нових вимог користувачів до IT-інфраструктури.

Розробка і дослідження роботи алгоритму променевого пошуку виконані для кластеру, який складається з фізичних серверів різної конфігурації. Кожен ФС надає для декількох ВМ декілька ресурсів: процесор (CPU), об’єм пам’яті (RAM), доступ до підсистеми зберігання даних (IOPS), доступ до мережевої підсистеми (NET) та інші. В залежності від наявної ємності ресурсів ФС, вимог до ресурсів з боку ВМ, часу виконання завдань всередині ВМ та інтенсивності надходження завдань кількість ВМ, що виконуються на ФС, постійно змінюється. Зміна кількості ВМ відбувається в таких станах: створення нової ВМ, видалення ВМ, міграція ВМ.

В запропонованому методі розглядаються ресурси CPU та RAM, що надаються для ВМ. Однак алгоритм може бути доповнений іншими ресурсами, які потребує ВМ. Вибір саме двох ресурсів обумовлено

використанням вхідних даних з набору Google cluster-usage traces (GCT) [38]. Для дослідження роботи алгоритму ПП з набору GCT використано 6000 ФС з таблиці "Machine events" та 70000 завдань з таблиці "Task events table" (вибраних випадково). З першої таблиці використано такі атрибути, як machine ID, CPU capacity, memory capacity. З таблиці "Task events table" для кожного ФС використано атрибути: task index within the job, machine ID, resource request for CPU cores, resource request for RAM. Дані в таблицях нормовані відносно ФС з найбільшим значенням ємності відповідного ресурсу.

4.1.3. Постановка задачі

Кластер управління складається з множини M фізичних серверів та N віртуальних машин, $N, M \in \mathbb{N}$. В процесі дослідження роботи алгоритму ПП кількість ФС та ВМ не змінюється. Кожне завдання з таблиці "Task events table" виконується в окремій ВМ. В загальному випадку, між запусками алгоритму ПП кількість ФС та ВМ може змінюватись.

Задана ємність j-ї ВМ для ресурсу k, $c_j^k \in (0,1]$, $k \in \{CPU, RAM\}$ визначається вимогами завдання і нормовано відносно ФС з найбільшою ємністю ресурсу k. Ємність фізичного сервера i для ресурсу k, $C_i^k \in (0,1]$ визначається типом ФС і нормовано відносно ФС з найбільшою ємністю ресурсу k.

Множина M складається з множини A фізичних серверів, які визначені для вимикання, та множини B фізичних серверів, які надають ресурси для ВМ, що будуть мігрувати з ФС, які належать до множини A, $A \cup B = M$, $A \cap B = \emptyset$.

Міграцію віртуальної машини j на фізичний сервер і позначимо як $U_{ij} \in \{0,1\}$. Міграція відбувається якщо $U_{ij} = 1$. Кожна ВМ має свій ID, що пов'язаний з номером ***j***. Кожний ФС теж має свій ID, що відображається в алгоритмі номером ***i***.

4.1.4. Метод консолідації ВМ на основі променевого пошуку

Опис основного алгоритму. Вхідними даними алгоритму ПП є список ФС з множини *A* та список ФС з множини *B*, в яких є вільні ресурси і є можливість розмістити додаткові завдання у вигляді ВМ.

Ідея алгоритму: перегляд *i*-го ФС зі списку *A* та пошук таких або такого ФС зі списку *B*, куди можливо мігрувати *j*-ту ВМ з *i*-го ФС. Якщо вдалося звільнити всі або частку ФС зі списку *A*, видаємо результат у вигляді матриці U_{ij} , яка є планом міграцій.

Опис алгоритму променевого пошуку:

Перша стадія: підготовка вхідних даних для другої стадії. Формування списку *A* фізичних серверів, які треба звільнити від ВМ, та списку *B* фізичних серверів для визначення плану міграцій.

Друга стадія виконується для кожного ФС зі списку *A*:

1. На кожному кроці обираємо одну ВМ, назначену на *i*-й ФС і розглядаємо наступні варіанти обміну (міграцій):
   a. мігрувати ВМ на інший ФС, в якого залишилось достатньо вільних ресурсів CPU та RAM;
   b. перевірити можливість обміну з іншим ФС віртуальною машиною, яка вимагає менше ресурсів (так ми розглядаємо лише стани з кращою оцінкою та уникаємо можливих зациклювань) і, в результаті, ФС не буде перевантаженим після обміну.
2. З усіх можливих обмінів обирається *n* (ширина променю) обмінів з найвищою оцінкою.
3. Завершуємо пошук якщо *i*-й ФС вдалося звільнити від віртуальних машин або якщо не можна побудувати нові стани (тобто не залишилось жодного варіанту для реалізації допустимого обміну a) або b)).

Порівняння станів виконується за допомогою критерію (1):

$$J = \sum_{i=1}^{m} u_i^2 + \sum_{i=1}^{n} f_i^2 \qquad (1)$$

де u_i – кількість використовуваних ресурсів на *i*-му ФС, f_i – кількість вільних ресурсів на *i*-му ФС, *m* – кількість ФС в списку *B*, *n* – кількість ФС в списку *A*.

Таким чином, алгоритм поступово зменшує використовувані ресурси на ФС, які належать до списку *A*, та завантажує ФС зі списку *B*.

Умови завершення алгоритму

Для завершення роботи алгоритму пропонуються наступні умови:

1. Вичерпання списку *A*.
2. Неможливість мігрувати всі ВМ з певної визначеної кількості ФС Th_A поспіль.

Якщо другу умову не застосовувати, цикли пошуку виконуються занадто довго, якщо співставити їх з часом на створення нової ВМ та часом на міграцію для ВМ з середніми вимогами до ресурсів пам'яті. Для визначення *ThA* пропонується застосувати евристику, яка полягає у розгляді певного відсотку фізичних серверів зі списку *A*, але не менше, ніж α фізичних серверів. В реалізації досліджуваного алгоритму прийнято $Th_A = 0.05$, $\alpha = 10$. З меншими значеннями α алгоритм пропускає значну кількість ФС, що могли бути вимкнуті, але в результаті завершення алгоритму були не звільнені від працюючих на них ВМ. Даний критерій завершення вводиться для зменшення часу виконання алгоритму.

Визначення вхідних даних (опис першої стадії роботи методу). Отримання списку фізичних серверів, з яких треба мігрувати всі ВМ з метою подальшого вимкнення ФС пропонується здійснювати за допомогою двох методик: *нижньої границі* та *порогу вільних ресурсів*. Розглянемо кожну з методик більш детально.

Методика нижньої границі полягає у визначенні такої кількості фізичних серверів, які вимкнути в результаті міграції усіх ВМ, що на них працює, не уявляється можливим. Таке уявлення виникає з гіпотез, що використовуються в роботі алгоритму визначення нижньої границі.

Пошук нижньої границі виконується наступним чином:

1. Знаходимо середній об'єм ресурсів по всім ФС, на яких працюють ВМ $\sum_{i=1}^{M} C_i^k$. Значення M_i для кожного ресурсу k розраховуються окремо.
2. По кожному ресурсу окремо рахуємо суму необхідних ресурсів для виконання всіх наявних ВМ, $\sum_{j=1}^{N} C_j^k$
3. Окремо по кожному ресурсу рахуємо відношення необхідних ресурсів до середнього об'єму ресурсів та округляємо значення до більшого цілого.
4. Нижньою границею буде найбільше число з отриманих на кроці 3.
5. Сортуємо фізичні сервери одним з наступних способів:
   a. за об'ємом ресурсів ФС, потім за відношенням використаних ресурсів до кількості працюючих ВМ;
   b. за об'ємом ресурсів ФС, потім за кількістю назначених завдань, потім відношенням використаних ресурсів до кількості працюючих ВМ.

В результаті досліджень роботи алгоритму з'ясувалося, що варіант b) виявився значно ефективнішим. При його використанні на одному й тому самому наборі даних вдалося вимкнути 82 ФС, тоді як при використанні варіанту а) вдалося вимкнути лише 33 ФС.

В результаті, знаходимо різницю між кількістю ФС, що використовуються, та отриманою нижньою границею. Знайдене число – це кількість ФС списку *A* на вимкнення, які є першими у відсортованому списку. Решта ФС потрапляє у список *B*.

Ідея методики порогу вільних ресурсів полягає в такому формуванні списку *A*, при якому до цього списку потрапляють ФС, загальна кількість невикористаних ресурсів яких перевищує об'єм ресурсів одного з ФС кластеру.

Побудова списку *A* з використанням порогу вільних ресурсів β виконується наступним чином:

1. Встановлюємо значення β .
2. Відбираємо ФС, у яких кожного вільного ресурсу більше за значення β
3. По кожному ресурсу окремо рахуємо суму вільних ресурсів.
4. Сортуємо обрані ФС аналогічно до варіанту b) з методики нижньої границі.
5. Проходимо по відсортованому списку. Поки сума ресурсів більша за об'єм поточного ФС віднімаємо від суми цей об'єм, додаємо поточний ФС в список *A* та переходимо до наступного ФС, інакше завершуємо проходження списку. Таким чином, отримуємо список *A* з ФС, які треба вимкнути, та список *B* з ФС для обміну віртуальними машинами.

4.1.5. Оцінка результатів моделювання

Дослідження роботи алгоритму виконане на фрагментах набору вхідних даних GCT [38]. Дані для тестування і дослідження алгоритму обрані з GCT для певного діапазону часу, за який збиралися дані на реальному кластері. Це необхідно, щоб врахувати всі дані за один і той самий період часу в журналі GCT. З набору випадковим чином відібрано 70000 завдань з певними вимогами до ресурсів k. Кожному завданню відповідатиме окрема ВМ. Обираємо 6000 фізичних серверів, з яких на 5832 серверах розміщено ВМ, та на 168 серверах завдань немає, але вони доступні для розміщення ВМ. Ширина променю для алгоритму дорівнює 5.

В даній роботі дослідження впливу ширини променю на роботу алгоритму не досліджувалося.

