Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи розподілення віртуальних машин за хмарними ресурсами

(тема)

Виконав:

студент II курсу, групи СПзм-20-1

Гвоздецький Д.П.
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Волк М.О.
(посада, прізвище, ініціали)

Допускається до захисту

В.о. зав. кафедри ЕОМ _____ Волк М.О.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Гвоздецькому Дмитру Павловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи розподілення віртуальних машин за хмарними ресурсами

затверджена наказом по університету від " 25 " березня 2022 р. № 33 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи

Середовище керування хмарними обчисленнями OpenStack

Основні типи віртуалізації

Система моделювання CloudSim

4. Перелік питань, що потрібно опрацювати у роботі

Аналіз предметної області

Програмне забезпечення моделювання та реалізації

Розробка методів та алгоритмів розподілення віртуальних машин за хмарними ресурсами

Реалізація експериментів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 12 слайдів презентації в форматі PowerPoint

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	29.03.21 – 04.04.21	
2	Розробка моделей	05.04.21 – 12.04.21	
3	Реалізація алгоритмів	13.04.21 – 20.04.21	
4	Розробка структури програмних засобів	21.04.21 – 25.04.21	
5	Розробка програмних модулів	26.04.21 – 30.04.21	
6	Оформлення матеріалів атестаційної роботи	01.05.21 – 10.05.21	
7	Подання атестаційної роботи керівникові та її	11.05.21 – 14.05.21	
	попередній захист		
8	Подання атестаційної роботи на рецензування	14.05.21 – 15.05.21	

Дата видачі завдання 28 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Волк М.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 59 с., 22 рис., 3 табл., 1 дод., 17 джерел.

ВІРТУАЛЬНА МАШИНА, ХМАРНИЙ РЕСУРС, ФІЗИЧНА МАШИНА, МОДЕЛЮВАННЯ, СЕРВЕР, РОЗМІЩЕННЯ, РОЗПОДІЛ РЕСУРСІВ

Метою кваліфікаційної роботи є дослідження методів підвищення ефективності розміщення віртуальних машин.

У ході виконання кваліфікаційної роботи проаналізовано сучасний стан хмарних технологій та віртуалізації як основи побудови хмарних систем різного призначення; проведено оцінку енергоспоживання центрів обробки даних за допомогою алгоритму розміщення; спроектовано тестовий стенд OpenStack та реалізовано упаковку контейнера; розроблено евристичний метод для оцінки таких показників продукт ивності, як деградація ресурсів і енергоспоживання; підвищено ефективність пакування віртуальних машин з мінімальною QoS.

ABSTRACT

Master’s thesis: 59 pages, 22 figures, 3 tables, 1 appendices, 17 sources.

VIRTUAL MACHINE, CLOUD RESOURCE, PHYSICAL MACHINE, SIMULATION, SERVER, PLACEMENT, DISTRIBUTION OF RESOURCES.

The purpose of the qualification work is to study methods of improving the efficiency of placement of virtual machines.

In the course of qualification work the current state of cloud technologies and virtualization as the basis for building cloud systems for various purposes is analyzed; the estimation of energy consumption of data processing centers by means of the algorithm of placement is carried out; the OpenStack test stand was designed and container packaging was implemented; developed a heuristic method for estimating productivity indicators such as resource degradation and energy consumption; increased efficiency of packing virtual machines with minimum QoS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВМ – віртуальна машина

ІТ – інформаційні технології

ЦОД – центр обробки даних

ЦП – центральний процесор

API – програмний інтерфейс застосувань (англ., Application Programming Interface)

IaaS – інфраструктура як послуга (англ., Infrastructure as a service)

PaaS – платформа як послуга (англ., Platform as a service)

PM – фізична машина (англ., Physical machine)

SaaS – програмне забезпечення як послуга (англ., Software as a service)

SLA – угода між постачальником послуг і користувачем про рівень послуг (англ., Service-level agreement)

VM – віртуальна машина (англ., Virtual Machine)

QoS – якість обслуговування (англ. Quality of service)

ВСТУП

Рішення про те, на яких фізичних машинах (РМ) розмістити кожну віртуальну машину (VM), дуже важливо для ефективної роботи хмари. З одного боку, важливо, щоб віртуальні машини отримували адекватні ресурси (наприклад, центральний процесор, пам’ять, мережу) від хостингового РМ, щоб його продуктивність не погіршувалася. З іншого боку, хмарний оператор хотів би об’єднати віртуальні машини на якомога меншій кількості РМ, щоб максимально використовувати ресурси та зменшити споживання енергії. Ефективне розміщення ВМ подібне до проблеми упаковки векторного бункера, яка, як відомо, є NP-складною[2].

Інша проблема полягає в тому, що споживання ресурсів у віртуальній машині не є постійним, а динамічно змінюється з часом. Один із способів вирішення цієї проблеми — розмістити віртуальні машини на основі пікового використання ресурсів. Однак це марнотратний підхід, оскільки пікове використання трапляється рідко[3]. Іншим варіантом є міграція віртуальних машин на РМ з додатковими ресурсами, якщо це необхідно, наприклад, шляхом живої міграції. На жаль, оперативна міграція сама по собі вимагає додаткових ресурсів і, таким чином, сприяє завантаженню всієї системи[4] [5]. Це означає, що постраждала не тільки віртуальна машина, що переміщується, а й інші пристрої на шляху міграції.

Бажано, щоб алгоритм розміщення знаходив баланс між стабільністю розміщення та ефективним використанням ресурсів (тобто пакування), враховуючи вартість реальної міграції. Крім того, в [6] та [7] було показано, що продуктивність QoS є неоптимальною, якщо мета QoS не може бути виражена явно в термінах незалежної від робочого навантаження метрики QoS [8].

Хмарні обчислення – це революційна технологія в інформаційно-комунікаційній індустрії. Це постійно збільшує попит на ресурси в сервісах

різних організацій, таких як доповнена реальність, самокеровані автомобілі та мобільні послуги 5G. Таким чином, попит на центри обробки даних продовжує зростати у всьому світі. Однак хмарні центри обробки даних споживають велику кількість енергії, а також виділяють тепло та вуглекислий газ у навколишнє середовище. Згідно зі звітом «Total Consumer Power Consumption Forecast», він передбачає, що в 2025 році центри обробки даних споживатимуть понад 20% загальної електроенергії у світі, а центри обробки даних також відповідають за 3% загального викиду парникових газів [8].

Це дуже велика проблема для великих технологічних галузей, таких як Facebook, Google, Amazon, Alibaba тощо. Щоб усунути неефективність та марнотратство ресурсів, що споживають електроенергію, можна вдосконалити як інфраструктуру ЦОД, так і ефективні інструменти управління за допомогою алгоритмів розподілу ресурсів. . Вони вкладають величезні гроші в мінімізацію використання ресурсів різними методами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Хмарні обчислення

Хмарні обчислення — це великий пул систем, підключених до загальнодоступної або приватної мережі, для надання користувачам інфраструктури для зберігання, додатків і даних (рисунок 1.1). У хмарних обчисленнях один РМ може одночасно розміщувати кілька віртуальних машин. Швидке зростання попиту на обчислювальну потужність ресурсів зміщується в бік моделі хмарних обчислень, таких як великомасштабні віртуалізовані центри обробки даних по всьому світу. Модель хмарних обчислень надає клієнту плату за використання. Замість того, щоб купувати інфраструктуру та оновлення IT-сервісів, багато організацій використовують хмарні оператори, такі як Amazon Web Services, Google Cloud та платформи Rackspace. Ці дата-центри споживають величезну кількість електроенергії під час роботи.

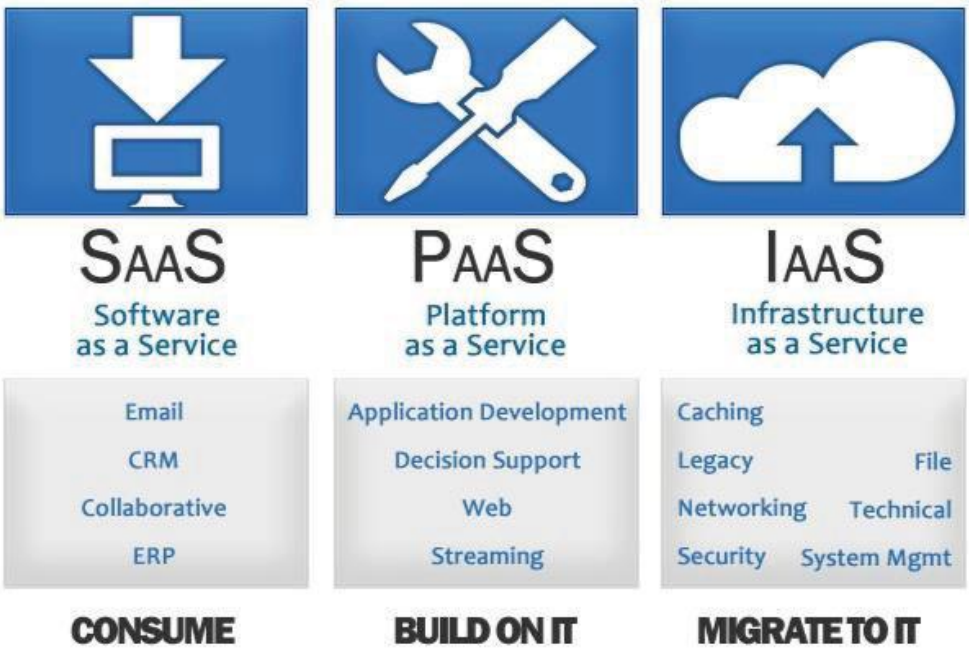

Рисунок 1.1 – Структурна схема хмарних обчислень

Хмарні провайдери пропонують послуги в трьох категоріях:

- програмне забезпечення як послуга (SaaS);
- платформа як послуга (PaaS);
- інфраструктура як послуга (IaaS).

Нині центри обробки даних використовують технологію віртуалізації, впроваджуючи свої послуги на багатьох віртуальних машинах, розміщених у пулі РМ. Важливо ретельно розподіляти фізичні ресурси для віртуальних машин, щоб не погіршити продуктивність служби та обмежити споживання енергії. Алгоритми, які використовуються для розміщення віртуальних машин, впливають на безліч факторів:

- енергоспоживання РМ;
- продуктивність віртуальних машин;
- міграція ВМ;
- енергоспоживання РМ.

Хороший алгоритм розміщення ВМ мінімізує кількість РМ, які використовуються для використання всіх віртуальних машин, не погіршуючи продуктивність розгорнутих програм. Споживання енергії не є постійним протягом усього часу. Динамічне масштабування напруги та частоти можна використовувати на основі навантаження РМ. Динамічне масштабування напруги збільшується, якщо навантаження РМ збільшується (перенапруга), або динамічне масштабування напруги зменшується, якщо навантаження РМ зменшується (перенапруга) [10].

Продуктивність віртуальної машини.

Для агресивної віртуальної машини розміщення може спричинити погіршення продуктивності програми, наприклад, споживання високої електричної енергії, випромінювання високого шуму та споживання великої кількості пам’яті [4]. У цьому випадку хост РМ не може вмістити кількість ресурсів, необхідних віртуалізованій програмі. Що в свою чергу, ймовірно, призводить до порушення рівня обслуговування угод (SLA) залежно від умов замовника та постачальника послуг. Занадто велике порушення рівня

обслуговування може призвести до штрафів, а також втрати задоволеності клієнтів.

Міграція віртуальної машини.

Жива міграція може знизити час простою ВМ і призведе до високого використання ресурсів під час міграції. Це означає, що якщо кількість міграції ВМ збільшується, то і споживання енергії також збільшується. Тому ефективні алгоритми розміщення повинні бути спрямовані на мінімізацію кількості міграцій ВМ[4].

1.2 Типи хмарних платформ

Хмарні обчислення доступні споживачам у трьох типах (рисунок 1.2):

- публічна хмара;
- приватна хмара;
- гібридна хмара.

Залежно від вимог і функціональності вони класифікуються на різні рівні безпеки, управління та доступності. Середовище OpenStack можна використовувати як загальнодоступні, приватні та гібридні хмарні моделі.

Загальнодоступна (публічна) хмара в основному використовує Інтернет для надання споживачам таких функцій, як обчислювальні ресурси, програми та сховище. Прикладами загальнодоступних хмар є Amazon Elastic Compute Cloud (EC2), IBM Blue Cloud, Windows Azure Services Platform, Sun Cloud і Google App Engine.

