Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи розподілення задач та моніторингу KPI в highload-системах обробки даних

(тема)

Виконав:

студент II курсу, групи СПм-23-1

Бондаренко О.В.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: ст. викл. Бугрій А.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Бондаренко Ользі Віталіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Методи розподілення задач та моніторингу KPI в highload-системах обробки даних

затверджена наказом по університету від " 22 " листопада 2024 р. № 1236 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи

1) специфікації потоків даних;
2) специфікації гібридного середовища обробки даних;
3) специфікації наборів KPI.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) метод розподілу обчислювальних завдань в гетерогенній обчислювальній системі з використанням клауд-обчислювачів;
3) методи моніторингу KPI в гетерогенній обчислювальній системі з використанням клауд-обчислювачів;
4) експериментальна оцінка запропонованого методу розподілу обчислювальних завдань;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	26.11.24-30.11.24	
2	Вибір алгоритмів розподілення задач	02.12.24-05.12.24	
3	Формування характеристик вхідних даних	06.12.24-10.12.24	
4	Розробка моделей	11.12.24-21.12.24	
5	Проведення експериментів	23.12.24-03.01.25	
6	Оформлення матеріалів кваліфікаційної роботи	04.01.25-07.01.25	
7	Подання кваліфікаційної роботи керівникові	08.01.25-11.01.25	
	і її попередній захист		
8	Подання кваліфікаційної роботи на	13.01.25-17.01.25	
	рецензування		

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Бугрій А.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 92 с., 24 рис., 9 табл., 1 додаток, 31 джерело.

ВІРТУАЛЬНА МАШИНА, ВИСОКОНАВАНТАЖЕНІ СИСТЕМИ, ВИСОКОПРОДУКТИВНІ ОБЧИСЛЕННЯ, ГІБРИДНА ОБЧИСЛЮВАЛЬНА СИСТЕМА, МОДЕЛЬ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ, ПРОЦЕС, РОЗПОДІЛЕННЯ ЗАДАЧ, VIRTUAL PRIVATE CLOUD

Метою кваліфікаційної роботи є розробка та дослідження ефективних методів розподілення обчислювальних завдань в гібридних обчислювальних системах з урахуванням обмежень ресурсів, прогнозування навантаження, а також забезпечення балансу навантаження між хмарною інфраструктурою та орендованими фізичними серверами. Зокрема, акцент робиться на гнучкості конфігурації ресурсів, забезпеченні високої продуктивності системи та оптимізації процесу перемикання задач між вузлами для зменшення витрат на додаткові ресурси при переміщенні даних.

У ході виконання кваліфікаційної роботи потрібно розглянути теоретичні та практичні аспекти розподілення обчислювальних задач у highload-системах, зокрема методи балансування навантаження, інтеграція з віртуальними мережами і приватними хмарними інфраструктурами (VPC), а також алгоритми для оптимізації використання обчислювальних ресурсів з різною конфігурацією. Оцінка ефективності розроблених рішень буде здійснюватися через моделювання та аналізу результатів у реальних умовах експлуатації гібридних обчислювальних систем.

ABSTRACT

Master’s thesis: 92 pages, 24 figures, 9 tables, 1 appendice, 31 sources.

HIGH-PERFORMANCE COMPUTING, HYBRID COMPUTING SYSTEM, MODEL OF COMPUTATIONAL RESOURCES, PROCESS, VIRTUAL MACHINE, VIRTUAL PRIVATE CLOUD, WORKLOAD DISTRIBUTION

The major goal of this thesis is to develop and investigate effective methods of task distribution in hybrid computing systems, taking into account resource limitations, load forecasting, and load balancing between cloud infrastructure and rented physical servers. In particular, the focus is on the flexibility of resource configuration, ensuring high system performance, and optimizing the process of task migration between nodes to reduce the costs of additional resources when transferring data.

During the qualification work, it is necessary to consider the theoretical and practical aspects of computational task distribution in high-load systems, including load balancing methods, integration with virtual networks and private cloud infrastructures (VPC), as well as algorithms for optimizing the use of computational resources with different configurations. The effectiveness of the developed solutions will be evaluated through modeling and analyzing results in real-world conditions of hybrid computing system operation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВМ – віртуальна машина

ЦП – центральний процесор

ВМСТ – Збалансований мінімальний час завершення (англ., Balanced Minimum Completion Time)

CPU – Центральний процесор (англ., Central Processing Unit)

DAG – Орієнтований ациклічний граф (англ., Directed Acyclic Graph)

FIFO – Перший прийшов – перший пішов (англ., First in First Out)

KPI – Ключові показники ефективності (англ., Key Performance Indicators)

LLF – Спочатку найменше завантажений (англ., Least Loaded First)

LIFO – Останній прийшов – перший пішов (англ., Last in First Out)

RAM – Пам'ять з довільним доступом (англ., Random Access Memory)

VPC – Віртуальна приватна хмара (англ., Virtual Private Cloud)

WDS – Бездротова розподільча система (англ., Wireless Distribution System)

ВСТУП

Системи планування завдань призначені для вирішення проблеми ефективного і гнучкого розподілу завдань обробки даних на доступні обчислювальні ресурси в гетерогенних високонавантажених системах. Основною проблемою під час розгортання та супроводу таких систем є складність налаштування програмного забезпечення, що зазвичай пов'язано з певними характеристиками високонавантажених обчислювальних систем: різнорідність завдань за ресурсними вимогами (частина завдань має різні вимоги до процесору, пам'яті), апаратна гетерогенність обчислювальних вузлів та різне завантаження вузлів високонавантаженої системи вимагають спеціального обліку, що веде до створення складних політик планування; відсутність повної інформації про ресурсні вимоги завдань ускладнює прийняття інтелектуальних рішень щодо їх планування.

Зростання популярності кластерних, грід- і хмарних систем обумовлено збільшенням кількості прикладних завдань та значним навантаженням на обчислювальні системи [1]. Постачальники мережевих сервісів, орієнтуючись на великі центри обробки даних, почали приділяти особливу увагу вдосконаленню методів планування завдань обробки даних.

Вимоги скорочення часових витрат рішення завдань, спрощення процедури супроводу гетерогенних високонавантажених систем обробки даних у існуючих умовах обґрунтовують актуальність дослідження і розробки нових методів планування завдань у таких системах.

Об'єктом дослідження є системи планування завдань у розподілених гетерогенних високонавантажених системах обробки даних.

Предметом дослідження виступають методи планування завдань та ключові індикатори продуктивності, які використовуються в системах планування обчислювальних завдань для розподілених гетерогенних високонавантажених обчислювальних систем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вступ до розділу

Гетерогенні високонавантажені системи, які частково використовують хмарні обчислювальні ресурси, характеризуються такими властивостями: розподіл компонентів, динамічна зміна конфігурації, неоднорідність (використання вузлів з різнорідними програмно-апаратними ресурсами) та часто неконтрольованою кількістю завдань. Важливо зазначити, що дані можуть бути розподілені на обчислювальні вузли не з метою виконання невідкладних обчислень для вирішення того чи іншого завдання. Попереднє збереження даних на обчислювачах (можливо, із дублюванням – для забезпечення стійкості до відмови) може слугувати частиною підготовки до початку розподіленого вирішення складного завдання у майбутньому або як незалежна процедура в рамках функціонування активного або пасивного розподіленого сховища даних, коли дані стають ресурсом [2]. Процеси збереження та читання даних також можуть розглядатися як операції, оскільки їх виконання потребує використання не лише фізичних пристроїв зберігання, а й процесорних ресурсів і певного обсягу оперативної пам'яті.

Найбільш відомими дослідженнями в цій галузі можна вважати роботи Tanenbaum A., Coulouris G., Agrawal D., Zhao H., Sakellariou R., Amar A. Вибір методу розподілу завдань між обчислювальними вузлами визначає загальну ефективність обчислювального середовища. Коректний вибір може зменшити час простою обчислювальних пристроїв, скоротити обсяги та час передачі даних між виконавчими пристроями, підвищити масштабованість системи та зменшити час доступу до даних. Цей розділ є оглядом предметної галузі, який надає уявлення про актуальні підходи до вирішення задачі розподілу обчислювального навантаження в умовах високонавантажених гетерогенних обчислювальних середовищ.

1.2 Класифікація стратегій розподілу обчислювального навантаження у високонавантажених обчислювальних середовищах

Класифікація стратегій розподілу обчислювального навантаження дозволяє систематизувати існуючі підходи, визначити їх сильні та слабкі сторони, а також адаптувати їх до специфічних вимог різних додатків і архітектур. Можна виділити наступну класифікацію стратегій розподілу обчислювального навантаження у високонавантажених обчислювальних середовищах.

У залежності від динаміки розподілу навантаження бувають: статичні, напівдинамічні та динамічні стратегії. Кожна з цих стратегій має свої особливості та підходи до управління ресурсами. Перша стратегія базується на фіксованому плані розподілу навантаження, який визначається заздалегідь [3]. У таких системах всі обчислювальні завдання розподіляються між вузлами відповідно до попередньо встановлених параметрів. Напівдинамічні стратегії поєднують елементи статичних і динамічних підходів. План розподілу навантаження формулюється на етапі ініціалізації системи, перед початком основних обчислень. Це дозволяє врахувати певні зміни в умовах, але не забезпечує гнучкості під час виконання завдань. Динамічна стратегія є найбільш гнучкою та адаптивною. Вона дозволяє змінювати план розподілу навантаження в реальному часі, реагуючи на зміни у навантаженні, появу нових обчислювальних вузлів, відмову вже працюючих або за заздалегідь визначеним графіком. Адаптивні стратегії динамічного розподілу навантаження враховують різноманітні фактори, такі як зміна характеристик завантаження, заплановані події, а також непередбачувані обставини. Вони дозволяють виконувати балансування навантаження та необхідну рекомбінацію ресурсів при переконфігуруванні розподіленого середовища. Наприклад, при виявленні нових обчислювальних вузлів система може автоматично перерозподілити навантаження, щоб максимально використовувати доступні ресурси. Також, у випадку відмови вузла,

необхідно швидко побудувати новий план розподілу навантаження, що дозволить уникнути затримок у виконанні завдань.

За принципом управління виділяють такі стратегії: з децентралізованим, централізованим та ієрархічним управлінням. Централізовані стратегії передбачають наявність центрального елемента – планувальника, який приймає рішення на основі інформації, отриманої від усіх вузлів системи. Цей підхід забезпечує єдине управління та координацію, але може стати вузьким місцем, якщо система піддається великому навантаженню. Децентралізовані алгоритми функціонують інакше: кожен обчислювальний вузол планує розподіл даних самостійно або спирається на інформацію від найближчих сусідніх вузлів. Це дозволяє знизити навантаження на центральний елемент, але може призвести до неузгодженості в управлінні. Ієрархічні стратегії поєднують елементи обох попередніх підходів. У цьому випадку існує головний планувальник, який називається метапланувальником. Він розподіляє завдання відповідно до певних правил, а глобальна черга завдань поділяється на кілька підчерг для кожного рівня ієрархії.

За ознакою універсальності існують: універсальні та спеціалізовані стратегії. Спеціалізовані стратегії можуть орієнтуватися на специфічність архітектури розподіленої системи, окремий випадок топології мережевого середовища та ін. Універсальні – розроблені для використання в різноманітних умовах і можуть адаптуватися до різних архітектур розподілених систем.

Прогностичні стратегії та стратегії без використання механізмів прогнозу станів системи (наприклад, завантаженості вузлів). Стратегії, які застосовують короткострокове та довгострокове прогнозування розвитку обчислювальних процесів, зазвичай показують значно кращі результати в порівнянні з тими, що не мають таких механізмів. Прогностичні стратегії, у свою чергу, поділяються на ті, які враховують причини розбалансування навантаження, і ті, які цього не роблять.

Стратегії, що враховують лише продуктивність обчислювальних вузлів розподіленої системи (у тому числі продуктивність наявних локальних сховищ даних), та стратегії, що також враховують продуктивність комунікаційної підсистеми. У цьому разі здійснюється аналіз характеру інформаційно-керуючого трафіку мережі.

Стратегії, які не використовують і використовують знання про доступні для виконання завдань ресурси (у тому числі – із залученням наближених та реалістичних методів оцінки їх кількості). В останньому випадку враховуються характеристики вузлів, такі як кількість активних потоків, швидкість і кількість наявних обчислювальних пристроїв (процесорів, прискорювачів), обсяг оперативної пам’яті, а також наявність вільного місця для виконання файлових операцій. Виконання підзадач може здійснюватися за різними принципами: FIFO – у порядку надходження задач; LIFO – обробка відбувається у зворотному порядку; Random Selection for Service – випадковий вибір завдань для виконання; Time Sharing – ресурси рівномірно розподіляються між усіма завданнями; Least Attained Service – вибирається завдання, яке має найменший час обслуговування. Ці підходи зазвичай ґрунтуються на припущеннях про подібну трудомісткість завдань, а також на використанні схожих обчислювальних вузлів, де в один момент на кожному вузлі виконується не більше одного завдання. Однак, за наявності знань про ресурси можуть використовуватися додаткові стратегії, наприклад Shortest Job First – вибирається підзадача з найменшими вимогами до ресурсів; Shortest Remaining processing time – виконується завдання з найменшою оцінкою часу обслуговування.

Стратегії з різними ініціаторами розподілу завдань. Ініціюючою стороною може бути як приймач навантаження (недовантажені вузли), так і джерело навантаження (центральні вузли і навіть перевантажені вузли).

Представлена класифікація дозволяє краще зрозуміти внутрішні принципи роботи програм-планувальників, дає підстави вважати, що універсальної стратегії на випадки життя немає. В одному і тому ж

програмному продукті може використовуватися кілька різнорідних стратегій розподілу навантаження

Далі пропонується перейти до розгляду придатних для використання часткових рішень для розподілу та балансування завдань.

1.3 Аналіз актуальних рішень для розподілу та балансування завдань у гетерогенних високонавантажених обчислювальних середовищах

У роботі [4] описаний підхід з поданням розв'язуваної задачі у вигляді ациклічного графа (DAG) [5]. Кожна вершина цього графа відповідає конкретному завданню, яке виконується в процесі роботи над загальним проектом, а дуги графа вказують на послідовність виконання цих завдань. Граф має одну вихідну вершину, що відповідає функції формування вихідних даних, та одну кінцеву вершину, яка представляє функцію формування фінального результату обробки. Вага кожної вершини графа відображає вартість виконання відповідної роботи, тоді як вага ребер графа показує вартість передачі даних між обробниками. Вартість виконання окремих завдань вимірюється в абстрактних умовних одиницях, так само, як і вартість передачі даних між незалежними обробниками (рисунок 1).

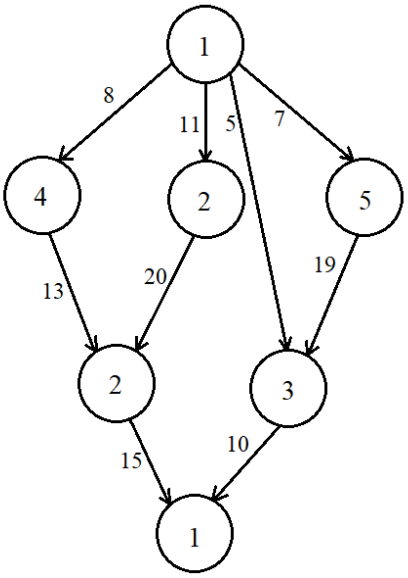

Рисунок 1.1 – Приклад ациклічного графа обчислень

Експерт, який працює з гетерогенною розподіленою високонавантаженою обчислювальною системою, що складається з різних обчислювачів, визначає поняття елементарної задачі та оцінює часову складність її виконання на кожному з цих пристроїв. Він також встановлює час, необхідний для передачі завдання між обчислювачами. Знаючи вартість виконання окремих частин загального завдання, експерт може легко оцінити загальний час, необхідний для виконання всього завдання на конкретному обчислювачі.

Алгоритм, який займається упорядкуванням вершин у DAG та формуванням груп робіт з можливістю незалежного планування, детально описано в роботі [6]. Для ефективного планування роботи цих груп рекомендується використовувати метод збалансованого мінімального часу завершення (BMCT).

У роботі [7] представлено нову математичну модель планувальника завдань, яка відрізняється від існуючих тим, що враховує як метадані про попереднє виконання завдань, так і метрики завантаження ресурсів при розподілі задач на доступні обчислювальні ресурси. Запропоновано новий алгоритм пошуку найближчих сусідів, що використовує локалізоване хешування. Цей алгоритм враховує типи атрибутів і їх значення для споживання ресурсів.

Модель планування включає кілька множин: завдань, атрибутів даних (метаданих), вузлів-обробників і метрик завантаження ресурсів обчислювальних вузлів. Вона встановлює зв’язок між множиною завдань і множиною вузлів-обробників, враховуючи експертні оцінки обчислювальних витрат під час виконання завдань. Наприклад, обчислювальний вузол може бути описаний за кількістю процесорів та обсягом вільної оперативної пам’яті або пам’яті тривалого зберігання.

Запропонований метод базується на метаданих та метриках ресурсів і складається з ряду спеціальних функцій. Серед них: отримання метрик завантаження обчислювальних ресурсів, оцінка обчислювальних витрат для

попередніх завдань, оцінка витрат ресурсів для запуску та виконання нових завдань на доступних обчислювальних вузлах, а також обчислення коефіцієнтів призначення на основі отриманих оцінок. Для кожної пари (вузол, завдання) проводиться аналіз накопиченої інформації про попередні виконання завдань і обчислюється відстань до всіх раніше вирішених завдань. При цьому враховується різна вартість атрибутів завдань, які стосуються споживання ресурсів.

Для порівняння з цією моделлю використовувалися підходи на основі черги FIFO для розподілу завдань між обчислювальними вузлами, а також підхід, який обирає найменш завантажений вузол для нових завдань LLF. Експериментальні дослідження показали таке:

- загальний час обробки заявок скорочується на 20% при високій інтенсивності надходження завдань. Проте, при середній та низькій інтенсивності він зростає на 4,5% та 10,7% відповідно;

- середній час очікування обробки скорочується на 7,6% при високій інтенсивності надходження завдань, але зростає на 10,6% та 70,6% у випадках середньої та низької інтенсивності. Це зумовлено значними накладними витратами;

- збільшення інтенсивності надходження заявок дозволяє більш ефективно використовувати попередню інформацію, що покращує якість прогнозування.

У цьому дослідженні згадані системи DIET, ProActive, Moab та Maui. Це одні з найпоширеніших західних систем для планування обчислень.

Система DIET [8] призначена для організації обчислень з урахуванням різноманітних типів розподілених ресурсів. Її архітектура (рисунок 2) складається з кількох ключових компонентів:

- клієнтська програма для виконання завдань (Client);

- керуючий агент (Master Agent, MA) – обробляє запити на виконання обчислень від клієнта, отримує інформацію про вільні ресурси з обчислювачів, обирає відповідний виконавчий вузол (обчислювач) та передає

його адресу клієнту;

- місцевий агент (Local Agent, LA) – забезпечує передачу запитів та інформації між керуючими агентами і обчислювачами, а також здійснює локальне планування на рівні обчислювачів;
- серверна служба (Server Daemon, SeD) – служба, що функціонує на кінцевих обчислювальних вузлах, отримуючи інформацію про різні типи завдань, кількість ресурсів та їх завантаженість.

