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)

Метод використання ресурсів в хмарних системах

(тема)

Виконав:

студент II курсу, групи КСМзм-21-1

Стельмахова А.С.
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: зав. каф. Коваленко А.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(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-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Стельмаховій Анастасії Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Метод використання ресурсів в хмарних системах

затверджена наказом по університету від " 24 " жовтня 2022 р. № 178 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2022 р.

3. Вхідні дані до роботи
Kubernetes
хмарні технології
ресурси

4. Перелік питань, що потрібно опрацювати у роботі
Технології забезпечення високошвидкісних обчислень
Методика вибору оптимальної хмарної інфраструктури
Реалізація формалізованого методу

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 15 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	Отримання завдання. Аналіз літератури	25.10.2022–04.11.2022	
2	Огляд існуючих методів та алгоритмів.	05.11.2022–11.11.2022	
3	Аналіз існуючих реалізацій	12.11.2022–19.11.2022	
4	Розробка та реалізація методу	20.11.2022–29.11.2022	
5	Моделювання	30.11.2022–03.12.2022	
6	Отримання результатів	04.12.2022–06.12.2022	
7	Оформлення ПЗ	07.12.2022–12.12.2022	

Дата видачі завдання 24 жовтня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ зав. каф. Коваленко А.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 58 с., 8 рис., 1 дод., 27 джерел.

ВІРТУАЛІЗАЦІЯ, ЕФЕКТИВНІСТЬ, ОПЕРАЦІЙНА СИСТЕМА, ОПЕРАЦІЙНЕ ОТОЧНЕННЯ, РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, DOCKER, KUBERNETES.

Метою кваліфікаційної роботи є розробка методу використання ресурсів для збільшення обчислювальної продуктивності хмарної системи за допомогою автоматичного ранжування доступних хмарних інфраструктур.

У ході виконання кваліфікаційної роботи Розроблено формалізований метод для автоматизації процесу розгортання застосунків у хмарних інфраструктурах, що дозволяє використовувати нефункціональні вимоги для досягнення високої якості обслуговування. Розроблена концепція ухвалення рішень для розгортання мікросервісів. Отримана ймовірнісна модель, яка використовує метод класифікації еквівалентності.

ABSTRACT

Master’s thesis: 58 pages, 8 figures, 1 appendices, 27 sources.

VIRTUALIZATION, PERFORMANCE, OPERATING SYSTEM, OPERATING ENVIRONMENT, DISTRIBUTED COMPUTING, DOCKER, KUBERNETES.

The major goal of this thesis is to develop a method of using resources to increase the computing performance of a cloud system by means of automatic ranking of available cloud infrastructures.

In order to formalized method was developed for automating the process of deploying applications in cloud infrastructures, which allows using non-functional requirements to achieve high quality of service. The concept of decision-making for the deployment of microservices has been developed. A probabilistic model using the equivalence classification method is obtained.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7
ВСТУП 8
1 ТЕХНОЛОГІЇ ЗАБЕЗПЕЧЕННЯ ВИСОКОШВИДКІСНИХ ОБЧИСЛЕНЬ 11
1.1 Хмарні обчислення 11
1.2 Обчислювальні парадигми 14
1.3 Типи хмарних обчислень 15
1.4 Віртуалізація обчислювальних ресурсів 17
1.4.1 Віртуалізація на основі віртуальних машин 18
1.4.2 Віртуалізація на основі контейнерів 19
1.5 Атрибути якості обчислень 21
1.6 Вибір оптимальної хмарної інфраструктури 22
1.7 Kubernetes 24
2 МЕТОДИКА ВИБОРУ ОПТИМАЛЬНОЇ ХМАРНОЇ ІНФРАСТРУКТУРИ 27
2.1 Імовірнісна модель та Марківські процеси 28
2.2 Розрахунок імовірностей моделі винагороди 29
3 РЕАЛІЗАЦІЯ ФОРМАЛІЗОВАНОГО МЕТОДУ 36
3.1 Концепція процесу автоматичного розгортання інфраструктур 36
3.2 Архітектура програмного комплексу 38
3.3 Система моніторингу 42
3.4 Автоматичне розгортання застосунків 43
ВИСНОВКИ 46
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 47
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 50

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

ПЗ – програмне забезпечення

СКБД – система керування базами даних

IOT – Інтернет речей

VM – віртуальна машина

ВСТУП

З появою концепції Інтернету речей виникла необхідність створення додатків у різних галузях, таких як розумний будинок, розумні міста та населені пункти, промисловість 4.0 та подібних, потребують високої обчислювальної потужності та швидкої передачі даних. Дотримуючись новітніх стандартів розробки програмного забезпечення для архітектури мікросервісів, програмне забезпечення може бути побудовано як набір модульних сервісів, що називаються мікросервісами.

Мікросервіс – це невеликий, незалежний і масштабований сервіс з конкретними бізнесменами, який взаємодіє з клієнтами та іншими сервісами стандартним протоколам із чітко визначеними інтерфейсами. Мікросервіси можна віртуалізувати у формі віртуальних машин чи контейнерів. Таким чином, вони можуть бути розгорнуті по всьому обчислювальному спектру, хмарних центрів обробки даних до обчислювальних вузлів на периферії

Розробка програмного забезпечення на основі мікросервісів підтримується новими підходами, методологіями та інструментами, які обіцяють радикальне покращення життєвого циклу програмного забезпечення [1], [2], [3]. Нові засоби розробки програмного забезпечення підтримують розробку програмного забезпечення шляхом об'єднання існуючих мікросервісів, виявлення та погодження обчислювальних ресурсів з хмарними провайдерами, розгортання та оркестрування під час виконання у мультихмарних обчислювальних середовищах. Однак процес розробки програмного забезпечення не залежить від хмар у тому сенсі, що програмне забезпечення може бути спроектовано та розроблено до вибору інфраструктури розгортання.

Хмарні технології пропонують низку переваг, у тому числі: еластичність, швидке масштабування та низькі витрати. На відміну від традиційних систем високопродуктивних обчислень, хмарні обчислення

надають можливість: використовувати ресурси як сервіс через мережу, такі як: «Інфраструктура як послуга» (IaaS), «Платформа як послуга» (PaaS), "Програмне забезпечення як послуга" (SaaS) і т.д.; варіювати кількість використовуваних ресурсів залежно від вимог користувача; контролювати витрати використання ресурсів.

У цій роботі будуть розглядатися питання, пов'язані з організацією високопродуктивних обчислень в хмарних інфраструктурах з використанням віртуальних машин та їх налаштування відповідно до потреб і переваг користувача до якості обслуговування. При реалізації Інтернету речей нині існує багатий вибір різноманітних програмного забезпечення. При цьому на основі різних сценаріїв використання програмне забезпечення запитує задоволення різних вимог, включаючи вимоги до якості обслуговування. Прикладом такого програмного забезпечення є програма для раннього попередження.

Щоб своєчасно прогнозувати та запобігати катастрофі, така програма взаємодіє з безліччю датчиків і має швидко обробляти великі обсяги даних та адекватно реагувати. З іншого боку, в контексті Інтернету речей також можуть бути розроблені додатки для зберігання та керування даними або відеоконференцій WebRTC у реальному часі та інших застосувань. Через величезної різниці між сценаріями всі ці програми мають різні вимоги до якості. Наприклад, система раннього попередження вимагатиме високої швидкості відгуку, додаток зберігання вимагатиме великої ємності сховища, а додаток для відеоконференцій у реальному часі вимагатиме високої пропускної спроможності та низької затримки. Тому ухвалення рішення про те, де розгорнути програмне забезпечення, яке має задовольняти вимог до високої обчислювальної продуктивності, в континуумі від вузла до хмари, де багато різних доступних інфраструктур для розгортання, є особливо складним.