Приватна хмара означає центр обробки даних, що знаходиться у власності та під контролем окремої організації. Ця хмара матиме повний контроль над гнучкістю, безпекою, масштабованістю, автоматизацією та моніторингом організації. Основна мета приватної хмари — не продавати «послугу» зовнішнім клієнтам, замість цього отримати вигоду від підтримки власних центрів обробки даних.

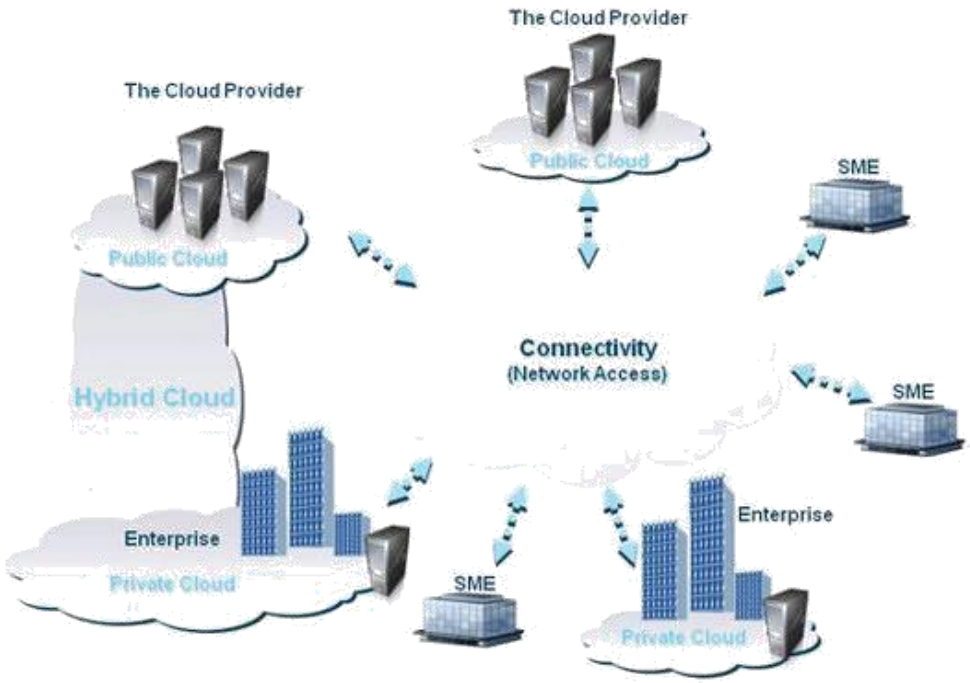

Рисунок 1.2 – Оглядова діаграма класифікації хмар

Цей тип хмар асоціюється з високою задоволеністю користувачів і ефективністю витрат. Тому що постачальник може розкрити всі ресурси, як-от обладнання, з безпекою [11]. Публічна хмара дотримується правила оплати за використання, тому користувач може платити лише за ресурси, які вони використали.

Деякі обмеження також перераховані у загальнодоступній хмарі. Для деяких організацій це некорисно, оскільки вони володіють умовами обміну внутрішніми даними.

Приватні хмарні центри обробки даних коштують дорого, оскільки організації потрібно придбати всю інфраструктуру для розгортання та наглядати за повсякденними операціями. Отже, ця хмара не підходить для малих галузей через обмежений бюджет.

Гібридна хмара — це інтегрований хмарний сервіс, який використовує як публічну, так і приватну хмарну інфраструктуру для виконання операцій в організації (рисунок 1.3).

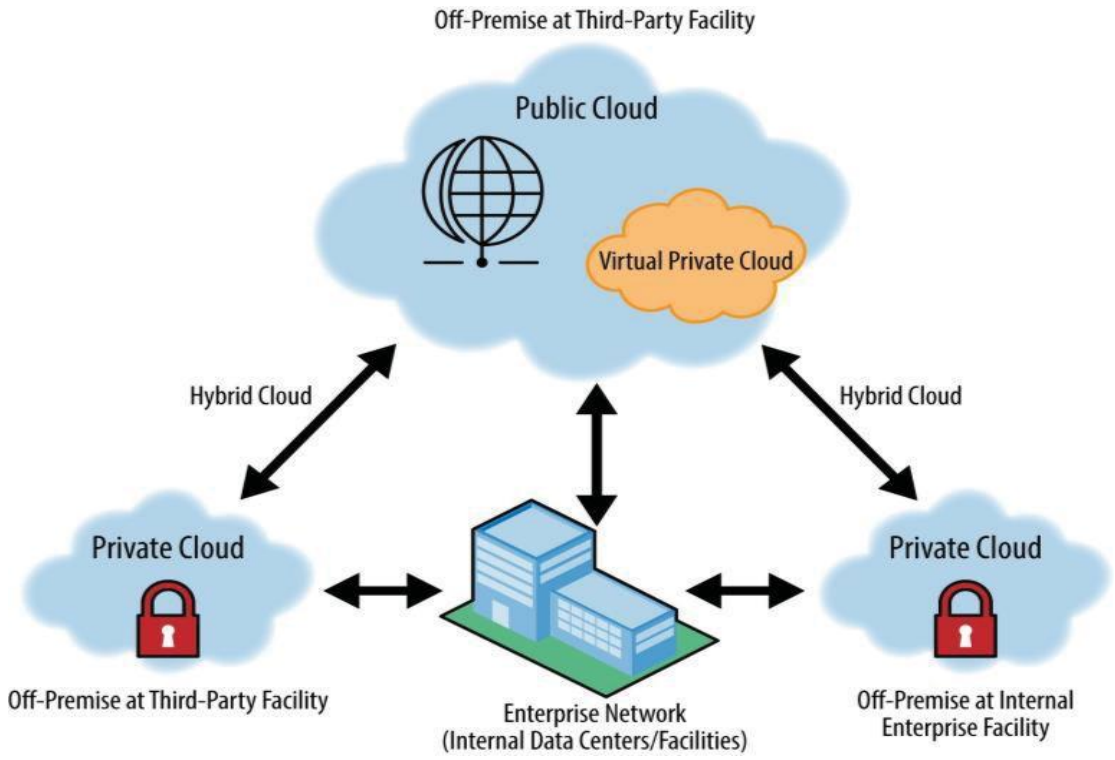

Рисунок 1.3 – Оглядова діаграма гібридної хмари

Публічна хмара має кращу масштабованість з обмеженими межами, оскільки ресурси використовуються постачальниками хмари. Використання приватної хмари залежить від конфігурації та ресурсів, що надає обмежену масштабованість. Використовуючи гібридну хмару, ми можемо об'єднати переваги масштабованості загальнодоступної хмари, одночасно значно зменшуючи попит на приватну хмару. Таким чином, збільшується ефективність та економічність такого рішення в порівнянні з загальнодоступною хмарою. Але залишається можливість використання загальнодоступної хмари для рішення окремих завдань, наприклад, використання поштових сервісів або сервісів аналізу даних.

1.3 Віртуалізація

Віртуалізація – це технологія, яка дозволяє створювати декілька віртуальних середовищ за допомогою одного фізичного обладнання. У цій технології Hypervisor має можливість належним чином розділити ресурси фізичних машин на віртуальні машини. Він розподіляє ресурси для віртуальних машин, такі як пам’ять, графіка, параметри мережі та віртуальні ядра, які використовуються для віртуальних машин. Ця технологія має постійний розвиток, щоб звести до мінімуму втрату ресурсів фізичних машин (рисунок 1.4). Віртуалізація має великий попит на ринку інформаційних технологій, бо надає користувачам доступ до обчислювальних ресурсів, якими вони не могли б скористатися через їх фізичну відсутність, велику вартість оренди, не навченість персоналу, складних алгоритмів використання та з інших причин. Тому тема кваліфікаційної роботи, що пов’язана з віртуалізацією є актуальною.

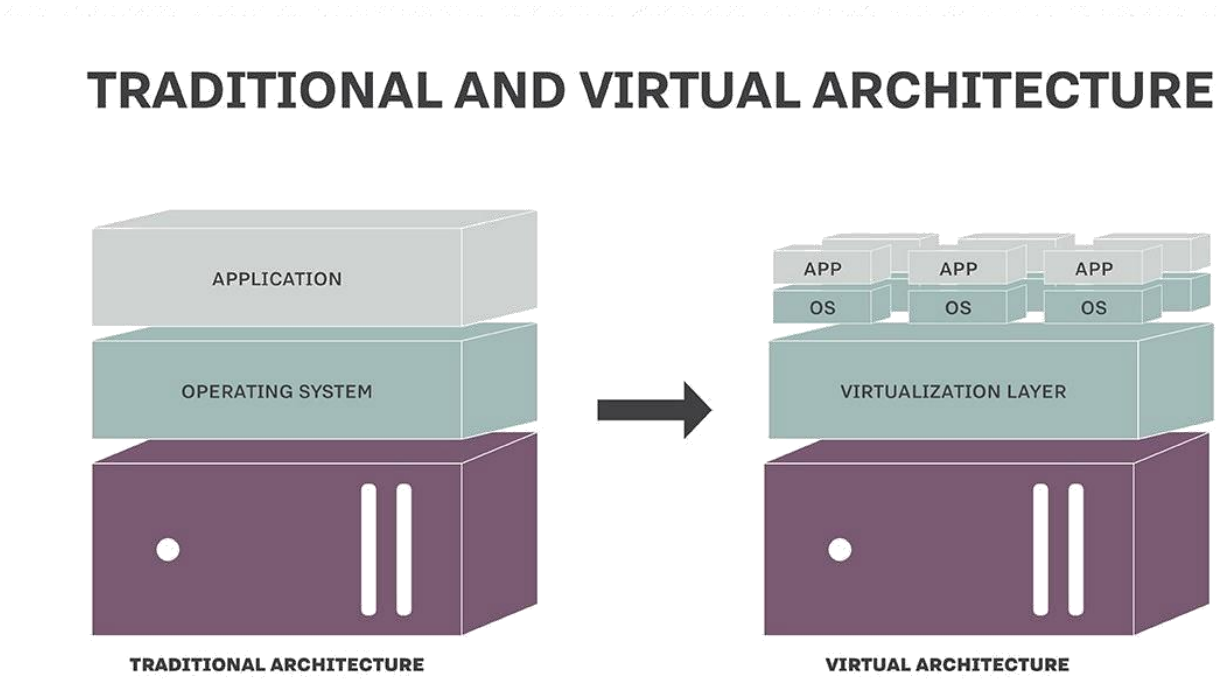

Рисунок 1.4 – Архітектура віртуальної машини

1.4 Типи віртуалізацій

1.4.1 Віртуалізація мережі

Віртуалізація мережі – це ключ і майбутнє хмарних обчислень. Комп’ютерна мережа починається з мережевої інтерфейсної карти, яка з’єднана з мережею рівня 2 з сегментами мережі рівня 2 (L2) (Ethernet, WiFi тощо) (Рисунок 1.5).

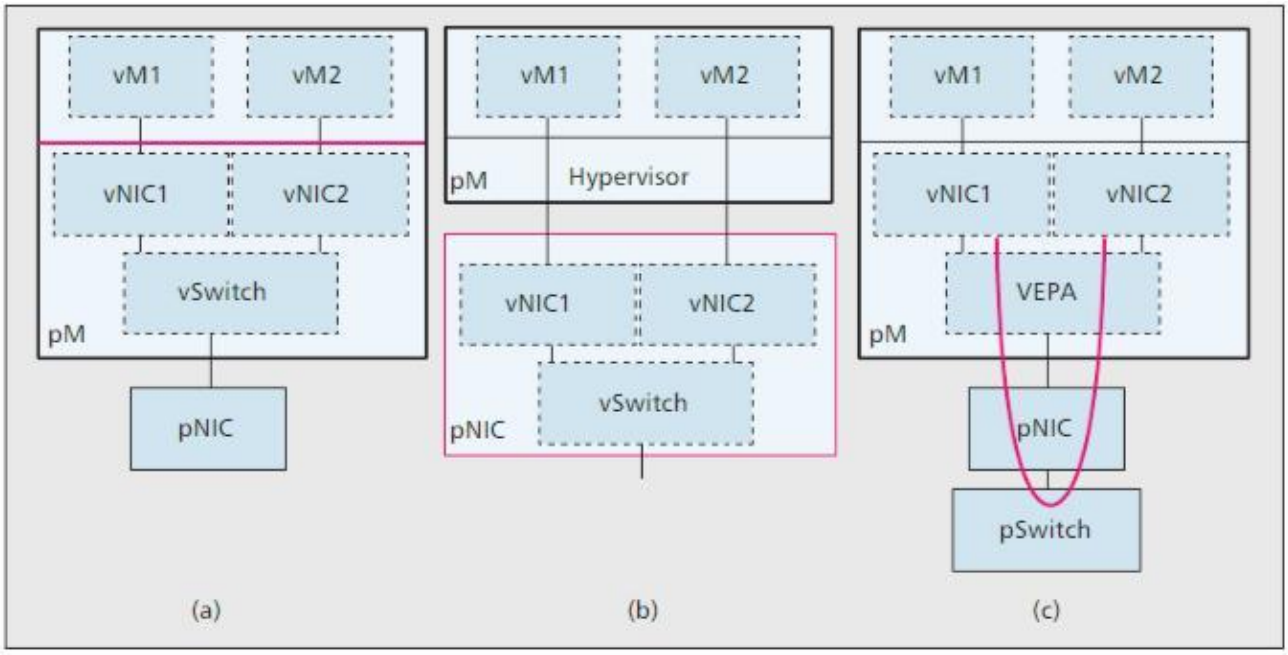

Рисунок 1.5 – Структурна схема віртуалізації мережі

Кілька сегментів мережі L2 можуть бути з'єднані між собою через комутатори (він же мости), щоб утворити мережу L2 [12]. Якщо ми запускаємо декілька віртуальних машин на фізичній машині, вона повинна мати принаймні один віртуальний NIC, як показано на малюнку вище. Один із способів вирішення цієї проблеми проблема за допомогою використання віртуального перемикача в гіпервізоре. Ми використовуємо позначення P означає фізичне V означає віртуальне.