Коли до системи DIET надходить нове завдання, серверні служби здійснюють оцінку характеристик ресурсів розподіленого обчислювача. Ці оцінки надсилаються до батьківського агента, де обчислювачі впорядковуються за певним оптимізаційним критерієм, за замовчуванням – за часом останнього використання. Якщо зберігати часові мітки неможливо, обчислювач вибирається випадковим чином.

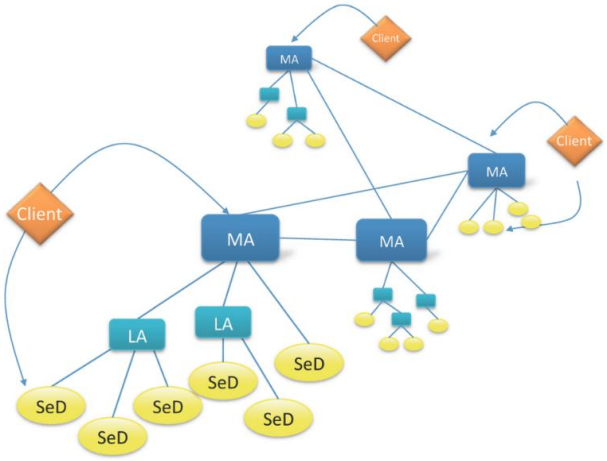

Рисунок 1.2 – Ієрархія компонентів системи DIET

Система DIET реалізує просту автоматизовану схему вибору ресурсів, таких як дані, обчислювальні пристрої та оперативна пам'ять, на основі моніторингу стану вузлів. При цьому вона не враховує інформацію про ресурсні обмеження. Крім того, DIET дозволяє модифікувати свій планувальник через модулі, які підключаються. Це дає змогу додавати нові метрики оцінки продуктивності та встановлювати критерії для

впорядкування вузлів-претендентів на виконання нових завдань. Серед недоліків системи можна відзначити вимоги до програмної реалізації підпрограми для планування нових додатків.

Кросплатформовий програмний комплекс ProActive [9] складається з кількох важливих компонентів: системи управління ресурсами ProActive Grids and Clouds, розподіленого планувальника ProActive Orchestration and Scheduling, програмного стеку ProActive Programming для грід-систем та ряду графічних додатків для користувачів. Планувальник отримує завдання з черги за принципом FIFO, але також надає можливість створення нових методів планування завдяки доступним інтерфейсам та рекомендаціям. Він не вибирає вузол-виконавця для завдання; натомість, він реалізує механізм пошуку ресурсів відповідно до специфікацій, зазначених у запитах клієнтів. Завдання виконується тільки за наявності достатньої кількості ресурсів, в іншому випадку воно залишається в черзі. Після виділення ресурсів клієнту (наприклад, коли на конкретному обчислювальному вузлі з'явилися необхідні дані), клієнт отримує винятковий доступ до вузла-обробника до завершення всіх необхідних операцій. Проте, ProActive має два основних недоліки: по-перше, навіть для простих завдань потрібно реалізовувати методи вибору вузлів-обчислювачів; по-друге, відсутні практично корисні механізми для налаштування політик вилучення завдань з черги.

Планувальник Moab [10] має два режими роботи: опитування системи контролю за вільними ресурсами та режим реєстрації подій. Цей планувальник активує новий цикл планування у відповідь на різні події, такі як надходження нового завдання, завершення виконання завдання, появу нового обчислювального вузла або коригування налаштувань. Кожна ітерація планування передбачає запит до системи контролю ресурсів для отримання актуальної інформації про їх стан, рівень завантаження та поточні налаштування планувальника. Далі Moab отримує список завдань, які відповідають умовам запуску, і впорядковує їх за відносним пріоритетом. Цей пріоритет визначається на основі таких факторів, як власник завдання,

його обсяг, тривалість очікування в черзі та інші параметри.

Основні цілі Moab при виборі обчислювального вузла полягають у підвищенні продуктивності виконання поточних завдань і забезпеченні гнучкості планування нових завдань. Гнучкість досягається завдяки можливості налаштування політики вибору виконавчих вузлів як на локальному рівні (для конкретного завдання), так і на рівні всієї системи. Проте, системі бракує автоматизації для визначення ресурсних вимог розв'язуваних завдань. Крім того, послідовний процес планування виконується лише для пріоритетних підзавдань, що може призвести до рішень, які не є найоптимальнішими для планування виконання всього завдання.

Система Maui [11] обробляє інформацію про завдання, яка включає атрибути, такі як обліковий запис, статус завдання, наявність попередніх даних, а також вимоги до ресурсів, включаючи загальну кількість і час резервування ресурсів. Вузол-обробник в Maui складається з ресурсних одиниць, таких як процесори, пам'ять та мережеві картки. Інформація про стан вузлів та їх доступні ресурси надходить від системи управління ресурсами. Maui підтримує різні політики планування, зокрема клас завдання, що асоціюється з конкретними атрибутами, такими як тривалість виконання, вимоги до ресурсів і обмеження на кількість вузлів-обробників. Класи завдань мають незалежні черги обробки. На відміну від Moab, Maui виконує резервування ресурсів, враховуючи часові інтервали. Проте система має недоліки: вона вимагає точного визначення умов вибору вузлів, класифікації завдань та обмежень доступу, що може ускладнювати процес планування.

Існує система, що працює з пластичними завданнями (moldable jobs), які можуть виконуватися на різній кількості процесорів, визначаючи свою розмірність. Експерт задає можливі розмірності, а програмний планувальник обирає відповідну в залежності від умов завдання та навантаження обчислювачів. Обчислювальні пристрої поділяються на несумісні класи,

об’єднуючи вузли з однаковою архітектурою та схожими характеристиками продуктивності. Кожен вузол може мати різну оперативну та постійну пам’ять, а також різні процесори. При постановці нового завдання формуються вимоги до ресурсів, які можуть бути постійними або витраченими. Якщо сума розмірностей активних завдань перевищує наявну кількість процесорів, виникає черга завдань. Власник завдання визначає клас обчислювачів, діапазон розмірностей та необхідні ресурси. Планувальник реалізується на основі централізованої архітектури клієнт-сервер, де агенти збирають інформацію про вузли та запускають завдання, відправляючи дані на сервер. Система підтримує пріоритети завдань, враховуючи їх простій, складність та активність власників.

У роботі [12] представлено гетерогенний комплекс потокової обробки інформації як розімкнуту мережу масового обслуговування з одноканальними вузлами, де кожен вузол має відому інтенсивність обслуговування. Час обслуговування є випадковою величиною з експотенційним розподілом. Якщо запит надходить, коли вузол зайнятий, він ставиться в чергу. Завершення обробки веде до видалення запиту з системи. Результати показують, що ймовірність обслуговування завдань змінюється залежно від інтенсивності потоку: з підвищенням інтенсивності зростає ймовірність передачі запитів вузлу, але зменшується максимальна ймовірність.

Метод з асинхронними обробниками виявляється найефективнішим для високої інтенсивності, тоді як другий метод підходить для низької інтенсивності, але може призвести до накопичення черг. Третій метод демонструє ефективність як при низькій, так і при високій інтенсивності вхідного потоку.

У роботі [13] представлено мультиагентний підхід до сервісно-орієнтованого управління розподіленими завданнями (рисунок 1.3).

У цій системі визначено правила групової поведінки агентів, які виконують ролі у плануванні, класифікації, моніторингу та виконанні

завдань. Агенти можуть співпрацювати або змагатися між собою.

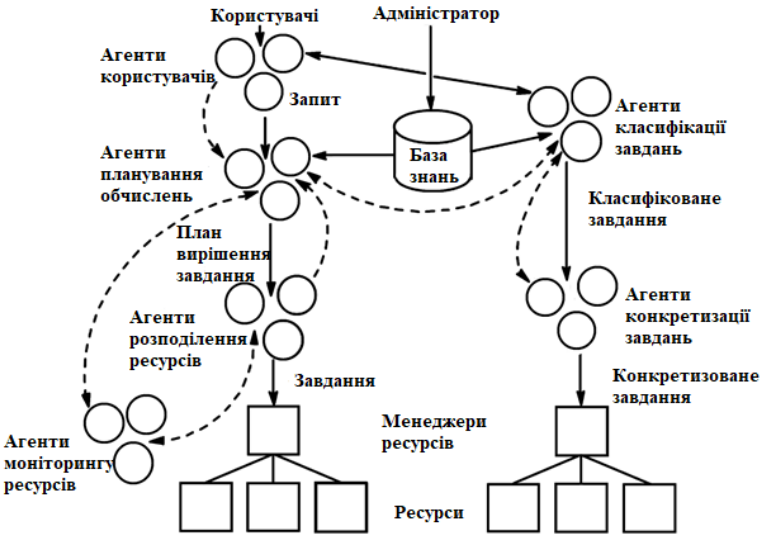

Рисунок 1.3 – Структура мультиагентної системи

Алгоритм управління потоками базується на економічних принципах попиту та пропозиції щодо ресурсів. Це дозволяє враховувати політику адміністрування обчислювальних вузлів і забезпечує справедливий розподіл ресурсів. Завдання, сформовані агентами, передаються агенту-менеджеру, який взаємодіє з агентом моніторингу ресурсів, що призводить до їх розподілу серед локальних агентів. Завдання розподіляються за допомогою тендерної моделі, де обчислювальні роботи виступають як лоти, а обчислювачі – як учасники. Однак таке управління може призводити до збільшення часу виконання завдань, оскільки не завжди можливо врахувати специфіку підзадач і переваги користувачів щодо ресурсів.

1.4 Висновки за розділом

Методи, використовувані в системах планування та розподілу обчислювального навантаження, часто базуються на інформації про ресурсні запити завдань, що покладає відповідальність за налаштування планування на кінцевого користувача. Врахування характеру завантаження обчислювачів

та короткострокове прогнозування вивільнення ресурсів (включаючи локальні фізичні сховища даних) на основі складених планів виконання робіт дозволяє ефективно управляти ресурсами для нових завдань, що сприяє балансуванню навантаження в розподіленій обчислювальній системі.

Методи, що використовують метадані та ресурсні метрики, демонструють високу ефективність лише за умови частого надходження заявок. Однак поєднання цих методів з традиційними, такими як FIFO та LLF, може зменшити тимчасові витрати. Використання адаптивних обмежень розмірності завдань і циклічного вирівнювання навантаження також сприяє зниженню часу виконання завдань.

У моделі гетерогенного комплексу потокової обробки даних, формалізованої як мережа масового обслуговування, метод з фіксованим розподілом навантаження вузлів-обчислювачів показує добрі результати. Застосування сервісно-орієнтованого агентного підходу дозволяє контролювати обчислення на рівні додатків і потоків завдань, а також ефективно розподіляти необхідні ресурси через агентів та спеціальних менеджерів.

2 МЕТОД РОЗПОДІЛУ ОБЧИСЛЮВАЛЬНИХ ЗАВДАНЬ В ГЕТЕРОГЕННІЙ ОБЧИСЛЮВАЛЬНІЙ СИСТЕМІ З ВИКОРИСТАННЯМ КЛАУД-ОБЧИСЛЮВАЧІВ

2.1 Вступ до розділу

Проведений аналіз основних методів та систем планування в високонавантажених гетерогенних обчислювальних системах показав наступне:

- методи, що використовуються в існуючих системах планування, часто спираються на наявність інформації про ресурсні вимоги задач обробки даних, перекладаючи проблему налаштування кінцевого користувача;

- методи планування, що не потребують інформації про ресурсні вимоги, мають наступний основний недолік – відсутність обліку спільних вимог для груп завдань, що призводить до неефективного використання обчислювальних ресурсів [14].

У зв'язку з цим потрібна розробка нового методу планування завдань у високонавантажених гетерогенних обчислювальних системах в умовах неповноти інформації про ресурсні вимоги, що дозволяє скоротити часові витрати на виконання прикладних завдань обробки даних.

2.2 Математична модель гетерогенної високонавантаженої обчислювальної системи з використанням клауд-обчислювачів

Гетерогенні високонавантажені обчислювальні системи, що використовують клауд-обчислювачі, відзначаються застосуванням програмно-конфігурованих мереж (ПКМ). Цей підхід розробляється в рамках ініціативи OpenNetworkFoundation та європейського проекту OFELIA [15]. Основна ідея програмно-конфігурованих мереж полягає в тому, що можна

динамічно керувати передачею даних у мережах, використовуючи так звані віртуальні приватні мережі (VPC). Це дозволяє ефективно організовувати маршрути передачі даних між процесами обчислювальних завдань ще до їхнього запуску, створюючи оптимальні маршрути для обміну інформацією. Такий підхід реалізує проактивну маршрутизацію, що знижує конкуренцію за ресурси.

У рамках цього підрозділу описується імітаційна модель високопродуктивної системи, що працює на хмарній платформі, а також визначаються метрики для оцінки ефективності алгоритмів планування завдань.

На рисунку 2.1 наведена імітаційна схема керуючої системи, що є частиною високопродуктивної системи.

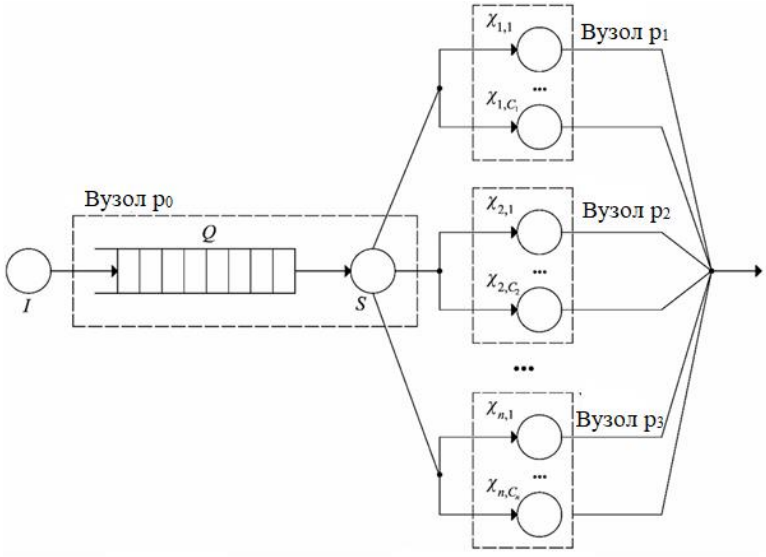

Рисунок 2.1 – Імітаційна схема керуючої системи

Джерело I створює потік паралельних обчислювальних завдань, які користувачі надсилають до черги Q на керуючому вузлі p0. Канал S виступає як планувальник, який обробляє завдання, вилучаючи їх із черги Q і розподіляючи на вільні обчислювальні ядра відповідних вузлів. Мережа високопродуктивної системи, яку моделюють, може бути представлена у вигляді неорієнтованого графа:

$$G = (D, L), \tag{2.1}$$

де D – множина вершин, що є мережевими пристроями: обчислювальні вузли p_i, керуючий вузол p_0, звичайні комутатори k_i (без підтримки OpenFlow), комутатори OpenFlow f_i і контролер OpenFlow Ω;

L – множина мережевих зв'язків між пристроями.

Імітаційну схему обчислювального вузла наведено на рисунку 2.2. Обчислювальний вузол з'єднаний з іншими вузлами та мережевими комутаторами за допомогою двосторонніх з'єднань (r_i). Усі вхідні пакети, а також повідомлення, що генеруються процесами на локальних обчислювальних ядрах, спочатку надходять до черги $Q_{inp.i}$. Після цього ці пакети обробляються через канал обслуговування R_i, який здійснює їх маршрутизацію.

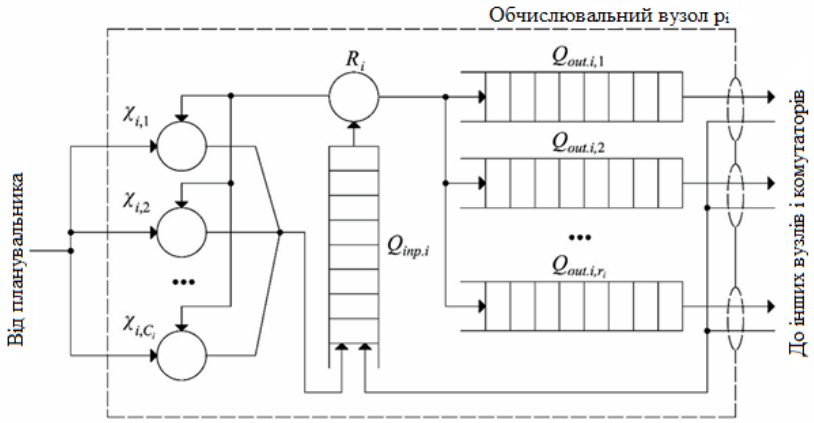

Рисунок 2.2 – Імітаційна схема обчислювального вузла

Якщо пакет спрямований на локальне обчислювальне ядро, він передається безпосередньо до нього, якщо ж ні, то алгоритм маршрутизації направляє пакет в одну з вихідних черг $Q_{out.i,1}$, $Q_{out.i,2}$, $Q_{out.i,\,ri}$. При маршрутизації через канал і моделюється тимчасова затримка. Якщо два процеси з однієї задачі виконуються на сусідніх ядрах того ж вузла, обмін пакетами між ними також відбувається через чергу $Q_{inp.i}$ і канал R_i. Імітаційна схема комутатора є спрощеною версією схеми обчислювального вузла,

оскільки комутатор не має обчислювальних ядер і не виконує обчислення. На рисунку 2.3 показано двосторонній зв'язок $L_{ij} = \{E_{ij}, E_{ji}\}$, який з'єднує мережеві пристрої $d_i \in D$ та $dj \in D$. Час передачі пакета через канали обслуговування E_{ij} і E_{ji} збільшується на затримку, що дорівнює $b(E_{ij}) = b(E_{ji})$.

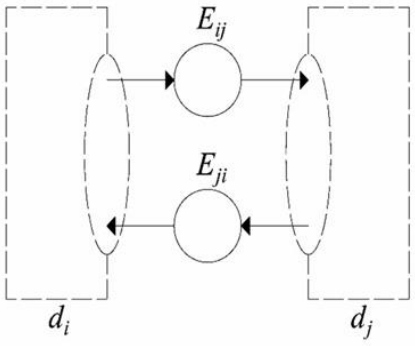

Рисунок 2.3 – Імітаційна схема мережевого зв'язку

Комутатор OpenFlowf_i (рисунок 2.4) має особливість порівняно зі звичайними комутаторами: його канал обслуговування R_i^{of} реалізує поведінку модуля OpenFlow. Коли комутатор отримує новий пакет, він шукає відповідне правило в таблиці потоків. Якщо правило не знайдено, пакет передається контролеру OpenFlow, і його додають до черги $Q^{of}_{out.i,r}$. Коли контролер відправляє відповідь, пакети поміщаються до черги $Q^{of}_{inp.i}$, де їм надається найвищий пріоритет для обробки.