Перш ніж прийняти рішення про розгортання, розробник програмного забезпечення забезпечення повинен розглянути різні вимоги, пов'язані з

якістю обслуговування, включаючи великий вибір інфраструктур, різні технології віртуалізації, географічний розподіл, якість мережі (наприклад, затримка, пропускна спроможність) та інші вимоги на рівні програми та користувача. Маючи на увазі кількість доступних інфраструктур і велика кількість вимог до якості, розробник програмного забезпечення практично не може вручну вибрати оптимальну інфраструктуру для розгортання програмного забезпечення. Отже, з'являється необхідність у розробці нових підходів, методів та технологій, покликаних покращити процес прийняття рішень та допомогти прийняти адекватні та оптимальні або близькі до оптимальних рішень щодо розгортання додатків у хмарах.

Метою роботи є розробка методу використання ресурсів для збільшення обчислювальної продуктивності хмарної системи за допомогою автоматичного ранжування доступних хмарних інфраструктур.

Об’єктом дослідження технології забезпечення високошвидкісних обчислень на базі хмарної інфраструктури.

Завдання:

- проведення огляду парадигм хмарних обчислень;
- аналіз методів віртуалізації обчислювальних ресурсів;
- аналіз методик вибору оптимальної хмарної інфраструктури та скорочення обчислювальної складності;
- розробка методу на основі розглянутих методик.

1 ТЕХНОЛОГІЇ ЗАБЕЗПЕЧЕННЯ ВИСОКОШВИДКІСНИХ ОБЧИСЛЕНЬ

1.1 Хмарні обчислення

Перші документовані обчислювальні машини були механічні. обчислювальні машини, спроектовані Вільгельмом Шикардом у 1623 та Чарльзом Беббіджем у 1837 році. Незабаром після винаходу електрики Аллан Маркванд підготував проект електричної логічної машини, який був реалізований у 1936 році Бенджаміном Бураком. Проте історія сучасних обчислень почалася у XX столітті з появи потужних обчислювальних машин Z3 та ENIAC, які вважаються першими комп'ютерами у світі [10], [11]. Ці машини поряд з винаходом транзисторів у 1947 році сприяли розвитку мейнфреймів, таких як IBM-704 та IBM System/360. Мейнфрейм є великим універсальним високопродуктивним відмовостійким комп'ютер для виконання сервісного програмного забезпечення. У 1950-х роках XX століття розпочалася поступова еволюція мейнфреймів. Через витрат на купівлю та обслуговування комп'ютерів для мейнфреймів організаціям було непрактично купувати та обслуговувати один комп'ютер для кожного співробітника.

У 1961 році вчений Джон Маккарті запропонував ідею надавати обчислювальні ресурси користувачам як послуга (сервіс) [12]. В результаті були зроблені зусилля для забезпечення розрахованого на багато користувачів доступу до16 мейнфреймів. Агенство DARPA (Defense Advanced Research Projects Agency) в 1963 розробило прототип мейнфрейму, який міг бути використаний двома чи більше користувачами одночасно. Більше того, вони вперше визначили термін «віртуалізація», який потім описував доступ до машини відразу кількома користувачами. Іншим важливим етапом для розвитку хмарних обчислень стала розробка система зв'язку Арпанет у 1969 році [13], яка вважається попередником Інтернету. Це

була система зв'язку, яка залишалася доступною, навіть якщо деякий з її вузлів відмовив в обслуговуванні, та була заснована на ідеї обміну даними між різними комп'ютерами або комп'ютерними мережами.

У наступний період, з 1970-х до 1990-х років, було представлено багато технологічних досягнень, необхідних для справжніх хмарних обчислень, таких як покращена концепція віртуалізації, а також зростання віртуальних приватних мереж (VPN) та Інтернету.

У 1970-х роках, IBM випустила операційну систему під назвою VM, яка дозволяла користувачам у її мейнфреймових IBM System/370 мати кілька віртуальних систем, тобто віртуальних машин однією фізичному вузлі. Це змінило існуюче визначення віртуалізації, яке почало описувати створення віртуальних машин, що діють як незалежні фізичні машини з повнофункціональною операційною системою. Кожна віртуальна машина працювала з власними операційними системами чи гостьовими операційними системами, які мали власну пам'ять, процесор та жорсткі диски, а також компакт-диски, клавіатури та мережні пристрої, незважаючи те що ці ресурси були загальними.

У 1990-х роках нові технології, такі як Інтернет та обчислювальні мережі, що дозволили забезпечити підключення достатньої кількості комп'ютерів, щоб створювати величезні взаємопов'язані загальні пули для обчислень та зберігання даних. Цей розподілений тип обчислень був названий гридобчислення [14].

До цієї парадигми обчислень виник особливий інтерес у фінансових компаній, які мали тисячі персональних, що не використовуються17 комп'ютерів та серверів, доступних для використання поза робочим часом. Тим не менш, дорогі інфраструктури, мережеві або програмні збори та обмеження, пов'язані з грід-обчисленням, призвели до появи парадигми хмарних обчислень. 1997 року професор Рамнат Челлапа з Університету Еморі визначив хмарні обчислення як нову «обчислювальну парадигму, в якій межі обчислень визначатиметься економічним обґрунтуванням, а не

лише технічними обмеженнями» [15].

Через два роки після визначення хмарних обчислень Salesforce.com стала першою компанією, яка запропонувала використання бізнес-додатків через Інтернет, і оголосила про появу програмного забезпечення як послуги (SaaS). Іншими словами, це вважається першою офіційною реалізацією хмарних обчислень. З початку 2000-х років хмарні обчислення набули популярності, оскільки компанії стали краще розуміти свої послуги та корисність. Наприклад, у 2002 році Amazon представила свої роздрібні веб-сервіси, а потім запустила Amazon Web Services, які пропонують онлайн-сервіси, такі як: зберігання, обчислення та «штучний інтелект» іншим веб-сайтам чи клієнтам.

Крім того, Amazon представила Elastic Compute Cloud (EC2), що дозволяє користувачам орендувати віртуальні машини і запускати там свої власні програми. У той же період Google та Apple також запустили свої власні хмарні сервіси, які дали користувачам можливість зберігати та редагувати документи. У 2012 році Oracle представила Oracle Cloud, пропонуючи три базові умови для бізнесу: IaaS (інфраструктура як послуга), PaaS (платформа як послуга) та SaaS (Програмне забезпечення як послуга). Однак виявилось, що хмарні обчислення також мають деякі недоліки і в деяких випадках вони не можуть задовольнити всі необхідні стандарти якості у багатьох IoT сценаріях.

У результаті в 2012 році компанія Cisco представила нову обчислювальну парадигму, звану туманним обчисленням [16]. Крім того, у 2014 році Карім Арабі запропонував іншу, більш розподілену обчислювальну парадигму, яку називають периферійними обчисленнями [17].

З 2015 року створено декілька організацій, таких як: OpenFog consortium [18], Cloud Native Computing Foundation [19] та European Edge Computing Consortium [20], покликаних допомогти просунути обчислювальні технології та узгодити технологічну галузь із розвитком обчислювальних технологій.

1.2 Обчислювальні парадигми

Терміни «периферійні обчислення», «туманні обчислення», «хмарні обчислення» розрізняють між собою такі властивості як: продуктивність обчислень, продуктивність мережі та географічний розподіл [21].