1.4.2 Віртуалізація сервера

Віртуалізація серверів здебільшого спрямована на мінімізацію використання ресурсів центрів обробки даних нового покоління. При віртуалізації серверів віртуальний рівень розміщується на вершині фізичного рівня [12] (рисунок 1.6).

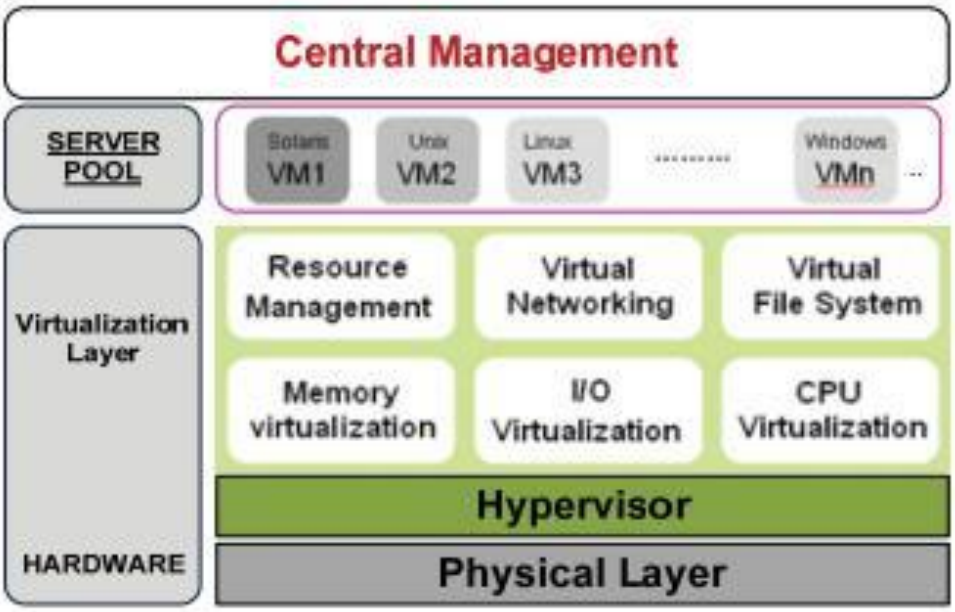

Рисунок 1.6 – Структурна схема віртуалізації Сервера

Віртуалізація серверів дозволяє використовувати декілька віртуальних серверів (ВМ) на фізичних серверах. Тут кожен віртуальний сервер працює зі своєю операційною системою за допомогою фізичних ресурсів[13]. Найпопулярнішими прикладами програмного забезпеченням є VMware, Microsoft та Citrix.

1.4.3 Віртуалізація робочого столу

Система віртуалізації робочого столу (рисунок 1.7), як правило, містить такі чотири специфічні модулі, які називаються модулем інфраструктури

сервера, хост-модулем віртуального робочого столу, модулем агента підключення та модулем тонкого клієнта[14].

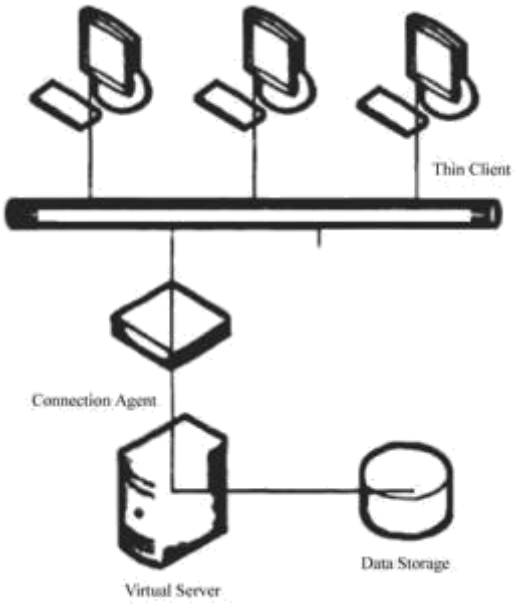

Рисунок 1.7 – Оглядова діаграма віртуалізації робочого столу

На рисунку вище показано, як модуль інфраструктури сервера виконує резервне копіювання даних, розподіл ресурсів та розгортання програми. Віртуальний сервер, який використовується для створення сервера для настільної операційної системи. Агент підключення надає користувачеві віддалений доступ для підключення операційної системи та клієнтського модуля – це віртуальна програма робочого столу, доступна для користувача.

1.5 Переваги використання віртуалізації

До переваг використання віртуалізації слід віднести наступне:

а) у віртуалізації він максимізує використання фізичних ресурсів машини;

б) зменшується час простою під час оновлення;

в) безпека покращує написання окремих політик безпеки за допомогою

окремих правил брандмауера в кожній віртуальній машині замість фізичної машини;

г) дуже легко імпортувати та переносити віртуальні машини.

1.6 Упаковка контейнера

Розміщення віртуальної машини важливе для управління ресурсами та живленням у хмарних обчисленнях. В основному це стосується як віртуальних машин, так і фізичних машин. Хороше розміщення віртуальної машини мінімізує перевантаження ресурсів, а також може зменшити споживання енергії. ефективність розміщення віртуальної машини вимірюється ефективністю упаковки (Tightness of the packing). Розподіл ресурсів віртуальної машини здійснюється в два етапи:

Етап 1. Кожна віртуальна машина призначена хосту за використанням ресурсів, таких як використання пам’яті, дискового простору та пропускної здатності зв’язку.

Етап 2. На другому етапі відбувається розподіл віртуальної машини для фізичного хоста шляхом реалізації алгоритму пакування.

Проблема упаковки одновимірного контейнера визначається як набір елементів змінного розміру і упаковує всі елементи в N контейнерів з ємністю кожного контейнера C [15]. Проблема упаковки контейнера має великий вплив на промисловий сектор, наприклад, у програмах планування (планування рекламних місць, планування завдань на процесорах, планування завдань з обмеженими ресурсами) і в транспортній сфері (завантаження вантажівок з обмеженням ваги або з обмеженням обсягу, наповнення контейнери, палети)[15].

Упаковка контейнерів - це об'єкти різного об'єму, які повинні бути упаковані в кінцеву кількість бункерів або контейнерів. У цьому підході об’єктами є віртуальні машини, а бункери — фізичні машини. Для мінімізації проблеми упаковки контейнерів ми вибираємо алгоритм, який дає найменшу

кількість використаних контейнерів. Існує багато варіантів вирішення проблеми упаковки контейнера, наприклад 2D упаковка для контейнерів, лінійна упаковка, пакування за вагою, упаковка за вартістю. Основна мета цієї упаковки Bin - мінімізувати кількість фізичних машин, що використовуються (рисунок 1.8).

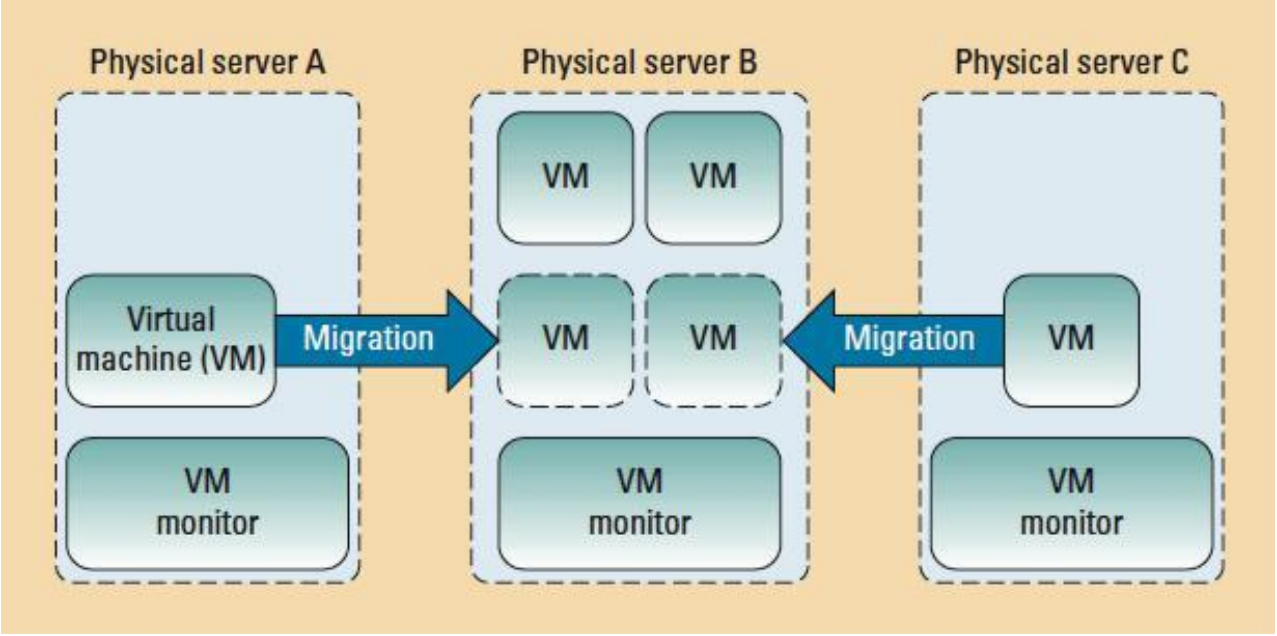

Рисунок 1.8 – Структурна схема консолідації сервера

Проблема упаковки контейнера в основному використовується під час реальної міграції віртуальних машин. Більшість використовуваних стратегій міграції віртуальних машин – це найкраща підгонка, перша підгонка та наступна підгонка відповідно (рисунок 1.9).

У двовимірній задачі упаковки контейнера кожен контейнер може обробляти прямокутник, визначений шириною та висотою. У цій упаковці бункера нам дано набір N= {N_1, N_2 ....., N_I} з I пунктів, ємністю C∈ I та розміри функцій s: N → I. Мета, щоб знайти мінімальну кількість використаних бункерів. Можливе віднесення до елементів у I контейнери є розділом P_1, P_2, ……, P_N. такий, що для кожного Рк , сума розмір предметів не перевищує ємність (C).

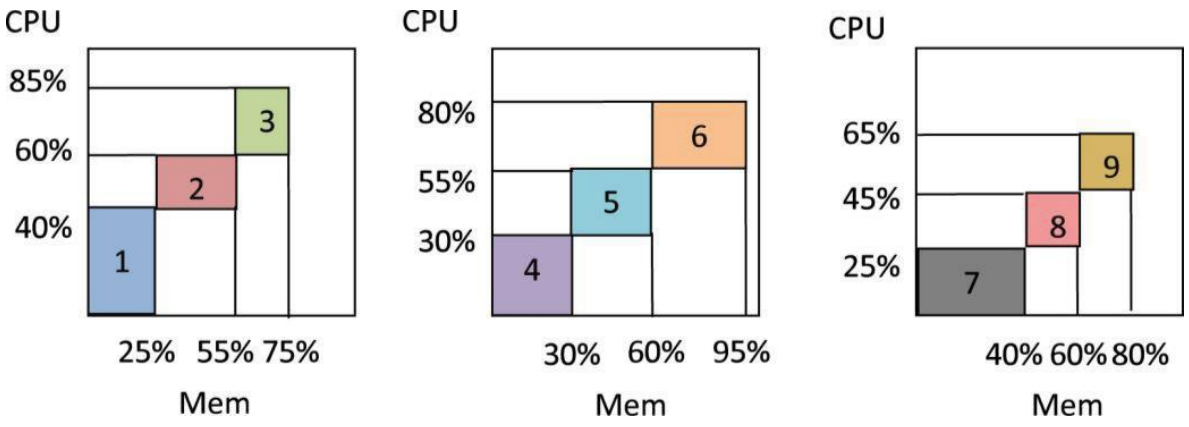

Рисунок 1.9 – Діаграма двовимірної упаковки контейнера

1.7 Постановка завдань на кваліфікаційну роботу

В результаті виконання першого етапу кваліфікаційної роботи проаналізовано сучасний стан хмарних технологій та віртуалізації як основи побудови хмарних систем різного призначення.