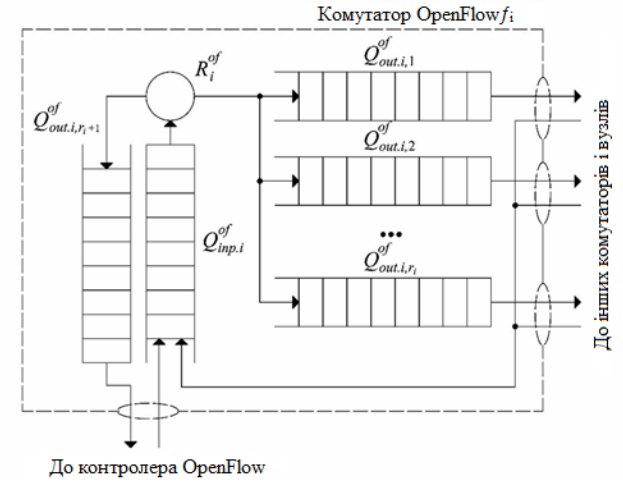

Рисунок 2.4 – Імітаційна схема комутатора OpenFlow

Контролер OpenFlow знаходиться на виділеному сервері, а його імітаційна схема аналогічна схемі обчислювального вузла (рисунок 2.5). Однак є суттєва відмінність: канал R^{con}, окрім звичайної маршрутизації пакетів, додає запити до контролера OpenFlow в окрему чергу Q_{req}^{con}.

Канал обслуговування L^{con} відповідає за реалізацію логіки контролера, який приймає рішення, як обробляти пакет. Результатом цього є управляючі пакети, які передаються через чергу Q_{inp}^{con} до комутаторів для того, щоб перенаправити пакет на відповідний порт або змінити правила в таблицях потоків. У черзі Q_{inp}^{con} і в чергах портів такі пакети мають найвищий пріоритет.

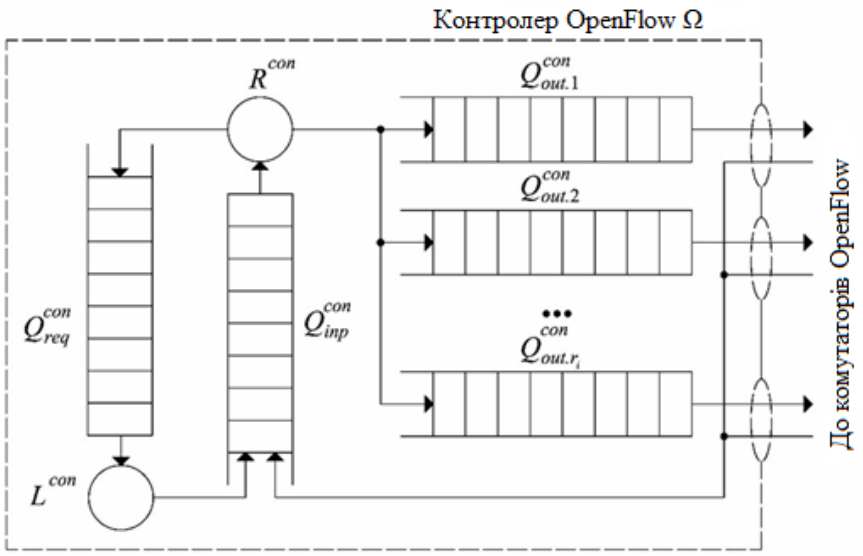

Рисунок 2.5 – Імітаційна схема контролера OpenFlow

Потоки завдань, трафіку та характеристики каналів обслуговування є випадковими величинами. Найоптимальніші закони їхнього розподілу описані в роботі [16]. Імітаційна модель високопродуктивної системи враховує процес передачі даних через мережу та роботу протоколу OpenFlow. Реалізація цієї моделі у вигляді симулятора дозволяє вивчати ефективність алгоритмів.

2.3 Математична модель планування задач в гетерогенній високонавантаженій обчислювальній системі з використанням клауд-обчислювачів

Модель завдання планування для гетерогенної високонавантаженої обчислювальної системи є кортежем:

$$P = (T, A, K, R, M, f, C, W), \tag{2.2}$$

де T – множина задач;

A – множина атрибутів даних (метаданих), пов’язаних із задачами;

K – множина вузлів-обробників;

R – множество характеристик узлов;

M – множина метрик завантаження ресурсів обчислювальних вузлів;

F – відображення множини T на множину K;

C – оцінки обчислювальних витрат виконання завдань з множини T на вузлах-обробниках з множини K;

W – коефіцієнти призначення задач з множини T на вузли-обробники з множини K.

У процесі обробки даних відбувається призначення завдань, що надійшли на вузли обробки, відповідно до деякого методу планування:

$$f: (t, k) \to w, \tag{2.3}$$

де $t \in T$ – задача з черги очікування; задача из очереди ожидания;

$k \in K$ – обчислювальний вузол-обробник;

w – елемент матриці призначення, яка є результатом роботи системи планування.

З кожним одиничним призначенням (t, k) пов'язана деяка оцінка ресурсних витрат c, а метод планування f будує матрицю ресурсних витрат C

та матрицю призначення W [17]:

$$C = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nm} \end{pmatrix},$$
$$W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}, \quad (2.4)$$
$$w_{ij} = \begin{cases} 0, \text{якщо завдання } i \text{ не призначено на вузол } j \\ 1, \text{ якщо завдання } i \text{ призначено на вузол } j \end{cases},$$

де $i \in [1, n]$;

$j \in [1, m]$;

$n = |T|$;

$m = |K|$.

При цьому виконується умова:

$$\sum_{i=1}^{n} w_{ij} = 1, i, j \in N \quad (2.5)$$

Тобто кожне завдання призначається для виконання лише одному обчислювальному вузлу.

Мета методу планування полягає в тому, щоб вирішити задачу призначення, підібравши коефіцієнти $w_{ij} \in \{0, 1\}$, з метою скорочення сумарних ресурсних витрат [17]:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} w_{ij} = sum \to min \quad (2.6)$$

Метод планування завдань у гетерогенних високонавантажених обчислювальних системах на основі метаданих та ресурсних метрик є послідовністю застосування функцій на етапі збору передісторії виконання:

$$f_{hist} = f_{hist2} \circ f_{hist1} \quad (2.7)$$

І на етапі прогнозування:

$$f = f_{assign} \circ f_{costestimation} \circ f_{nearest} \quad (2.8)$$

На етапі збору даних передісторії застосовуються такі функції:

а) функція $f_{hist,1}$: $(t, k) \mapsto (m_1, m_2 \cdots m_r) \in M_{hist,r} \in M_{hist,r} \in \mathrm{N}$ – задає спосіб отримання метрик завантаження обчислювальних ресурсів, де: $t \in T_{hist}$ – завдання обробки даних, $k \in K_{hist}$ – вузол-обробник, M_{hist} – множина кортежів метрик завантаження ресурсів, отриманих в результаті виконання;

б) функція f_{hist2}: $(m_1, m_2, \cdots m_r) \mapsto c$ – задає спосіб отримання оцінок обчислювальних витрат $c \in C_{hist}$ для завдань передісторії. В результаті будується множина $(t, k, c) \in G_{hist}$, де c – оцінка обчислювальних витрат виконання завдання t на вузлі k. Ці дані використовуються для прогнозування витрат на виконання нових завдань.

На етапі прогнозування застосовуються наступні функції:

а) функція $f_{nearest}$: $T \times K \mapsto G_{histnearest}$ – задає спосіб отримання найближчих до нової пари (t, k) трійок $(t_{hist}, k_{hist}, c) \in G_{hist.nearest} \subset G_{hist}$ з передісторії, для яких відомі оцінки обчислювальних витрат c;

б) функція $f_{costestimation}$: $T \times K \times C_{histnearest} \mapsto C$ – визначає спосіб отримання оцінок обчислювальних витрат на виконання нових завдань на доступних обчислювальних вузлах, де $C_{histnearest} = \{c_1, c_2 \cdots c_n\}$ для деякого $n \in N$ – оцінки обчислювальних витрат із отриманої множини $G_{histnearest}$;

в) функція f_{assign}: $c \mapsto w$ – задає спосіб отримання коефіцієнтів призначення на основі отриманих оцінок, де: $c \in C$ – коефіцієнт оцінки

обчислювальних витрат виконання завдань із черги очікування; *w* – елемент матриці призначення, яка є результатом роботи системи планування задач.

Описаний метод на етапі а) передбачає пошук найближчих задач. Однак, основною проблемою цього пошуку є висока розмірність. Для кожної пари (вузол, задача) потрібно обчислити відстань до всіх n задач з попереднього виконання, при цьому асимптотична складність кожного запиту становить O(n).

Крім того, при обчисленні відстані потрібно враховувати різну вагу атрибутів задач з огляду на споживання ресурсів. Додатково слід брати до уваги гетерогенність атрибутів, оскільки прикладні задачі мають як категоріальні, так і числові метадані.

Далі пропонується модифікація алгоритму пошуку на основі методу локалізованого хешування, що дозволяє виконати частину пошуку за константний час O(1), а іншу частину – за сублінійне O(r), $r << n$.

Нижче наведено кроки модифікованого алгоритму:

а) вибір атрибутів для хешування:

1) розділити множину атрибутів на категоріальні та чисельні;

2) для кожного з отриманих множин обчислити коефіцієнти значущості використовуючи навчальну вибірку;

3) вибрати n найбільш значущих для споживання ресурсів категоріальних атрибутів і n найбільш значимих чисельних атрибутів;

б) побудова хеш-таблиці на основі навчальної вибірки для кожного об'єкта даних:

1) обчислити хеш значення окремо для категоріальних та окремо для чисельних типів з множини обраних на кроці а.3 атрибутів;

2) використовувати отриманий сумарний хеш як індекс у хеш таблиці для вибору кошика;

3) зберегти ідентифікатор об'єкта за отриманою адресою в хеш таблицю;

в) пошук k найближчих сусідів для нового об'єкту:

1) обчислити хеш значення окремо для категоріальних та окремо для чисельних типів з множини обраних на кроці а.3 атрибутів;

2) використовувати отриманий сумарний хеш як індекс у хеш таблиці для вибору кошика;

3) обчислити відстань до кожного з об'єктів, ідентифікатори яких зберігаються в кошику, отриманому на попередньому кроці;

4) відсортувати отриманий вектор відстаней;

5) вибрати k об'єктів з найменшою відстанню.

Для застосування описаного методу планування задач у гетерогенних високонавантажених обчислювальних системах на практиці була розроблена методика планування задач на основі метаданих та ресурсних метрик (рисунок 2.6).

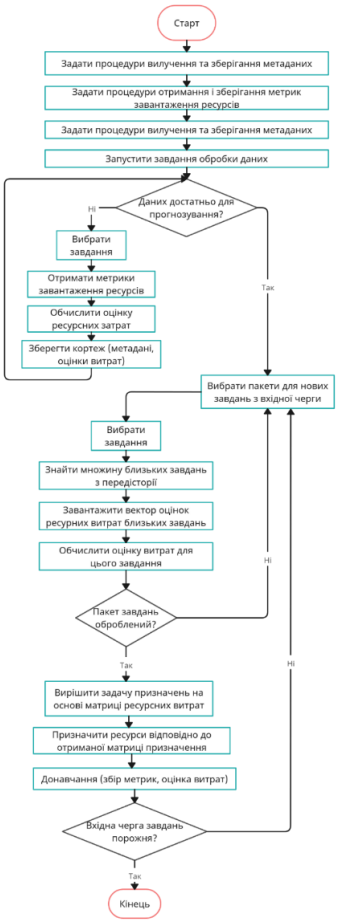

Рисунок 2.6 – Метод планування задач на основі метаданих та ресурсних метрик

2.4 Висновки за розділом

У цьому розділі викладено основні теоретичні положення методу розподілу обчислювальних завдань у гетерогенних високонавантажених обчислювальних системах з використанням клауд-обчислювачів. Розглянуто, як на проблему розподілу задач впливають наявність клауд-обчислювачів та можливість створення конфігурованих комп'ютерних мереж. Окремо представлено імітаційну модель такої системи, яка враховує специфіку розгортання обчислювальних кластерів з конфігурованими мережами. Модель побудована на основі математичних моделей систем масового обслуговування, що дозволить у наступному розділі визначити критерії якості обслуговування та виділити KPI для таких систем. Також наведено формальну модель алгоритму розподілу завдань серед обчислювальних вузлів цієї гетерогенної високонавантаженої системи, що значно зменшує складність процесу завдяки використанню метаданих та ресурсних метрик задач.

3 МЕТОДИ МОНІТОРИНГУ KPI В ГЕТЕРОГЕННІЙ ОБЧИСЛЮВАЛЬНІЙ СИСТЕМІ З ВИКОРИСТАННЯМ КЛАУД-ОБЧИСЛЮВАЧІВ

3.1 Вступ до розділу

Ефективне функціонування гетерогенних обчислювальних систем, які об'єднують різні апаратні та програмні компоненти, значною мірою залежить від здатності системи підтримувати стабільну продуктивність та відповідати заданим вимогам. Для досягнення цих цілей критично важливим є моніторинг ключових показників ефективності, що дозволяє своєчасно виявляти проблеми, оптимізувати використання ресурсів та забезпечувати безперервність роботи системи.

У паралельних і розподілених системах виникають складнощі ефективного використання обчислювальної потужності. Для вирішення цих викликів необхідні спеціалізовані методи моніторингу. Одним із найефективніших підходів є керований подіями моніторинг, який дозволяє аналізувати поведінку та продуктивність програм. Він може бути апаратним, програмним або гібридним, причому останній поєднує переваги обох підходів, забезпечуючи глибший аналіз і підвищену точність.

У гетерогенних системах, що використовують хмарні обчислювальні ресурси, моніторинг стає більш складним через розподілений характер архітектури, динамічність хмарного середовища та різноманітність типів обчислювальних ресурсів. Основним завданням є забезпечення наскрізного відстеження KPI на всіх рівнях системи – від фізичного обладнання до віртуальних сервісів у хмарі.

Моніторинг KPI включає:

- відстеження навантаження на ресурси – моніторинг використання CPU, пам'яті, мережевих ресурсів;
- прогнозування продуктивності – використання історичних даних для

прогнозування майбутнього навантаження та виявлення потенційних вузьких місць;

- автоматичне налаштування ресурсів – використання інструментів автоматичного масштабування для динамічної адаптації ресурсів відповідно до навантаження.

Цей розділ присвячений аналізу методів моніторингу KPI у гетерогенних обчислювальних системах із використанням хмарних обчислень. Розглядаються сучасні підходи до збору, обробки та аналізу даних, а також інструменти, які забезпечують гнучкість, масштабованість та надійність моніторингу в умовах сучасних викликів обчислювальної інфраструктури.

3.2 Теоретичні основи моніторингу KPI

Ключові показники ефективності (KPI) – це числові показники, які використовуються для оцінки продуктивності компонентів, процесів чи системи загалом у досягненні поставлених цілей. Вони слугують інструментом, що дозволяє вимірювати прогрес і результативність в реальному часі, допомагаючи приймати швидкі та обґрунтовані управлінські рішення.

Історично, продуктивність сервера залежала від ефективності використання його процесора та пам'яті. Якщо процесор працює майже на повну потужність, або сервер має дуже обмежені запаси вільної пам'яті, це може негативно вплинути на роботу програм, запущених на ньому. Тому потрібно мати можливість своєчасно ідентифікувати процеси, які найбільше споживають ці ресурси, щоб оперативно усунути проблеми та забезпечити стабільність системи [18].

Окрім моніторингу використання процесора та пам’яті, сучасні серверні операційні системи вимагають відстеження низки інших ключових показників продуктивності, які можуть свідчити про наявність вузьких місць.

Виділено важливі характеристики для гетерогенної обчислювальної системи:

- завантаження ЦП: вимірювання відсотка використання процесора дозволяє оцінити, чи знаходиться завантаження ЦП в межах очікуваних значень для конкретної системи. Високі показники використання процесора протягом тривалого часу можуть свідчити про необхідність оптимізації ресурсів або додаткові налаштування;
- вільне місце для зберігання: відстежування обсягу вільного місця для зберігання або використання диска у системі та тенденції використання диска допомагає уникнути простою сервера, збоїв та втрати даних [19];
- безвідмовна робота сервера: час безперервної роботи (Uptime) визначає, скільки часу система знаходиться в робочому стані без збоїв або перезавантаження. За допомогою цієї характеристики можна виявити неочікувані перезавантаження, контролювати їх;
- активність диска: основні показники включають час зайнятості диска, кількість операцій введення/виведення (IOPs), та час читання/запису. Довжина черги запитів повинна бути мінімальною, щоб запобігти накопиченню запитів. Особливо важливо контролювати ці показники для дискоінтенсивних серверів, таких як бази даних і файлові сервери;
- перемикання контексту: відбувається, коли процесор переходить від виконання одного процесу або потоку до іншого. Це важлива операція для управління багатозадачністю, але вона також є ресурсозатратною. Високий рівень перемикань контексту може свідчити про наявність надмірної кількості активних процесів або помилок у програмному забезпеченні, що призводить до непотрібних переключень.

Моніторинг CPU дозволяє контролювати загальне навантаження та завантаженість ядер, встановлюючи пороги для попередження при перевищенні заданих меж. Використання оперативної пам’яті слід відстежувати, щоб уникнути переповнення, зі встановленням порогу сповіщення при використанні понад 80%. Контроль диска передбачає перевірку вільного простору та швидкостей читання/запису, де алерти

можуть спрацьовувати при заповненні менш ніж на 10%. Для мережі важливо контролювати пропускну здатність і використовувати пороги для сигналізації при перевищенні 90%.

Визначення порогів має відповідати потребам інфраструктури, з обов'язковим тестуванням для оптимізації налаштувань і мінімізації хибних тривог. Алерти мають бути налаштовані для автоматичного сповіщення через різні канали і визначення пріоритетів за критичністю. Необхідно також планувати дії у разі отримання алерту, які можуть включати автоматичні заходи.

Для аналізу ефективності взаємодії алгоритмів планування та методів управління потоками даних у високопродуктивній системі, розгорнутій у хмарному середовищі, розроблено набір показників:

- час виконання еталонного набору завдань, цей показник демонструє основну мету роботи високопродуктивної системи – ефективне виконання завдань її користувачів:

$$\Delta T = T_1 - T_0, \tag{3.1}$$

де T_1 – час завершення виконання всіх завдань набору;

T_0 – час запуску системи для виконання еталонного набору завдань.

- середнє завантаження обчислювальних ядер системи U, розраховується як середнє арифметичне значення середніх завантажень усіх обчислювальних ядер системи, вимірюється у відсотках:

$$\overline{U} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{c_j} \bar{U}_{ij}}{\sum_{i=1}^{n} c_i}, \tag{3.2}$$

де C_i – кількість обчислювальних ядер і-го вузла системи;

$\bar{U}_{ij}$ – середнє завантаження j-го ядра на і-му вузлі системи;

n – загальна кількість вузлів у системі.