В таблиці 4.1 представлені результати роботи алгоритму променевого пошуку з консолідації ВМ для різних значень β. В таблиці 4.2. представлені результати роботи алгоритму променевого пошуку при консолідації ВМ з використанням методики нижньої границі. Якісними показниками роботи алгоритму є кількість вимкнених ФС та кількість міграцій ВМ, за допомогою

яких ФС вивільнюються від ВМ.

Таблиця 4.1. Результати моделювання консолідації ВМ алгоритмом променевого пошуку з використанням методики порогу вільних ресурсів

β	*B*	*A*	Вимкнено ФС	Міграцій	Час, с
0.005	199	3	0	0	0.53
0.004	299	3	0	0	0.69
0.003	1098	9	0	0	4.05
0.002	1393	10	0	0	6.55
0.001	1868	28	13	219	38.08
0.0009	1943	31	15	263	42.29
0.0008	2002	39	22	349	67.93
0.0007	2081	43	24	399	81.02
0.0006	2204	48	28	432	94.14
0.0005	2302	55	32	504	105.48
0.0004	2407	63	36	554	141.43
0.0003	2565	73	43	683	143.16
0.0002	2764	82	48	782	166.54
0.0001	3706	95	59	992	238.06
0.00005	4323	106	68	1199	330.18
0.00001	5095	116	75	1336	463.36
0	5328	125	84	1459	518.34

Таблиця 4.2. Результати моделювання консолідації ВМ алгоритмом променевого пошуку з використанням методики нижньої границі

B	*A*	Вимкнено ФС	Міграцій	Час, с
5707	125	82	1460	508.09

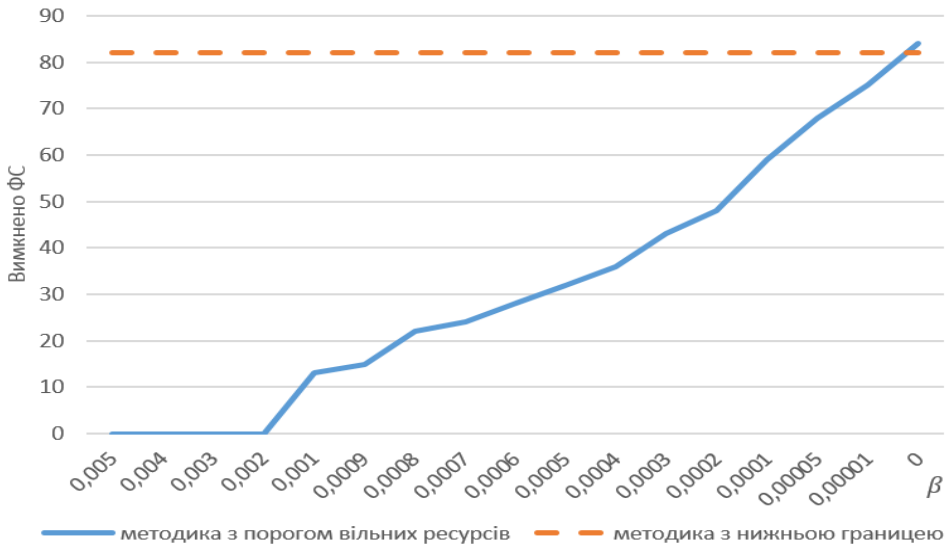

Рис. 4.1. Кількість вимкнених ФС

Вплив значення порогу β на якісні показники роботи алгоритму є суттєвим (рис. 4.1 та рис. 4.2), залежність виявилася майже лінійною. Чим менший поріг β тим більше ФС потрапляють на вхід алгоритму.

При β = 0 ФС у яких хоча б один ресурс повністю зайнятий не потрапляють на вхід алгоритму, тому що при перевірці наявності ресурсів використовується строга нерівність. При β < 0.0002 алгоритм з методикою порогу вільних ресурсів починає працювати більш ефективно і на граничних значеннях працює краще чим з методикою нижньої границі. Для всебічної перевірки роботи алгоритму з різними методиками та їх порівняння планується створити декілька наборів даних з різною кількістю ФС та ВМ.

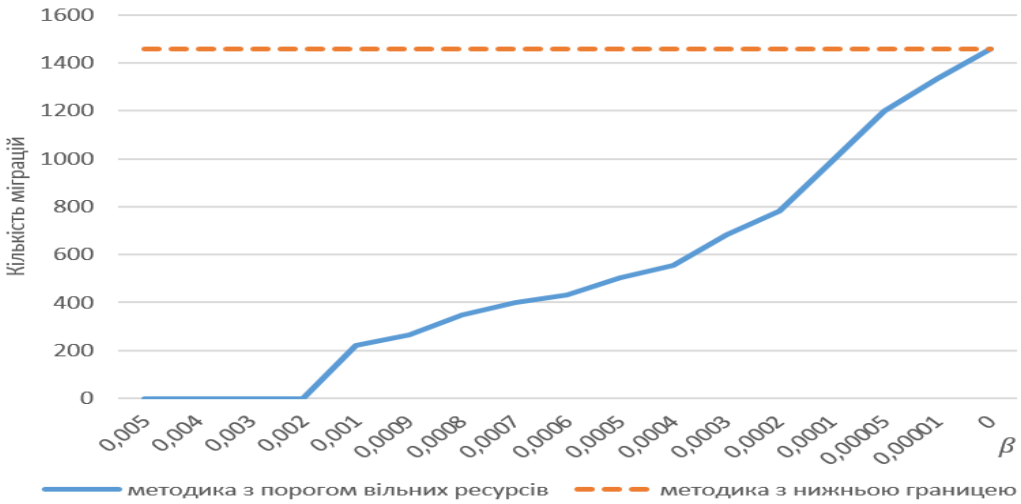

Рис. 4.2. Кількість міграцій

Перевагою використання методики з порогом вільних ресурсів є можливість адаптуватися до стану кластера, враховуючи інші ресурси, час міграції ВМ та інтенсивність надходження заявок на створення нових ВМ.

4.2. Спосіб динамічного балансування навантаження на основі алгоритму Least Connections

Підходи розподіленого програмування все частіше використовуються для створення розподілених обчислювальних систем шляхом розробки засобів передачі даних. Велике завдання ділиться на кілька менших підзавдань, що працюють на декількох комп'ютерах, що мають спільну мережу, і коли всі розрахунки зроблені, результати завдання можна використовувати для

обчислення початкового завдання. При вирішенні деяких проблем розподілені системи балансування використовуються для підвищення ефективності роботи системи в цілому [42]. Це означає зниження витрат, підвищення надійності, досягнення певного рівня продуктивності системи та простоту розширення.

Веб-служби зазвичай використовують архітектуру клієнт-сервер. У цьому випадку сервер - це веб-служба (кінцева точка URL-адреси), яка обробляє вхідні повідомлення від клієнтів і відповідає на них за допомогою певного протоколу (SOAP, WebSockets або REST API) [40].

Балансування навантаження в хмарному середовищі відрізняється від традиційної моделі архітектури балансування навантаження та реалізації серверами для виконання балансування навантаження, оскільки кількість запитів, надісланих на сервер, є непередбачуваною. Він пропонує нові функції, а також пропонує унікальний набір завдань. Балансування навантаження - одна з ключових проблем хмарних обчислень. Це механізм рівномірного розподілу динамічного локального навантаження по всіх вузлах у хмарі, щоб уникнути ситуацій, коли одні вузли перевантажені, а інші не працюють.

Оптимальним рішенням поставленого завдання може бути спосіб динамічного балансування навантаження на основі модернізованого алгоритму Weighted Least Connections, що використовує метод міграції задач, а саме алгоритм динамічного навантаження з урахуванням наступних параметрів: часу обробки завдання та середнього коефіцієнта використання ресурсів.

Загальна схема запропонованого способу представлена на рис. 4.3.

Планування виконання n завдань в m вузлах (віртуальних машинах) необхідно виконати так, щоб користувачі хмарного середовища могли виконувати свої завдання у мінімальні часи роботи з максимальними використаннями ресурсів.

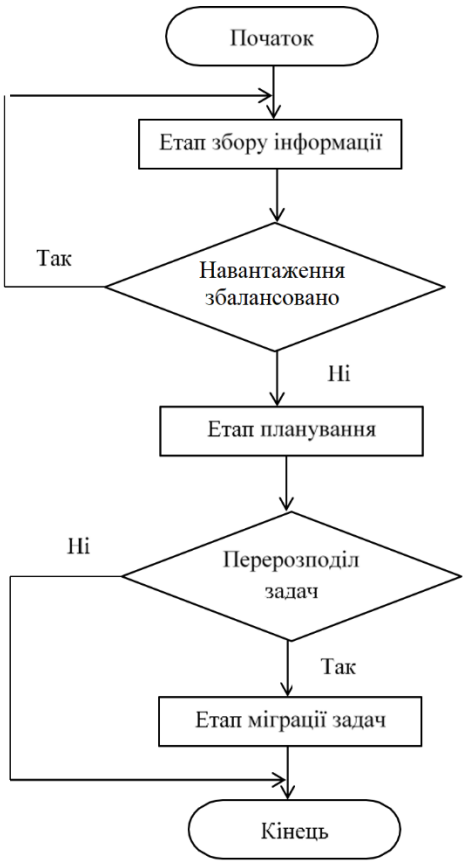

Рис. 4.3. Загальна схема запропонованого способу динамічного балансування навантаження

Планувальник задач отримує N запитів на завдання T_1,T_2 T_3…T_N. Всі завдання не являються пріоритетними та незалежними, кожне завдання має довжину T *L1*, швидкість обробки *p*, кількість процесорів *q*, обсяг основної пам'яті *r* та обов'язково смугу пропускання *B*. Планувальник завдань (рис.4.4.) в хмарному середовищі має інформацію про віртуальну машину M, а саме швидкість обробки процесора, кількість процесорів, пам'ять, пропускну спроможність VM_1,VM_2,VM_3,VM_4……VM_N [40].