Хмарні обчислення – це обчислювальна парадигма, що пропонує централізовані обчислення, що забезпечують високу обчислювальну потужність та високу пропускну здатність у великих центрах обробки даних. Тим не менш, вона також має деякі недоліки, особливо в області IoT. Наприклад, на продуктивність хмарних обчислень зазвичай сильно впливає кругова затримка мережі через відстань між клієнтськими пристроями та інфраструктурами обробки даних. Таким чином, хмарні обчислення доповнюються новими обчислювальними парадигмами, що з'являються розширюють обчислювальні можливості ближче до периферії мережі. Туманні та хмарні обчислення взаємопов'язані (рисунок 1.1).

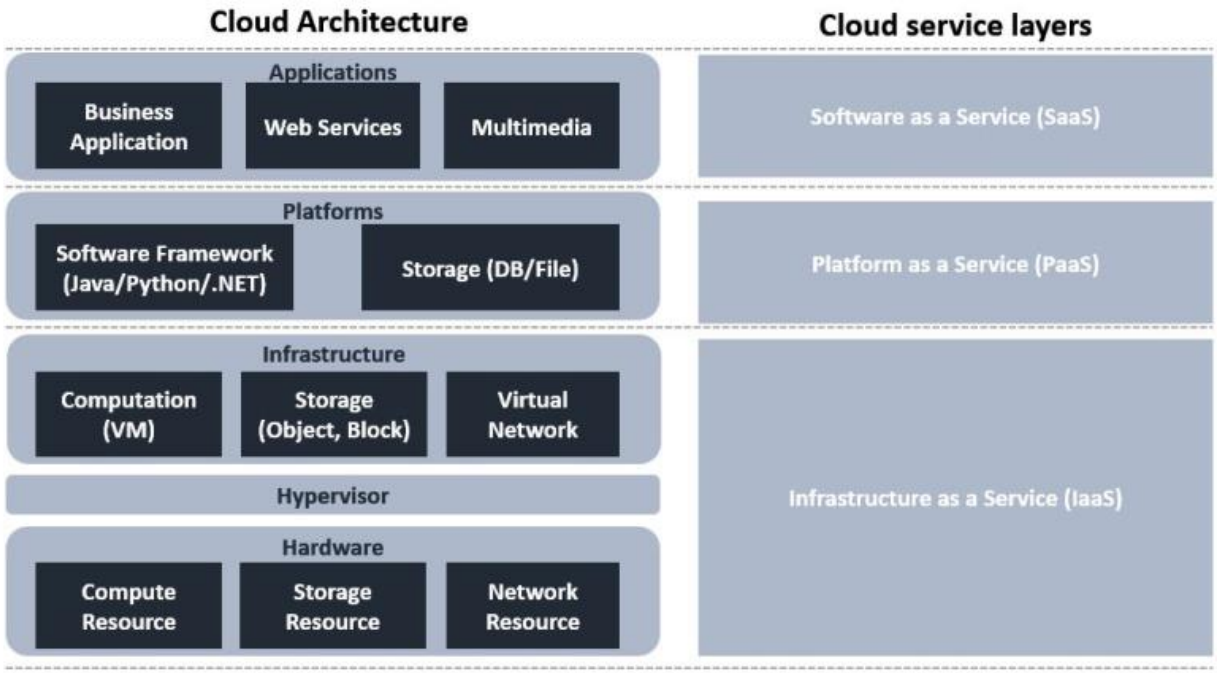

Рисунок 1.1 – Архітектура хмарних обчислень та рівні сервісу

В природі туман знаходиться ближче до землі, ніж хмари, тому в технологічному світі обчислення туман може сприйматися як обчислювальні ресурси, які існують між сенсорними пристроями та традиційними хмарними інфраструктурами [16]. Інфраструктура туманних обчислень складається з невеликих і досить потужних центрів обробки даних у безпосередній близькості від периферії мережі. Їх метою є підтримка IoT додатків, які вимагають високопродуктивних обчислень з покращеною продуктивністю мережі. Крім того, основна відмінність між туманними обчисленнями та хмарними обчисленнями полягає в тому, що хмара – це централізована система, а туман – це розподілена децентралізована інфраструктури.

Периферійні обчислення – це високорозподілений підхід обчислення, що дозволяє обробляти дані на різних багатопроцесорних пристроях, що працюють у безпосередній близькості від датчиків, таких як Raspberry Pi, BeagleBoard або PCDuino [22]. Ця парадигма може мати високу продуктивність мережі, але вона не рекомендується для інтенсивних обчислювальних операцій. Отже, ця парадигма корисна, коли IoT додаток вимагає легких обчислювальних операцій при досягненні високої швидкості відгуку.

1.3 Типи хмарних обчислень

Хмарні обчислення можна класифікувати з двох точок зору: класифікація за типом послуги та класифікація за типом моделі [23].

Найбільш поширеними та широко прийнятими сервісами хмарних обчислень є: інфраструктура як послуга (IaaS), платформа як послуга (PaaS) та програмне забезпечення як послуга (SaaS). IaaS – це модель послуг, яка пропонує користувачам віртуальні інфраструктури, такі як: сховища, сервери або мережеві компоненти як послуга, що підключається. Таким чином, замість того, щоб мати всі компоненти інфраструктури у приміщеннях компанії, ця модель послуг дозволяє компаніям орендувати віртуальні

інфраструктури на платній основі у провайдерів хмарних послуг. PaaS – це модель сервісу, який відноситься до постачання хмарного середовища по вимоги для розробки, тестування, доставки та управління програмними додатками. Вона призначена для швидкої розробки програмного забезпечення забезпечення без необхідності керування базовою інфраструктурою серверів, сховищ, мереж та баз даних.

SaaS – це модель сервісу хмарних обчислень, який включає IaaS і PaaS. SaaS постачає програмні програми через Інтернет відповідно до вимоги та на основі підписки. За допомогою цього виду послуг, постачальник хмарних послуг надає весь програмний пакет у вигляді моделі з оплатою використання. Грунтуючись на можливості організації управляти даними та забезпечувати їхню безпеку, а також на потребах бізнесу, існують три типу моделей хмарного розгортання: публічна, приватна та гібридна модель та модель спільноти.

Публічна хмара – це тип моделі хмарного розгортання, який підтримує всіх користувачів, які хочуть використовувати обчислювальні ресурси, такі як апаратне забезпечення (ОС, ЦП, пам'ять, сховище) або програмне забезпечення (сервер-додаток, база даних) на основі підписки. Така хмарна інфраструктура може належати, керуватися та експлуатуватися комерційною, академічною чи державною організацією. Він існує на території хмарного провайдера.

Приватне Хмара – це інфраструктура, використовувана однією організацією. Така інфраструктура може керуватися самою організацією для підтримки різних груп користувачів, або вона може керуватися постачальником послуг, який дбає про неї як на території організації, так і дистанційно. Приватні хмари дорожчі за публічні через капітальні витрати на їх придбання та обслуговування. Тим не менш, приватні хмари краще справляються з проблемами безпеки та конфіденційності організацій.

Гібридна хмара дозволяє організації одночасно використовувати взаємопов'язані приватні та публічні хмарні інфраструктури. Ця модель

використовується багатьма організаціями, коли їм необхідно швидко розширити свою IT-інфраструктуру, наприклад, при використання загальнодоступних хмар для доповнення ємності, доступної в приватній хмарі. Модель спільноти є хмарною інфраструктуру, яка надається для виключного використання конкретним співтовариством споживачів із організацій, які поділяють спільні стурбованості (наприклад, місія, вимоги безпеки, політика та міркування відповідності). Вона може належати та керуватися однією або декількома організаціями у співтоваристві, третьою стороною чи якоюсь їх комбінацією, а також може існувати всередині організації або за її межами.