Зроблений аналіз надає підстави для проведення дослідженнь ефективності алгоритмів розміщення віртуальної машини (Bin placement).

Основні цілі кваліфікаційної роботи виглядають наступним чином:

- провести оцінку енергоспоживання центрів обробки даних за допомогою алгоритму розміщення;
- спроектувати тестовий стенд OpenStack та реалізувати упаковку Bin;
- розробити евристичний алгоритм для оцінки таких показників продукт ивності, як деградація ресурсів і енергоспоживання;
- підвищити ефективність пакування віртуальних машин з мінімальною QOS.

2 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОДЕЛЮВАННЯ ТА РЕАЛІЗАЦІЇ

2.1 Симулятор CloudSim

Симулятор CloudSim — це фреймворк для моделювання та оцінки ефективності хмарної інфраструктури та сервісів [17]. Цей симулятор в першу чергу розроблений дослідниками з лабораторії хмарних обчислень і розподілених систем Університету Мельбурна. Симулятор CloudSim в основному розроблений з використанням Java і є загальнодоступним за ліцензією LGPL[17]. Архітектура стимулятора та зв'язок між модулями системи наведена на рисунку 2.1.

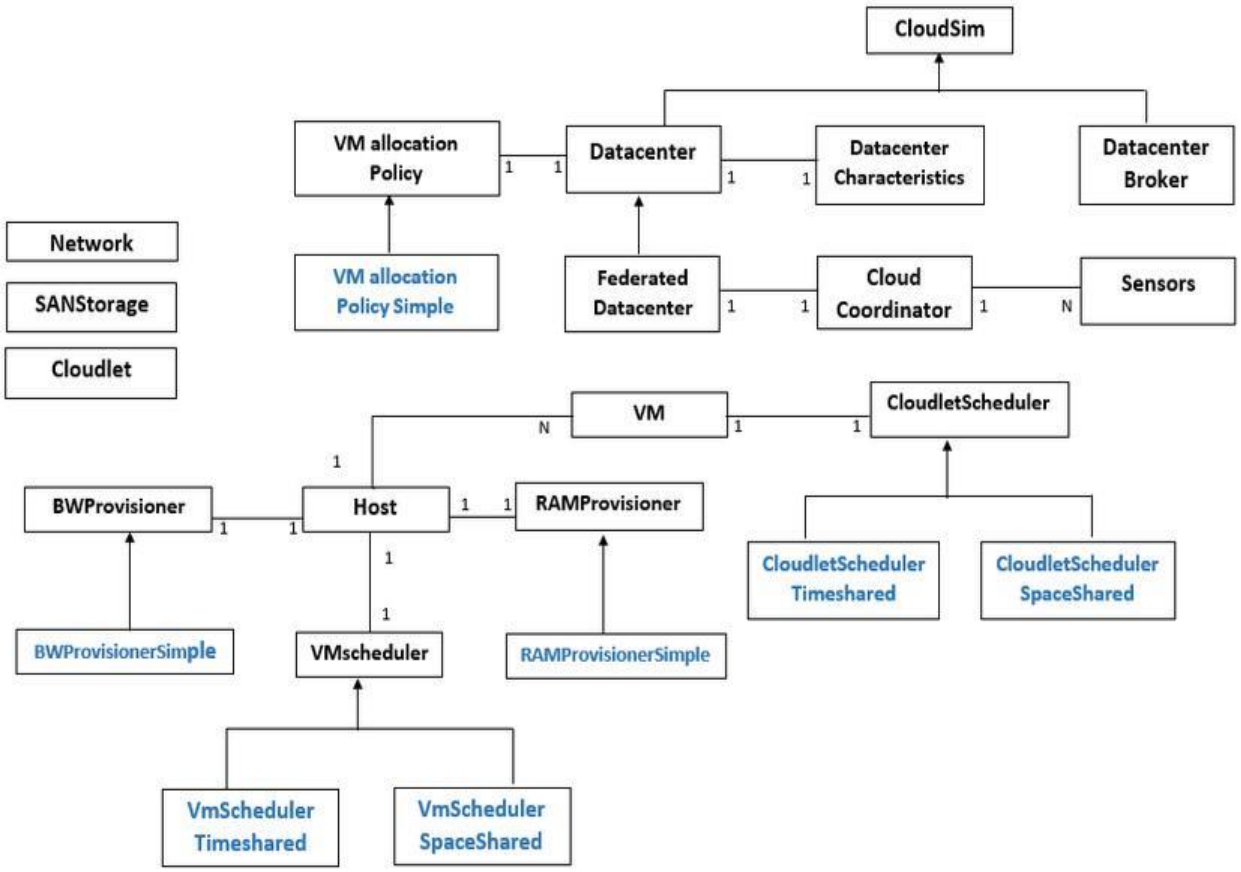

Рисунок 2.1 – Структурна діаграма архітектури симулятора CloudSim

Основні переваги хмарного симулятора:

- підтримка процесівмоделювання для великомасштабних центрів обробки даних;
- підтримка моделювання для обчислювальних ресурсів з урахуванням енергії;
- підтримка моделювання для розширення політики розподілу віртуальної машини та хоста політики розподілу;
- підтримка моделювання мережевих топологій і програми для передачі повідомлень;
- легка реалізація завдяки розширенню визначеної користувачем хмарної політики для хмари.

Недоліком симулятора CloudSim є те, що він не підтримує паралельний процес. Він має обмежену функціональність у параметрах, пов’язаних із мережею хоста та віртуальної машини. У симуляторі CloudSim ми можемо надати тільки пропускну здатність кожного хоста, і неможливо надати окремі компоненти (коммутатор, посилання). Однак це не стосується цього дипломного експерименту. Альтернативними інструментами моделювання для CloudSim є CloudAnalyst, GreenCloud, NetworkCloudSim, EmuSim, SPECI та GroudSim. Більшість із цих інструментів є розширенням симулятора CloudSim.

2.2 Програмні засоби OpenStack

OpenStack — це набір проектів програмного забезпечення з відкритим вихідним кодом, якими може керувати хмарна інфраструктура (рисунок 2.2). Здебільшого це розгортається як інфраструктура як послуга (Iaas). OpenStack розроблений спільним проектом NASA і Rackspace у 2010 році. Нині ним керує фонд OpenStack, некомерційна корпорація, що об'єднує понад 500 компаній. OpenStack виходить приблизно кожні шість місяців. В основному він написаний мовою Python і покладається на зовнішні бібліотеки [9].

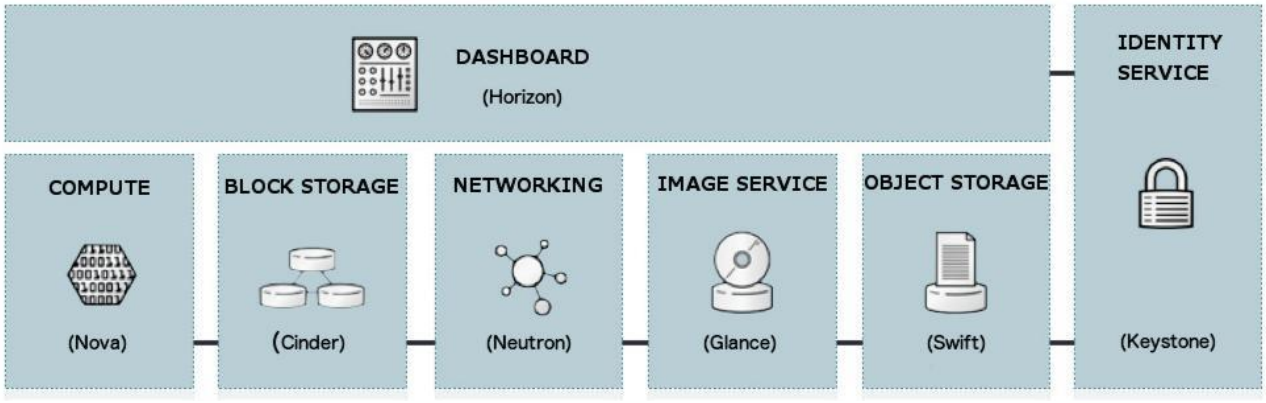

Рисунок 2.2 – Структурна діаграма служби OpenStack

Основними компонентами системи є наступні: Nova, Cinder, Neotron, Glance, Swift, Keystone, Horizon. Також OpenStack має кілька інших додаткових сервісів, які можуть покращити роботу користувача (рисунок 2.3).

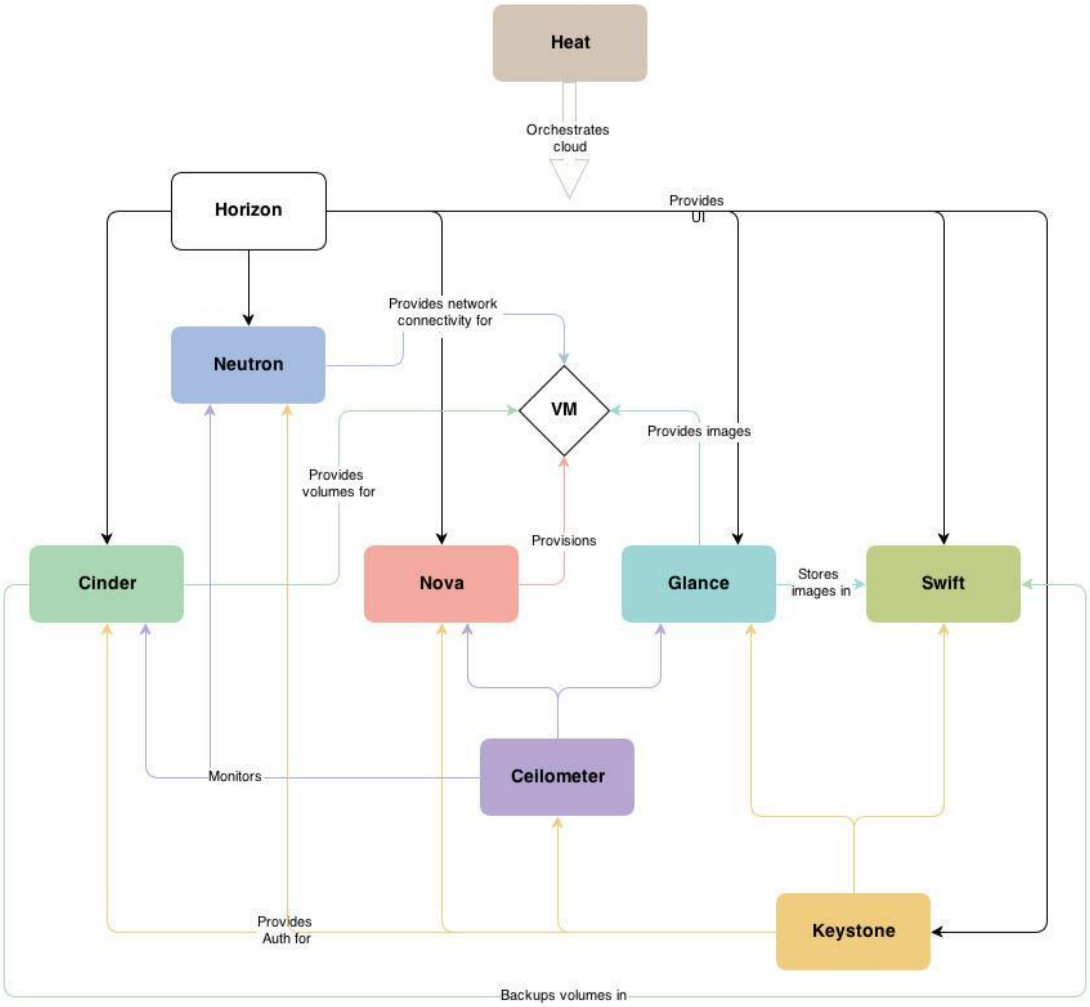

Рисунок 2.3 – Архітектура OpenStack

2.3 Огляд служби OpenStack

OpenStack Compute (Nova) — це контролер для середовища OpenStack. Nova в основному працює з доступними технологіями віртуалізації, такими як доступні гіпервізори KVM, VMware та Xen. Це написано на Python і використовує зовнішні бібліотеки, такі як SQL Alchemy (база даних), Kombu і Eventlet (одночасне програмування). Крім того, RabbitMQ використовується для служб обміну повідомленнями між компонентами.