- індекс збалансованості використання ресурсів обчислювальних ядер σ, визначає рівномірність завантаження обчислювальних ресурсів системи, допомагає виявляти вузли, які перевантажені чи недовантажені, і оптимізувати розподіл завдань у системі. Рівень збалансованості вважається високим, якщо значення σ_{CPU} не перевищує 10%, розраховується, як:

$$\bar{U} = \sqrt{\frac{\sum_{i=1}^{n}(\bar{U}_i - \bar{U})^2}{n-1}}, \quad (3.3)$$

де $\bar{U}i$ – середнє завантаження і-го вузла системи;

$\bar{U}$ – загальне середнє завантаження обчислювальних ядер у системі;

n – загальна кількість вузлів у системі.

- вартість володіння обчислювальною інфраструктурою:

$$Cost = \sum_{i=1}^{n} Cost_{VM_i} + Cost_{Data}, \quad (3.4)$$

де $Cost_{VM_i}$ – витрати на віртуальні машини;

$Cost_{Data}$ – вартість вихідного трафіку.

Ми не враховуємо вартість мережевих інтерфейсів, вважаємо, що кожна віртуальна машина має гігабітний мережевий інтерфейс за замовченням. При цьому вартість віртуальних машин розраховується як:

$$Cost_{VM} = \sum_{i=1}^{n} Cost_{CPU} \times CPUPerfomance + Cost_{MemoryGb} \times MemorySizeGb, \quad (3.5)$$

де $Cost_{CPU}$ – витрати на віртуальні машини;

CPUPerfomance – продуктивність процесора, що використовується віртуальною машиною;

$Cost_{MemoryGb}$ – вартість одного гігабайта оперативної пам’яті;

MemorySizeGb – обсяг оперативної пам’яті, виділений для віртуальної машини (у гігабайтах);

n – кількість віртуальних машин у системі.

Вартість трафіку який виходить із GCP обчислюється так:

$$Cost_{Data} = Cost_{EgressGb} \times EgressGb, \quad (3.6)$$

де CostEgressGb – вартість передачі одного гігабайта вихідного трафіку з хмарної платформи;

EgressGb – обсяг вихідного трафіку, переданого з GCP, виміряний у гігабайтах.

Моніторинг KPI в гетерогенних обчислювальних системах стикається з численними викликами через різноманіття використовуваних технологій та інфраструктури, що ускладнює інтеграцію даних, їх обробку та узгодження. Системи часто мають різні формати і стандарти, що потребує адаптації для забезпечення сумісності. Крім того, обробка великих обсягів даних і моніторинг у реальному часі вимагає потужних обчислювальних ресурсів.

3.3 Підходи до моніторингу KPI

3.3.1 Агентний моніторинг

Моніторинг на основі агентів передбачає розгортання програмних агентів на окремих пристроях або серверах в IT-інфраструктурі. Ці агенти постійно збирають дані про продуктивність системи, використання ресурсів, мережевий трафік і поведінку програм. Кожен агент працює автономно, збираючи та аналізуючи дані локально перед передачею відповідної інформації на централізовану платформу моніторингу [20].

Головні складові інструментів агентного моніторингу включають:

- програмне забезпечення агента – легкий, специфічний для

платформи код, який встановлюється на окремих пристроях або серверах, і призначений для збору метрик продуктивності системи;

- платформа моніторингу, що збирає дані, зібрані окремими агентами, проводить їх аналіз у реальному часі та надає узагальнену інформацію;

- комунікаційні протоколи для безперешкодної передачі даних між агентами та платформою моніторингу. Серед звичних протоколів – HTTP/HTTPS, SNMP (Simple Network Management Protocol), SSH (Secure Shell).

Такий вид моніторингу може використовуватися для моніторингу серверів та додатків для відстеження продуктивності, доступності та стану ресурсів; для мережевої інфраструктури, в цьому випадку агенти збирають дані про трафік, використання пропускної здатності, втрати пакетів і затримки; для хмарних і віртуалізованих середовищ, щоб оптимізувати використання ресурсів і забезпечити відповідність угодам про рівень обслуговування (SLA); для баз даних збираються дані про час відгуку SQL-запитів, пропускну здатність транзакцій, використання кешу буфера та операції введення-виведення на дисках [20].

System Center Operations Manager є одним із прикладів агентного моніторингу. Він використовується для моніторингу складних IT-інфраструктур, охоплюючи як локальні, так і хмарні середовища. SCOM – це програмне забезпечення від Microsoft, яке використовується для моніторингу і управління інфраструктурою. SCOM дозволяє відстежувати стан, здоров'я та продуктивність комп'ютерних систем та застосунків через єдиний інтерфейс. Він використовує агентів, які збирають дані про різні події та алерти, і передають їх на центральний сервер для аналізу та відображення. Він також надає можливість отримувати сповіщення, що створюються у разі порушень доступності, продуктивності, конфігураційних змін або загроз безпеці, і підтримує інтеграцію з різними системами, зокрема Microsoft Windows Server та Unix/Linux. Архітектура зображена на рисунку 3.1.

Основна концепція SCOM полягає в тому, що на моніторингованому

комп'ютері розташовується агент, який спостерігає за різними джерелами, включаючи журнали подій Windows, і виявляє події чи попередження від запущених програм. Коли агент виявляє проблему, він надсилає повідомлення на центральний сервер SCOM, де дані зберігаються у базі даних Operation DataBase для подальшого аналізу. Сервер застосовує фільтри для сповіщень і може автоматично надсилати їх адміністраторам через електронну пошту чи месенджери, створювати запити на підтримку мережі або запускати інші робочі процеси для усунення проблеми.

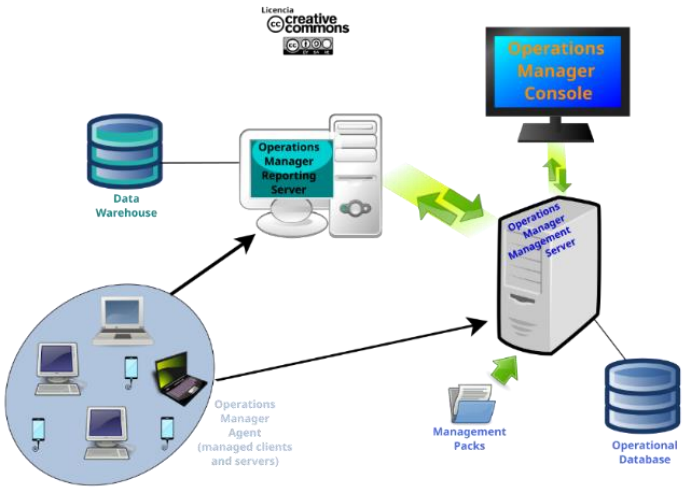

Рисунок 3.1 – Основні компоненти продукта SCOM

3.3.2 Безагентний моніторинг

Безагентний моніторинг – це менш інвазивний метод спостереження за мережею, який використовує специфічні для застосунків API та різні мережеві протоколи, такі як SNMP і WMI, для відстеження загальної продуктивності мережевих ресурсів, зокрема серверів і додатків. Він не потребує встановлення окремого програмного забезпечення на кожен компонент мережі, що робить його простішим у використанні порівняно з агентним методом. Остаточного вибору між цими методами немає, оскільки кожен має свої переваги та недоліки. Вибір залежить від потреб. Наприклад, якщо всі пристрої в мережі однорідні та використовують однакові протоколи,

безагентний моніторинг буде доцільним. Але для детального контролю за інфраструктурою більше підійде агентний підхід [21]. Процес роботи бзагентного моніторингу представлено на рисунку 3.2.

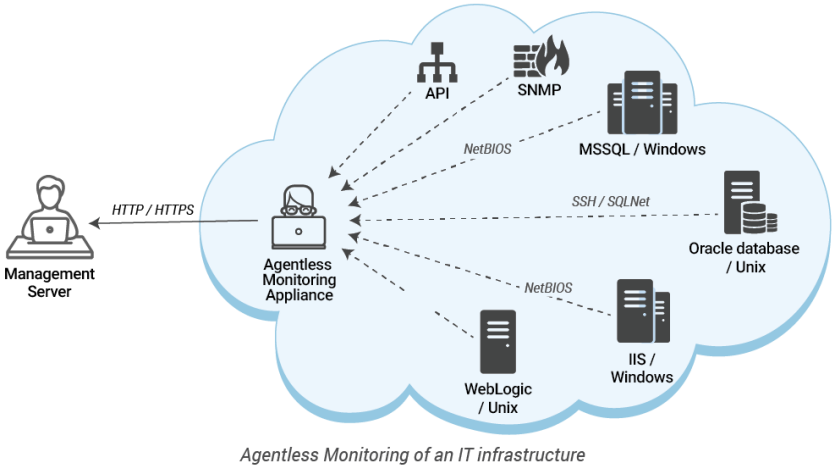

Рисунок 3.2 – Робота безагентного моніторингу

Основна перевага цього підходу – простота впровадження, адже немає потреби встановлювати та обслуговувати програмні агенти на кожному пристрої. Це значно скорочує час і зусилля, особливо у дата-центрах з великою кількістю пристроїв. Також зменшуються витрати на обслуговування, оскільки не потрібно регулярно оновлювати агенти чи вирішувати конфлікти програмного забезпечення. Безагентний підхід не створює навантаження на пристрої, оскільки не використовує їхні ресурси. Це особливо корисно для пристроїв із обмеженими ресурсами. Безагентний моніторинг також ідеально підходить для хмарних і віртуальних середовищ, оскільки використовує API хмарних сервісів, таких як AWS, Azure чи Google Cloud.

Nagios підтримує агентний і безагентний підходи до моніторингу, залежно від налаштувань. Розглянемо у випадку безагентного моніторингу.

Nagios – це інструмент з відкритим вихідним кодом для безперервного моніторингу мережі, серверів і додатків. Його завдання – виявляти та усувати проблеми в IT-інфраструктурі, а також запобігати їхньому виникненню до

того, як вони вплинуть на користувачів. Nagios забезпечує повну інформацію про стан та продуктивність системи.

Серед переваг використання Nagios:

- автоматизація процесу моніторингу, позбавляючи потреби у ручному тестуванні;
- миттєве виявлення неполадок навіть у переривчастій фазі їхнього прояву;
- зниження витрат на підтримку інфраструктури без втрати продуктивності;
- забезпечення своєчасним сповіщенням про помилки та критичні зміни в системі.

Основний процес виглядає так: сервер Nagios ініціює запити до плагінів через планувальник завдань, які, у свою чергу, збирають необхідну інформацію про стан системи, наприклад, навантаження CPU та використання пам'яті. Зібрані дані передаються назад до сервера Nagios, який аналізує інформацію, оновлює графічний інтерфейс і, за потреби, надсилає сповіщення адміністраторам.

Ще одним прикладом може виступати Prometheus, який зазвичай класифікується як безагентний метод моніторингу на основі запитів. Він працює шляхом вибірки метрик з систем на визначених інтервалах за допомогою HTTP-ендпоїнтів, а не через агенти, які передають дані з моніторингових систем. Такий підхід є характерною особливістю Prometheus і робить його ідеальним для динамічних середовищ типу хмарних інфраструктур та мікросервісів.

3.3.3 Гібридний моніторинг

Гібридний моніторинг дозволяє ефективно спостерігати за інфраструктурою, що складається з поєднання локальних дата-центрів і хмарних середовищ. Такий моніторинг допомагає забезпечити цілісну

видимість усіх компонентів IT-інфраструктури, дозволяючи відстежувати продуктивність і здоров'я систем незалежно від того, де вони фізично розташовані. Інтегруючи локальну інфраструктуру з хмарними послугами, гібридні хмарні рішення надають організаціям більшу масштабованість, покращену продуктивність і оптимізоване використання ресурсів [22].

Замість використання окремих інструментів для кожного середовища, гібридна система об'єднує всі дані в одному інтерфейсі. Вона повинна мати змогу збирати дані з різних джерел, таких як локальні сервери, приватні хмари, публічні хмари (AWS, Azure, Google Cloud тощо) і зовнішні сервіси.

Datadog вирішує цю проблему, об'єднуючи дані моніторингу (метрики, логи, трасування, дані про продуктивність мережі) з локальних дата-центрів та різних хмарних платформ в єдине відображення. Такий підхід дозволяє швидше усувати проблеми. Datadog масштабується з динамічною інфраструктурою, автоматично відстежуючи нові інстанси та контейнери, як тільки вони запускаються. Підтримуючи основні публічні хмари (AWS, Azure, Google Cloud, Alibaba) і приватні хмарні технології (OpenStack, VMware), Datadog пропонує легку інтеграцію, попередньо налаштовані інформаційні панелі та всебічний моніторинг для всіх середовищ.

Інші приклади включають Zabbix, Nagios та Prometheus, які також підтримують моніторинг як локальних, так і хмарних ресурсів за допомогою відповідних агентів і безагентних методів.

Zabbix – це відкрите програмне забезпечення для моніторингу та управління мережею, серверами, додатками та різноманітними іншими IT-ресурсами. Як інструмент системного моніторингу з відкритим вихідним кодом, Zabbix дозволяє виявляти, вирішувати та передбачати проблеми в інфраструктурі комп'ютерних систем. Встановлення Zabbix відбувається на операційних системах Linux, використовуючи різні комбінації стеків, такі як LAMP (Linux, Apache, MySQL, PHP), LEMP (Linux, Nginx, MariaDB, PHP) або LAPP (Linux, Apache, PostgreSQL, PHP) [22].

Zabbix збирає дані про мережеві пристрої, моніторинг хмарних

сервісів, контейнерів і віртуальних машин, стеження за операційними системами, логи, бази даних, IoT сенсори та інші, забезпечує систему оповіщення про критичні проблеми за допомогою різних каналів комунікації, таких як: email та SMS; платформи для командної роботи, як-от Slack, MS Teams, Telegram; вебхуки для інтеграцій з зовнішніми системами обміну повідомленнями, ITSM і системами тикетингу.

Ще однією важливою особливістю Zabbix є можливість автоматичного усунення проблем. Система може виконувати сценарії чи команди для виправлення ситуацій, наприклад: перезапускати сервіси, керувати хмарними ресурсами, виконувати автоматичне масштабування, запускати інші користувацькі логіки та сценарії.

3.3.4 Моніторинг у хмарних системах

Хмари мають складну природу через велику кількість залучених ресурсів та необхідність виконання угод рівня обслуговування (SLA) для різних користувачів. Однією з основних характеристик хмар є еластичність, яка дозволяє ресурсам адаптуватися до поточних завдань у будь-який момент. Це означає, що розподіл ресурсів повинен бути гнучким і динамічним, що робить цю задачу ще складнішою. Потреби моніторингу для користувачів хмари відрізняються від потреб постачальників хмарних послуг (CSP), а також моніторинг віртуальних систем має свої відмінності від моніторингу традиційних фізичних систем. Моніторинг хмари залежить від ряду аспектів, серед яких:

- модель надання хмарних послуг, наприклад, інфраструктура як послуга (IaaS), платформа як послуга (PaaS) або програмне забезпечення як послуга (SaaS);
- цільове використання отриманої інформації, наприклад, зворотний зв'язок від користувачів, управління системами;
- початкове джерело моніторингу, таке як фізичні ресурси, віртуальні

машини, програмне забезпечення тощо [23].

Зважаючи на специфічні потреби та виклики, що виникають у хмарних системах, важливо використовувати надійні інструменти для моніторингу. Інструменти від Google Cloud є одними з найкращих рішень у цій галузі, пропонуючи широкий спектр можливостей для моніторингу та аналізу.

Cloud Logging – це повністю керована служба, що забезпечує високу масштабованість і дозволяє збирати дані логів від додатків і платформ, а також індивідуальні логи з середовищ GKE, віртуальних машин та інших сервісів як всередині, так і поза межами Google Cloud. За допомогою Log Analytics, інтегрованого з BigQuery, користувачі отримують розширені можливості для проведення аналізу, усунення неполадок, забезпечення безпеки та отримання бізнес-інсайтів.

Cloud Monitoring забезпечує нагляд за продуктивністю, доступністю та загальним станом хмарних додатків. Це інструмент для збору метрик, подій і мета-даних із сервісів Google Cloud, перевірки доступності, інструментування додатків і багатьох компонентів. Дані можна відображати на графіках і панелях, а також налаштовувати сповіщення для отримання повідомлень у разі відхилення метрик від очікуваних значень.

Managed Service for Prometheus – це повністю кероване рішення для моніторингу, сумісне з Prometheus, побудоване на основі глобально масштабованого сховища даних, яке використовується також у Cloud Monitoring. Це дає змогу зберігати існуючі сервіси для візуалізації, аналізу та сповіщення, адже дані можна запитувати за допомогою PromQL або Cloud Monitoring [24].

3.3.5 Розподілені методи моніторингу

Розподілені системи моніторингу здатні обробляти величезні обсяги даних у реальному часі, що є їхньою значною перевагою, особливо коли обчислювальні навантаження швидко зростають. Вони розподіляють обробку

даних між кількома вузлами, що допомагає знижувати затримки, покращувати надійність і забезпечувати масштабованість моніторингової інфраструктури.

Платформа ELK Stack – це комплексне рішення для аналізу логів, побудоване на трьох складових із відкритим кодом: Elasticsearch, Logstash і Kibana. Це рішення створене для подолання ключових проблем у сфері управління логами та аналізу даних, забезпечуючи надійний та масштабований підхід до збору та обробки інформації [25].

Elasticsearch – це розподілений пошуковий механізм, розроблений на базі Apache Lucene. Його можна безкоштовно завантажувати, використовувати та змінювати. Цей інструмент дозволяє масштабуватися горизонтально, забезпечує надійність і підтримку одночасного використання для пошуку в реальному часі.

Доступ до функцій Elasticsearch здійснюється через RESTful API у форматі JSON. Він використовує технологію, яка автоматично індексує дані без потреби знати їхню структуру заздалегідь, що забезпечує швидкий пошук текстових та інших даних, легко інтегрується з хмарними інфраструктурами.

Це система, яка працює на кількох серверах, може масштабуватись і залишатися доступною навіть при великих навантаженнях. Вона дозволяє шукати і аналізувати дані в режимі реального часу. Містить потужний інтерфейс (API), який дає змогу виконувати різні завдання, такі як пошук на кількох мовах, визначення геолокації, автозаповнення запитів, підказки типу «напевно, ви мали на увазі» і фрагменти результатів.

Logstash – це інструмент для обробки даних, який допомагає збирати, аналізувати і парсити як структуровані, так і неструктуровані дані з різних джерел. Він працює як конвеєр, який може підключатися до численних систем за допомогою плагінів і гнучко обробляти дані для їх подальшого використання, наприклад, збереження в файлах, виведення на консоль або відправка в Elasticsearch.