Для обчислення потенціалу окремої віртуальної машини та ємності використовуємо формулу:

$$C_{VM} = p*q \qquad (2)$$

де:

p – швидкість обробки процесора в мільйонах інструкцій за секунду;

q – кількість процесорів зайнятих для виконання завдання.

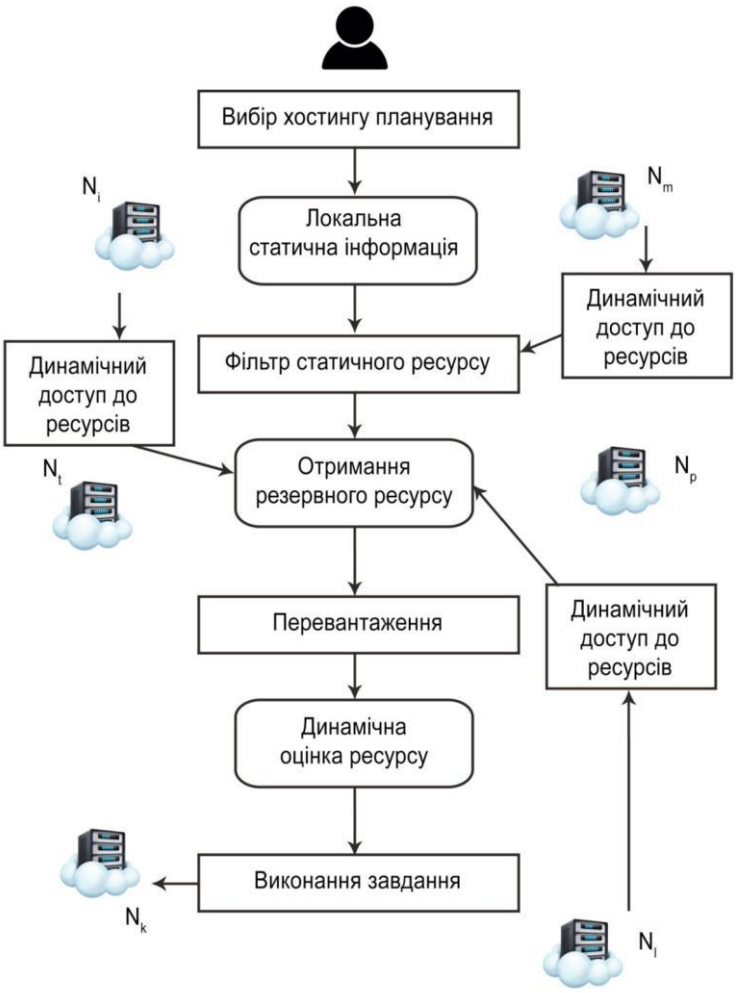

Рис. 4.4. Опис процесу планування завдань

Завантаження інформації на віртуальну машину: планувальник завдань із хмарного середовища розподіляє завдання на віртуальну машину, кожна віртуальна машина має чергу для зберігання навантаження. Загальна довжина черги у віртуальній машині визначається як навантаження на цю віртуальну машину [39].

Завантаження віртуальної машини можна розрахувати як

$$LVM_{i,t} = \frac{K*TL_{i(t)}}{S(VM_{i,t})} \quad (3)$$

де K=1,2,3….N завдання.

S(VMi,t) визначається як швидкість обслуговування віртуальної машини в момент часу *t*, який можна виразити у формі потужності *p* та кількості процесорів *q* як *p**x(*t*), де x=1,2,3…*q*.

Навантаження віртуальної машини за часом *t* обчислюємо як кількість задач на конкретній віртуальній машині, поділену на швидкість

обслуговування віртуальної машини.

Отже, загальне навантаження на всю віртуальну машину дорівнює

$$L = \sum_{j=1}^{M} LVM_j \quad (4)$$

Якщо загальне заплановане робоче навантаження системи перевищує її потенціал, то центр обробки даних не може обробляти всі майбутні запити, відмовляючись від майбутніх робочих запитів або збільшуючи віртуальну машину з концепцією еластичності [40]. Якщо майбутнє навантаження менше потенціалу всієї віртуальної машини, тоді кожна окрема віртуальна машина має навантаження і визначається, чи є перевантажена або завантажена віртуальна машина. Коли всі віртуальні машини перевантажені, завдання переносяться на завантажені віртуальні машини, щоб усі завдання могли бути виконані за мінімум часу.

Час передачі завдання може бути розрахований TT = довжина завдання (TL) / пропускна спроможність (B).

Знаходимо час виконання завдання T_i на віртуальній машині VM_j

$$T_{exej} = \frac{\sum_{i}^{N} E_{ij} * TL_i}{p * q} \quad (4)$$

де E_{ij} = 1, якщо завдання T_i призначене віртуальній машині VM_j, інакше значення = 0.

Необхідно визначити час, який складається з часу виконання та передачі завдання (за умови, що будь-яке завдання може переміщуватися з перевантаженої віртуальної машини до менш завантаженої).

$$MST = \max\{\sum_{j=1}^{M} T_{exej}\} \quad (5)$$

4.2.1. Метод міграції задач

Щоб перевірити можливість перенесення одного процесу або суперпроцесу, вам потрібна лише така інформація про один вузол [40]:

- кількість даних для обміну через зовнішній комунікаційний процес,
- кількість даних, якими обмінюється внутрішній (локальний) процес спілкування, де були розміщені дані
- найближчі сусіди, які беруть участь у вищезазначеному зовнішньому

спілкуванні.

Оскільки вся ця інформація вже відома вузлу в системі, реалізація алгоритму зважених найменших зв’язків (WLC) базується виключно на інформації доступній локально.

Алгоритм WLC не вимагає локальної інформації, яка ще не доступна або не може бути зібрана, і вона мало корисна. Прикладами такої інформації є характеристики навантаження вузлів / процесів, від яких алгоритм не залежить. Щоб алгоритм працював належним чином, кожен процес повинен знати, де його знайти. Має сенс припустити, що до цієї інформації можна легко отримати доступ для здійснення комунікаційного процесу або вона може повільно поширюватися, коли процеси взаємодіють між собою.

Використовується метод усереднення для відображення динамічних зміни в трафіку між двома процесами.

Стандартизація атрибуту ресурсу.

Серед рішень проблем багатоцільового атрибуту спосіб квантості атрибутів безпосередньо впливає на об'єктивну оцінку. У процесі оцінки ресурсів використовується кілька показників; кожен індикатор має іншу одиницю і розмірність. Щоб краще відображати фактичний стан ресурсів, кожен індикатор потрібно порівнювати за уніфікованими показниками; тому, перш ніж оцінювати кожен ресурс обчислювального вузла всебічно, нам потрібні вимірювання та стандартизація відповідних показників [40]. У запропонованому методі використовується метод середнього геометричного методу в методі стандартизації векторів для стандартизації атрибутів ресурсу, метод простий для розрахунку та вплив екстремального значення менше [42]. Спосіб полягає в наступному:

$$Z_{ij} = \frac{X_{ij}}{\sqrt{\Sigma (X_{ij})^2}} \tag{6}$$

Кожен з атрибутів ресурсу має власну одиницю вимірювання, використовуємо формулу 6 та стандартизуємо атрибути статичного ресурсу,

як показано нижче:

$$d_i = \frac{d_i}{\sqrt{\sum_{j=1}^{u} {d_j}^2}} \; c_i = \frac{c_i}{\sqrt{\sum_{j=1}^{u} {c_j}^2}} \tag{7}$$

У формулі 7. *p* означає частоту процесора, *k* означає число ядер, *m* – об'єм пам'яті, *d* – пропускна спроможність периферійного зберігання, а *c* – пропускна спроможність; *i*={1,2,…,u} означає всі доступні обчислюванні ресурси.

Кількісне розміщення динамічних атрибутів

Прогнозування перевантажень видаляє обчислювальні ресурси з великим навантаженням і визначає кількість найефективніших хостів у системі та обчислювальних вузлів із низьким навантаженням. Алгоритм використовує динамічну оцінку для вибору оптимального обчислювального вузла, який буде представлений у задачі. Використання вагового коефіцієнту в модифікованомі алгоритмі Least Connections відіграє ключову роль у представленому методі.

Перед динамічним оцінюванням потрібно стандартизувати динамічну інформацію обчислювального вузла. Оскільки кожен обчислювальний процесор може мати кілька ядер, процес індикаторів виглядає наступним чином:

$$p_i'(t) = \sum_{i=1}^{k_i} [(1 - p_i(t)) * p_i] \tag{8}$$

У формулі k_i означає основний номер хоста N_i.

Ми можемо знайти стандартизоване значення показників динамічного атрибуту одиночного обчислювального ресурсу, використовуючи формули(7) та (8), наступним чином:

$$\begin{gathered} \mathrm{p_i(t)} = \frac{\mathrm{p_i} * \mathrm{k_i}}{\sqrt{\sum_{j=1}^{g} (\mathrm{p_i'(t)})^2}} \; m_i(t) = \frac{\left(1 - m_i(t)\right) * m_i}{\sum_{j=1}^{g} [\left(1 - m_j(t)\right) * m_j]^2} \\ \mathrm{d_i(t)} = \frac{(1 - \mathrm{d_i(t)}) * \mathrm{d_i}}{\sqrt{\sum_{j=1}^{g} [\left(1 - \mathrm{d_j(t)}\right) * \mathrm{d_j}}^2} \; c_i(t) = \frac{(1 - c_i(t)) * c_i}{\sqrt{\sum_{j=1}^{g} [\left(1 - c_j(t)\right) * c_j]^2}} \end{gathered} \tag{9}$$

Завантаження інформаційної системи зворотного зв'язку.

Після фільтрування ресурсів відповідно до попиту та квантування їх статичних властивостей, використовуємо вікно перехоплення продукту *v* та стохастичні ваги, виключаючи низькопродуктивні обчислювальні вузли та виділяючи *l* більш ефективних ресурсів для резервування [42].