OpenStack Neutron в основному використовується для мережевих завдань у середовищі OpenStack. В основному він використовується для розподілу IP-адрес, балансування навантаження, брандмауерів і віртуальних приватних мереж. Нейтрон керує IP-адресою та керує статичною IP-адресою за допомогою DHCP. Динамічна IP-адреса спрямовує трафік в межах інфраструктури під час технічного обслуговування та випадків збою.

Cinder — це служба зберігання блоків у середовищі OpenStack. Він надає постійний диск для віртуальних машин у хмарному середовищі. Це дозволяє віртуалізувати керування блочними пристроями зберігання даних.

OpenStack Keystone забезпечує ідентифікацію користувача в середовищі OpenStack і підвищує безпеку запитів без меж проекту. Ця послуга в основному заснована на аутентифікації на основі маркерів. Keystone керує деталями облікового запису користувача, інформацією про проект, інформацією про групу та внутрішньою політикою організації.

Проект OpenStack Glance в основному займається виявленням служби зображень, зображеннями сервера та реєстрацією зображень. Це також можливість зберігати сервер екземпляра у знімку зображення, щоб його можна було використовувати для створення нового сервера протягом короткого часу. Зображення Glance, що зберігаються в різних місцях зберігання, як-от файлова система, Swift за допомогою Glance API і розташування, зберігатимуться в реєстрі Glance.

OpenStack Swift є одним із головних проектів, розроблених Rackspace та NASA під час початкових випусків OpenStack. Основні функції swift включають резервне копіювання, зберігання та архівування неструктурованих даних, включаючи файли, зображення, документи та образи віртуальних машин.

Панель інструментів OpenStack надає веб-інтерфейс користувача, який інтегрує компоненти OpenStack, такі як Nova, Glance, Cinder, Heat тощо. В основному вона розроблена за допомогою Django і керує всіма обчисленнями за допомогою однієї програми.

Heat — це множина шаблонів для використання викликів API OpenStack для обміну інформацією з хмарним додатком у OpenStack. Цей шаблон описує кожен параметр у службі, який можна читати та розуміти для людей. Цей шаблон в основному використовується для керування компонентами в OpenStack, але він також поширюється на інструменти керування, такі як Ansible і Puppet.

2.4 Генератор навантаження

Для функціонування системи потрібно запустити генератор навантаження ЦП віртуально в обчислювальному вузлі для виконання міграції віртуальної машини. Щоразу, коли навантаження ЦП змінюється, буде виконуватися сценарій обгортки порогового значення. Для генерації навантаження ЦП в Інтернеті доступно багато інструментів з відкритим кодом. У цій кваліфікаційній роботі ми використали CPULoadGenerator який доступний на Github [18]. Основна перевага цього інструменту полягає в тому, що він генерує фіксовану величину навантаження за кінцевий час вибірки з вибраними ядрами. Мова програмування – Python, приклад запуску: <CPULoadGenerator -l 0.5 -d 20 -c 1>. У наведеній команді скрипт виконує 50% навантаження за 20 секунд в одному ядрі.

2.5 Створення контейнерів

Проблема упаковки контейнера була запропонована Чекурі та Ханною у 1990 році і довела, що процес упаковки контейнера є APX-складною. Суніл [19] та інші запропонували концепцію об'єднання використання пам'яті та ЦП для вирішення проблеми упаковки. У цьому підрозділі надано результати детального аналізу та порівняння порушення SLA, споживання енергії під час міграції VM.

Векторні контейнери різного розміру є важливою проблемою, яка привертає увагу людини. Узагальнення для цієї проблеми було введено Michael [20], яке корелює генерацію можливих рішень для проблем розміщення віртуальних машин. В роботі також йдеться про різні розглянуті додаткові обмеження та розклад великої проблеми на менші фрагменти.

Консолідація сервера відіграє дуже важливу роль в ефективному управлінні хмарним центром обробки даних і оптимізації витрат і споживання енергії. Звичайний процес консолідації серверів був здійснений шляхом заміни фізичних серверів на віртуальні машини. Але Sunil [19] вводить нову техніку для консолідації серверів, а саме RFAware Server Consolidation, яка виконує дефрагментацію залишкових ресурсів, що зменшує залишкові ресурси і, у свою чергу, створює основу для нових виділень віртуальних машин.

Підвищення продуктивності розподілу хмарних ресурсів та консолідації віртуальних машин було викликано Huda [21] за допомогою гібридної моделі шляхом поєднання існуючих генетичних алгоритмів, а саме комбінаційного упорядкування першого підходящого генетичного алгоритму (COFFGA) і комбінаційного упорядкування наступного відповідного генетичного алгоритму (CONFGA). Результати показують, що запропонована гібридна модель перевершує попередні генетичні алгоритми. Мохамед [22] також пропонує гібридний генетичний алгоритм із застосуванням найкращого придатного зменшення (BFD). Результати доводять, що

розгорнута модель дає бажані результати.

Статичний розподіл ресурсів у віртуальних машинах був неефективним і завжди знижує ефективність усієї системи. Використання динамічного розподілу ресурсів у віртуальних машинах може вирішити проблему такої ефективності. Et el Zhuo [23] пропонує алгоритм планування, а саме планування динамічного прогнозу віртуальної машини (VM-DFS), яке аналізує історію споживання пам'яті та приймає рішення про розподіл ресурсів на основі оцінки майбутніх потреб, що дозволяє зменшити кількість фізичних машин. Було виконано кілька оцінок запитів віртуальної машини, і нормалізовані результати показують, що близько 17,08 % фізичних машин можна зберегти.

Кілька віртуальних машин було розгорнуто на одній фізичній машині, щоб підвищити ефективність використання фізичного пристрою, а також знизити вартість і компроміс щодо продуктивності, щоб уникнути покупки нових фізичних пристроїв, які є дорогими. Але, як згадувалося раніше, динамічний розподіл ресурсів для віртуальної машини значно зменшить кількість додаткових фізичних блоків. Chowdhury [24] надає оптимізовані алгоритми, які відповідно зменшують кількість ресурсів і іноді деактивують хости, якщо ресурси використовуються недостатньо, тим самим зменшуючи споживання енергії. Техніка для кластеризації віртуальних машин була також запропоновано, що уможливлює новий шлях для міграції ресурсів за допомогою зворотного зв'язку з ЦП і пам’яттю.

Відомий підхід до консолідації сервера за допомогою технології віртуалізації для оптимізації ефективності центру обробки даних. Гра з розподілом і перерозподілом ресурсів для кожної віртуальної машини призведе до досягнення цільових показників продуктивності фізичної машини, але це призведе до деяких обмежень в управлінні системою, особливо під час роботи з великомасштабними центрами обробки даних. Оптимізована методика одночасного управління витратами ресурсів, споживанням електроенергії та витратами на теплову розсіювання була

розроблена та Ель Джінг [25], яка показує гарні результати порівняно з іншими методами та підходами.

Аналіз продуктивності генетичного алгоритму перевпорядкування групування (RGGA) у різних сценаріях був пояснений та Ель Девідом [26]. Це показує ефективність та результативність цього алгоритму, який розглядав і тестував у сценарії з одним обмеженням із подальшим сценарієм із багатьма обмеженнями та порівнював обидва підходи.

Пропускна здатність мережі відіграє важливу роль для контролю трафіку даних через декілька віртуальних машин на фізичній машині, що призводить до погіршення продуктивності. Jianhai [27] пропонує новий метод, який надає можливість вирішення проблеми пропускної здатності мережі, а саме метод Affinity-Aware Grouping для розподілу віртуальних машин (AAGA). Була розгорнута та протестована експериментальна установка використання віртуальних машин на фізичній машині з евристичним алгоритмом упаковки ящиків. Результати показують, що AAGA перевершує NAGA.

3 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РОЗПОДІЛЕННЯ ВІРТУАЛЬНИХ МАШИН ЗА ХМАРНИМИ РЕСУРСАМИ

3.1 Огляд застосування методів вирішення задач роботи

Упаковка контейнера у загальному випадку є NP-складною проблемою. Отже, ця кваліфікаційна робота поділяється на дві частини. Це теоретичний підхід (з використанням CloudSim) та практичний підхід (з реалізацією у середовищі OpenStack).

Етапи впровадження вказаних підходів:

Етап 1. Виконується моделювання алгоритмів на базі розроблених методів у симуляторі CloudSim з різними наборами даних.

Етап 2. Встановлюються порогові значення та запускається сценарій обгортки Python.

Етап 3. Зберігаються метрики вузла в локальних (VM) і глобальних (Host) параметрах у базі даних.

Етап 4. Обирається кращий алгоритм упаковки для розміщення ВМ у середовищі OpenStack.

Для того, щоб отримати відповідні результати, дуже важливо протестувати модель на реальних даних. За замовчуванням симулятор CloudSim використовує алгоритм белоглазова для розміщення ВМ. Розширення класу PowerVmAllocationPolicyMigrationAbstract optimizeAllocation для реалізації алгоритмів розміщення Bin в симуляторі CloudSim. Порівняти всі алгоритми поруч за змінними даними не так просто. Отже, ми використовуємо фіксовані параметри для тестування цього алгоритму з різними наборами даних. Основною метою цієї роботи є мінімізація використання ресурсів РМ без впливу на SLA. Тому ми фіксуємо політику міграції як «мінімальний час міграції» під час розміщення.

Симулятор CloudSim за замовчуванням містить трасування робочого навантаження. Для обох кластерів Google параметри Bitbrain та індекс

параметрів відрізняються. Отже, нам потрібно конвертувати кластер Google і набір даних Bitbrain у потрібний формат введення YAML. Після перетворення набору даних ми надаємо вихідний файл в симулятор. Для порівняння алгоритмів нам потрібно мати фіксовані параметри в конфігурації хоста і віртуальної машини.

3.2 Сбор даних про рабоче навантаження

3.2.1 Параметри кластера Google

Набір даних робочого навантаження кластера Google збирається з обчислювальних ресурсів Google (кластер екземплярів kubernetes). Цей кластер Google являє собою набір машин, упакованих у стійки та підключених до високошвидкісної кластерної мережі. Цей набір даних містить відстеження робочого навантаження за 29 днів, що містить 12 500 машин. Нижче пояснюються параметри, які містять набір даних у таблиці подій завдання. Параметри в наборі даних:

- ID;
- час початку;
- ядра;
- використання ЦП (МГц);
- використання пам'яті (КБ);
- забезпечена пам’ять (КБ);
- використовуваний процесор.

3.2.2 Параметри кластера Bitbrains

Цей набір даних є розподіленим центром обробки даних від Bitbrains (голландського постачальника послуг), який керував хостингом та бізнес-обчисленнями для підприємств (1750 віртуальних машин). Відстеження

робочого навантаження Bitbrain — це дві частини, які є швидким сховищем і трасуванням робочого навантаження Rnd. Швидке відстеження робочого навантаження зібрала 1250 віртуальних машин за 1 місяць із загальною кількістю 4057 ядер. Там, де трасування робочого навантаження Rnd зібрала дані про 500 віртуальних машин за 3 місяці з 1444 ядрами. Кожні параметри віртуальної машини зберігаються окремо у форматі .csv[28].

Параметри в наборі даних:

- позначка часу;
- ядра процесора;
- передбачена ємність ЦП;
- продуктивність ЦП (МГц);
- пам’ять забезпечена (КБ);
- використана пам'ять (КБ).
- пропускна здатність читання диска (КБ/с);
- пропускна здатність запису на диск (КБ/с);
- отримана мережа пропускна здатність (КБ/с);
- пропускна здатність мережі (КБ/с).

3.2.3 Параметри кластера Planet Lab

Набір відстежень використання віртуальних машин Planet lab (1000 віртуальних машин), зібраних протягом 10 випадкових днів у березні та квітні 2021р. За замовчуванням, відстеження робочого навантаження лабораторії включено в симулятор CloudSim. Дані містять характеристики використання ЦП тисячі віртуальних машин, розташованих у більш ніж 500 місцях по всьому світу. У цьому наборі даних використання ЦП кожної віртуальної машини вимірюється за 5 хвилин.