Logstash дозволяє створювати конвеєри для централізованої обробки

даних, об'єднуючи інформацію з різних джерел у єдиний формат за допомогою плагінів для введення та виведення даних. Підтримує кастомні формати логів: може обробляти логи з різних додатків, навіть якщо вони мають специфічний формат, і має можливість створювати власні фільтри або використовувати вже готові рішення для парсингу даних.

Kibana – це платформа для візуалізації даних з відкритим кодом, ліцензована за Apache 2.0, допомагає створювати візуальні представлення даних, збережених в індексах Elasticsearch. Від базового бізнес-аналізу до візуалізації в реальному часі, Kibana допомагає презентувати дані у вигляді гістограм, географічних карт, кругових діаграм, графіків, таблиць тощо. Kibana забезпечує зручний інтерфейс для аналітики та візуалізації даних, підтримуючи аналіз у реальному часі, побудову графіків і підсумовування. Інтерфейс простий у використанні та дозволяє налаштовувати панелі за допомогою функцій перетягування та вирівнювання. Користувачі можуть зберігати та управляти кількома панелями одночасно, а також ділитися ними та інтегрувати в інші системи. Крім того, Kibana дозволяє швидко створювати знімки логів для перевірки та ізолювання проблемних транзакцій.

Як працює стек ELK: Logstash збирає, перетворює та надсилає дані в потрібний пункт призначення. Elasticsearch індексує та аналізує зібрані дані та здійснює пошук у них. Kibana надає візуалізацію результатів аналізу [26]. Процес роботи стеку показано на рисунку 3.3

У звичайному процесі обробки даних в стеку ELK логи з різних серверів додатків передаються через Logstash на центральний Logstash indexer. Цей індексатор виводить дані в кластер Elasticsearch, який потім використовується для запитів через Kibana для створення візуалізацій та побудови інформаційних панелей, що дають змогу аналізувати дані логів.

Logstash діє як посередник, збираючи й обробляючи логи з різних джерел, а Elasticsearch забезпечує швидкий пошук і індексацію, тоді як Kibana дозволяє користувачам зручно візуалізувати та взаємодіяти з даними,

створюючи інформативні дашборди для аналізу.

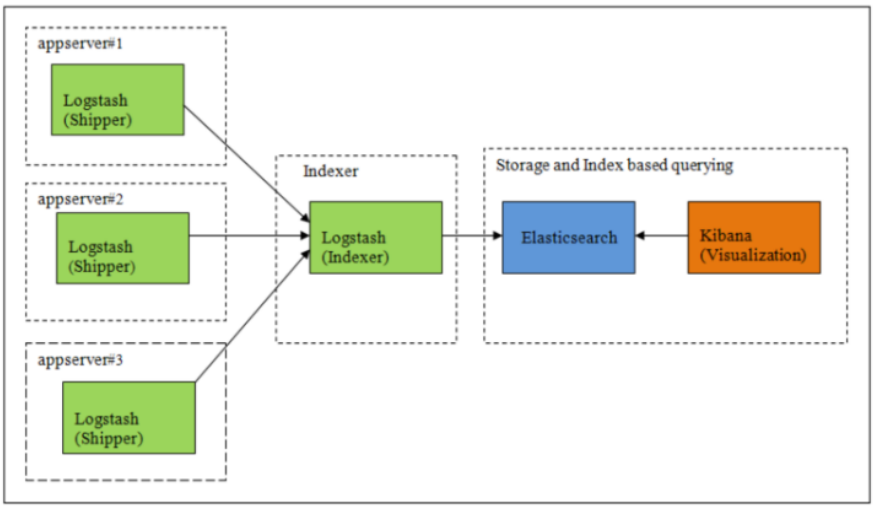

Рисунок 3.3 – Процес обробки даних в ELK

3.3 Висновки за розділом

У розділі було описано основні підходи до моніторингу ключових показників ефективності (KPI) серверів та обчислювальних систем, які є важливими для забезпечення стабільності сучасної IT-інфраструктури. Було визначено, що моніторинг ресурсів включає відстеження таких метрик, як завантаження процесора, використання оперативної пам’яті, вільне місце на диску, активність введення/виведення та тривалість безвідмовної роботи. Встановлення порогових значень для цих показників є важливим для своєчасного реагування на потенційні загрози.

У ході аналізу були розглянуті агентний та безагентний методи моніторингу. Агентний підхід забезпечує детальнішу інформацію, проте вимагає додаткового програмного забезпечення, тоді як безагентний є менш інвазивним, але має обмежену функціональність. Вибір методу залежить від специфіки інфраструктури та вимог до інтеграції. Наведені приклади реалізації, такі як System Center Operations Manager (SCOM), Nagios та Prometheus, продемонстрували різні можливості цих підходів. Особливу увагу було приділено гібридному моніторингу, який дозволяє об’єднувати

локальні та хмарні середовища для централізованого спостереження.

Окремо розглянуто Cloud Logging – технологію, яка використовується для централізованого збору, збереження та аналізу логів з різних компонентів обчислювальних систем, включаючи хмарні сервіси. Цей підхід дозволяє інтегрувати логи з різних джерел у єдине середовище, автоматизувати виявлення аномалій та спрощує діагностику.

Також було досліджено розподілені методи моніторингу, які є критично важливими для великих систем із мікросервісною архітектурою. Ці методи дозволяють збирати дані з окремих вузлів або сервісів і аналізувати їх у реальному часі.

Виявлено основні виклики впровадження моніторингу у гетерогенних системах. Серед них – несумісність форматів даних, високе навантаження на обчислювальні ресурси та необхідність уніфікації даних. Разом із тим, ефективний моніторинг KPI, зокрема із застосуванням Cloud Logging та розподілених методів, сприяє стабільній роботі серверних систем, дозволяючи мінімізувати ризики збоїв, забезпечити продуктивність та оптимізувати використання ресурсів.

4 ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА ЗАПРОПОНОВАНОГО МЕТОДУ РОЗПОДІЛУ ОБЧИСЛЮВАЛЬНИХ ЗАВДАНЬ

4.1 Вступ до розділу

Розподіл процесів на віртуальні машини є складною комбінаторною задачею, яка належить до класу NP-повних задач. Її складність полягає в необхідності оптимального розташування великої кількості процесів з урахуванням обмежених ресурсів віртуальних машин (процесор, пам’ять, мережеві ресурси) та специфічних умов, таких як відповідність тегів і мінімізація сумарної вартості володіння інфраструктурою. Оскільки кількість можливих розташувань процесів зростає експоненційно зі збільшенням кількості процесів і віртуальних машин, прямий перебір усіх можливих рішень є неможливим для великих задач через обчислювальні обмеження.

Задача статичного розподілення процесів на віртуальні машини тісно пов’язана із задачею пакування в контейнери (Bin Packing Problem), яка є однією з найвідоміших задач комбінаторної оптимізації. Вперше задачі такого типу почали досліджуватися ще в 1930-х роках, і з тих пір було розроблено широкий спектр теоретичних і практичних підходів. Ця область досліджень залишається актуальною і сьогодні [27].

У класичній задачі пакування в контейнери (Bin Packing Problem) необхідно розмістити набір об’єктів із різними розмірами у контейнери так, щоб кількість використаних контейнерів була мінімальною [28]. У випадку задачі розподілу процесів, об’єктами є процеси, а контейнерами — віртуальні машини з обмеженими ресурсами. Як і в задачі пакування, кожен процес має власні характеристики, що визначають його вимоги до ресурсів (наприклад, використання процесора, пам’яті та мережі), а кожна віртуальна машина має обмежену місткість цих ресурсів.

Однак розподіл процесів додає ще більшу складність через додаткові

умови. Наприклад, процеси з певними тегами можуть бути розміщені тільки на віртуальних машинах із відповідними тегами, що значно ускладнює можливість їх оптимального розташування. Крім того, у задачі ставиться мета мінімізувати сумарну вартість володіння інфраструктурою, що вимагає врахування фінансових параметрів віртуальних машин та пошуку компромісу між продуктивністю й витратами.

Оскільки задача є NP-повною, знаходження точного оптимального розв’язання у загальному випадку є обчислювально неефективним. У зв’язку з цим застосовуються евристичні та метагевристичні підходи, які дозволяють знайти наближені розв’язання в розумний час. Зокрема, для початкового розподілу процесів можуть використовуватися жадібні алгоритми, які базуються на принципі поступового заповнення віртуальних машин процесами до досягнення обмеження у 70% ресурсів.

Таким чином, задача розподілу процесів на віртуальні машини є розширенням класичної задачі пакування в контейнери з додатковими обмеженнями. Її складність та значущість в умовах сучасних хмарних обчислень роблять її важливим об’єктом дослідження, особливо з огляду на необхідність знаходження ефективних та швидких алгоритмів для вирішення реальних задач із великим масштабом даних. Графік кількості статей на тему проблеми можна представлено на рисунку 4.1

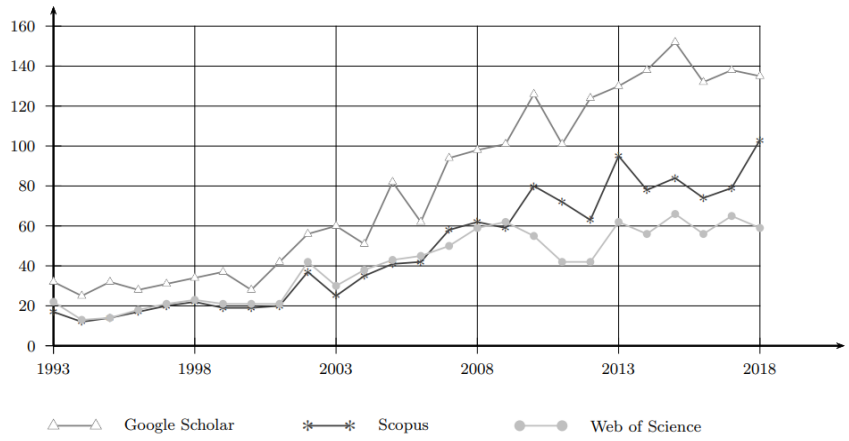

Рисунок 4.1 – Графік кількості статей на тему Bin Packing Problem та обмеженої кількості ресурсів

4.2 Підходи та алгоритми розподілу процесів на віртуальні машини

У задачі розподілу процесів на віртуальні машини можна застосовувати два основні типи алгоритмів. Онлайн-алгоритми дозволяють призначати процеси до віртуальних машин в реальному часі, без попереднього знання усіх вхідних даних, що робить їх корисними для динамічного розподілу навантаження. Офлайн-алгоритми, в свою чергу, потребують попереднього аналізу всіх процесів і доступу до їх характеристик, що дозволяє більш ефективно оптимізувати розподіл та знижувати витрати ресурсів.

Для вирішення задачі підходять жадібні алгоритми, які працюють за принципом «краще зараз», тобто на кожному кроці вибирається найбільш вигідне рішення, яке здається оптимальним у цей момент [29]. Однак таке підходить не завжди, оскільки глобальне оптимальне рішення може бути досягнуте лише за допомогою певних компромісів між окремими кроками, а жадібний алгоритм їх не враховує.

Одним із простих і ефективних підходів є алгоритм First Fit. Цей алгоритм намагається запакувати процес у першу доступну ВМ, яка здатна вмістити цей процес, як тільки знайдеться підходяща, алгоритм припиняє пошук. Якщо жодна з наявних ВМ не може вмістити процес, то алгоритм відмовляється від розміщення цього процесу. Перевагою First Fit є його простота і швидкість реалізації. Однак, його недоліком є те, що він може не забезпечувати оптимального використання ресурсів ВМ, оскільки може заповнити ВМ не повністю, залишаючи великі невикористані ділянки.

Іншим можливим варіантом є алгоритм Best Fit. Цей алгоритм пакує процес у ту ВМ, яка залишає найменше вільного простору після його розміщення, що забезпечує більш рівномірний розподіл навантаження на ВМ. Зазвичай цей підхід є ефективнішим за First Fit, оскільки дозволяє краще використовувати доступні ресурси, мінімізуючи залишкові простори у ВМ. Однак, Best Fit є дещо складнішим у реалізації та може вимагати використання додаткових структур даних для ефективного пошуку.

Ще одним підходом є алгоритм Worth Fit, який враховує не лише відповідність ресурсів, але й вартість використання кожної ВМ. Алгоритм визначає «вартісний показник» для кожної ВМ, який залежить від її вартості, і обирає ту машину, яка мінімізує співвідношення між витраченими ресурсами та вартістю. Цей підхід дозволяє оптимізувати фінансові витрати, забезпечуючи економічно ефективний розподіл процесів. Недоліком Worth Fit є можливість фрагментації ресурсів через фокусування на мінімізації вартості, а не на ефективності використання ресурсів. Такий підхід є найбільш корисним у середовищах, де витрати мають ключове значення.

First Fit Decreasing та Best Fit Decreasing є жадібними алгоритмами з найгіршою ефективністю 3/2. Це означає, що у найгіршому випадку кількість бінів, яку використовують ці алгоритми, не перевищить 1,5 від оптимального рішення, що є найкращою досяжною ефективністю для поліноміальних наближувальних алгоритмів.

Що стосується питання поліноміальних наближувальних алгоритмів для задачі упаковки в бін, то існує важливий теоретичний висновок: неможливо побудувати поліноміальний наближувальний алгоритм з ефективністю меншою ніж 3/2, якщо тільки не доведено, що P = NP. Це твердження базується на тому, що існує класична NP-повна задача Partition, яка полягає в тому, щоб розділити множину чисел на дві підмножини з однаковими сумами. Проблема Partition є NP-повною, і існує теоретичний зв'язок між цією задачею та задачею упаковки в бін.

Якщо б існував поліноміальний наближувальний алгоритм для задачі упаковки в бін з ефективністю меншою ніж 3/2, то це дозволило б вирішити задачу Partition за поліноміальний час. Такий алгоритм міг би бути перетворений на розв'язок для Partition, що суперечило б припущенню про NP-повноту цієї задачі, оскільки Partition не може бути вирішено за поліноміальний час (якщо тільки $P \neq NP$). Тому, якщо б існував такий алгоритм для BPP, це означало б, що P = NP, що є відкритим питанням у теорії складності [27].

Так як задача передбачає розподіл процесів, що належать до різних класів (наприклад, класи обробки, типи навантаження тощо), то більш підходящим може бути використання алгоритму Class Constrained Bin Packing Problem (CCBP). Цей алгоритм враховує класові обмеження на розміщення процесів: кожен клас процесів може бути оброблений тільки на ВМ, що підтримує цей клас.

Серед особливостей поставленої задачі також будуть: розподіл за розкладом, стримінгові процеси, рестарти. Оскільки деякі процеси виконуються за розкладом, алгоритм повинен враховувати час виконання і забезпечити ресурси в цей період. Це важливо, щоб уникнути перевантаження ВМ. А для стрімінгових процесів, які постійно споживають ресурси, алгоритм має враховувати їх постійне навантаження на ресурси. Оскільки ці процеси не призводять до тривалих перерв, алгоритм повинен ефективно балансувати їх наявність у часі.

4.3 Характеристики вхідних даних

Для проведення дослідження було сформовано набір характеристик, які будуть належати кожній з віртуальних машин та кожному з процесів. Основні характеристики віртуальних машин наведено в таблиці 4.1.

Таблиця 4.1 – Характеристики віртуальної машини

Назва параметру	Характеристика
1	2
Кількість vCPU	Кількість ядер процесора, віртуальні еквіваленти фізичним ядрам
Обсяг оперативної пам'яті (RAM)	Визначає кількість пам’яті, в GB
Пропускна здатність мережі	Виділяється для кожного ядра процесора (vCPU). EgressBandwidth множиться на кількість ядер, в Gbps

Продовження таблиці 4.1

1	2
Локація	Позначає, де фізично знаходиться машина. GCP (Google Cloud Platform) – хмарна платформа, Host – виділений хостинг.
Вартість	Для GCP – розраховується на основі кількості використаних vCPU, RAM за формулою 3.5. У разі хостингу вартість є фіксованою та додається вартість вихідного трафіку в GCP за формулою 3.4.

Для процесів характеристики виглядають так, щоб враховувалися у віртуальних машинах, вони вказані в таблиці 4.2.

Таблиця 4.2 – Характеристики процесу

Назва параметру	Характеристика
1	2
Швидкодія CPU	Кількість обчислювальних ресурсів (в умовних одиницях)
Обсяг оперативної пам'яті (RAM)	Кількість оперативної пам’яті, необхідної процесу (в GB)
Обсяг зовнішніх даних	Окремо обсяг даних, який отримується з ресурсів, які розташовано в GCP, і окремо з будь-яких інших ресурсів, які не розташовані в GCP (в Gbps)
Тип розкладу виконання	Стрімінговий, який виконується постійно, або за рокладом, який визначено CRON-виразом
CRON-вираз	Вказується, якщо тип запуску процесу – за розкладом
Тривалість виконання	Вказується для тих, які виконуються за розкладом (в хвилинах)

Продовження таблиці 4.2

1	2
Ймовірність отримання помилки	Для стрімінгових не має сенсу, оскільки вони одразу рестартують, для інших виконуються до 4 ретраїв. Для рестартів використовується експоненційно збільшуваний період.

Обрані характеристики дають змогу точно описати ресурси, що виділяються для кожної машини, і забезпечити баланс між продуктивністю, ефективністю використання ресурсів та витратами.

Характеристики віртуальних машин дозволяють враховувати як технічні обмеження, так і економічну складову. Наприклад, розрахунок пропускної здатності, прив’язаний до кількості ядер, дозволяє відображати реальну залежність між обчислювальною потужністю та швидкістю обміну даними. Вибір між хмарним розміщенням (GCP) та виділеним хостингом надає гнучкість у розрахунку вартості та ресурсів.

Для процесів характеристики відображають їхню функціональність та потреби у ресурсах. Зокрема, швидкодія CPU та оперативна пам'ять дозволяють визначити, скільки ресурсів необхідно для виконання завдання. Обсяг зовнішніх даних поділений на джерела, розташовані в GCP та поза ним, що важливо для точного врахування мережевих витрат у загальну вартість інфраструктури. Тип розкладу та CRON-вираз забезпечують деталізацію режиму роботи процесів, а тривалість виконання допомагає розрахувати загальне навантаження на систему. Особливий акцент на ймовірності помилки та механізмі рестарту процесів робить систему більш стійкою до збоїв, враховуючи можливість поступового збільшення часу між повторними спробами. CRON-вираз, тривалість роботи та ймовірність помилок вказується для процесів, які виконується за розкладом, бо для постійних процесів ця інформація неактуальна.

4.4 Реалізація алгоритмів розподілу

4.4.1 Еталонний алгоритм

Так як одним із підходів до вирішення таких задач є застосування алгоритму повного перебору (Brute Force), застосуємо його для невеликої кількості процесів та машин. Алгоритм повного перебору здійснює генерацію всіх можливих комбінацій розподілу процесів між наявними віртуальними машинами. Для кожної комбінації він перевіряє:

- чи вистачає ресурсів на віртуальних машинах для виконання всіх призначених завдань;
- рівень завантаженості CPU, RAM та мережевих ресурсів;
- загальну вартість використання машин.