1.4 Віртуалізація обчислювальних ресурсів

Віртуалізація ресурсів хмарних обчислень – це фундаментальна технологія, яка дозволяє створити програмне (віртуальне) подання операційних систем, інфраструктур, сховищ та мереж для їх одночасного використання кількох обчислювальних пристроях [24].

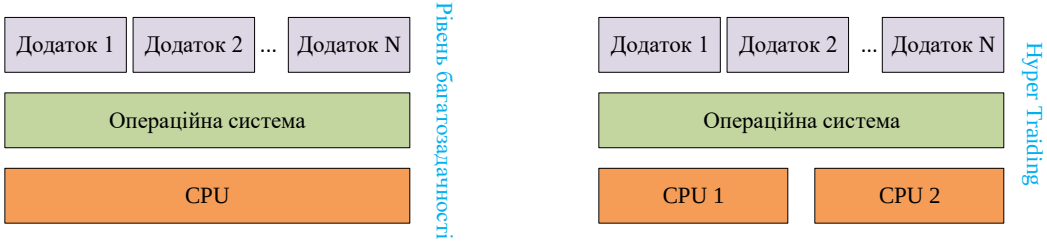

Рисунок 1.2 – Віртуалізація

Порівняно з традиційним обчислювальним підходом (рисунок 1.2), де додатки запускаються в хостовий ОС, віртуалізація є новий рівень, що відокремлює хостову ОС від додатків. Переваг у такої системи багато, наприклад: гнучкість і масштабованість при одночасному суттєвому зниження витрат, збільшення продуктивності та доступності ресурсів,

автоматизації операцій, мінімізації чи усунення простоїв, підвищення безпеки, значне спрощення адміністрування та підтримки тощо.

Найбільш поширеними концепціями віртуалізації є віртуалізація на базі віртуальних машин та віртуалізація на основі контейнерів.

1.4.1 Віртуалізація на основі віртуальних машин

Віртуалізація на основі віртуальних машин – це одна з сучасних методів доставки програмного забезпечення, який використовує концепцію віртуальної машини як ефективного інструменту для пакування програмного забезпечення. Віртуальна машина – це віртуальна вистава чи емуляція фізичний комп'ютер. Віртуальна машина називається гостем, а фізична машина, де вона працює, називається хостом. Рисунок 1.3 зображує інфраструктуру, в якій розміщено 2 віртуальні машини. Кожна з цих віртуальних машин може незалежно запускати свої власні операційні системи та програми, хоча вони спільно використовують вихідні ресурси хостової інфраструктури.

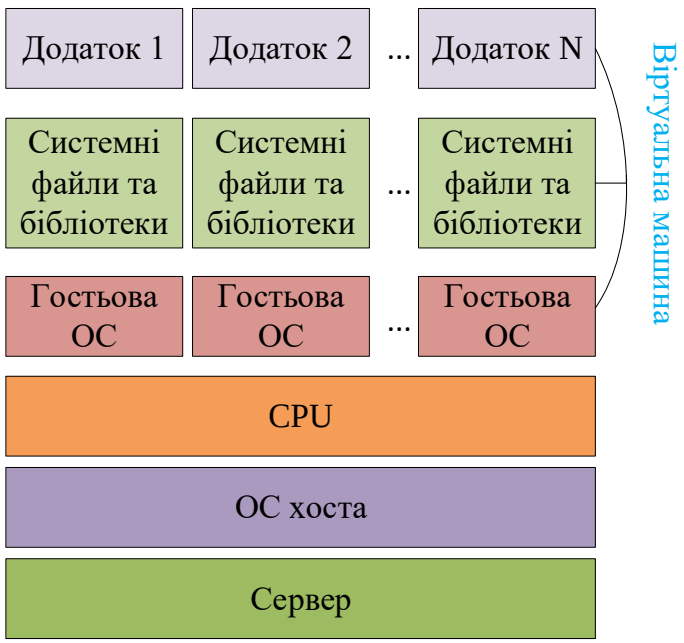

Рисунок 1.3 – Віртуалізація на основі віртуальних машин

Віртуальна машина не може безпосередньо взаємодіяти з фізичною інфраструктурою, натомість вона використовує спеціалізоване програмне забезпечення, зване гіпервізором. Гіпервізор розподіляє фізичні обчислювальні ресурси кожної віртуальної машини і поділяє їх, щоб віртуальні машини не заважали одна одній. Використання цієї концепції дозволяє ефективно масштабувати програми, забезпечуючи гнучке надання ресурсів віртуальних машин у відповідь на їх змінне навантаження, підвищуючи таким чином ефективність використання при менших фінансових витратах.

1.4.2 Віртуалізація на основі контейнерів

Віртуалізація на основі контейнерів є інкапсуляцією. програмного коду та всіх його залежностей, щоб він працював рівномірно та постійно у будь-якій інфраструктурі. Цей тип віртуалізації був запропонований у як альтернатива віртуалізації на основі гіпервізора.

Хоча концепція контейнеризації та ізоляції процесів існує вже кілька десятиліть, поява Docker Engine, LXC Linux Containers, Google Kubernetes Engine (GKE), EC2 Container Service (ECS), як стандартів для віртуалізації включаючи інструменти розробки та універсального підходу до віртуалізації прискорило впровадження віртуалізації на основі контейнерів. Серед усіх платформ віртуалізації на основі контейнерів останнім часом Docker користується великою популярністю та швидко розвивається. Тому, в цій роботі термін «контейнер» ставитиметься до Docker контейнерів.

Docker контейнери – це підхід до віртуалізації, який дозволяє процесів працювати в ізоляції. Як показано на рисунку 1.4, Docker контейнери не включають окрему операційну систему, натомість вони спираються на операційну систему, надану базовою інфраструктурою. У результаті вони запускаються набагато швидше і використовують частину пам'яті. порівняно із завантаженням повної операційної системи. Тому ця технологія

віртуалізації є відповідним способом побудови масштабованих, сервісів, що швидко розгортаються в хмарі.