Параметри в наборі даних:

- ID;
- ядра процесора;

- потужність процесора;
- ОЗУ ВМ (КБ);
- пропускна здатність (КБ/с);
- ємність диска (КБ).

3.2.4 Параметри моделі генерації ресурсів

Цей набір даних генерується програмою на основі python за допомогою бібліотеки pyyaml. У цьому наборі даних навантаження ЦП генерується постійним значенням, наприклад 50%, 30% використання ЦП. Ми можемо генерувати набір даних за такими параметрами, як оперативна пам’ять, ядра, потужність ЦП тощо.

Параметри в набір даних:

- ID;
- ядра процесора;
- потужність процесора;
- ОЗУ ВМ (КБ);
- пропускна здатність (КБ/с);
- ємність диска (КБ).

3.2.4 Параметри постійної бази даних

Він також має ті самі параметри, що й постійна база даних. Цей набір даних генерує динамічну зміну політики використання за періодичною, випадковою, постійною, спадною закономірністю.

Параметри в набір даних:

- ID;
- ядра процесора;
- потужність процесора;
- ОЗУ ВМ (КБ);

- пропускна здатність (КБ/с);
- ємність диска (КБ).

3.3 Підготовка набору даних

Існує два типи вхідних даних, які ми повинні надати симулятору CloudSim. За цим в основному слідує формат файлу YAML (зчитувана мова серіалізації даних): ресурси хосту та ресурси ВМ.

Зразок введення для ресурсів хоста:

Пропускна здатність: 1000000

Оперативна пам'ять: 4096

Ядра: 2

ID: [3]

PowerModel: *id002

Ємність диска: 1000000

Ємність процесора: [2660]

4 РЕАЛІЗАЦІЯ ЕКСПЕРЕМЕНТІВ

4.1 Експериментальне налаштування середовища OpenStack

Експеримент для оцінки упаковки Bin установки віртуальних машин передбачав використання 5 вузлів, наданих через платформу Mirantis. Специфікації кожного сервера:

- 4 обчислювальні вузли;
- 1 вузол контролера;
- 8-ядерний процесор;
- 16 ГБ ОЗП;
- ОС Ubuntu 16.04 (Xenial);
- 256 ГБ пам'яті.

OpenStack — це дуже потужне хмарне середовище. В основному він класифікується як обчислювальний, контролер і мережевий вузол. У цьому практичному підході обчислювальний вузол схожий на фізичну машину, а всередині віртуального екземпляра знаходиться віртуальна машина. Нарешті, консолідація ВМ буде виконана відповідно до алгоритму розміщення.

У цій реалізації використовуються два типи зберігання даних. Сховище даних-1 розгорнуто в кожному обчислювальному вузлі, щоб воно могло в реальному часі збирати використання ЦП кожної віртуальної машини та використання ЦП вузла гіпервізора. Ці відомості про сховище даних надаються центральній базі даних для обчислення показників продуктивності. Ця центральна база даних розгорнута у вузлі контролера OpenStack. Дані, що зберігаються як середній мегагерц, споживаний кожною віртуальною машиною під час останнього вимірювання часу t. На практиці використання CPU C(t1, t2) функції віртуальної машини описано нижче. Тут t1 — початковий час, а t2 — кінцевий час використання ЦП віртуальної машини. Це використання ЦП кожного вузла, що зберігається відповідно до

відмітки часу. Обчислення виконується згідно наступного рівняння:

$$\overline{C_i^V}(t_1,t_2)=\frac{n_i^v F\left(\tau_i^v(t_2)-\tau_i^v(t_1)\right)}{t_2-t_1} \quad (4.1)$$

Використання ЦП C є функцією VM з номером i на інтервалі часу між t_1, t_2 з n кількістю ядер віртуальної машини. F – частота одноядерної віртуальної машини τ – це процесор, який використовує ВМ[2].

Жива міграція віртуальної машини — це перенесення віртуальної машини одного обчислювального вузла на інший обчислювальний вузол без простоїв. Ця оперативна міграція віртуальної машини вплине на продуктивність програмної системи. Таким чином, затримка збільшується, якщо виконується величезна кількість операцій розміщення ВМ. Вільям [29] пояснює погіршення продуктивності програми та час простою. Середній очікуваний час простою очікується на рівні 10% використання ЦП. За запропонованим алгоритмом необхідно виконати мінімальну міграцію ВМ для усунення порушення SLA.

4.2 Дизайн контейнера клієнта OpenStack

OpenStack має дуже широку підтримку в API клієнта python. У цій імплантації контейнера підтримує операції з міграції та підтримки ресурсів середовища OpenStack (рисунок 4.1). Цей контейнер класифікується трима етапами.

Етап 1. Ініціалізація контейнеру.

Етап 2. Виконання алгоритму прийняття рішень.

Етап 3. Розміщення ВМ.

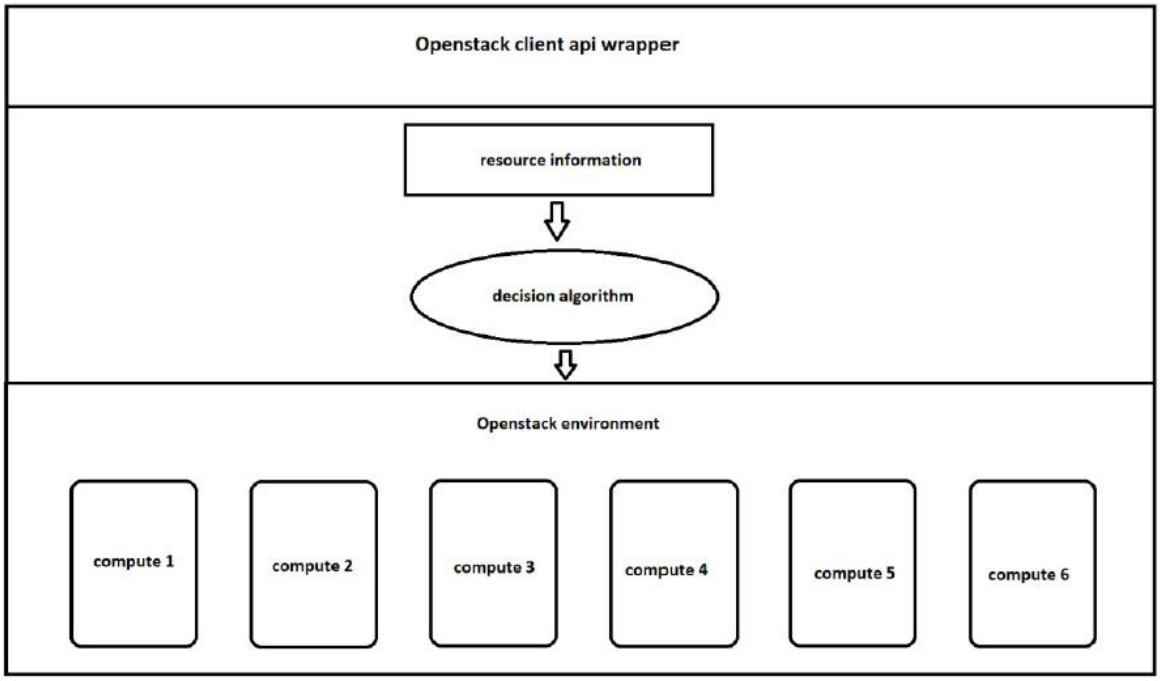

Рисунок 4.1 –Діаграма реалізації OpenStack

4.2.1 Ініціалізація обгортки

Ця обгортка в основному розроблена з використанням OpenStack python API і numpy. Щоб запустити екземпляр у середовищі OpenStack, нам потрібно передати такі параметри, як ім’я зображення, назва тому, ім’я ключа, ідентифікатор мережі та зона доступності обчислень. За замовчуванням планувальник OpenStack Nova розміщуватиме екземпляри в обчислювальному вузлі за допомогою циклічного алгоритму. Тому, використовуючи цю обгортку, перед запуском екземпляра в середовищі OpenStack виконайте внутрішні обчислення за допомогою алгоритму прийняття рішень. Ця обгортка надасть всю необхідну інформацію про ресурси алгоритму прийняття рішень.

4.2.2 Алгоритм прийняття рішень

Обчислюючи доступність ресурсів кожного обчислювального вузла, цей алгоритм прийняття рішень забезпечує вихід імені обчислювального

вузла, де буде запущено екземпляр. У цій частині використовуватимуться алгоритми First fit, Best Fit і Enhanced Best Fit.

4.2.3 Розміщення ВМ

Розміщення ВМ виконується двох типів. Це такі

- нормальне розміщення;
- порогове розміщення.

Звичайне розміщення буде виконуватися щоразу, коли буде запущено екземпляр. Порогові значення кожного обчислювального вузла будуть встановлені користувачем (максимальне використання ЦП, використання пам’яті тощо). Щоразу, коли використання вузла перевищує порогове значення, скрипт автоматично запускається для виконання живої міграції екземплярів між обчислювальними вузлами. Він використовує алгоритм упаковки Bin для вибору нового хоста для віртуальної машини.

Налаштування BIOS кожного РМ необхідно встановити відповідно до технології Wake-on-LAN. Щоб ми могли контролювати параметри потужності обчислювального вузла. Необхідно відправити чарівний пакет за допомогою програми ether-wake.

Після операції розміщення ВМ цей скрипт автоматично вимикає невикористаний обчислювальний вузол за допомогою MAC-адреси:

ether-wake <MAC-адреса>.

Цей контейнер інтегрован з Grafana для відображення використання ресурсів у реальному часі в середовищі OpenStack.

4.3 Монітор OpenStack

Монітор OpenStack в основному розроблений для візуалізації стану хмарного середовища в реальному часі. Цей інструмент розроблено за допомогою клієнтського API клієнта OpenStack і клієнта Influx. InfluxDB —

це база даних часових рядів, тому вона може ефективно зберігати всі показники, використовуючи позначку часу (рисунок 4.2).

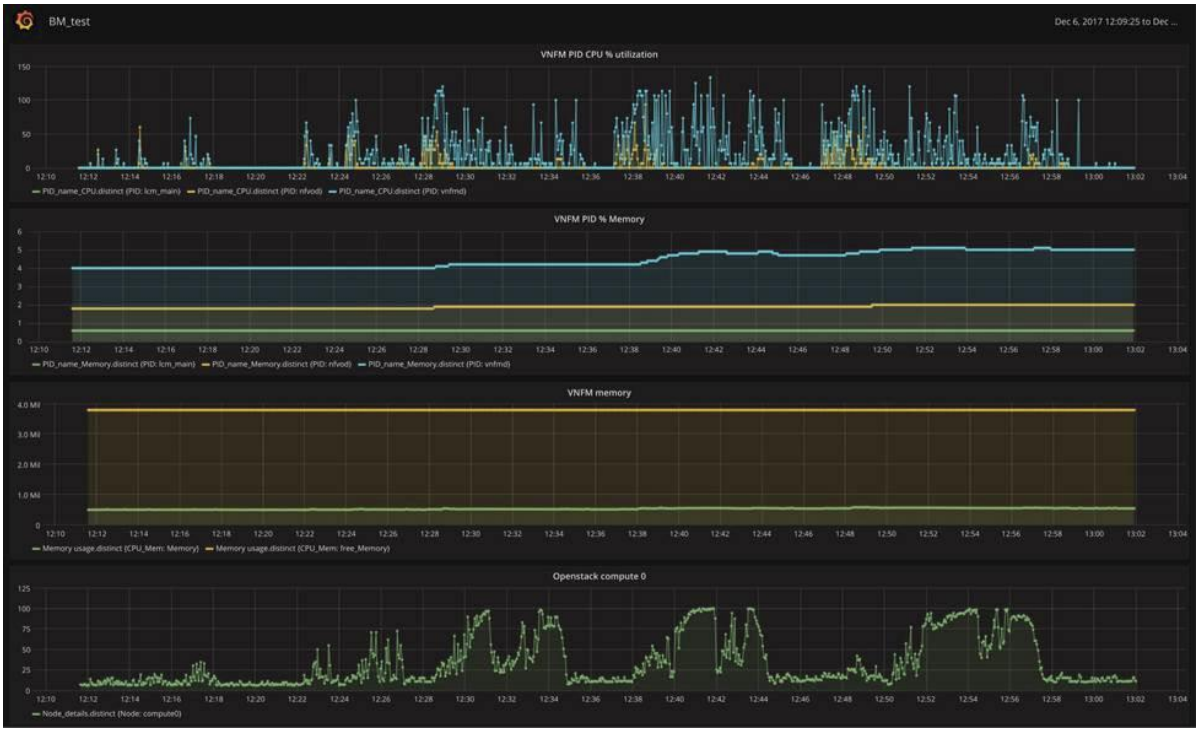

Рисунок 4.2 – Візуалізація приладової панелі Grafana

Знаючи інформацію про всі метрики, ми знаємо, що відбувається збій під час тестування. Цей інструмент вивантажує всі дані метрики з бази даних -2 (обчислювальний вузол) до клієнта InfluxDb.