Кількість комбінацій розподілу визначається за формулою:

$$C = m^n , \tag{4.1}$$

де C – загальна кількість комбінацій;

m – кількість машин;

n – кількість процесів.

Кожен процес може бути призначений одній із m машин. Таким чином, для одного процесу є m варіантів. Для n процесів всі варіанти комбінуються, що призводить до $m \times m \times \cdots \times m = mn$ комбінацій. Кількість комбінацій C має експоненційну складність. Це означає, що час виконання алгоритму зростає дуже швидко зі збільшенням кількості процесів n або машин m. Наприклад, для чотирьох процесів і двох машин кількість комбінацій становить 16, але вже для десяти процесів це число зростає до 1024. При збільшенні числа процесів до 20 кількість комбінацій досягає понад мільйон, що унеможливлює практичне використання цього методу в системах із великими обсягами даних. Тому в реальних сценаріях він використовується переважно

для тестування та порівняння ефективності інших підходів. Наприклад, отримані результати повного перебору можуть використовуватися як контрольні дані для перевірки, наскільки наближеними є результати жадібних алгоритмів або алгоритмів із використанням евристик. Завдяки цьому можна визначити, наскільки ефективно ці алгоритми знаходять рішення в умовах, де повний перебір стає обчислювально недоцільним.

Реалізуємо алгоритм повного перебору як еталонний метод для визначення оптимального розподілу. Оскільки він забезпечує точний розрахунок найкращого варіанта. Зокрема, повний перебір дозволяє точно визначити мінімальну вартість використання інфраструктури, рівномірність розподілу навантаження та кількість необхідних віртуальних машин. Такі дані є важливими для калібрування та перевірки інших алгоритмів, які можуть не завжди знайти глобально оптимальне рішення через обмеження часу або обчислювальних ресурсів. Він забезпечує гарантію того, що знайдене розв’язання відповідає глобальному мінімуму за критерієм сумарної вартості.

Створено вибірку з 2 віртуальних машин з різною конфігурацією, одна з них знаходиться у GCP, інша на орендованій інфраструктурі іншого провайдера. Обрано різні локації, бо якщо машина знаходиться за межами GCP і отримує дані з ресурсів GCP, це класифікується як вихідний трафік, який враховується потім в загальну вартість машин. Для розрахунку швидкодії CPU кількість vCPU множиться на фіксоване число: 2000. Параметри зазаначені у таблиці 4.3.

Таблиця 4.3 – Конфігурації віртуальних машин для експерименту

Характеристика	VM 1	VM 2
1	2	3
Локація	GCP	Host
Кількість vCPU	8	4

Продовження таблиці 4.3

1	2	3
RAM, GB	16	8
Пропускна здатність мережі, Gbps	16 (2 Gbps * vCPU)	8 (2 Gbps * vCPU)
Вартість	$ 246	$80 + вартість отримання даних з GCP

Для тестування було обрано 4 процеси, кожен із яких представляє різний тип завдань з урахуванням специфіки використання ресурсів і особливостей розкладу виконання. Два з них виконуєюься в режимі реального часу, виконуючи задачі типу стримінгу, наприклад, трансляції даних або потокове оброблення інформації. Один з процесів запускається кожні 2 години та виконується протягом 15 хвилин. Цей процес виконує завдання для виконання проміжних обчислень. Ще один процес запускається кожного дня о 9:00 та виконується протягом 60 хвилин. Цей процес призначений для резервного копіювання або щоденних звітів. Характеристики подані у таблиці 4.4.

Таблиця 4.4 – Параметри процесів для експерименту

Параметр	Процес 1	Процес 2	Процес 3	Процес 4
1	2	3	4	5
Швидкодія CPU	1000	2100	1200	2000
Обсяг оперативної пам’яті (RAM), GB	2	1	1	2
Обсяг мережі, Gbps	2.5	0.5	1	2
Обсяг зовнішніх даних, Gbps	1	-	-	-

Продовження таблиці 4.4

1	2	3	4	5
ScheduleType	Постійний	За розкладом	За розкладом	Постійний
CronExpression	-	0 */2 * * *	0 9 * * *	-
ExecutionTime (хвилини)	-	15	60	-

Ці процеси були відібрані для моделювання сценаріїв, які охоплюють широкий спектр завдань: від стримінгових додатків до періодичних обчислень і звітності. Це дозволяє оцінити, наскільки ефективно алгоритм розподілу може мінімізувати витрати, раціонально використовувати ресурси.

Для того, щоб визначити, чи може той чи інший процес бути розташований на машині, потрібно розробити алгоритм. Головною умовою розміщення процесів на машинах є те, що в будь-який часу сума споживаного ресурсу процесору, пам'яті та мережевого ресурсу не буде перевищувати 70% від ресурсу віртуальної машини. Щоб розмістити якнайбільше процесів на одну машину та тим самим зменшити загальну вартість володіння інфраструктурою, потрібно враховувати особливість процесів, які запускаються за розкладом. Можна було б просто брати їх характеристики та додавати до загальної суми зайнятих ресурсів, але це зовсім неефективне використання ресурсів. Бо може бути розміщено декілька процесів за розкладом, вони в один момент часу виконаються, а далі машина буде простоювати, відповідно неефективно витрачаються кошти. Тому розроблено механізм, який не враховує характеристики процесу до загальної суми, а створює список інтервалів, може бути задано на день/тиждень/місяць/рік. В нашому прикладі тривалість інтервалу – 10 хвилин, для кожного інтервалу зберігається інформація про використання ресурсів. І потім кожен процес додає своє навантаження до відповідних інтервалів, у які він виконується. Початок і кінець виконання процесу визначається CRON-виразом та тривалістю роботи. Загальну картину

завантаження ресурсами можна побачити, якщо додати максимальне значення відповідного ресурсу зі списку інтервалів до суми завантаження стрімінговими процесами. Інтервали дозволяють чітко визначити, чи перетинається процес із іншими завданнями у часових рамках, уникнути конфліктів та перевантаження.

Для розміщення стрімінгових процесів, які працюють постійно, обчислюється сума використовуваних ресурсів разом із поточними навантаженнями машини. Якщо всі ці показники не перевищують 70%, то процес може бути розміщений. Для завдань по розкладу: визначаються інтервали, перевіряються перетини з іншими процесами і сукупне використання ресурсів у цьому інтервалі.

Перейдемо до результатів роботи алгоритму повного перебору всіх варіантів розміщення з вказаними даними для експерименту. Було оцінено 16 можливих комбінацій розподілу процесів між двома віртуальними машинами. Краща комбінація виявилася такою: усі процеси були розміщені на одній машині (VM 2). Завантаження ресурсів машини залишилося в межах допустимих значень: CPU – 63,75%, RAM – 62,5%, мережа – 68,75%. Для наочності використання ресурсів наведено на рисунку 4.2

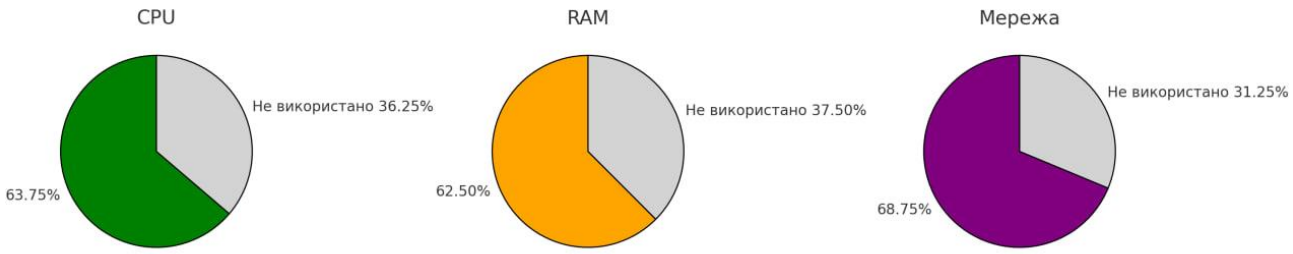

Рисунок 4.2 – Використання ресурсів після розподілу

Машина VM 1 не була використана, оскільки всі процеси змогли розміститися на VM 2. До оптимізації загальна вартість використання двох машин становила $326.12. Після оптимізації вартість знизилася до $80.12, оскільки одна машина була виключена з використання. Загальна економія:

$246 (показано на рисунку 4.3). Процеси, які виконуються за розкладом, вдалось суміщати із стрімінговими та іншими процесами за розкладом, уникнувши перевантаження машини.

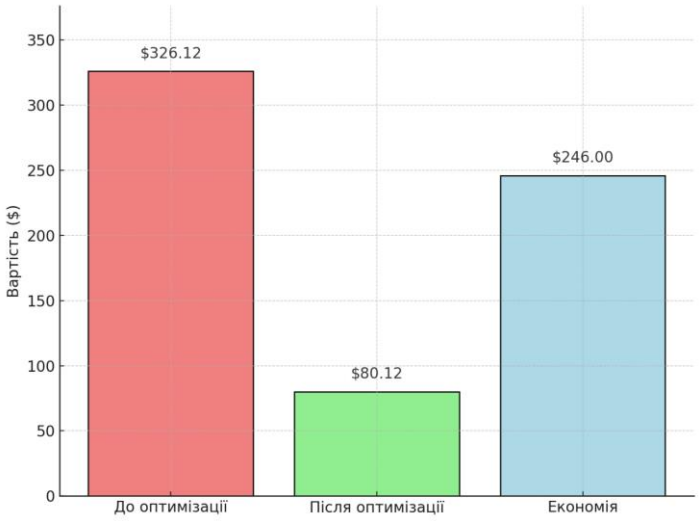

Рисунок 4.3 – Економія витрат на віртуальні машини

Для того, щоб покращити навіть такий варіант алгоритму, кількість операцій якого невпинно буде зростати, запропоновано використовувати теги. Вони використовуються для класифікації та позначення специфічних характеристик як процесів, так і машин. Завдяки тегам можна впроваджувати політики відповідності між ресурсами машин і вимогами процесів [30]. Теги дозволяють групувати процеси за типами завдань, наприклад: HighPerformance – для процесів, які потребують високої продуктивності, BatchProcessing – для пакетної обробки даних, RealTime – для процесів із низькою затримкою. Машини також повинні мати аналогічні теги, що визначають їх здатність обслуговувати певні типи завдань. Це запобігає невідповідності вимог процесів і можливостей машин, забезпечуючи стабільність роботи. Те, що нам потрібно: теги зменшують кількість перевірок для розміщення процесів, оскільки не потрібно аналізувати всі можливі комбінації, алгоритм одразу виключає машини, які не мають потрібних тегів.

Ще серед переваг використання тегів є те, що вони дозволяють швидко

адаптувати систему до змін. Наприклад, додавання нового типу процесів або ресурсів може бути реалізовано шляхом додавання нових тегів. Таком можна таким чином управляти пріорітетністю розміщення процесів. Наприклад, процеси з тегом Critical отримують пріоритет.

В алгоритм це впроваджено наступним чином: першим етапом при розміщенні процесу на машині є перевірка, чи містять теги машини всі теги процесу. Для цього виконується порівняння списків тегів. Якщо теги не збігаються, процес не може бути розміщений на цій машині.

Для проведення експерименту на тих же даних, було додано 2 різних тега для машин та процесів: HighPerformance та BatchProcessing. Тобто на такому невеликому наборі даних не складно здогадатися, що буде всього 1 варіант розподілу, так як 2 процеси з тегом HighPerformance та 2 процеси з тегом BatchProcessing, і кожна машина має свій тег. Це назвемо строге тегування. Для нестрогого тегування будуть або комбінації тегів, або декілька машин з однаковими тегами.

Для різних варіантів розподілу проведено дослідження та сформовано діаграму кількості комбінацій – рисунок 4.4. Було взято такі варіанти:

- 15 процесів з тегом HighPerformance, 2 машини з тегом HighPerformance (в цьому випадку теги не впливають).

- 10 процесів з тегом HighPerformance, 5 процеси з тегом BatchProcessing, 2 машини з тегом HighPerformance, 1 машина з тегом BatchProcessing;

- 5 процесів з тегом HighPerformance, 5 процеси з тегом BatchProcessing, 5 процесів з тегом RealTime, 1 машина з комбінацію тегів HighPerformance, BatchProcessing та 1 машина з тегом RealTime (те ж саме, що і строге тегування);

- 5 процесів з комбінацією тегів HighPerformance та BatchProcessing, та 10 процесів з тегом RealTime, 1 машина з тегом HighPerformance, 1 машина з тегом BatchProcessing, 1 машина з RealTime:

- строге тегування(кількість машин = кількість тегів), 10 процесів з

тегом HighPerformance, 5 процесів з тегом BatchProcessing, 1 машина з тегом HighPerformance, 1 машина з тегом BatchProcessing;

- 5 процесів з тегом HighPerformance, 5 процеси з тегом BatchProcessing, 5 процесів з тегом RealTime, 1 машина з комбінацію тегів HighPerformance, BatchProcessing та 1 машина з тегом RealTime (те ж саме, що і строге тегування).

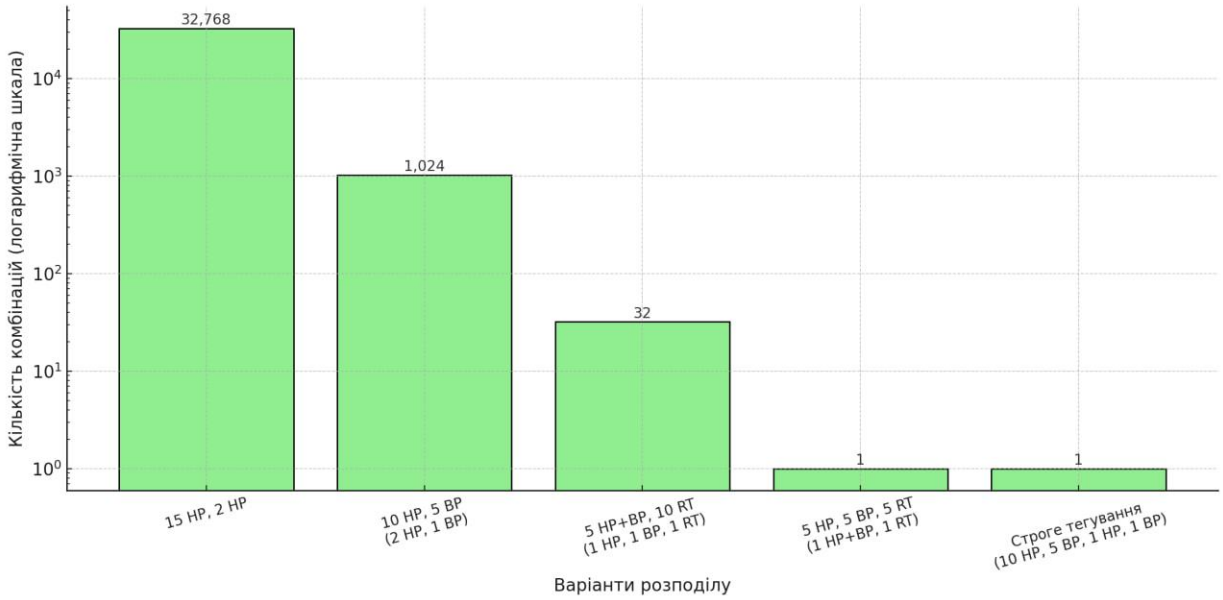

Рисунок 4.4 – Діаграма кількості комбінацій для різних варіантів тегування

Додавання тегів до процесів і машин дозволило суттєво зменшити кількість можливих комбінацій, які аналізує алгоритм, що, у свою чергу, значно спростило та пришвидшило процес розрахунку. Результати експерименту продемонстрували, що всі процеси були успішно розподілені відповідно до їхніх вимог і характеристик. Це стало можливим завдяки попередньому визначенню, які машини можуть приймати конкретні процеси, виходячи з відповідності їхніх тегів.

4.4.2 Реалізація жадібних алгоритмів

Для розв'язання задач розподілу процесів між віртуальними машинами

спробуємо використовувати жадібні алгоритми [31] з модифікаціями (сортування, використання тегів, обмежень), такі як: First Fit, Best Fit, Worth Fit. Ці алгоритми дозволяють скоротити кількість операцій у порівнянні з повним перебором, але не гарантують глобально оптимального рішення. Вони мають швидший час виконання, оскільки працюють за локальним принципом: обробляють кожен процес або машину незалежно, приймаючи рішення на основі поточного стану.

Жадібні алгоритми базуються на ітеративному підході до розподілу процесів між машинами:

- First Fit – процеси обробляються у порядку надходження. Кожен процес розміщується на першій доступній машині, яка має достатньо ресурсів для його виконання. Як тільки відповідна машина знайдена, процес їй призначається без подальшої перевірки інших машин;
- Best Fit – кожен процес намагається бути розміщеним на машині, яка залишить найменший обсяг невикористаних ресурсів після його призначення. Це дозволяє зменшити фрагментацію ресурсів;
- Worth Fit – для кожного процесу обирається машина, на якій співвідношення витрачених ресурсів до вартості машини буде найвигіднішим, що дозволяє мінімізувати витрати.

Сортування це додаткова оптимізація та крок до покращення ефективності алгоритму. Для алгоритма First Fit можна використати сортування за ресурсними вимогами. У Best Fit машини можуть бути відсортовані за залишковими ресурсами, що забезпечує оптимальне використання потужностей, а процеси – за максимальним використанням ресурсів. У Worth Fit сортування може враховувати вартість машин, дозволяючи мінімізувати загальні витрати.

Для реалізації жадібного алгоритму First Fit з сортуванням було обрано підхід, який оптимізує процес розподілу шляхом зменшення кількості операцій і спрощення логіки розміщення процесів на віртуальних машинах. Основна ідея цього методу полягає в тому, щоб спочатку відсортувати всі

процеси за їхніми ресурсними вимогами у спадному порядку, а потім послідовно розміщувати кожен процес на першій доступній машині, яка має достатньо ресурсів. Завдяки сортуванню спочатку обробляються найбільш «складні» процеси, що знижує ризик їхньої відмови у розподілі.

Реалізація Best Fit відрізняється від попередньо розглянутого основним підходом до вибору віртуальної машини для процесу. У First Fit процес розподілу зупиняється на першій віртуальній машині, яка може розмістити процес. Це швидше, але не завжди оптимально, оскільки машина, яка підходить найкраще, може залишитися незавантаженою. У Best Fit процес шукає машину з мінімальним залишковим ресурсом після розміщення процесу. Це означає, що алгоритм намагається оптимізувати використання ресурсів кожної машини, зменшуючи фрагментацію ресурсів.

Алгоритм Worth Fit відрізняється від інших жадібних алгоритмів тим, що його головним критерієм вибору віртуальної машини є вартість. Алгоритм перебирає всі доступні віртуальні машини, які можуть розмістити процес. Для кожної машини обчислюється «вартісний показник», який дорівнює вартості машини. Обирається машина з найменшим показником вартості, яка може задовольнити вимоги процесу.

Для порівняння реалізованих алгоритмів наведено таблицю 4.5, яка показує їх позитивні, негативні аспекти та в яких ситуаціях слід використовувати відповідний метод.