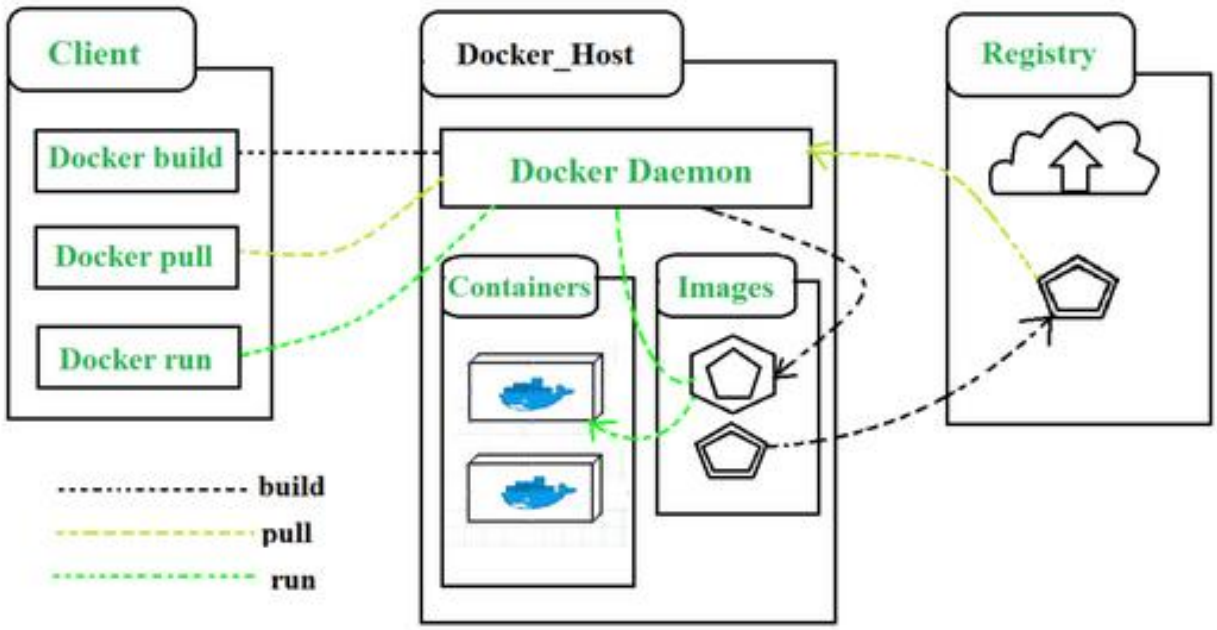

Рисунок 1.4 – Архітектура платформи Docker

Використання Docker контейнерів у процесі виробництва програмного забезпечення може гарантувати швидке та безперервне розгортання програмного забезпечення, переносимість через хмарні інфраструктури, ізоляцію та сегрегацію ресурсів. При запуску контейнера є можливість вказати, як контейнер взаємодіє із хост-системою. Наприклад, мережні порти можуть бути налаштовані з переадресацією між внутрішнім (тобто контейнерним) та зовнішнім (тобто хостовим) номерами портів. Таким чином, різні сервіси в однієї і тієї ж мережі або через Інтернет можуть зв'язуватися з контейнером.

Docker пропонує набір API інтерфейсів, що надаються з боку Docker-движка, що охоплюють весь життєвий цикл контейнерів, а саме створення, запуск, зупинку та управління контейнерами. Крім того, його API інтерфейси дозволяють створювати та спільно використовувати образи Docker

контейнерів через сховища, а також витягувати образи, які були створені та надані іншим розробникам.

1.5 Атрибути якості обчислень

Збої хмарної інфраструктури можуть призвести до руйнівних наслідків, таких як: безліч незадоволених користувачів та порушення безперервності бізнес-процесів. Більше того, простої хмарних інфраструктур можуть призвести до зниження продуктивності, втрати доходів та погіршення репутації користувача. Тому, щоб уникнути цього та розробити надійну середу хмарних обчислень, необхідно враховувати ряд нефункціональних вимог на етапі проектування. Нефункціональні вимоги відносяться до якостям, які повинна мати система та обмеженням, за яких вона має працювати. Тим не менш, не існує універсального набору нефункціональних вимог, оскільки вони різняться залежно від мети програмного компонента, наприклад: перегляд веб-сторінок, мережеве спілкування, аудіо трансляція, рендеринг відео.

Більше того, автори дійшли висновку, що нефункціональні вимоги, такі як місце розташування, доступність, час відгуки інфраструктури та завантаження процесора, є важливими атрибутами якості для додатків, які є частиною сценаріїв обробки великих даних. В іншому дослідженні автори запропонували розглядати продуктивність мережі, енергоефективність інфраструктури та вартість як важливі атрибути для досягнення високої якості обслуговування у своїх сценарії IoT.

У ході цієї роботи було розглянуто безліч нефункціональних атрибутів, які можна використовувати для автоматичного методу прийняття рішень та які пов'язані з різними контейнерними програмними компонентами. Рішення, розроблене в цій роботі, призначене для роботи з потенційно великою кількістю дисфункційних вимог. Хоча пропоноване рішення може використовуватися для підтримки будь-якого кількісного атрибуту, який

може становити інтерес для розробника програмного забезпечення, на основі огляду NFR (Non-Functional Requirements) атрибутів.

1.6 Вибір оптимальної хмарної інфраструктури

У багатьох дослідженнях, присвячених балансуванню навантаження, управління та розподілу ресурсів, забезпечення ресурсів, системам розгортання та управління послугами, вивчався вибір оптимального IaaS з погляду очікуваної якості обслуговування. Через велику кількість різних нефункціональних вимог, які можна використовувати, у розглянутих нижче дослідженнях пропонуються багатокритеріальні підходи для різних сценаріїв розгортання додатків у хмарних інфраструктурах. Автори [15], [16], [17] описують детерміновані підходи, засновані методі аналізу ієрархій (MAI).

MAI – це метод, який широко використовується на практиці та активно розвивається вченими всього світу, він був найчастіше використовуваним методом прийняття багатокритеріальних рішень у періоді 2000-2014 р. Цей метод не враховує QoS вимог, обрані користувачем, але натомість він виконує попарне порівняння інфраструктур, якими ранжуються інфраструктури. Тим не менш, MAI може бути обчислювально непрактичним у тих випадках, коли використовується для порівняння великої кількості параметрів, оскільки він генеруватиме велика кількість вбудованих рішень. Наприклад, якщо MAI використовується для ранжування 500 доступних хмарних інфраструктур шляхом порівняння 20 нефункціональних атрибутів, метод згенерує дерево рішень з 10000 вузлів.

Автори [19] розробили фреймворк для надання оптимального вибору хмарних інфраструктур, який також ранжує послуги відповідно з QoS вимог. Використовуючи переваги минулого досвіду використання інших користувачів, платформа ідентифікує та поєднує переваги між парами сервісів, щоб зробити ранжування сервісів. Платформа використовує два алгоритми прогнозування та ранжування для обчислення ранжування послуг

на основі переваг розробника хмарних програм. Однак для ранжування хмарних інфраструктур у цій платформі використовується лише минуле використання даних на основі двох атрибутів мережевого рівня (пропускна спроможність та час відгуку).

Іншим часто використовуваним методом, який використовується для отримання оптимальної хмарної інфраструктури, є метод Парето, який використовується для пошуку оптимальної інфраструктури шляхом виконання компромісів між двома чи більше конфліктуючими цілями. Автори [20] представили підхід до розподілу ресурсів контейнера, який заснований на Парето-оптимізації шляхом реалізації недетермінованого генетичного алгоритм сортування (NSGA-II). Крім того, в іншому дослідженні [21] також використовується оптимізація Парето для компромісу нефункціональних вимог на ранніх етапах процесу розробки програмного забезпечення і, таким чином, розгортання програми на оптимальній хмарної інфраструктури.

Однак цей метод має обмеження, що дозволяє включати або виключати нефункціональні атрибути і приймати рішення щодо не більш ніж трьох атрибутів. На відміну від розглянутих вище досліджень, у яких хмарні обчислення розглядаються як детермінований процес, існують дослідження, у яких природа хмарних обчислень розглядається як стохастична. Зокрема, вони досліджують явища невизначеності надійності інфраструктури, її доступності, надання ресурсів, надання послуг і т.і.

Стохастичні підходи дозволяють отримати оцінку, визначальну ймовірність досягнення високої якості обслуговування, яка є важливою для критично важливих бізнес-додатків.

Численні дослідження різних областей використовують Марківські процеси прийняття рішень (MDP) для надання результатів прийняття рішень у випадках, коли можуть виникнути непередбачені ситуації. Наприклад, MDP використовується для полегшення оптимального лікування пацієнтів.