Основними характеристиками цього монітора OpenStack є:

- використання ЦП обчислювальних вузлів та контролеров;
- кількість VCPU, розгорнутого на вузлі;
- дисковий простір, що використовується в кожному вузлі;
- використання пам'яті вузла.

4.4 Алгоритми

Ефективність упаковки завжди змінюється відповідно до змінних розмірів контейнерів. Таким чином, алгоритм пакування має бути

протестований на контейнерах змінного розміру за короткий проміжок часу з більшою ефективністю.

$$PackingEfficiency = \frac{sum\ of\ VM\ \ resources}{Used\ \ PM\ \ resources} \tag{4.2}$$

Опис алгоритмів пакування контейнерів у наступному порядку:

- алгоритм зменшення першої відповідності та першого підбору;
- найкраще підходить алгоритм зменшення;
- покращений алгоритм найкращої підгонки.

4.4.1 Алгоритм першої відповідності

Алгоритм першої відповідності є одним із найстаріших алгоритмів в управлінні пам'яттю. Реалазацію алгоритма представлено на рисунку 4.3.

Цей алгоритм розміщує контейнери відповідно до принципу FIFO. Цей алгоритм поділяється на два типи:

а) перша підгонка;

б) перша підгонка зменшується.

```
Algorithm 1: First fit decreasing algorithm
Input: Host list, VM list
Output: map_to_VM
1. Sort VM list in order of decreasing CPU utilization
2. For VM in VM list do
3.    for host in Host list do
4.       if host has enough resources to VM then
5.          map_host ← Host
6.    if  map_host != NULL then
7.       add (map_host, VM) to map_to_VM
8.    return map_to_VM
```

Рисунок 4.3 – Алгоритм першої відповідності

4.4.2 Алгоритм зменшення навантаження

У першому алгоритмі зменшення спочатку всі завдання сортуються відповідно до розмірів завдання. Таким чином, у більшості випадків він дає кращу ефективність пакування порівняно зі звичайним алгоритмом першого підбору. Обидва алгоритми подібні, за винятком сортування списку ВМ на початковому кроці. Отже, нижче згадується лише перший алгоритм зменшення (рисунок 4.4).

```
Algorithm 2: Best fit decreasing algorithm
Input: Host list, VM list
Output: map_to_VM
1. Sort VM list in order of decreasing CPU utilization
2. For VM in VM list do
3.    for host in Host list do
4.       if (0 < host utilization > threshold (host, VM))
5.          power _difference ← power (host, VM) - power (host)
6.          If power _difference > max(power) then
7.             map_host ← host
8.             max(power) ← power _difference
9.    if map_host != NULL then
10.       add (map_host, VM) to map_to_VM
11.   return map_to_VM
```

Рисунок 4.4 – Алгоритм зменшення навантаження

4.4.3 Розширений алгоритм найкращої відповідності

Цей алгоритм мало відрізняється від найкращого алгоритму зменшення. У алгоритмі зменшення оптимальної відповідності ми спочатку сортуємо ресурси ВМ і розміщуємо завдання відповідно до різниці потужності між ресурсами після міграції ВМ і до міграції ВМ хоста. Але в цьому підході ми сортуємо як віртуальну машину, так і хост кожну ітерацію (рисунок 4.5).

```
Algorithm 3: Enhanced Best fit algorithm
Input: Host list, VM list
Output: map_to_VM
1. Sort VM list in order of decreasing CPU utilization
2. For VM in VM list do
3. Sort host list in order of decreasing CPU utilization
4.    for host in Host list do
5.       if (0 < host utilization > threshold (host, VM))
6.             map_host ← host
7.    if map_host ! = NULL then
8.       add (map_host, VM) to map_to_VM
9.    return map_to_VM
```

Рисунок 4.5 – Розширений алгоритм найкращої відповідності

4.5 Результати моделювання та експерементів

У цьому розділі потрібно порівняти різні параметри для алгоритму «Найкраще підходження», «Зменшення найкращого підходу», «Зменшення першої підгонки» та «Першої підгонки» за допомогою набору даних Planet lab, Google compute, Bitbrain і Raw. Отже, ми провели моделювання 16 разів, щоб охопити всі наші тестові випадки.

Порівняння показників для алгоритму розміщення:

- споживання енергії;
- кількість міграції ВМ;
- кількість вимкнених хостів;
- середній час міграції;
- порушення SLA.

Споживання енергії залежить від того, скільки РМ використовується під час роботи. Він також змінюється відповідно до використання ресурсів РМ. З рисунка 4.6 чітко видно, що покращений алгоритм найкращої

відповідності дає кращі результати в усіх тестових випадках, оскільки ми сортуємо ресурси хоста та ресурси віртуальної машини на кожній ітерації.

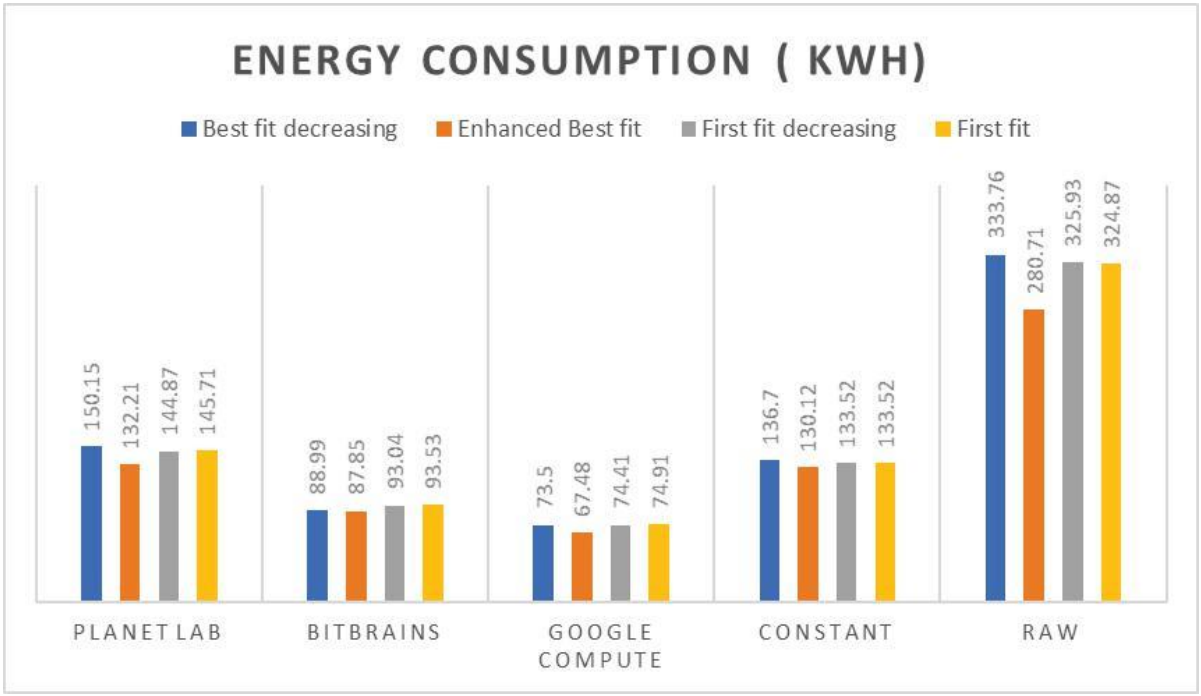

Рисунок 4.6 – Порівняння споживання енергії (кВт год)

У таблиці нижче представлені середнє значення та довірчий інтервал для кожного набору даних.

Таблиця 4.1 – Дані експерементів: порівняння спожитої енергії

Набір даних	Середній	Стандартне відхилення	95% ДІ
Planet Lab	143,24	7.71	135 до 151
Bitbrains	90,85	2,85	88.1 до 93.7
Google compute	72,58	3.45	69,2 до 76
Постійний набір даних	133,47	2.69	130-136
Набір необроблених даних	316,32	24.1	292 до 340

Менша кількість міграцій ВМ означає ефективну консолідацію, менше трафіку в хмарній мережі та менше порушення SLA для міграції ВМ. У цьому сценарії покращений алгоритм найкращого підходу дає кращі результати порівняно з іншими алгоритмами, за винятком постійної бази даних. У постійному наборі даних зменшення найкращої відповідності дає кращі результати.

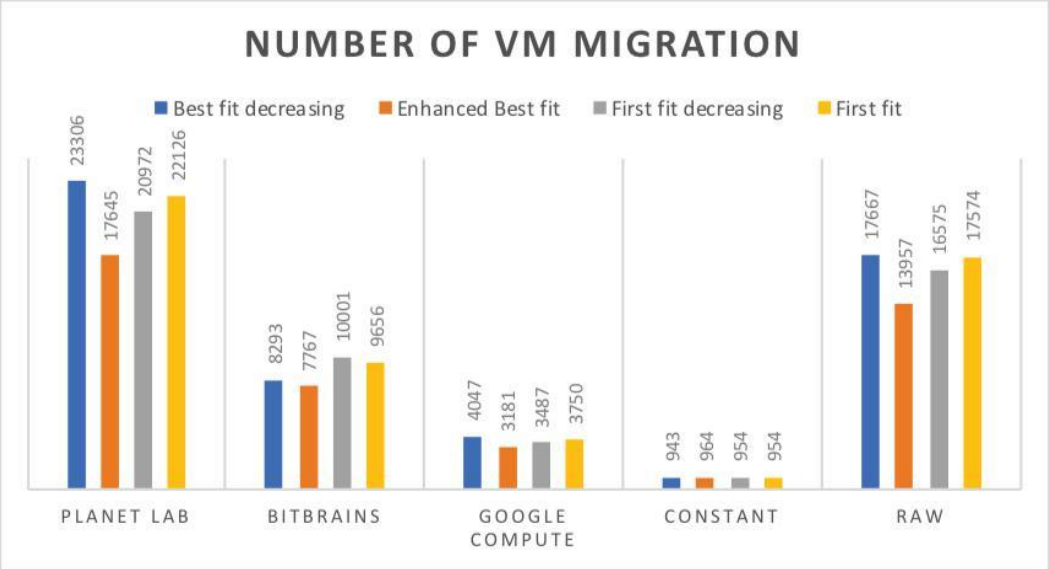

Рисунок 4.7 – Порівняння кількості міграції ВМ

У таблиці нижче представлені середнє значення та довірчий інтервал для кожного набору даних.

Таблиця 4.2 – Дані експерементів: кількість міграції

Набір даних	Середній	Стандартне відхилення	95% ДІ
Planet Lab	21012.25	2440	18600 до 23400
Bitbrains	8929,25	1070	7880 до 9980
Google compute	3616,25	369	3260 до 3980
Постійний набір даних	953,75	8.59	946 до 962
Набір необроблених даних	16443,25	1730	14700 до 18100

У випадку зміни вимкнених хостів мінімальна кількість вимкнення хоста відноситься до найкращого ефективного алгоритму. З рисунку 4.8 легко помітити, що кількість вимкнених хостів, які найкраще підходять, значно зменшує кількість методів. Це означає, що коли хост переводиться в сплячий режим, він залишається там протягом тривалого часу, що доводить ефективність наших алгоритмів.