У процесі вибору запасного ресурсу коефіцієнт *y* безпосередньо впливає на розподіл ймовірності ресурсу. Коли *y* дорівнює 1, то ймовірність вибору сортованих обчислювальних ресурсів для резервування близька до 0; при зменшенні λ, ймовірність вибору відмінного обчислювального вузла буде збільшуватися. Наприклад, коли *v* беруться 4, λ прийнято 0.5, можна отримати вірогідність вибору чотирьох сортованих обчислювальних ресурсів відповідно:

$$a_1 = \frac{8}{15} a_2 = \frac{4}{15} a_3 = \frac{2}{15} a_4 = \frac{1}{15} \qquad (10)$$

У наведеному вище прикладі ймовірність ресурсу з найбільш відмінними статичними атрибутами становить більше 0,5.

4.2.2. Опис модифікованого алгоритму Weighted Least Connections.

Запропонований алгоритм є динамічним, оскільки він використовується кожним перевантаженим вузлом і управляється основним вузлом. Алгоритм WLC використовує “зважений” компонент на основі можливостей кожного сервера. Однак необхідно заздалегідь вказати або визначити «вагу» кожного сервера [41].

Балансир навантаження з WLC алгоритмом враховує такі параметри, як вага (ємність) кожного сервера та кількість клієнтів, які в даний час підключені до кожного сервера. Вибір системних вузлів для цього методу балансування навантаження базується на кількості активних з'єднань, тобто пропускній здатності сервера.

Розглянемо розрахунок часу обробки під час випробувань на навантаження. Після розподілу завдань віртуальним машинам відповідно до

алгоритму планування здійснюється моніторинг системи віртуальної машини. Як результат, ми можемо перевірити стан кожної віртуальної машини та визначити необхідність повторного балансування навантаження - передачі завдань із перевантаженої віртуальної машини на менш завантажену. Потрібно збільшити кількість і довжину повідомлень, щоб досягти максимальної оптимізації та визначити надійні результати. Граничне значення - це стан перевантаження всієї системи.

Середній коефіцієнт використання ресурсів (Average Resource Utilization Ratio, ARUR): основна мета розподілу навантаження полягає в тому, щоб рівень використання наявних ресурсів був максимально можливим [40]. Середній коефіцієнт використання ресурсів розраховуємо за формулою:

$$\text{ARUR} = \left(\frac{\text{середній час}}{\text{час обробки}}\right) * 100 \quad (11)$$

де середній час = Σ часу, витраченого ресурсом (VMj), щоб закінчити всю роботу ресурсу, де j={1,2,3,…M}.

Діапазон середнього коефіцієнта використання ресурсів становить від 0 до 1, максимальне значення для ARUR становить 1 (використання ресурсу становить 100%), найгірше – 0 (ресурс знаходиться в ідеальному стані).

4.2.3. Тестування запропонованого алгоритму та аналіз отриманих результатів

Опис системи під час тестування

Запущена симуляція обробляла дані більше двох годин з різною кількістю завдань (діапазон задач від 10 до 50) з довільною довжиною. Отримуємо результат, використовуючи загальну політику та функції cloudsim. При старті розглядаємо 5 віртуальних машин різної потужності обробки (від 200 до 1000 MIPS), діапазон пропускної спроможності віртуальної машини становив від 100 до 1000 Мбайт/с, кількість процесорів для кожної віртуальної машини становила 1, як показано в таблиці 4.3, і

враховує від 10 до 50 завдань з довжиною від 20000 MI до 400000 MI, як показано в таблиці 4.3.

Таблиця 4.3. Властивості задач

Task ID	Length	File size	Output file	№ of CPU
0	116228	300	300	1
1	52118	300	300	1
2	33734	300	300	1
3	67488	300	300	1
4	290543	300	300	1
5	60246	300	300	1
6	366852	300	300	1
7	73891	300	300	1
8	197914	300	300	1
9	54199	300	300	1

Таблиця 4.4 Властивості ВМ

VM ID	VM MIPS	VM image size	Memory	№ of CPU	VMM
0	1000	1000	512	1	Xen
1	800	1000	512	1	Xen
2	600	1000	256	1	Xen
3	400	1000	512	1	Xen
4	200	1000	256	1	Xen

Результати таблиці 4.3 і таблиці 4.4 представлені на наступних рисунках із результатами роботи при експериментальних запусках та виконанні завдань. Вісь *x* представляє кількість завдань та вісь *y* представляє час виконання завдання.

Завдання виділяється всій віртуальній машині за алгоритмом планування Weighted Least Connections, після чого починається моніторинг віртуальної машини і виявляється, що віртуальна машина 5 (VM ID 4) має перевантажений стан. Наступним кроком проводиться моніторинг віртуальних машин із найменшим показником стану завантаженості.

Завдання перевантаженої віртуальної машини мігрує для балансування

віртуальної системи, тому буде перенесене на віртуальну машину 1, а також додано час міграції завдання. Всі завдання будуть виконані за 505 секунд.

Налаштування системи

В ході роботи при отриманні результатів завдяки проведеним ряду експериментів було визначено, що іноді час Max-Min алгоритму близький до запропонованого методу з використанням алгоритму (WLC).

Для реалізації вищезгаданих функціональних можливостей CloudSim використовуються базові функції моделювання дискретних подій та моделювання основних компонентів хмарного обчислення. Оскільки об'єкти та компоненти CloudSim спілкуються через операції проходження повідомлення, основний рівень відповідає за керування події та обробку взаємодій між компонентами [42]. Основні класи ContainerCloudSimare зображені на рисунку 4.5.

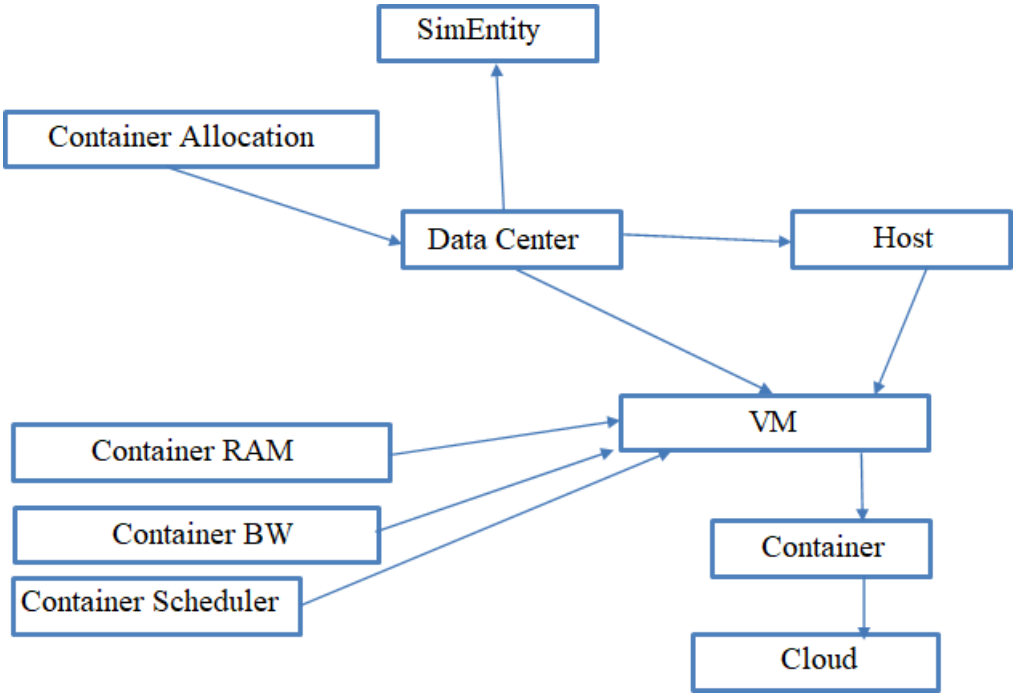

Рисунок 4.5 Діаграма ContainerCloudSimclass

У якості контейнерного хмарного симулятора ContainerCloudSim масштабується з мінімальним накладними витратами на пам'ять та часом виконання. Для того, щоб дослідити масштабованість розробленого симулятора, проведено експеримент, який був встановлений, розглядаючи

алгоритм Weighted Least Connections як політику первинного розміщення контейнера. Цей самий експеримент повторюється для різного числа контейнерів від 50 до 5000.

Для експерименту порівняння числа доступних хостів і віртуальних машин вважається постійним і дорівнює 700 і 1000, відповідно. Для кожного експерименту, що зображені на рисункках 4.6-4.8, показано час виконання симуляції, який визначається як час, необхідний для закінчення моделювання, та використання пам'яті програми Java (імітація).