Крім того, MDP застосовується в галузі штучного інтелекту та

навчання з підкріпленням. У цьому контексті MDP також підходить для застосування в області хмарних обчислень через його стохастичну природу.

Тим не менш, наскільки нам відомо, використання MDP для забезпечення високої якості обслуговування при розгортанні програмного компонента у контексті контейнерів вже розглядалося. MDP також дозволяє формально перевірити правильність розміщення рішення про розгортання. Ллерена та співавт.

Крім інтересу, який академічне суспільство виявляє до підходів до оркестрування додатків у хмарі, існує також безліч комерційних рішень, як наприклад: Apache Mesos, Docker Swarm та Kubernetes. Найбільш широко використовуються комерційним рішенням для оркестрування контейнерів, що охоплює понад 50% галузі, є Kubernetes.

1.7 Kubernetes

Kubernetes – це система оркестрування контейнерів із відкритим вихідним кодом для автоматичного розгортання завдань та управління ресурсами кластера з високою гнучкістю та масштабованістю, заснована на контейнерноорієнтованій системі управління кластерами Borg.

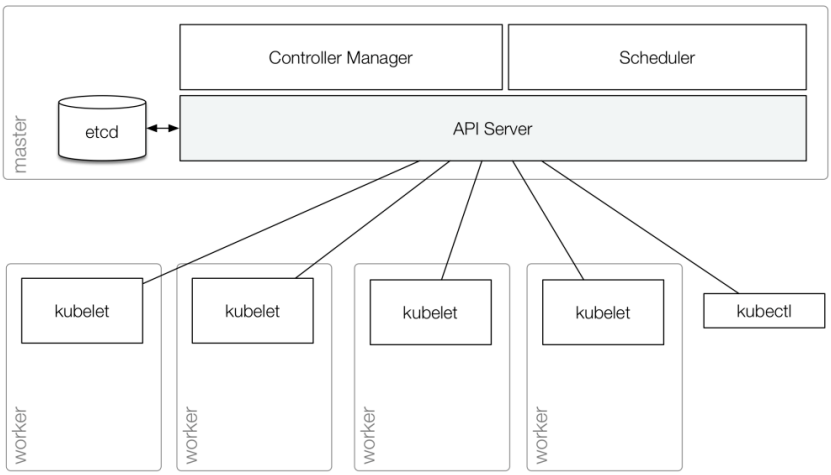

Рисунок 1.5 – Використання Kubernetes

Основним компонентом Kubernetes є кластер (hисунок 1.5), який складається з безлічі віртуальних або фізичних машин, кожна з яких може служити або як головний вузол, або як вузол. Програмні контейнери Kubernetes є важливими інструментами для побудови, розгортання, запуску та управління сучасними корпоративними застосунками в масштабі підприємства і швидкої та надійної доставки корпоративного програмного забезпечення кінцевому користувачеві при більш ефективному використанні ресурсів і зниження витрат.

Основними компонентами Kubernetes є Kubernetes master, який містить сервер API, планувальник і диспетчер контролерів.

Вузли містять групи з одного або кількох контейнерів, а головний вузол зв'язується з вузлами у разі необхідності створення чи знищення контейнер. Головний вузол є точкою доступу, яка використовується для управління плануванням та розгортанням контейнерів. Крім того, головний вузол дозволяє кожному вузлу отримувати доступ до даних стану та конфігурації. для всього кластера, які зберігаються в ectd.

Головний вузол зв'язується з рештою компонентів кластера та іншими сторонніми компонентами через kube-apiserver. Кожен вузол включає: агента, званий kubelet, який відповідає за керування станом вузла: запуск, зупинка та обслуговування контейнерів додатків на основі вимог щодо якості обслуговування; мережевий проксі kubeproxy, який працює на вузлах у кластері та відповідає за маршрутизацію вхідного трафіку на конкретні контейнери на одному вузлі.

У Kubernetes група з однієї або невеликої кількості контейнерів, які тісно пов'язані разом із загальною IP-адресою та простором порту, може бути визначена як pod. Отже, Kubernetes pod вказує на один єдиний екземпляр програми. Kubernetes розгортає програми лише на вузлах, які мають достатні обчислювальні ресурси. Іншими словами, Kubernetes віддає перевагу вузлу з найнижчим використанням обчислювальних ресурсів (ЦП та пам'ять).

У процесі прийняття рішення Kubernetes спочатку фільтрує вузли, які задовольняють жорстким обмеженням (тобто предикатам), а потім він ранжує інші вузли за допомогою певних методів пріоритету, які приймають як вхідні м'які обмеження (тобто пріоритетів). Тим не менш, Kubernetes має встановлені визначення жорстких та м'яких обмежень, які враховуються у процесі ухвалення рішень. Наприклад, в даний час він не використовується і не дозволяє розширювати список вимог під час виконання за допомогою показників мережного рівня (наприклад, пропускна здатність, затримка, втрата пакетів) або інших нефункціональних вимог (наприклад, вартість, конфіденційність і т.і.).

2 МЕТОДИКА ВИБОРУ ОПТИМАЛЬНОЇ ХМАРНОЇ ІНФРАСТРУКТУРИ

В основі роботи лежить гіпотеза про ефективність використання Марківського методу прийняття рішень (MDP) як механізму прийняття рішень щодо розгортання мікросервісів на оптимальній хмарній інфраструктурі, враховуючи конкретні вимоги, контекст використання, актуальні інфраструктурні та мережеві вимірювання та метадані. Тому в цій роботі застосовується стохастичний метод прийняття рішень, що спирається на MDP процес прийняття рішень. Це метод прийняття рішень для динамічних середовищ дозволяє об'єднати кілька випадкових поведінок системи у одній моделі. Ця характеристика дозволяє в нашому випадку методу виконувати кілька симуляцій з кожного стану та спрямована на отримання оптимальної хмарної інфраструктури для розгортання у моделі.

У цій роботі MDP підходить для використання, оскільки метод здатний: автономно вибирати одну з низки можливих дій; реалізовувати функцію корисності, яка генерує оцінку рангу для кожної інфраструктури; перевіряти поведінку моделі у певний момент у майбутньому, забезпечуючи цим надійне прийняття рішень.

MDP – це імовірнісна структура M = (S, A, P, R, γ), де:

$S=\{S_0, \ldots, S_n\}$ – кінцева безліч станів, що в даній роботі являє собою інфраструктури розгортання програмних компонентів у «Edge-to-Cloud» континуумі;

$A=\{a_0, \ldots, a_n\}$ – кінцеве безліч дій, які у цій роботі показані як розгортання програми в інфраструктурі з більш високим/низьким QoS;

$P=\{s_{t+1}=s'|s_t=s, a_t=a\}$ – можливість переходу зі стану s на етапі t стан s′ внаслідок дії a;

R(s,s') – винагорода, яка отримується після переходу в стан S' з стану S внаслідок дії a;

γ – коефіцієнт дисконтування, який має значення в інтервалі [0,1].

Модель MDP, розроблена в рамках цієї роботи, слідує стандартній структурі наведеного вище визначення.

2.1 Імовірнісна модель та Марківські процеси

Імовірнісна модель – це кінцевий автомат (рисунок 2.1), який є необхідним компонентом, який використовується для отримання результатів ранжування інфраструктури. Імовірнісні моделі будуються для кожного програмного компонента окремо і можуть динамічно змінюватися через мінливості вхідних NFR атрибутів інфраструктур. Переходи в моделі є ймовірним вибором для кількох наступних станів. Кожен стан імовірнісної моделі представляє різну інфраструктуру розгортання. Переходи між станами відбуваються через різні дії в моделі, які дають стохастичну поведінку моделі.