Таблиця 4.5 – Порівняльна характеристика жадібних алгоритмів

Алгоритм	Переваги	Недоліки	Для чого
1	2	3	4
First Fit	Швидка реалізація та виконання, мінімальна кількість порівнянь для кожного процесу.	Можлива фрагментація ресурсів, не гарантує ефективного використання машин.	Для задач, де важлива швидкість розподілу.

Продовження таблиці 4.5

1	2	3	4
Best Fit	Оптимізує використання залишкових ресурсів, мінімізує фрагментацію ресурсів.	Більша кількість порівнянь для кожного процесу, не враховує вартість.	Для задач із критично обмеженими ресурсами, де важлива рівномірність завантаження.
Worth Fit	Вибирає найдешевші машини з урахуванням їхньої доступності.	Можлива фрагментація, не враховує ефективність використання ресурсів.	Для задач, орієнтованих на економію коштів, у хмарних середовищах із неоднорідними витратами.

Так як наша задача полягає в тому, щоб мінімізувати розташування процесів за критерієм сумарної вартості володіння інфраструктури, тому за описом більше підходить алгоритм Worth Fit. Але реалізувавши, порівняємо результати розподілу та кількості операцій.

4.5 Аналіз результатів розподілу

Для перевірки ефективності різних підходів до розподілу процесів було проведено експеримент на невеликій кількості даних, що дозволило використати алгоритм повного перебору (Brute Force). Конфігурації віртуальних машин було взято з таблиці 4.3. У таблиці наведено ключові характеристики розподілу:

Таблиця 4.6 – Результати розподілу процесів за допомогою різних алгоритмів

	Brute Force	First Fit	Best Fit	Worth Fit
Кількість операцій	16	5	9	9
Процеси на VM 1	-	1, 2, 3, 4	-	-
Процеси на VM 2	1, 2, 3, 4	-	1, 2, 3, 4	1, 2, 3, 4

Результати показують, що повний перебір розмістив усі процеси на одній машині, досягнувши оптимального використання ресурсів. First Fit також розмістив усі процеси на одній машині, застосувавши меншу кількість операцій, але при цьому обрана машина по вартості дорожче, ніж друга. Best Fit і Worth Fit розмістили процеси так, як це зробив еталонний алгоритм. Можна зробити висновок, що жадібні алгоритми є швидшими, але можуть поступатися повному перебору в точності для невеликих задач. Але додатково процеси та машини можна регулювати сортуванням. Якщо до First Fit ще додати сортування машин по вартості, то цей би алгоритм показав оптимальніший результат.

Було проведено ще один експеримент із більшою кількістю даних, а саме 12 процесів і 2 віртуальні машини, без використання тегів. Результати, відображені в таблиці 4.7, показали, що Brute Force потребував 4096 операцій, тоді як жадібні алгоритми значно зменшили кількість обчислень: First Fit – 14, Best Fit та Worth Fit – по 25. Brute Force забезпечив точне рішення, максимізуючи використання ресурсів, але його обчислювальна складність є неприйнятною для великих задач. Best Fit і Worth Fit продемонстрували дуже схожі результати, при чому вони виявилися найбільш наближеними до повного перебору.

При використанні тегів для обмеження вибору можливих комбінацій кількість операцій у Brute Force знизилася до 156, а у Best/Worth Fit – до 21, що значно покращило швидкість обчислень без втрати точності для жадібних алгоритмів. Це підтверджує доцільність застосування тегів для оптимізації процесу розподілу у великих системах.

Таблиця 4.7 – Результати завантаженості віртуальних машин після розподілу

	Brute Force		First Fit		Best Fit		Worth Fit	
	VM1	VM2	VM1	VM2	VM1	VM2	VM1	VM2
Кількість операцій	4096		14		25		25	
Завантаження CPU	49,06%	25%	55,31%	12,50%	42,81%	37,50%	42,81%	37,50%
Завантаження RAM	57,81%	25%	64,06%	12,50%	45,31%	50,00%	45,31%	50,00%
Завантаження мережі	32,50%	50%	38,75%	18,75%	28,75%	68,75%	28,75%	68,75%

Результати порівняння алгоритмів розподілу процесів на віртуальні машини демонструють суттєві відмінності в ефективності залежно від кількості процесів і машин. Дивлячись на рисунок 4.5, можна побачити, що Brute Force швидко стає нерелевантним при збільшенні розмірності задачі через експоненційне зростання кількості операцій. Алгоритми First Fit, Best Fit та Worth Fit є значно ефективнішими, навіть для великих обсягів процесів і машин, забезпечуючи помірно зростаючу кількість операцій. Для складних конфігурацій із великою кількістю процесів і машин жадібні підходи є оптимальним вибором.

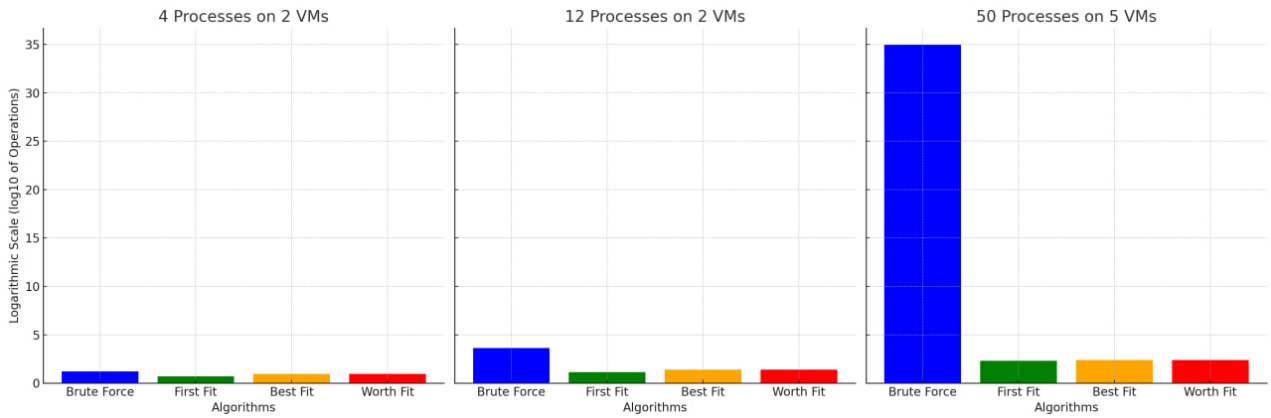

Рисунок 4.5 – Діаграма росту кількості операцій для різних алгоритмів

Оскільки не всі процеси виконуються на постійній основі, а деякі мають розклад та час виконання, у реалізації застосовувалися інтервали, які розбивали заданий період на однакові відрізки, та для кожного інтервалу біли записані свої значення навантаження на ресурси. Їх можна використати не лише для підрахунку найбільшої завантаженості ресурсів за весь час, а ще й для побудови графіків розподілу ресурсів, які наочно покажуть, яким чином буде завантажуватися віртуальна машина за добу.

Графіки на рисунках 4.6, 4.7, 4.8 відображають динамічне використання ресурсів віртуальної машини, яка працює із 100 процесами. Ці процеси розподілено на стрімінгові та такі, що виконуються за розкладом. Для моделювання роботи процесів використано віртуальну машину з характеристиками: максимальна продуктивність CPU становить 100%, максимальний обсяг пам’яті – 64 GB, а пропускна здатність мережі – 20 Gbps. Для всіх ресурсів було встановлено обмеження в 70% від їхніх максимальних значень, тобто 70% для CPU, 44.8 GB для RAM та 14 Gbps для мережі.

У моделюванні враховано 80 стрімінгових процесів, які створюють постійне фонове навантаження на ресурси, і 20 процесів, які виконуються за розкладом. Стрімінгові процеси працюють безперервно протягом доби. Загальне навантаження від стрімінгових процесів додається до всіх часових інтервалів. Процеси за розкладом виконуються в різні моменти часу з випадковою тривалістю від 1 до 120 хвилин. Процеси за розкладом розподіляються по часових інтервалах із перевіркою, щоб їх сумарне навантаження разом зі стрімінговими процесами не перевищувало встановлених лімітів.

На графіках зображено використання кожного ресурсу протягом доби. CPU показано у відсотках від максимальної продуктивності. Для RAM і Network використано абсолютні значення в GB і Gbps відповідно. Червоні пунктирні лінії позначають ліміти для кожного ресурсу: 70% від максимальних значень.

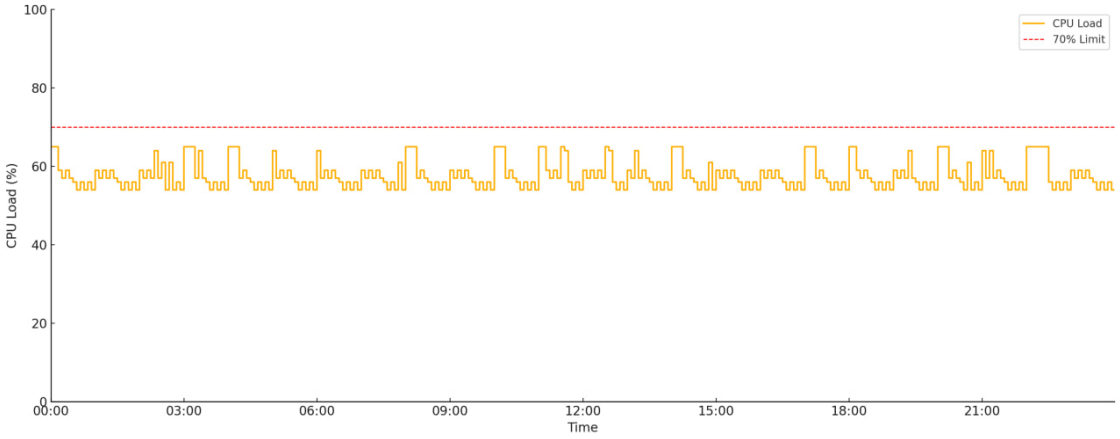

Рисунок 4.6 – Графік розподілу навантаження на CPU за добу

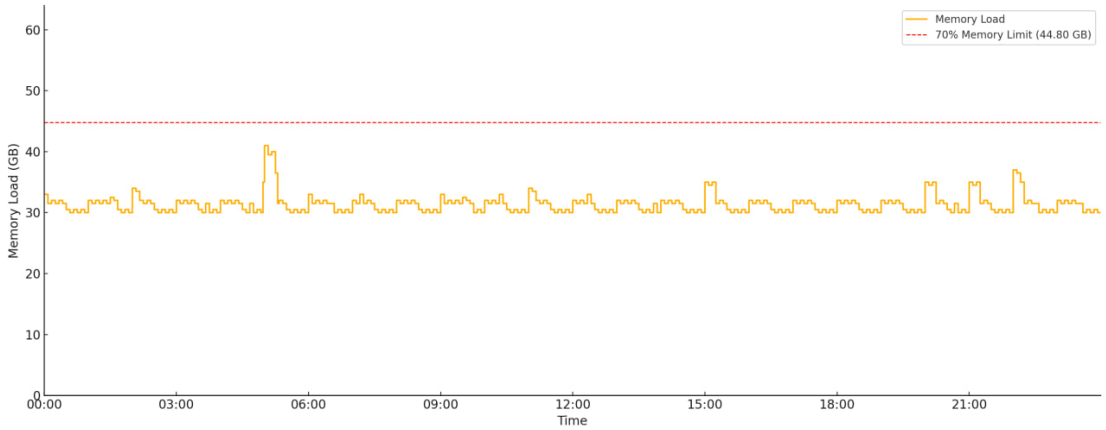

Рисунок 4.7 – Графік розподілу навантаження на пам’ять за добу

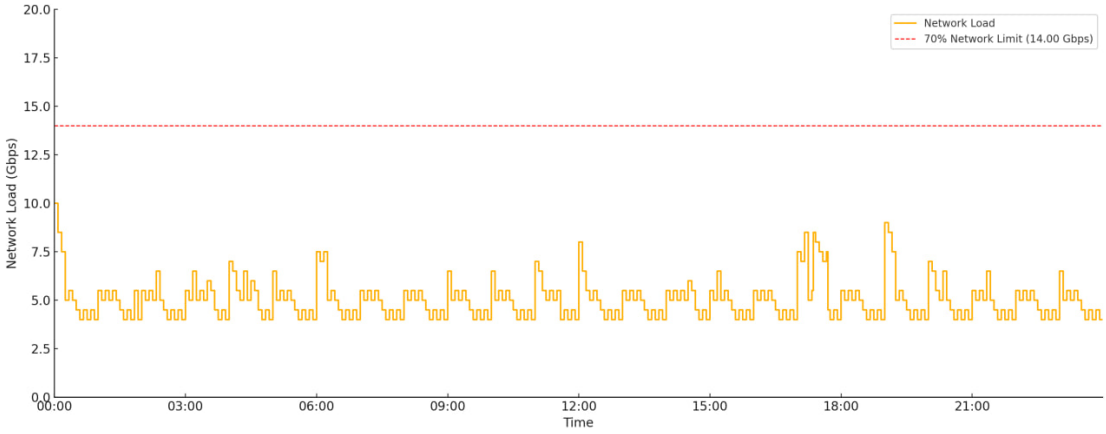

Рисунок 4.8 – Графік розподілу навантаження на мережу за добу

Графіки демонструють, що всі ресурси використовуються в межах встановлених лімітів. Завантаження змінюється протягом доби залежно від активності процесів, але жодного разу не перевищує допустимі значення. Це

підтверджує, що розподіл процесів на віртуальній машині виконано ефективно, і встановлені обмеження дозволяють забезпечити стабільну роботу системи без перевантажень.

4.6 Вирішення задачі оптимізації використання ресурсів

Алгоритм оптимізації розподілу процесів на віртуальні машини реалізовано як набір кроків для аналізу, оптимізації та створення нових машин, видалення невикористаних. Його основна ідея полягає в аналізі поточного використання ресурсів віртуальних машин, визначенні надлишкових чи недостатніх конфігурацій, а також у створенні оптимальних рішень для нерозподілених процесів.

Спочатку всі віртуальні машини аналізуються на наявність активних процесів і рівень використання ресурсів. Якщо машина не має активних процесів, вона визначається як кандидат на видалення. Якщо завантаження машини нижче 70%, алгоритм перевіряє, чи можливо оптимізувати її конфігурацію. Для цього обчислюються нові значення CPU, RAM і пропускної здатності мережі, виходячи із реального завантаження, і підбираються конфігурації з меншою вартістю, що забезпечують необхідну продуктивність.

Для нерозподілених процесів алгоритм обчислює їх загальні потреби в ресурсах, об’єднує їх у групи та створює пропозиції щодо нових віртуальних машин. Алгоритм також розглядає можливість створення кількох маленьких машин або однієї великої для виконання завдань. Розподіл процесів відбувається поступово. Для кожного процесу перевіряється, чи може він бути розміщений на існуючій машині, не перевищуючи обмежень у 70% завантаження ресурсів. Якщо це можливо, процес додається до машини, і її характеристики оновлюються. Якщо процес не можна розмістити, він додається до групи для створення нових машин.

На фінальному етапі всі зміни групуються: машини, які потрібно

видалити, оптимізовані конфігурації, а також нові машини з їхніми характеристиками та вартістю. Це дозволяє створити оптимальну структуру інфраструктури, яка забезпечує виконання всіх процесів із мінімальними витратами.

Проведемо експеримент з 6 віртуальними машинами, на які вже розподілено 56 процесів, але також існує 9 процесів, які не знайшли підходящу машину через специфічний тег High priority. Початкові дані задані у таблиці:

Таблиця 4.8 – Характеристики наявних віртуальних машин

Локація	vCPU	Пам’ять (GB)	Мережа (Gbps)	Теги	Використання (%)	Вартість
Host	4	16	5	General	68	$140.16
GCP	8	32	10	Data-processing	33	$298.7
GCP	16	64	20	General	0	$596.39
Host	2	8	2	AI, ML	70	$80
GCP	16	64	20	Data storage	35	$596.39
Host	8	32	10	General	56	$230.5
Загальна вартість:						$1942.14

Таблиця 4.9 представляє результати роботи програми та наданих рекомендації щодо управління 6 існуючими віртуальними машинами та додавання однієї нової. Вона містить характеристики кожної машини, запропоновані дії, поточне використання ресурсів і пояснення причин для кожної дії, та також вартість після оптимізації. Також для кращого розуміння різниці зображено рисунок 4.9.

З наданих рекомендацій: утримувати машини, які мають адекватний рівень завантаження ресурсів (56–70%), що відповідає оптимальному

використанню, оптимізувати машини, які мають низьке завантаження (33% для VM2 і 35% для VM5), тому пропонується зменшити їхні ресурси для економії, видалити VM3, бо вона не має процесів, і її використання не є економічно доцільним, для процесів із тегом «High priority» немає відповідної машини серед існуючих, тому пропонується додати нову машину з характеристиками: 4 vCPU, 16 GB пам’яті, 5 Gbps мережі, розміщену в арендованій інфраструктурі. Загальна економія становить 902.77$ на місяць: видалення машини – 596.39$, оптимізація машин – 446.54$, додавання нової машини коштувало 140.16$.

Таблиця 4.9 – Рекомендації по оптимізації віртуальних машин

Локація	Дія	vCPU	Пам’ять (GB)	Мережа (Gbps)	Теги	Використання (%)	Причина	Вартість після
Host	-	4	16	5	General	68	-	$140.16
GCP	Оптимізувати	4	16	5	Data-processing	55	Низьке завантаження	$149.85
GCP	Видалити	16	64	20	General	0	Завантаження відсутнє	$0
Host	-	2	8	2	AI, ML	70	-	$80
GCP	Оптимізувати	8	32	10	Data storage	57	Низьке завантаження	$298.7
Host	-	8	32	10	General	56	-	$230.5
Host	Додати	4	16	5	High priority	63	Для процесів з вказаним тегом немає машини	$140.16
Загальна вартість:								$1039.37

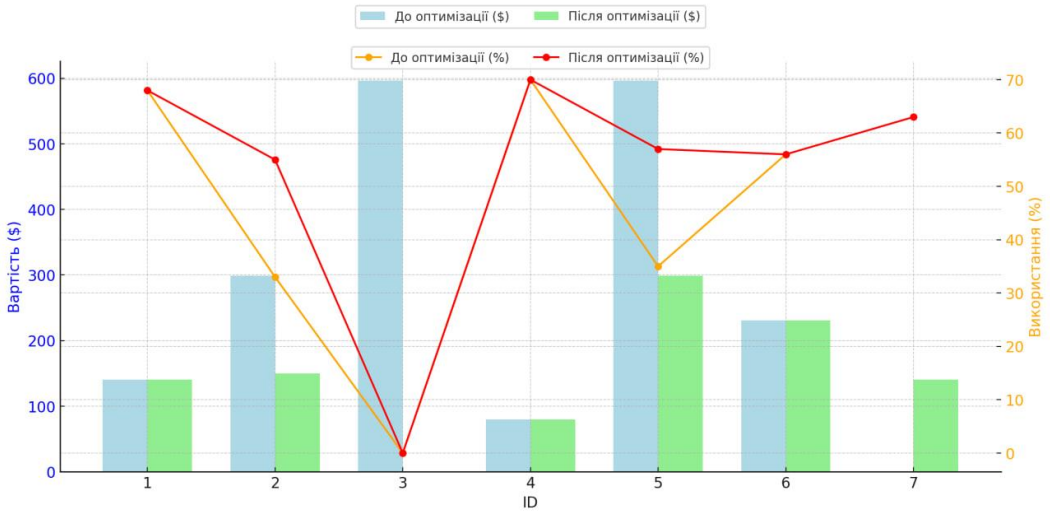

Рисунок 4.9 – Візуалізація результатів оптимізації віртуальних машин

4.7 Імітаційне моделювання виконання процесів

Метою імітаційного моделювання є дослідження особливостей роботи серверів у змішаному середовищі, де одночасно виконуються стрімінгові процеси та задачі за розкладом, які можуть отримувати помилки і виконувати до 4 рестартів, що впливає на продуктивну здатність системи. Аналіз надає можливість оптимізувати використання обчислювальних ресурсів і підвищити надійність системи.