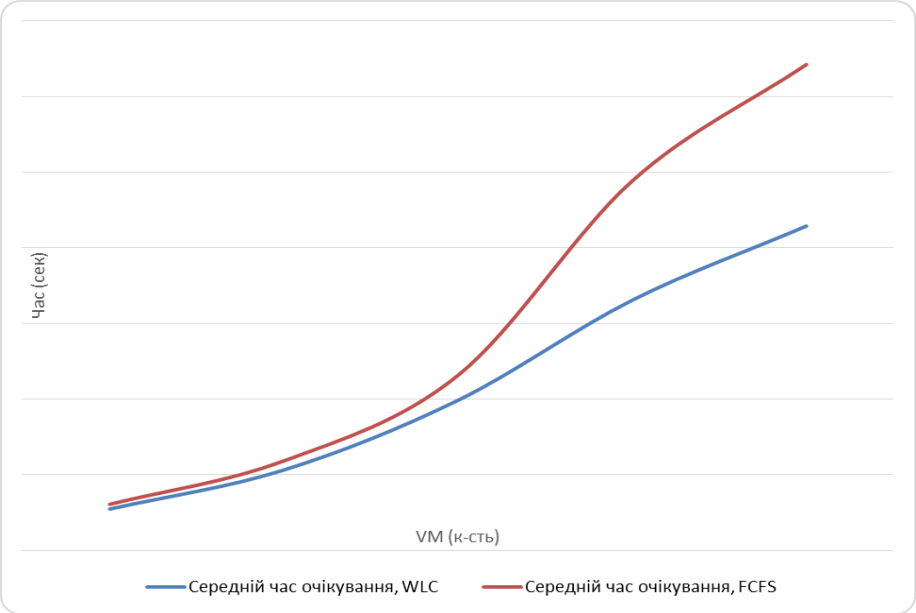

Рисунок 4.6. Використання алгоритмів WLC та FCFS

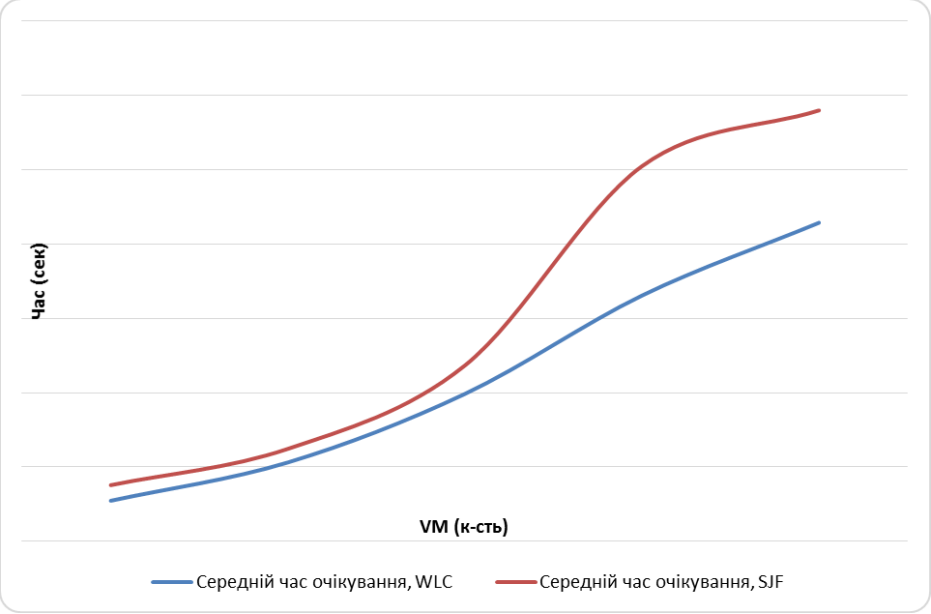

Рисунок 4.7 Використання алгоритмів WLC та SJF

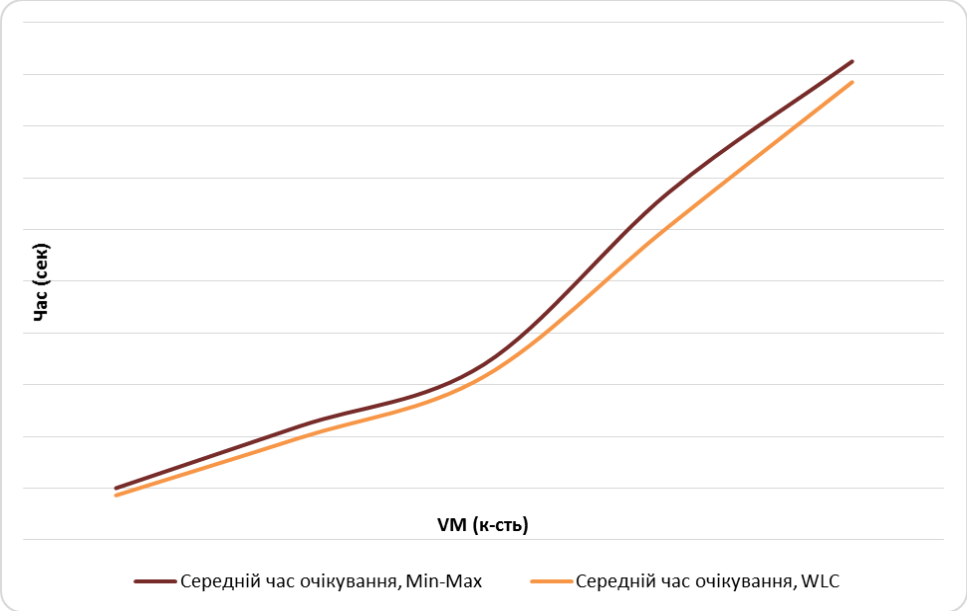

Рисунок 4.8 Використання алгоритмів WLC та Min-Max

Оскільки алгоритм Max-Min дотримується евристичного підходу, а його коефіцієнт використання ресурсів є низьким у порівнянні з запропонованою WLC (завдання не виділяється на віртуальну машину 5). Ми збільшуємо кількість завдань до 50 з різною довжиною і застосовуємо один і той же алгоритм до завдання, результати показані у наступному підрозділі.

Важливою операцією в середовищах хмарних обчислень є інсталяція віртуальних машин. Цей час є незначним і може вплинути на ефективність додатків, що працюють у хмарах. На основі дослідження поточна версія симулятора включає статичну затримку 100 секунд для кожної віртуальної машини. Також важливим компонентом стала затримка запуску контейнерів, оскільки жива міграція контейнерів не застосовна в реальних сценаріях. Тому міграція контейнера виконується шляхом вимкнення контейнера на вихідному хості, як тільки починається той самий контейнер у головному сервері. Аналогічно VM, затримка запуску контейнера може бути встановлена як константа у поточній версії CloudSim.

Результати запропонованого способу

Технологія віртуалізації є однією з основних концепцій інфраструктури Cloud Computing. CloudSim надзвичайно розгортає технології віртуалізації,

щоб імітувати постачання IaaS та PaaS та використовувати її як базу для виконання програм користувачів. З тієї ж точки зору виникає завдання розгортання найкращого розподілу ресурсів та алгоритму планування. Наприклад, один центр обробки даних, який складається з одного хосту з двома процесорами, а хмарний користувач намагається виконати інстанцію двох віртуальних машин з двома обробними одиницями кожна. Далі на рисунку 4.9. представлено серію експериментальних даних, які показують залежність часу обробки задач від алгоритму балансування кількість виконаних задач (також представлено результати запропонованого способу).

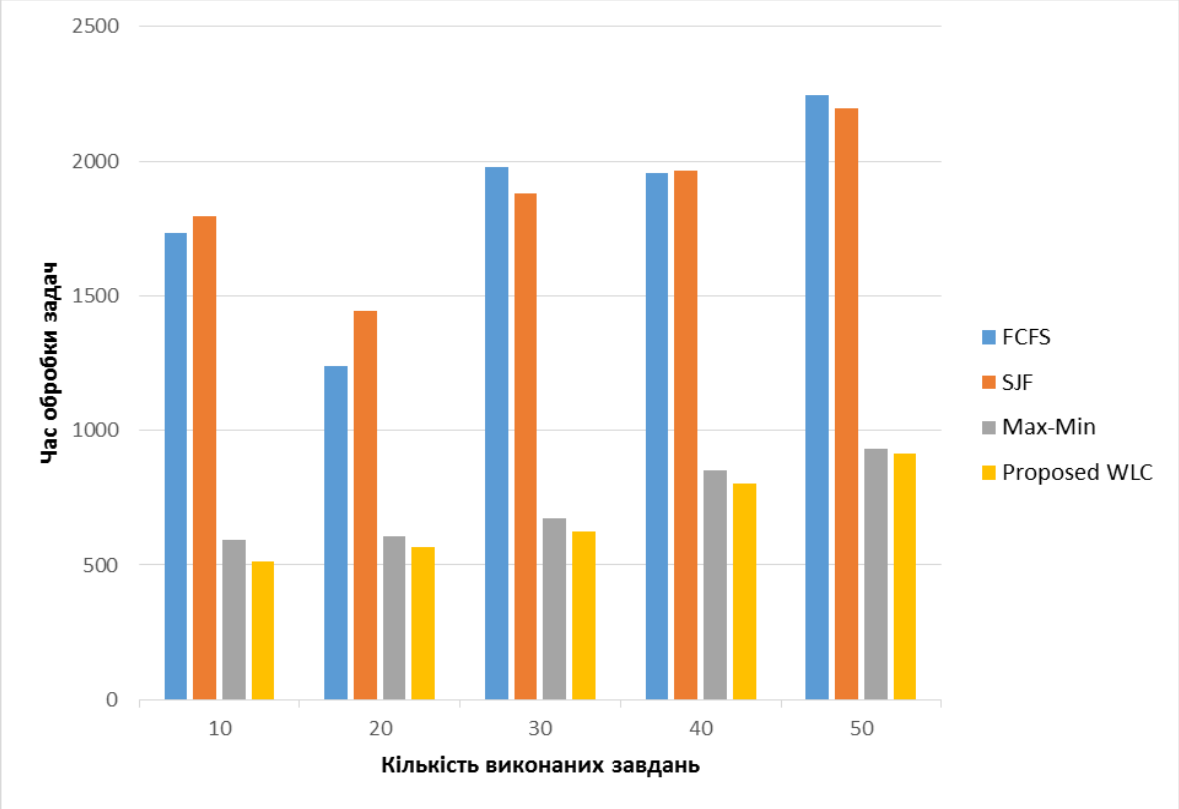

Рисунок 4.9 Результати часового проміжку обробки завдань із застосуванням алгоритмів при VM=5.

Збільшуємо кількість завдань від 10 до 20 і випадково вибираємо довжину завдання від 20000 до 200000 MI. Ми зменшуємо довжину завдання, тому що, якщо ми створимо 20 завдань довжиною від 20K до 400K і виділимо на 5 існуючих віртуальних машин, то вся віртуальна машина буде перевантажена, і планувальник задач потребує завантаження менш

навантаженої віртуальної машини. Тому для балансуванням навантаження, тому зменшуємо довжину завдання.

В ході експериментальної роботи було обчислено середнє співвідношення використання ресурсів запропонованого алгоритму за допомогою рівняння 4.11, а результати було порівняно з існуючими алгоритмами. Також отримані результати були згруповані та занесені до таблиці 2 та таблиці 3 для ресурсу та кількості завдань, після було використано для отримання результатів виконання задач при навантаженні системи при різних умовах.