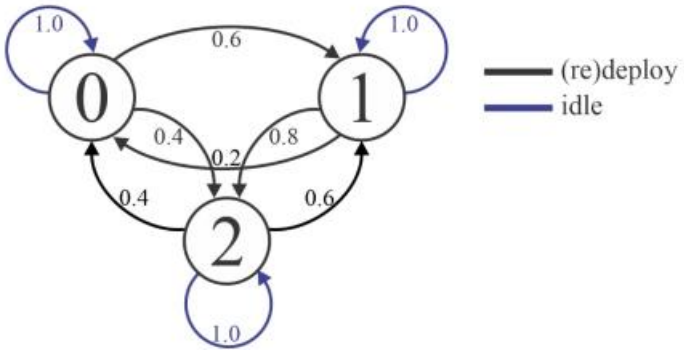

Рисунок 2.1 – Приклад імовірнісної моделі

Поточна модель реалізує дві дії: дія (re)deploy, яка відповідає за вибір інфраструктури розгортання, та дія idle, яка активується, якщо поточна інфраструктура розгортання пропонує оптимальна якість обслуговування, запитана розробником програмного забезпечення. У імовірнісній моделі кожна дія між двома станами має відповідний перехід. Наприклад, стан матиме дійсні переходи у всі стани, які представляють інфраструктури з

більш високою чи нижчою якістю обслуговування, через дію (re)deployment. Крім того, той самий стан також матиме перехід до самого себе через дії idle. Щоразу, коли відбувається перехід з одного стану в інший, призначається винагорода, оскільки кожен стан моделі пов'язаний з значенням винагороди. Значення винагороди застосовуються у функціях, які реалізують поточні виміри від моніторингу, що передує використання та порогові значення, встановлені програмним інженером, щоб оцінити корисність станів.

2.2 Розрахунок імовірностей моделі винагороди

При розробці ймовірнісної моделі важливим завданням є обчислення ймовірностей для кожного переходу та винагород за досягнення кожного стану. Крім того, ймовірності переходу повинні задовольняти правила Маркова, що свідчить, що для будь-якого заданого часу умовне розподіл майбутніх станів процесу з урахуванням нинішніх та минулих станів залежить лише від поточного стану. Математична властивість Маркова можна виразити так:

$$\mathrm{P}[\mathrm{S}_{t+1} \mid S_t\ ] = \mathrm{P}[\mathrm{S}_{t+1} \mid S_1, \dots, S_t\ ] . \qquad (2.1)$$

Отже, імовірності, які передбачають майбутню поведінку моделі залежать тільки від поточного стану моделі. Іншими словами, при розрахунку ймовірності переходу між інфраструктурами розгортання S_t та S_{t+1} ймовірність переходу не залежить від значень ймовірності, які були необхідні досягнення інфраструктури розгортання S_t. Перехідні ймовірності зображуються у вигляді шарів у ймовірнісному дереві рішень. Перший рівень показує, що ймовірність досягнення будь-якої доступної інфраструктури, коли немає даних моніторингу або знань про попереднє використання, однакові. Враховуючи, що N_{tran} є кількістю переходів однієї дії з кожного

стану до інших станів, а n є кількістю станів в автоматі, матриця матиме такий вигляд:

$$P_1 = (P1_{ij}),$$
$$P1_{ij} = \frac{1}{N_{tran}}, i = \overline{1,n}, j = \overline{1,n} \quad . \tag{2.2}$$

Другий рівень є ймовірністю переходу між станами в автоматі, які розраховуються за наявності знань про використання перерозподіл мікросервісів між інфраструктурами. Враховуючи N_{chosen} – кількість виборів інфраструктури як мета перерозподілу мікросервісу з іншої інфраструктури, а N_{listed} – кількість входжень інфраструктури в клас еквівалентності, рівняння матиме наступний вигляд:

$$P_2 = (P2_{ij}),$$
$$P2_{ij} = \frac{N_{chosen}}{N_{listed}}, i = \overline{1,n}, j = \overline{1,n} \quad . \tag{2.3}$$

Загалом, якщо автомат має знання про попереднє використання, матрицю ймовірностей остаточного переходу можна розрахувати як добуток матриць P_1 і P_2:

$$P = P_1 \cdot P_2 \, . \tag{2.4}$$

Значення винагороди для кожного стану залежить від даних, отриманих в результаті вимірювань моніторингу та історичних даних попередніх рішень. Чим більше м'яких обмежень NSC порушуються конкретним станом (тобто інфраструктурою), тим нижче буде винагороду за досягнення цього стану. Враховуючи counti – кількість порушень, вектор винагород можна розрахувати за допомогою наступного рівняння:

$$R(S_i) = \begin{cases} \frac{1}{count_i}, & count_i > 0 \\ 1, & count_i = 0 \\ 0, & count_i = n_{sc} \end{cases}. \quad (2.5)$$

Однак для більш точного розподілу винагород розроблена в цій роботі система слідувала наступному алгоритму (лістинг 2.1).

Лістинг 2.1 – Алгоритм розподілення винагород

```
1: count ← 1
2: for each i in size do
if NT<NTT and NL>NLT and PL>PLT and CPU>CPUT and MEM<MEMT and
CPUU>CPUUT and MEMU>MEMUT and C>CT and DT<DTT and DL>DLT and
QoE<QoET then
rewardsi ← 0
else if (NT*4)<NTT or NL>(NLT*4) or (DT*2)>DTT or DL>(DLT*2) or
C>(CT*4) then
rewardsi ← 0
else
if NT<NTT then count ← count + 1
if NL>NLT then count ← count + 1
if PL>PLT then count ← count + 1
if CPU>CPUT then count ← count + 1
if MEM<MEMT then count ← count + 1
if CPUU>CPUUT then count ← count + 1
if MEMU>MEMUT then count ← count + 1
if DT<DTT then count ← count + 1
if DL>DLT then count ← count + 1
if C>CT then count ← count + 1
if QoE<QoET then count ← count + 1
rewardsi ← 1/(count)
end if
count ← 1
```

Вхідні параметри: пропускна здатність (NT), кругова затримка (NL), втрата пакетів (PL), CPU ядра (CPU), пам'ять (MEM), CPU завантаження (CPUU), завантаження пам'яті (MEMU), вартість (C), пропускна здатність бази даних (DT), затримка через базу даних (DL), якість досвіду користувача (QoE), пороги пропускної здатності (NTT), пороги пропускної спроможності (NLT), пороги втрати пакетів (PLT), пороги кількості CPU ядер (CPUT),

порог пам'яті (MEMT), порог завантаження CPU (CPUUT), поріг завантаження пам'яті (MEMUT), пороги вартості (CT), пороги пропускної спроможності бази даних (DTT), пороги затримки через базу даних (DLT), пороги якості досвіду користувача (QoET). Вихідні параметри: масив винагород (rewards).