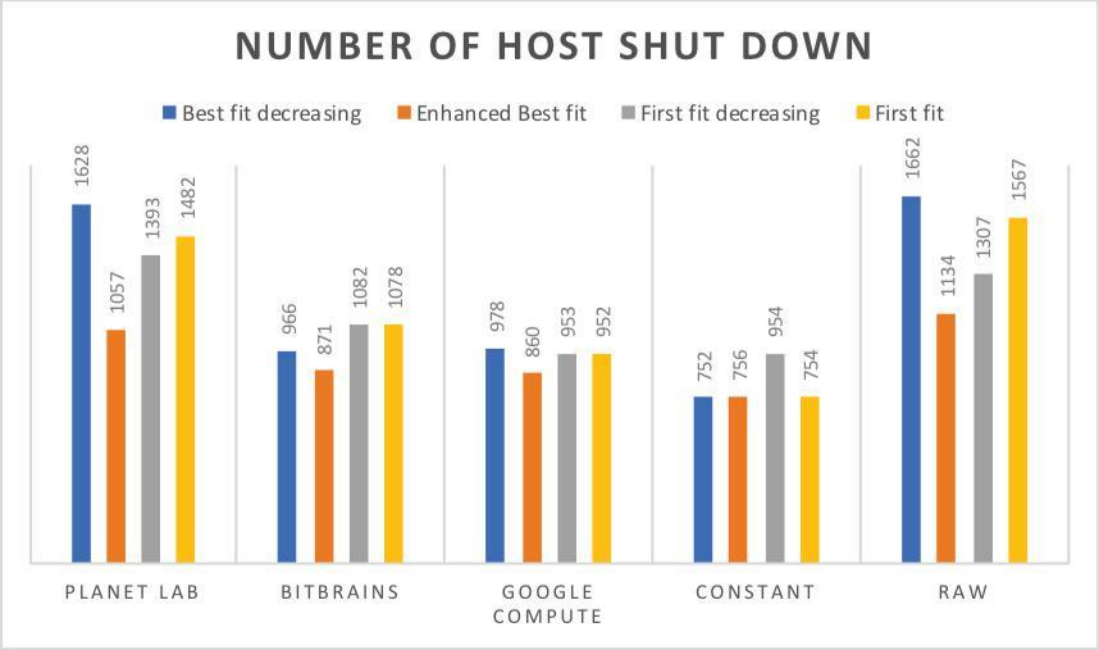

Рисунок 4.8 – Порівняння кількості вимкнених хостів

У цьому сценарії покращений алгоритм найкращого підходу дає кращі результати порівняно з іншими алгоритмами, за винятком бази даних із постійним завантаженням. У постійному наборі даних алгоритм першого підбору дає кращі результати.

Середнє порушення SLA – один із важливих показників для вимірювання ефективності упаковки в PM. Як обговорювалося раніше, мінімальне порушення SLA буде заявлено як ефективний алгоритм. Отже, у цій симуляції ми використовуємо мінімальний час міграції як бажану політику розподілу в CloudSim. Отже, результати цього розділу виглядають майже схожими на інші алгоритми (рисунок 4.9).

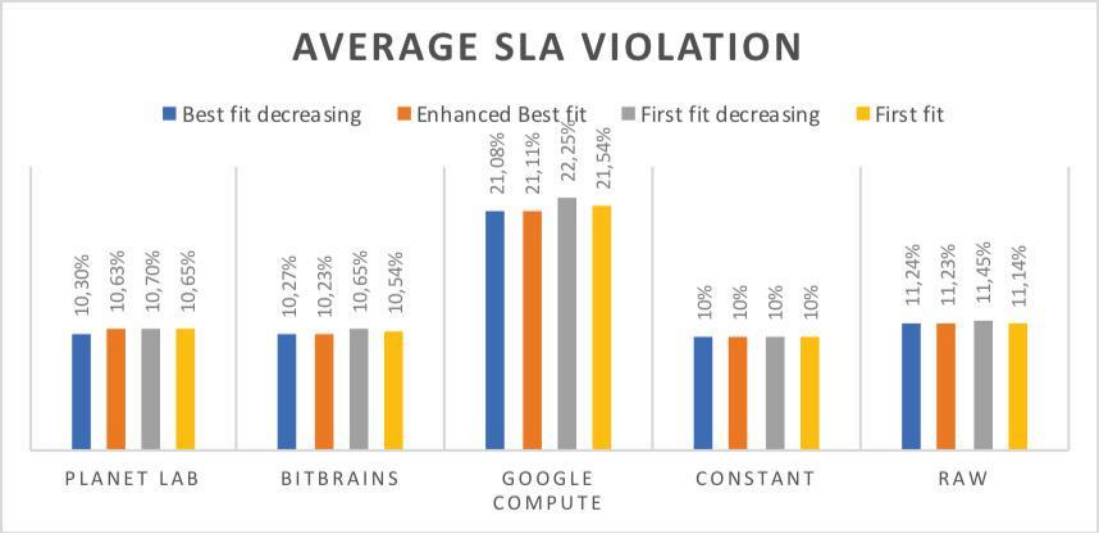

Рисунок 4.9 – Порівняння середніх порушень SLA

У таблиці нижче представлені середнє значення та довірчий інтервал для кожного набору даних (таблиця 4.3).

Таблиця 4.3 – Дані експерементів: значення порушень SLA

Набір даних	Середній	Стандартне відхилення	95% ДІ
Planet Lab	10.6	0,182	10.4-10.8
Bitbrains	10.4	0,205	10.2-10.6
Google compute	21.5	0,545	21-22
Постійний набір даних	10	0	10 до 10
Набір необроблених даних	11.3	0,131	11.2-11.4

Час виконання називається часом обробки моделювання для розміщення всіх віртуальних машин. У цьому випадку як кращий підхід буде вказано мінімальний час виконання. Тому що він завершує розміщення всіх операцій міграції віртуальної машини в Datacenter. У цьому сценарії розширений алгоритм зменшення першої відповідності дає кращі результати порівняно з іншими алгоритмами (рисунок 4.10).

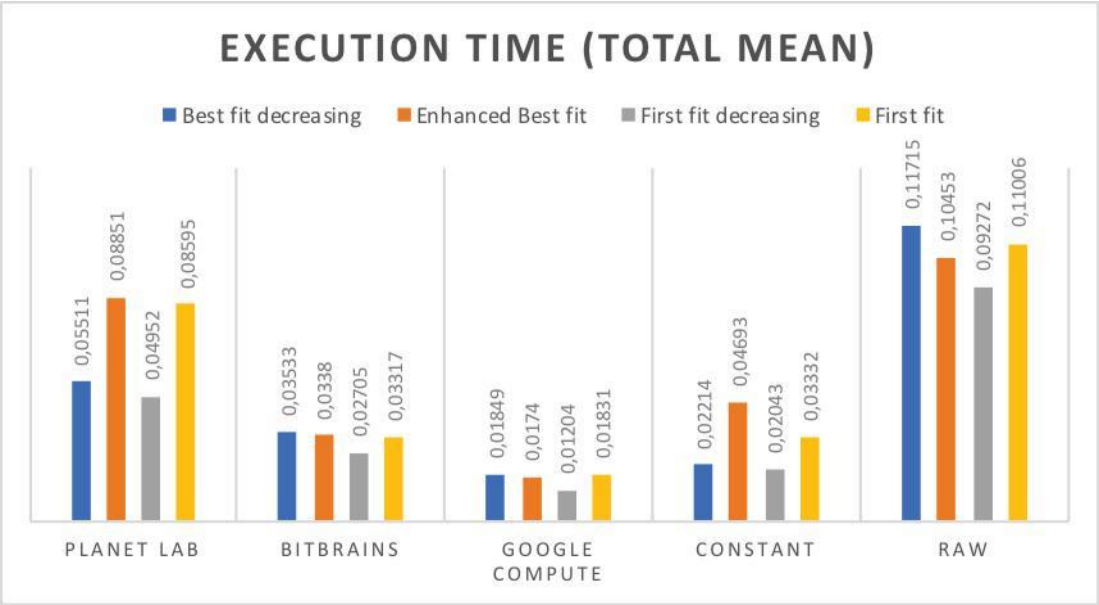

Рисунок 4.10 – Порівняння часу виконання (сек)

ВИСНОВКИ

Це дослідження було зроблено для того, щоб знайти, оцінити та оптимізувати енергоспоживання центрів обробки даних за допомогою алгоритмів Bin Packing. Результати дослідження відповідають очікуванням, і споживання енергії можна оптимізувати за допомогою алгоритмів розміщення. Для моделювання алгоритмів був використаний симулятор CloudSim.

У наведеному вище розділі ми порівнюємо різні параметри з різними алгоритмами. У більшості випадків вдосконалений алгоритм найкращого підбору дає кращі результати. У сценарії Час виконання, алгоритм зменшення першого підбору дає кращі результати порівняно з іншими алгоритмами.

Для практичного підходу використовується платформа OpenStack, і було розроблено оболонку python для інтеграції цієї моделі в середовище OpenStack за допомогою клієнта python. Панель інструментів Grafana використовується для моніторингу інформації про ресурси OpenStack в реальному часі. Переміщення операцій ВМ виконується між обчислювальними вузлами. Для вибору обчислювального вузла був використаний розширений алгоритм Best fit.

Результати довели, що енергію можна оптимізувати в хмарних центрах обробки даних за допомогою алгоритмів упаковки Bin, а продуктивність фізичних машин істотно не погіршується.

Це дослідження в основному було зосереджено на проблемі упаковки контейнера в центрах обробки даних. У майбутньому, застосовуючи машинне навчання, це може дати вдосконалене рішення для вирішення проблеми упаковки Bin[30]. Необхідні дослідження для вирішення 3-D алгоритму упаковки контейнера в середовищі реального часу. Цю модель можна покращити, використовуючи алгоритми навчання з підкріпленням.

Краще мати нову систему моделювання замість CloudSim з великою кількістю нових функцій, як-от енергоспоживання обладнання. Спроектуйте цей симулятор на Python, щоб він міг легко інтегруватися з іншими проектами з відкритим кодом.

Ми припускали, що дослідження проводитимуться в неоднорідному середовищі. Однак реалізувати цю модель у великому середовищі було б краще для досягнення найкращих результатів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Волк М. О., Саранча С. М., Гвоздецький Д. П., Ольшанська Т. І. Методи розподілення віртуальних машин за хмарними ресурсами. XII міжнародна науково-технічна конференція “Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління”. Баку, Харків, Жиліна. Том 1., 27-28 квітня 2022р., с. 70

2. M. Mishra and A. Sahoo, “On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach,” in 2011 IEEE 4th International Conference on Cloud Computing, 2011, pp. 275–282.

3. M. Mishra and U. Bellur, “Whither Tightness of Packing? The Case for Stable VM Placement,” IEEE Transactions on Cloud Computing, vol. 4, no. 4, pp. 481–494, Oct. 2016.

4. M. Forsman, A. Glad, L. Lundberg, and D. Ilie, “Algorithms for Automated Live Migration of Virtual Machines,” Journal of Systems and Software, vol. 101, pp. 110–126, 2015.

5. H. Liu, H. Jin, C.-Z. Xu, and X. Liao, “Performance and energy modeling for live migration of virtual machines,” Cluster Comput, vol. 16, no. 2, pp. 249–264, Jun. 2013.

6. A. Beloglazov and R. Buyya, “Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints,” IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 7, pp. 1366–1379, Jul. 2013.

7. A. Beloglazov and R. Buyya, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers,” Concurrency Computat.: Pract. Exper., vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

8. A. Beloglazov and R. Buyya, “OpenStack Neat: a framework for

dynamic and energyefficient consolidation of virtual machines in OpenStack clouds," Concurrency Computat.: Pract. Exper., vol. 27, no. 5, pp. 1310–1333, Apr. 2015.

9. A. Andrae, "Total Consumer Power Consumption Forecast," 05-Oct-2017.

10. T. D. Burd and R. W. Brodersen, "Design issues for Dynamic Voltage Scaling," in Proceedings of the 2000 International Symposium on Low Power Electronics and Design, 2000. ISLPED '00, 2000, pp. 9–14.

11. K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," IEEE Internet Computing, vol. 16, no. 1, pp. 69–73, Jan. 2012.

12. R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," IEEE Communications Magazine, vol. 51, no. 11, pp. 24–31, Nov. 2013.

13. M. R. Ahmadi and D. Maleki, "Performance evaluation of server virtualization in data center applications," in 2010 5th International Symposium on Telecommunications, 2010, pp. 638–644.

14. N. Jain and S. Choudhary, "Overview of virtualization in cloud computing," in 2016 Symposium on Colossal Data Analysis and Networking (CDAN), 2016, pp. 1–4.

15. L. Yan, "Development and application of desktop virtualization technology," in 2011 IEEE 3rd International Conference on Communication Software and Networks, 2011, pp. 326–329.

16. S. Abidi, S. Krichen, E. Alba, and J. M. Molina, "Improvement heuristic for solving the one-dimensional bin-packing problem," in 2013 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO), 2013, pp. 1–5.

17. «Хмарна лабораторія: флагманські проекти – Gridbus та Cloudbus». [Онлайн]. Доступно: http://www.cloudbus.org/cloudsim/. [Дата звернення: 10 травня 2018].