Було застосовано запропонований алгоритм балансування навантаження для знаходження середнього коефіцієнта використання ресурсів. Графік представляє собою результати запропонованого алгоритму балансування навантаження, що виділяє завдання на віртуальну машину таким чином, що не з'являються віртуальні машини в перевантаженому або завантаженому станах. Розрахований ARUR пропонованого алгоритму балансування навантаження становить 73%. Ми застосовуємо алгоритм Min-Min з однаковим значенням даних і отримуємо 50,8% ARUR, оскільки ресурс R3 та R4 в режимі очікування. Дані кожного з алгоритмів наведені у таблиці 4.5 для VM = 5.

Таблиця 4.5 Результати виконання завдань VM = 5

	10	**20**	**30**	**40**	**50**
CFS	1734	1238	1977	1958	2246
SJF	1795	1446	1881	1966	2196
MAX-MIN	594	605	672	854	934
Proposed WLC	515	569	625	803	912

Логічно, що існує розділення між двома віртуальними машинами, однак насправді кожна віртуальна машина обмежується потужністю обробки, яку пропонує фізичний вузол, тому неможливо одночасно ініціалізувати обидві віртуальні машини на тому ж хості без відповідного алгоритму планування. Дані при збільшенні VM для кожного відповідного алгоритму

наведені у таблиці 4.6.

Таблиця 4.6 Результати виконання завдань VM = 10

	10	**20**	**30**	**40**	**50**
FCFS	2900	2952	5321	6312	7456
SJF	2330	3456	5073	5623	6022
MAX-MIN	1830	1935	1669	1860	1900
Proposed WLC	1190	1017	1145	1234	1389

Експеримент повторювався з різними комбінаціями віртуальних машин (від 5 до 10), з складною математичною операцією, яка виконується для кожної хмари як навантаження, або без неї. Розподілені об'єкти та розподілене виконання контролювалися центром управління. Об'єкти рівномірно розподілені серед доступних екземплярів, споживаючи практично таку ж кількість вхідної пам'яті з кожного екземпляра. Перегородки різних екземплярів однаково виражені або доступні. Це показує ефективний розподіл розподілених об'єктів.

Виявлено чотири окремі випадки масштабованості, як описано нижче.

Динамічне масштабування: за умови використання динамічного масштабування, цей випадок представляє найбільше прикладів виконання, оскільки навантаження найвище. Пам'ять, яка використовується програмою у відсотках від загальної використовуваної пам'яті, використовувалася як міра моніторингу стану системи. З адаптивним масштабуванням середовище 5 VM і 10 перевантаження системи збільшено до 3 разів.

Зниження максимального порогу змусило основний шар масштабувати в першу чергу та швидше, залучаючи всі доступні приклади до моделювання. Час виконання збільшується, оскільки додається більше вузлів до хмарної системи. Отже, введення додаткових вузлів за межі певної максимальної кількості вузлів може бути економічно недоцільним, і в певному пункті це може стати мірою, яка пояснюється нижче як загальна справа. Далі на рисунку 4.4.10 представлено серію експериментальних даних, які показують залежність часу обробки задач від алгоритму балансування кількість

виконаних задач (також представлено результати запропонованого способу), але вже на VM=10.

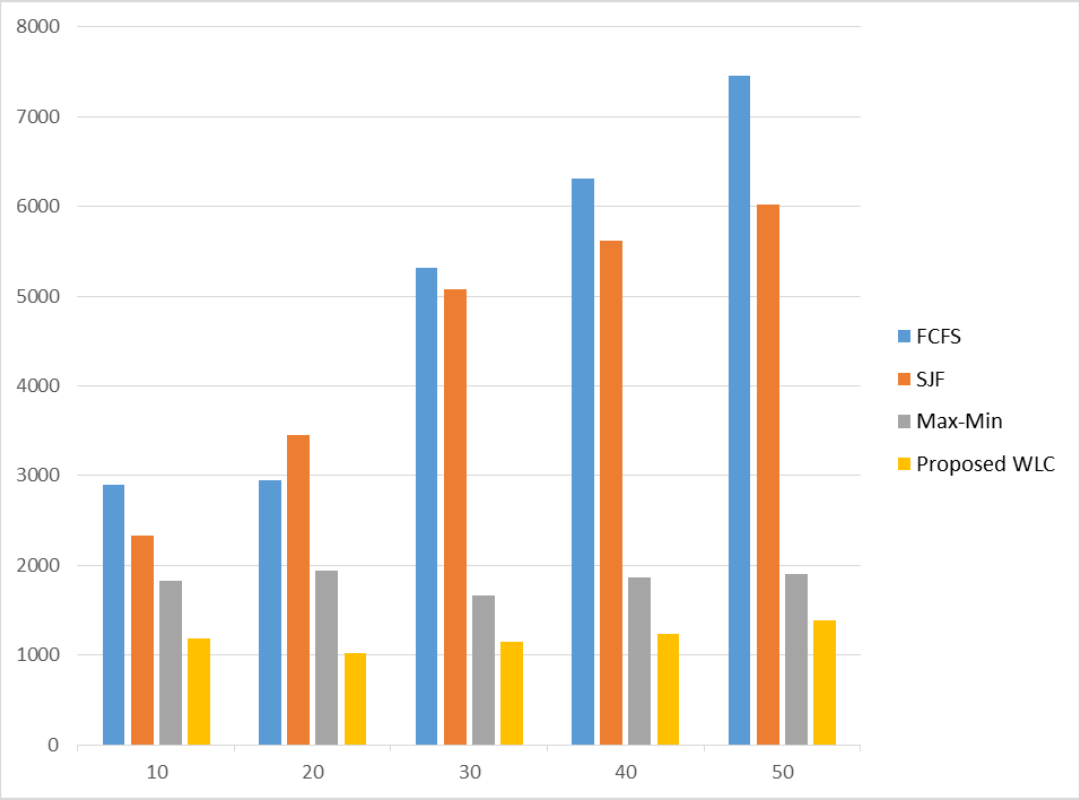

Рисунок 4.10. Результати часового проміжку обробки завдань із застосуванням алгоритмів VM=10.

Висновки до розділу 4

В даному підрозділі було визначено 2 методи, що зосереджуються на збільшенні навантаження, щоб задовольнити вимоги користувача для продовження сервісу шляхом розподілу робочого навантаження в балансовому порядку, щоб отримати максимальне використання ресурсів та зменшити час простою системи.

Для управління процесом консолідації віртуальних машин хмарного центру обробки даних запропоновано використовувати двостадійний метод на основі алгоритму локального променевого пошуку. Розроблені евристики першої та другої стадій роботи алгоритму, розроблений алгоритм променевого пошуку для консолідації ВМ, розроблені функція оцінювання та умови закінчення роботи алгоритму. Для перевірки роботи

методу використані дані про надходження завдань в кластер Google. методики нижньої границі та порогу вільних ресурсів, виконано їх порівняльний аналіз. В результаті пропонується використовувати методику з порогом вільних ресурсів, що показала більш якісні результати консолідації ВМ. Запропонований метод дозволяє вимкнути майже 50 відсотків фізичних серверів, потенційно визначених для вимикання, за рахунок виконання допустимої кількості міграцій віртуальних машин.

Також представлений метод динамічного балансування, який має за основу метод міграції задач та модернізований алгоритм Weighted Least Connections, враховує час проходження запитів та середній коефіцієнт використання ресурсів як параметри. Проведено експериментальні дослідження по роботі даного способу, який показав ефективність в порівнянні з інтегрованими в CloudSim алгоритмами. У ході експериментальних досліджень було виявлено зменшення часу обробки завдання та збільшення середнього коефіцієнта використання ресурсів. Час обробки завдань на 5-ти віртуальних машин при 10, 20 та 30 завдань зменшився від 30% до 10% (у порівнянні з алгоритмом FCFS та з алгоритмом Max-Min відповідно).

РОЗДІЛ 5. РОЗРОБКА СТАРТАП ПРОЕКТУ

В даному розділі приведено економічний аналіз перспектив впровадження досліджуваних методів підвищення ефективності способів балансування навантаження, а також оцінено можливості їх ринкового впровадження.

5.1 Опис ідеї стартап проекту

Стартап проект направлений на підвищення ефективності існуючих рішень для балансування навантаження задля отримання кращих показників ефективності систем та мінімізації витрат на віртуальні серверні ресурси при дотриманні бізнес-вимог по доступності та стабільності інфокомунікаційної системи та таких як пропускна здатність, середня завантаженість обчислювальних вузлів та середній час відповіді на вхідний запит.

Таблиця 5.1. Опис ідеї проекту

Зміст ідеї	Напрямки застосування	Вигода для користувача
Балансувач навантаження що використовує метод консолідації віртуальних машин на основі алгоритму променевого пошуку.	1. Зменшення кількості обчислювальних вузлів. 2. Зменшення точок відмови системи. 3. Підвищення параметрів доступності та масштабованості системи.	Оператори хмарної системи зможуть зменшити економічні витрати на обчислювальні ресурси без негативного впливу на рівень надання послуг клієнтам. Система стане більш гнучкою та масштабовною.

Таблиця 5.2. Технологічна здійсненність проекту

N п/	Ідея проекту	Методи реалізації	Можливості реалізації	Доступність реалізації
1	Імплементація підсистеми розподілу вхідних запитів на основі алгоритму променевого пошуку.	Реалізація у вигляді Lua-модулю для nginx	Існуючий тип HTTP-модулів – load balancers та можливості кастомізації через інтерфейс balancer_by_lua	Є доступним та безкоштовним для використання
2		Скрипт для автоматичної модифікації конфігурації HAProxy	Наявність Runtime API для динамічного конфігурування HAProxy	Є доступним та безкоштовним для використання
Обрана технологія реалізації проекту: Lua-модуль для nginx				

5.2. Аналіз ринкових можливостей

При дослідженні можливостей ринку, в першу чергу аналізовано попит: наявність попиту, обсяг, динаміка розвитку ринку. Дані приведені у таблицях нижче.