Поведінка системи у разі виникнення помилок залежить від типу процесів. Стрімінгові задачі негайно перезапускаються після збою, забезпечуючи постійний рівень навантаження без значних коливань. Натомість задачі, що виконуються за розкладом, мають складніший механізм обробки помилок, який включає до чотирьох спроб перезапуску із експоненційною затримкою між ними:

- 1-й рестарт через 1 секунду;
- 2-й рестарт через 10 секунд;
- 3-й рестарт через 100 секунд;
- 4-й рестарт через 1000 секунд.

Якщо після 4 спроб процес не завершується успішно, він виконується при наступному запланованому запуску. Це призводить до помітних спадів і

поступового відновлення навантаження.

Змоделюємо роботу сервера, який буде працювати протягом 12 годин безперервно, переключаючись між стрімінговими та джобами за розкладом, які можуть отримувати помилки та рестартувати.

Сформовані графіки навантаження на сервери, зображені на рисунках 4.10, 4.11, 4.12, ілюструють інтегральне використання ресурсів під впливом стрімінгових процесів та задач, виконуваних за розкладом, які можуть отримувати помилки та рестартувати. Разом ці процеси створюють динамічну, але контрольовану картину роботи серверів.

Процесорне навантаження здебільшого утримується в межах 60–70%, іноді досягаючи короткочасних піків до 75%. Такі сплески зумовлені одночасним виконанням кількох завдань та процесами перезапуску після можливих збоїв. Використання оперативної пам’яті демонструє стабільність із поступовим збільшенням під час інтенсивних обчислень та зменшенням після завершення задач. Мережева активність характеризується помірними коливаннями, які відображають зміну обсягу переданих даних у процесі виконання завдань.

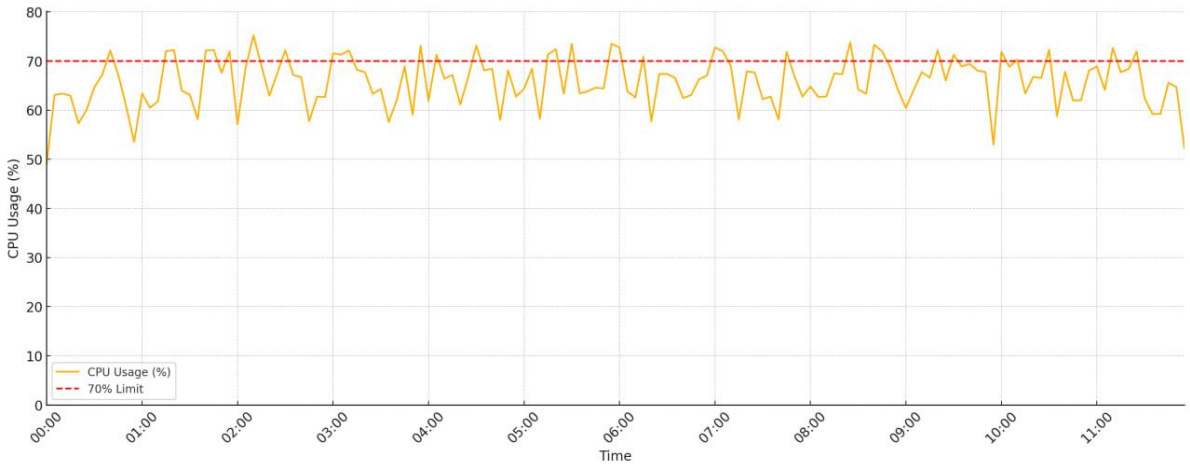

Рисунок 4.10 – Графік динаміки завантаження CPU з рестартами

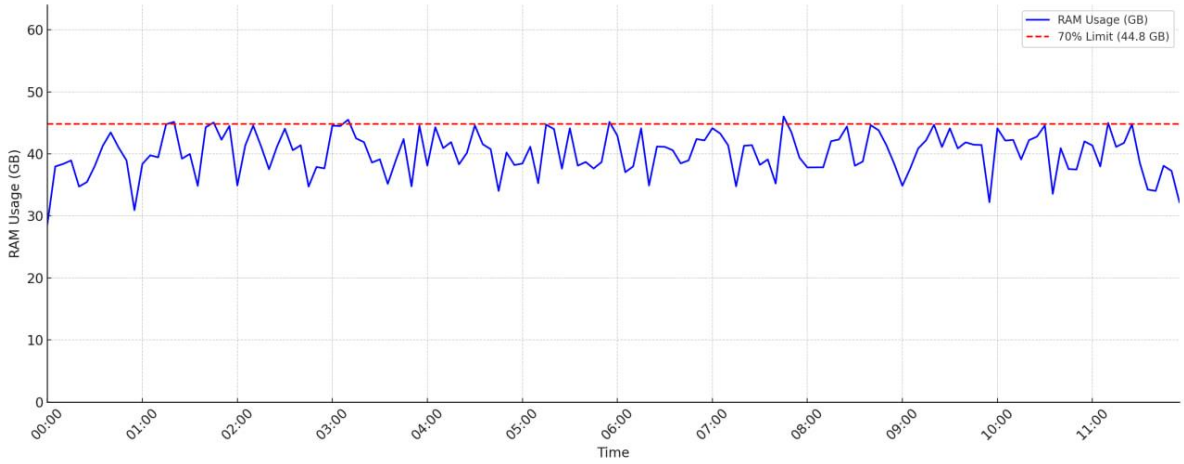

Рисунок 4.11 – Графік динаміки завантаження RAM з рестартами

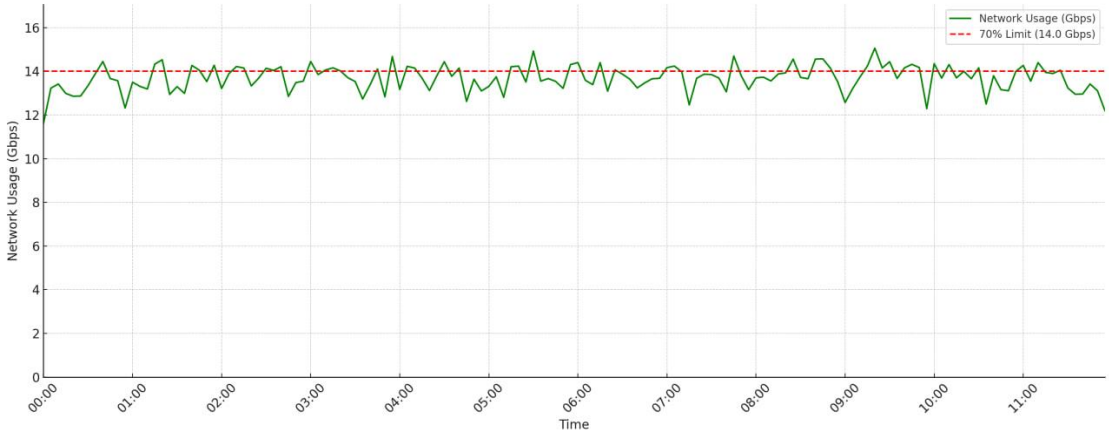

Рисунок 4.12 – Графік динаміки завантаження мережі з рестартами

Гнучке управління рестартами дозволяє уникнути критичних збоїв і значних втрат ресурсів. Це забезпечує стабільність і безперервність роботи системи. Завдяки застосуванню стратегії експоненційних рестартів вдалося мінімізувати ризики тривалих збоїв, що є ключовим чинником для стабільної роботи системи навіть за умов пікових навантажень.

4.8 Висновки за розділом

У розділі розглянуто проблему розподілу процесів на віртуальні машини, яка є складною комбінаторною задачею. Проведені експерименти підтвердили, що для малих задач повний перебір дозволяє отримати

оптимальне рішення, але його обчислювальна складність робить його непридатним для задач із великою кількістю процесів і машин. Жадібні алгоритми, такі як First Fit, Best Fit та Worth Fit, показали значно кращі результати за швидкістю виконання, хоча й поступаються повному перебору за точністю. Застосування тегів суттєво знижує кількість можливих комбінацій для аналізу, що прискорює обчислення без втрати ефективності.

Алгоритми розподілу довели свою ефективність у випадках з різними типами процесів: стрімінговими та такими, що виконуються за розкладом. Розглянуті підходи дозволили визначити ключові аспекти оптимізації інфраструктури, зокрема мінімізацію витрат, рівномірність завантаження ресурсів і уникнення перевантажень.

Дослідження роботи серверів у змішаному середовищі з імовірністю отримання помилок, де одночасно виконуються стрімінгові процеси та задачі за розкладом, виявило, що такий підхід дозволяє уникати довготривалих збоїв, але водночас спричиняє тимчасові спади навантаження з поступовим відновленням.

Таким чином, обрані методи та алгоритми демонструють придатність до практичного застосування, особливо у великих системах, і є основою для подальшої оптимізації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто мети, що полягала у розробці та дослідженні методів розподілення задач та моніторингу KPI в highload-системах обробки даних. Виконані завдання відповідають поставленому плану, зокрема: проведено аналіз наявних науково-технічних рішень, розроблено математичну модель і алгоритми розподілу завдань, а також запропоновано підходи до моніторингу показників ефективності. Під час випробувань із використанням тестових сценаріїв було підтверджено, що розроблений метод забезпечує поліпшення кількісних показників і підвищення якісних характеристик.

У практичній частині роботи було розглянуто задачу статичного розподілення процесів на множину віртуальних машин із метою зниження загальної вартості володіння інфраструктурою та забезпечення гарантованого запасу продуктивності. У процесі дослідження було детально проаналізовано характеристики як віртуальних машин (продуктивність процесора, обсяг оперативної пам’яті, пропускна здатність мережі, розташування в GCP чи на виділеному хостингу, а також додаткові теги), так і процесів (споживання ресурсів ЦПУ, пам’яті, зовнішніх даних, необхідність отримання даних із GCP або з інших джерел, розклад запуску та час виконання, ймовірність помилок та кількість перезапусків).

Запропонований підхід до статичного розподілення дозволяє дотримуватися критичних обмежень щодо максимальної завантаженості (не більше 70% від доступних у кожній віртуальній машині), що дає змогу уникати перевантажень і створює необхідний резерв продуктивності на випадок пікових навантажень. Крім того, враховується відповідність тегів між процесами та віртуальними машинами, аби забезпечити сумісність за вимогами безпеки, специфікою обчислень чи іншими критеріями. Завдяки цьому можливо досягти мінімізації сукупної вартості володіння, адже

витрати складаються з вартості обраних конфігурацій віртуальних машин і обсягу вихідного трафіку з GCP. Зокрема, якщо процеси активно використовують трафік із сервісів GCP, доцільно розміщувати їх у GCP, зменшуючи, таким чином, плату за вихідний трафік.

Результати роботи свідчать, що для ефективної побудови інфраструктури варто зважати на можливі сценарії: у разі, коли наявні потужності віртуальних машин є недостатніми, рекомендовано додати або масштабувати конфігурації; коли ж вони надмірні – раціонально скоротити кількість машин або зменшити їх параметри (частоту ЦПУ, обсяг пам’яті тощо), аби знизити щоденні витрати без шкоди для продуктивності. У разі застосування CRON-завдань чи стримінгових процесів додатково слід брати до уваги часові особливості запуску, механізми перезапуску й можливі помилки.

Загалом, дослідження продемонструвало, що грамотно реалізоване статичне розподілення процесів на віртуальні машини зі збереженням необхідних резервів продуктивності й дотриманням принципів сумісності за тегами дає змогу суттєво знизити загальну вартість використання інфраструктури та підвищити стабільність роботи highload-систем. На основі отриманих результатів програма формує рекомендації щодо подальшої інтеграції динамічних механізмів у вже створену модель, щоб у випадку змінних умов завантаження система могла самостійно вносити корективи в розподіл процесів. Перспективним напрямком також є використання методів прогнозної аналітики та машинного навчання, що дасть змогу ще більш точно визначати необхідну кількість ресурсів і запобігати можливим простоям або перевитратам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гора М. В. Моделі управління ресурсами для забезпечення функціональної стійкості процесу розподілених обчислень / М. В. Гора, М. О. Волк. // Вісник ХНТУ. – 2023. – Т.4, №87. – С. 244—251.

2. Piernas J. Active Storage User's Manual / J. Piernas, J. Nieplocha. – Pacific Northwest National Laboratory, 2007. – 23 с.

3. Демчик В. В. Аналіз сучасних методів планування обчислень та балансування навантаження в розподілених комп'ютерних системах / В. В. Демчик. // The scientific heritage. – 2021. – №72. – С. 30–39.

4. Pokusin N. V. Load Balancing in Distributed Heterogeneous Computing. System with a Priori Indefinite Input Stream Nature / Pokusin. // Naukovedenie. – 2013. – №3.

5. An Evolutionary Approach to Optimizing Cloud Services / D.Agrawal, L. H. Jaiswal, I. Singh, I. Chandrasekaran. // Computer Engineering and Intelligent System. – 2012. – Т.3, №4. – С. 47–54.

6. Zhao H. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems [Електронний ресурс] / H. Zhao, R. Sakellariou // 18th International Parallel and Distributed Processing Symposium. – 2004. – Режим доступу до ресурсу: https://doi.org/10.1109/IPDPS.2004.1303065

7. Golubev I. A. Scheduling in Distributed Computing Systems Based on Metadata : дис. канд. наук / Golubev, 2014. – 135 с.

8. DIET user's manual [Електронний ресурс] / [A. Amar, R. Bolze, E. Boix та ін.]. – 2010. – Режим доступу до ресурсу: https://graal.ens-lyon.fr/diet/download/doc/UsersManualDiet2.4.pdf

9. Amedro B. ProActive Scheduling v.3.3.2 user's manual / B. Amedro, V. Bodnartchouk, L. Baduel., 2013. – 152 с.

10. Moab Workload Manager. Administrator's Guide [Електронний ресурс] // Adaptive Computing Enterprises. – 2011. – Режим доступу до

ресурсу: https://docs.adaptivecomputing.com/mwm/archive/6-0/MWMAdminGuide.pdf

11. Maui Administrator's Guide [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: https://docs.adaptivecomputing.com/maui/pdf/mauiadmin.pdf

12. Розрахунок параметрів якості обслуговування у фотонних транспортних мережах / [М. В. Кайдан, С. С. Думич, Т. А. Максимюк та ін.]. // Вісник Національного університету «Львівська політехніка». Радіоелектроніка та телекомунікації. – 2014. – №796. – С. 147–156.

13. Feoktistov A. G. Development and application of subject-oriented multi-agent systems for distributed computing management. / A. G. Feoktistov, R. O. Kostromin. // Izvestiya SFedU. Engineering Sciences. – 2016. – №11. – С. 65–75.

14. Resource allocation for distributed cloud: concepts and research challenges / [P. Endo, A. Palhares, N. Pereira та ін.]. // Ieee Network. – 2011. – Т. 25, №4. – С. 42–46.

15. Openflow: enabling innovation in campus networks / [N. McKeown, T. Anderson, H. Balakrishnan та ін.]. // ACM SIGCOMM Computer Communication Revie. – 2008. – №38. – С. 69–74.

16. Feitelson G. Workload Modeling for Computer Systems Performance Evaluation / Feitelson., 2015. – 564 с.

17. Golubev I. A. Metadata-driven task scheduling in computer clusters / I. A. Golubev, M. S. Kupryianov. // Proceedings of 9th International Conference on Computer Science and Information Technologies. – 2013. – С. 249–252.

18. Balasubramaniam P. Important KPIs and Server Monitoring Metrics [Електронний ресурс] / Priya Balasubramaniam. – 2020. – Режим доступу до ресурсу: https://www.eginnovations.com/blog/server-performance-monitoring/

19. Key Performance Indicators [Електронний ресурс] – Режим доступу до ресурсу: https://help.hcl-software.com/commerce/9.1.0/admin/refs/rpmkpi.html

20. Agent-Based Monitoring [Електронний ресурс] – Режим доступу до ресурсу: https://www.motadata.com/it-glossary/agent-based-monitoring/

21. What is agentless monitoring? [Електронний ресурс] – Режим доступу до ресурсу: https://www.solarwinds.com/resources/it-glossary/agentless-monitoring

22. Teske K. Hybrid Cloud Monitoring: Best Practices and Strategies [Електронний ресурс] / Kelsey Teske. – 2024. – Режим доступу до ресурсу: https://www.veeam.com/blog/hybrid-cloud-monitoring-best-practices.html

23. GMonE: A complete approach to cloud monitoring / [J. Montes, A. Sanchez, B. Memishi та ін.]. // Future Generation Computer Systems. – 2013. – Т. 29, №8. – С. 2026–2040.

24. Google Cloud's Observability [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/products/observability

25. Chhajed S. Learning ELK Stack / Saurabh Chhajed. – Birmingham: Packt Publishing Ltd, 2015. – 206 с.

26. What is ELK Stack? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/elk-stack/?nc1=h_ls

27. Martello S. Bin Packing Problems [Електронний ресурс] / Silvano Martello – Режим доступу до ресурсу: https://mathopt.be/Slides_LaRoche_Martello.pdf

28. Kang J., Park S. Algorithms for the variable sized bin packing problem // European Journal of Operational Research. – 2003. – Т. 147, № 2. – С. 365–372.

29. Бульба С. С., Давидов В. В., Кучук Г. А. Метод розподілу ресурсів між композитними застосунками // Системи управління, навігації та зв’язку. – 2018. – Т. 4, № 50. – С. 99–104.

30. Бугрій А. М., Бондаренко О. В., Волк Д. М. Аналіз методів розподілу обчислювальних задач в гетерогенних високонавантажених обчислювальних системах. The most difficult problems of youth and ways to solve them : тези доп. III міжнар. наук.-практ. конф., Краків, Польща, 20-22 січня 2025 р. – С. 264-267.

31. Wang Y. Review on greedy algorithm // Theoretical and Natural Science. – 2023. – Т. 14, № 1. – С. 233–239.