Таблиця 5.3. Попередня характеристика потенційного ринку стартап-проекту

No п/п	Показники стану ринку	Характер стану
1	Динаміка трафіку в хмарних сервісах (якісна оцінка)	Зростає

2	Доступні для вибору алгоритми балансування в готових рішеннях	Обмежені виробником, не розвиваються
3	Наявність обмежень для входу	Немає
4	Специфічні вимоги для стандартизації, специфікації	Немає

Враховуючи сьогоднішню необхідність ринку рішень стосовно підвищення ефективності систем балансування навантаження, за попереднім оцінюванням ринок є привабливим для входження.

Таблиця 5.4. Характеристика потенційних клієнтів стартап проекту

No п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Хаотичний характер трафіку в хмарних інфокому-нікаційних системах	Хмарні провайдери, оператори хмарних веб-сервісів, приватні компанії з існуючою хмарною інфраструктурою	Хмарні провайдери можуть визначати самостійно формат взаємодії та ввести свої коригування. Приватні компанії можуть мати відмінні потреби в доступних функціях	Збереження або покращення рівня надання послуг Зменшення витрат за рахунок більш ефективного масштабування системи

Таблиця 5.5. Фактори загроз

No п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Відсутність зацікавленості у продукті	Успіх застосування пропонованого методу залежить від бажання хмарних провайдерів модифікувати свої поточні рішення, оскільки внесення змін до ядра логіки прийняття рішень може привести до неочікуваних наслідків та негативного впливу на вже існуючі інсталяції. Низька популярність кінцевого рішення серед користувачів хмарних обчислень.	Глибоке тестуванню впливу внесених в логіку роботи підсистеми змін наперед погоджене з провайдерами. Маркетингова компанія по просуванню нового рішенння через профільні конференції та канали зв'язку.
2	Складність технічної реалізації	Інтеграція запропонованого методу в існуючі балансувальники навантаження може бути ускладнена пропрієтарними API або вузькоспецифічними рішеннями, що потребують зворотної сумісності.	Ітеративне впровадження компонентів модифікованої системи, узгодження плану розробки і релізу з провайдерами.

Висновки до розділу 5

В даному розділі приведені можливості висування методу консолідації віртуальних машин в хмарній системі, заснованого на алгоритмі променевого

пошуку, на ринок систем балансування навантаження та оцінювання можливостей ринкового впровадження даного методу. Конкурентна ситуація на ринку дозволяє розглянути запропоноване рішення як економічно ефективне.

Існуючі рішення-аналоги не створюють прямої конкуренції на ринку, оскільки не мають . Основною проблемою є ймовірна складність розробки готового рішення на основі наведених методів та можливість появи сильних конкурентів в майбутньому.

Проведений аналіз підтверджує, що подальша імплементація проекту є доцільною.

ВИСНОВКИ

В даній роботі було виконано систематизацію і порівняльну характеристику основних методів і алгоритмів балансування навантаження та методів масштабування додатків в "хмарному" середовищі та надано відповідні практичні рекомендації.

Було проведено загальний огляд технології хмарних обчислень, визначені основні види хмарних систем, їх моделі обслуговування та моделі розгортання. Описано такі ключові аспекти хмарних систем як масштабованість та балансування навантаження. Показана важливість систем розподілення навантаження в хмарних обчисленнях для телекомунікаціної галузі. Описані основні способи розподілу навантаження, що використовуються в різних типах хмарних систем. Проаналізовані їх недоліки та переваги, що є передумовою для створення нових рішень в області балансування навантаження.

При розгляді способів балансуванні навантаження «хмарних» додатків були враховані особливості роботи додатків, особливості інфраструктури обчислювальних комплексів та додаткові вимоги, обумовлені принципами роботи веб-сервісів.

Були виділені наступні класи рішень, що використовуються при побудові багатовузлових розподілених систем: з єдиними балансиром; балансування засобами NLB-кластера; балансування без пропуску трафіку через один пристрій балансування. Визначені основні переваги та недоліки кожної з цих конфігурацій та виконано їх порівняння з урахуванням особливостей хмарних сервісів.

В ході роботи були проаналізовані типові алгоритми та методи балансування навантаження. Описані принципи роботи наступних алгоритмів: Round Robin; Opportunistic Load Balancing; Min-Min; Max-Min; Ant Colony Optimization; Honey-Bee Foraging; Active Clustering; Throttled Load

Balancing; ESCE; Biased-Random Sampling; Least Connections (звичайний та його модифікації – Weighted, Locality-Based Scheduling); Destination/Source Hash Scheduling; Sticky Sessions. На основі проведено аналізу було надано рекомендації по вибору кінцевого алгоритму в залежності від поставлених вимог проектованої інфокомунікаційної системи.

Розглядались методи, що працюють на трьох рівнях моделі OSI:

- мережевому (DNS-балансування, побудова NLB-кластеру, балансування за IP-адресою засобами вхідного шлюзу та балансування засобами маршрутизаторів глобальної мережі),
- транспортному (метод NAPT та проксі-сервери 4-го рівня OSI),
- прикладному (проксінг та HTTP Redirect).

Були визначені переваги та недоліки, спільні для всіх алгоритмів та методів на кожному з рівнів, та проведено аналіз описаних способів балансування навантаження.

На основі проведеного деталізованого аналітичного огляду технології хмарних обчислень показано, що споживачі хмарних обчислень можуть значно зменшити витрати на інфраструктуру інфокомунікаційних систем (в короткостроковому і середньостроковому планах) і гнучко реагувати на зміни обчислювальних потреб, використовуючи властивості обчислювальної еластичності хмарних послуг.

В ході роботи було обґрунтовано вибір конфігурацій та способів реалізації системи балансування навантаження для розгалуженої інфраструктури "хмарного" додатку в залежності від критеріїв, що встановлюються при проектуванні веб-сервісу. Надано практичні рекомендації щодо вибору алгоритмів та методів розподілу навантаження хмарних веб-додатків.

Запропоновано 2 методи підвищення ефективності розподілу навантаження: метод консолідації віртуальних мащин на основі алгоритму променевого пошуку та модифікований алгоритм Weighted Lest Connections

на основі методу міграції задач. Було обгрунтовано доцільність використання цих методів та проведено маркетинговий аналіз перспектив реалізації системи розподілу навантаження на основі запропонованих способів та проведено оцінювання можливостей її ринкового впровадження.

Результати даних досліджень можуть бути використані для проектуванні систем балансування навантаження для створенні "хмарних" інфокомунікаційних систем або впровадженню публічних «хмарних» сервісів (як приклад, сервіси дистанційного навчання та відео-конференцій).

ПЕРЕЛІК ПОСИЛАНЬ

1. Mell, Peter. The NIST Definition of Cloud Computing / Mell, Peter and Grance, Timothy // Recommendations of the National Institute of Standards and Technology. NIST, 2011 – https://csrc.nist.gov/publications/detail/sp/800-145/final
2. Carl Hewitt. ORGs for Scalable, Robust, Privacy-Friendly Client Cloud Computing // IEEE Internet Computing, September/October 2008, pp. 96-99, vol. 12.
3. Риз Дж., Облачные вычисления: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 288 с.: ил.
4. Hesham El-Rewini and Mostafa Abd-El-Barr (April 2005). Advanced Computer Architecture and Parallel Processing. John Wiley & Sons. p. 66. ISBN 978-0-471-47839-3.
5. IBM Redbook: The RS/6000 SP Inside Out, id: SG24-5374-00, стр.15
6. Масштабирование нагрузки web-приложений. – http://habrahabr.ru/post/113992/
7. Системы балансировки нагрузки Web-серверов. – http://citforum.ru/internet/webservers/websbal.shtml
8. Балансировка нагрузки в распределенных системах. – http://www.intuit.ru/studies/courses/1146/238/lecture/6153
9. Балансировка в облаках. – http://www.osp.ru/os/2011/09/13011562/
10. Балансировка нагрузки в распределенных системах. – http://www.intuit.ru/studies/courses/1146/238/lecture/6153?page=2
11. Балансировка нагрузки: основные алгоритмы и методы. – http://habrahabr.ru/company/selectel/blog/250201/
12. DNS балансировка. –http://ruhighload.com/post/DNS+балансировка+
13. Що таке балансування навантаження? Огляд балансування мережного навантаження. – https://svetvtebe.ru/uk/what-is-load-balancing-network-load-balancing-overview/

14. A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches / H. T. Dinhet al. // Wireless Communications and Mobile Computing, 2011.
15. Кравчук С.О., Міночкін. Д.А., Рибак О.О. МОЖЛИВОСТІ ЗАСТОСУВАННЯ «ХМАРНИХ» ТЕХНОЛОГІЙ В ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМАХ. / Збірник матеріалів Міжнародної науково-технічної конференції «ПЕРСПЕКТИВИ ТЕЛЕКОМУНІКАЦІЙ» – 2016. – С. 2–3 – http://conferenc.its.kpi.ua/proc/article/download/70930/66101 (Режим доступу)
16. CloudIQ: A Framework for Processing Base Stations in a Data Center / S. Bhaumik et al. // Proc.ACM Mobicom'12, (Istanbul, Turkey), 2012.
17. Сейдаметова З. С.Обучение облачным технологиям инженеров-программистов / З. С. Сейдаметова, Г. С. Сейдаметов // Інформаційні технології в освіті. – № 15. – 2013. – С. 74-82.
18. Above the Clouds: A Berkeley View of Cloud Computing. Berkeley EECS. February 10, 2009. – http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf
19. Wu, Qiang (August 8, 2014). Making Facebook's software infrastructure more energy efficient with Autoscale. Facebook Code Blog. – https://code.facebook.com/posts/816473015039157
20. Orzell, Greg; Becker, Justin (January 18, 2012). Auto Scaling in the Amazon Cloud. Netflix Tech Blog. – http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html
21. Autoscaling: How the Cloud Provides a Tremendous Boost. Morpheus. November 2, 2016. – https://www.morpheusdata.com/blog/2016-11-02-autoscaling-how-the-cloud-provides-a-tremendous-boost
22. Kameda Hisao. Optimal Load Balancing in Distributed Computer Systems / Hisao Kameda, Li Lie, Kim Chonggun, Zhang Yongbing. – Springer, Verlag London Limited.- London, 1997. – P. 238.

23. Bohdan Borovskyi. A Comparison between Open-Source Platforms for Cloud Computing / XIII International Scientific Conference "Modern Challenges in Telecommunications" MCT-2019. / Conference proceedings. Kyiv. Igor Sikorsky Kyiv Polytechnic Institute, 2019 – p. 377. / ISSN (print) 2663-502X
24. Bohdan Borovskyi. Load Balancing Algorithms and challenges in cloud computing. / 7th International Scientific and Practical Internet Conference “Modern Movement of Science” June 6-7, Dnipro / International Electronic Scientific and Practical Journal "WayScience" – 2019 – №1.
25. Zhang Min, Chen Yun-hai. Research and development of cloud computing for telecommunication operators. Business and Operations, 2010-12
26. Iang Wei. Telecom operators cloud computing development strategy analysis. Telecommunications Technology, Industry and Information Technology Ministry of Telecommunications Institute 2011
27. Shanghai Research Institute, China Telecom Co., Ltd., Shanghai 200122, China; China Telecom Co., Ltd., Sichuan Telecom Co., Ltd., Company Chengdu 610021
28. LIU Guo-hong. Research on cloud computing application of telecom operators in China. Mobile Communications. China Mobile Communications Group Shaanxi Co., Ltd. 2009-5
29. Yu Yulan, Wei Min, Xiao Yi. The cloud computing platform and application of telecommunication operators with core capability as the core. Telecommunication Science 2010.
30. Cai Y., Yu F. R., Bu S. Cloud Computing Meets Mobile Wireless Communications in Next Generation Cellular Networks // IEEE Network. – 2014. – № 9/10. – P. 1–6.
31. Иванисенко И.Н. Анализ методов балансировки нагрузки в распределённых системах / И.Н. Иванисенко, Т.А. Радивилова // Сучасні напрямки розвитку інформаційно-комунікаційних технологій та засобів управління: П’ята міжнар. наук.-техн. конф., 23 – 25 квіт. 2015: матер. конф. – Полтава – Баку – Кіровоград – Харків, Україна. – С. 20-21 - http://www.foibg.com/ijitk/ijitk-vol09/ijitk09-04-p04.pdf (Режим доступу)

32. Бабенко О.А. Балансування навантаження "хмарних" додатків / Бабенко О.А., Гіоргізова-Гай В.Ш. // Системний аналіз та інформаційні технології: материалы 17-й Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-25 червня 2015 р. – 2015. – с.184 – http://cad.kpi.ua/attachments/043_2017_03_gg.pdf (Режим доступу)
33. F. Lopez Pires and B. Baran, “A virtual machine placement taxonomy,” in Proc. of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015, pp. 159–168.
34. S. Telenyk, E. Zharikov, O. Rolik, “An approach to virtual machine placement in cloud data centers,” In proc. of the 2016 International Conference Radio Electronics & Info Communications (UkrMiCo) 11–16 September, Kyiv, Ukraine. – 2016 pp. 1–6.
35. R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, “A survey on virtual machine migration and server consolidation frameworks for cloud data centers,” Journal of Network and Computer Applications, vol. 52, pp. 11–25, 2015.
36. B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, “Modeling and synthesizing task placement constraints in google compute clusters,” In Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC), 2011.
37. S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, and U. Wieder, “Validating heuristics for virtual machines consolidation,” Microsoft Research, MSR-TR-2011- 9, 2011.
38. C. Reiss, J. Wilkes, and J. L. Hellerstein, “Google cluster-usage traces: format+ schema,” Google Inc., Mountain View, CA, USA, Technical Report, 2011.
39. Иванисенко И.Н. Методы решения задачи балансировки вычислительной нагрузки в сети распределенных вычислений / И.Н. Иванисенко // Информационные технологии в навигации и управлении: состояние и перспективы развития: Первая научно-техн. конф., 5 – 6 июля 2010: матер. конф. – Киев, Украина, 2010. – С. 26

40. Карун М.Є. БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ДОДАТКІВ У «ХМАРНОМУ» СЕРЕДОВИЩІ // Карун М.Є., Хмельницький Ю.В. - International Scientific Journal "Internauka" – https://www.inter-nauka.com/uploads/public/1543880159577.pdf (Режим доступу)
41. Телелейко І. С. Аналіз методів динамічного балансування навантаження в хмарному середовищі / І. С. Телелейко, М. М. Орлова // X конференція молодих вчених «Прикладна математика та комп'ютинг – ПМК'2018», Київ, 21-23 березня 2018р.: Збірник тез доповідей. – К.: Просвіта, 2018.
42. Телелейко І. С. Спосіб динамічного балансування навантаження в хмарному середовищі / І.С. Телелейко, М.М. Орлова // Науковий журнал «Інтернаука. – 2018. – №7. – Електронні дані. – https://www.inter-nauka.com/issues/2018/7/3687/ (Режим доступу).
43. Ільченко М.Ю., Кравчук С.О. Телекомунікаційні системи. – Київ: Наукова думка, 2017. – 730 с.
44. Досягнення в телекомунікаціях 2019 / за наук. ред. М.Ю.Ільченка, С.О.Кравчука: монографія. - Київ: Інститут обдарованої дитини НАПН України, 2019.- 336 с. Рекомендовано до друку ВР КПІ ім.І.Сікорського (прот.№10 від 04.11.2019 р.) ISBN 978-617-7734-12-2
45. Priya, V., Kumar, C. S., & Kannan, R. (2019). Resource scheduling algorithm with load balancing for cloud service provisioning. Applied Soft Computing, 76, 416-424. doi:10.1016/j.asoc.2018.12.021
46. Rahman, Mazedur & Iqbal, Samira & Gao, Jerry. (2014). Load Balancer as a Service in Cloud Computing. Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014. 204-211. 10.1109/SOSE.2014.31.
47. Milan, S. T., Rajabion, L., Ranjbar, H., & Navimipour, N. J. (2019). Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments. Computers & Operations Research, 110, 159-187. doi:10.1016/j.cor.2019.05.022

48. Joshi, Shalini & Kumari, Uma. (2017). A Comprehensive Analysis of Load Balancing Algorithms in Cloud Computing. 10.13140/RG.2.2.15001.06247.
49. Thakkar, Rikin & Ramavat, Jalpa. (2017). A Survey on Various Load Balancing Algorithms in Cloud Computing. International Journal for Science and Advance Research in Technology. 3. 10341039.
50. Rajeshkannan, R. & Murugaiyan, Aramudhan. (2016). Comparative Study of Load Balancing Algorithms in Cloud Computing Environment. Indian Journal of Science and Technology. 9. 10.17485/ijst/2016/v9i20/85866.
51. Q. Wang, S. Chen, "Latency–minimum offloading decision and resource allocation for fog–enabled Internet of Things networks", Trans. Emerg. Telecommun. Technol., Jan. 2020, [online] Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3880.
52. Gopinath, P.P. & Vasudevan, Shriram. (2015). An In-depth Analysis and Study of Load Balancing Techniques in the Cloud Computing Environment. Procedia Computer Science. 50. 427432. 10.1016/j.procs.2015.04.009.
53. L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, "Cost efficient resource management in fog computing supported medical cyberphysical system", IEEE Trans. Emerg. Topics Comput., vol. 5, no. 1, pp. 108-119, Jan. 2017.
54. Shah, Nadeem & Farik, Mohammed. (2015). Static Load Balancing Algorithms In Cloud Computing: Challenges & Solutions. International Journal of Scientific & Technology Research. 4. 353-355.
55. Hussein, M. K., & Mousa, M. H. (2020). Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization. IEEE Access, 8, 37191-37201. doi:10.1109/access.2020.2975741
56. B. Ghutke and U. Shrawankar, "Pros and cons of load balancing algorithms for cloud computing," 2014 International Conference on Information Systems and Computer Networks (ISCON), Mathura, 2014, pp. 123-127.
57. K. Nishant et al., "Load Balancing of Nodes in Cloud Using Ant Colony Optimization," 2012 UKSim 14th International Conference on Computer Modelling and Simulation, Cambridge, 2012, pp. 3-8.

58. M. Randles, D. Lamb and A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing," 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, Perth, WA, 2010, pp. 551-556.
59. J. M. Shah, K. Kotecha, S. Pandya, D. B. Choksi and N. Joshi, "Load balancing in cloud computing: Methodological survey on different types of algorithm," 2017 International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, 2017, pp. 100-107.
60. Dong, Y., Xu, G., Ding, Y., Meng, X., & Zhao, J. (2019). A 'JointMe' Task Deployment Strategy for Load Balancing in Edge Computing. IEEE Access, 7, 99658-99669. doi:10.1109/access.2019.2928582
61. S. Aslam and M. A. Shah, "Load balancing algorithms in cloud computing: A survey of modern techniques," 2015 National Software Engineering Conference (NSEC), Rawalpindi, 2015, pp. 30-35.
62. K. Garala, N. Goswami and P. D. Maheta, "A performance analysis of load Balancing algorithms in Cloud environment," 2015 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, 2015, pp. 1-6.
63. Ala'anzy, M., & Othman, M. (2019). Load Balancing and Server Consolidation in Cloud Computing Environments: A Meta-Study. IEEE Access, 7, 141868-141887. doi:10.1109/access.2019.2944420
64. Zhao, J., Yang, K., Wei, X., Ding, Y., Hu, L., & Xu, G. (2016). A Heuristic Clustering-Based Task Deployment Approach for Load Balancing Using Bayes Theorem in Cloud Environment. IEEE Transactions on Parallel and Distributed Systems, 27(2), 305-316. doi:10.1109/tpds.2015.2402655
65. Tang, F., Yang, L. T., Tang, C., Li, J., & Guo, M. (2018). A Dynamical and Load-Balanced Flow Scheduling Approach for Big Data Centers in Clouds. IEEE Transactions
66. Кононюк А.Е. Фундаментальная теория облачных технологий / «Освіта України». – Київ 2018. - С. 11–51.