Щоб розрахувати вектор рейтингових балів та визначити оптимальну інфраструктуру, необхідно оцінити корисність кожного стану використовуючи рівняння Беллмана:

$$U_i(S) = R(S) + \gamma max_{a \in A(S)} \sum_{S\prime} P(S'|S,a)U(S') , \quad (2.6)$$

де S є поточним станом, а S' є наступним станом. В цьому у разі кожного стану моделі існує окреме рівняння Беллмана, тому механізм прийняття рішень має вирішувати систему нелінійних рівнянь. Ця нелінійна система може бути вирішена за допомогою ітеративного підходу. В результаті, процес прийняття рішення є ітераційну процедуру, яка обчислює очікувану корисність кожного стану, використовуючи корисність сусідніх станів, доки корисності, розраховані на двох послідовних кроках, не стануть достатньо близькими:

$$|U(S_i) - U'(S_i)| < \theta , \quad (2.7)$$

де θ є зумовленим граничним значенням. Чим менше граничне значення, тим вища точність алгоритму. Алгоритм ітерації за значеннями, який використовувався у цій роботі, представлений нижче.

Вхідні параметри: вектор станів (S), матриця ймовірностей переходів (P), вектор винагород (R), вектор дій (A), коефіцієнт дисконтування (γ), максимальна зміна корисності будь-якого стану в ітерації (θ).

Вихідні параметри: матриця корисностей станів (U).

Лістинг 2.2 – Алгоритм ітерації по значенням

```
1: Repeat
2: U ← U′; δ ← 0
3: for each s in S do
4: U′(s) ← R(s) + γ*max!∈#(%) ∑%' P(s'|s, a)U(S')
5: if |U′(s) − U(s)| > δ then δ ← |U′(s) − U(s)|
6: end if
7: until δ < θ
8: return U
```

2.3 Методика скорочення обчислювальної складності

Щоб формально визначити класи еквівалентності, необхідно спочатку визначити дві множини: безліч нефункціональних атрибутів і безліч хмарних інфраструктур.

Безліч нефункціональних атрибутів є безліч підмножин атрибутів:

$$NFR = \{\{nr_{00}, nr_{01},..., nr_{0n}\},...,\{nr_{m0}, nr_{m1},..., nr_{mn}\}\}, \quad (2.8)$$

де nr представляє конкретний дисфункційний атрибут, наприклад,

$$NFR = \{$$
$$\{vCPU=1, location=Europe, latency=30\},$$
$$\{vCPU=1, location=Europe, latency=45\},$$
$$\{vCPU=1, location=USA, latency=200\}$$
$$\}.$$

Безліч хмарних інфраструктур визначається як:

$$I = \{inf_0, inf_1,..., inf_m\},$$

Співвідношення між елементами набору двох множин можна визначити, як бієктивну функцію:

$$f: NFR \rightarrow I, \text{ где: } inf_m = f(nr_{m0}, nr_{m1}, \ldots, nr_{mn}).$$

Іншими словами, кожне підмножина безлічі NFR відображається точно на одну хмарну інфраструктуру з множини I. Це означає, що кожна інфраструктура пов'язана з унікальним набором нефункціональних атрибутів. Наприклад, множини, представлені вище можуть бути пов'язані з інфраструктурами наступним чином:

$$f(vCPU=1, location=Europe, latency=30)=g1\text{-}small\text{-}EU,$$
$$f(vCPU=1, location=Europe, latency=45)=n1\text{-}standard1\text{-}EU,$$
$$f(vCPU=1, location=USA, latency=200)=n1\text{-}standard1\text{-}USA.$$

Наступним кроком є створення класу еквівалентності. Нехай e буде бажаною інфраструктурою розгортання, яка задовольняє всім жорстким обмеженням та еєI. Такий клас еквівалентності визначається як:

$$[e] = \{inf \in I \mid inf \sim e\}.$$

Це означає, що в рамках класу еквівалентності дві інфраструктури e та inf будуть еквівалентними, якщо кожному атрибуту інфраструктури e відповідає відповідний атрибут з інфраструктури inf з рівним значенням. Наприклад, клас еквівалентності, представлених вище, який базується на бажаній інфраструктурі $e = f(location = Europe)$, буде наступним: $[e] = \{g1\text{-}small\text{-}EU, n1\text{-}standard1\text{-}EU\}$.

Поряд із використанням категоріальних значень для формування класів еквівалентності (таких як використання місцеположенні), метод також дозволяє використання порогових значень, наприклад, класифікації всіх інфраструктур в одному класі, у яких завантаження процесора не перевищує 80%.

3 РЕАЛІЗАЦІЯ ФОРМАЛІЗОВАНОГО МЕТОДУ

Нерідко на на практиці після створення програмного забезпечення розробники повинні вручну вибрати інфраструктуру, в якій буде розміщено розроблене програмне забезпечення. Тому часто ці інфраструктури не є оптимальними хостами для програмного забезпечення, що призводить до зниження якості обслуговування. Щоб вирішити цю проблему, у цій роботі пропонується концепція процесу автоматичного ранжування всіх доступних обчислювальних інфраструктур з урахуванням нефункціональних вимог та унікального контексту використання мікросервісів.

3.1 Концепція процесу автоматичного розгортання інфраструктур

Концепція складається з п'яти загальних кроків (рисунок 3.1), де вхідними параметрами є всі доступні інфраструктури, дані моніторингу інфраструктури та вимоги до якості, задані розробниками програмного забезпечення, а вихідними даними є список ранжованих інфраструктур, де перша інфраструктура вважається оптимальною до роботи програмного забезпечення. Далі ми приступимо до пояснення окремих кроків.

Крок 1: На цьому етапі програмний інженер розробляє хмарне додаток з контейнерних мікросервісів, які необхідно розгорнути в хмарі. Він вибирає всі важливі нефункціональні вимоги (наприклад, конкретне розташування, вартість і т.д.). Крім того, програмному інженеру дозволено визначити порогові значення для будь-яких метрик якості (наприклад, пропускна здатність, кругова затримка, продуктивність та і т.д.). Тим не менш, цей вибір може значно відрізнятися між різними типами мікросервісів.

Іншими словами, інженер повинен також вирішити, які вимоги становлять жорсткі обмеження та повинні постійно задовольнятися під час виконання, а які вимоги є бажаними, але з обов'язковими (м'які обмеження).

Як тільки жорсткі та м'які обмеження визначені, вони використовуються на двох заключних етапах автоматизованого процесу прийняття рішень Жорсткі обмеження51 використовуються як вхідні параметри для класифікації еквівалентності (другий крок), а м'які обмеження використовуються на етапі генерації та перевірки імовірнісної моделі (третій крок).

Рисунок 3.1 – Концепція ухвалення рішень для розгортання мікросервісів

Крок 2: На цьому кроці зменшується кількість необхідних обчислень при визначення оптимальної хмарної інфраструктури Тому для подальших обчислень розглядаються лише інфраструктури, які задовольняють жорстким обмеженням розробника програмного забезпечення. На початку цього етапу генерується кінцевий автомат із використанням списку всіх доступних інфраструктур для розгортання хмарної програми (наприклад, 1000 доступних хмарних інфраструктур), де кожен стан цієї вихідної Модель

представляє один варіант розгортання. Потім слідує автоматизований процес, в якому інфраструктури класифікуються за класи еквівалентності. Наприклад, клас еквівалентності може містити все інфраструктури, які містять щонайменше 8 процесорів, або всі інфраструктури, для яких завантаження центрального процесора складає менше 80%.

Крок 3: Метою цього кроку є створення імовірнісного автомата, який являє собою ймовірнісну модель випадково-змінних систем. Для побудови ймовірнісної моделі використовуються різні метрики якості обслуговування, які становлять минуле та сучасне стан продуктивність інфраструктур. Вони збираються і зберігаються в базі даних з використання багаторівневої системи моніторингу. В процесі використовуються лише ті нефункціональні вимоги, які є частиною м'яких обмежень. Потім метод обчислює рейтингові оцінки для всіх інфраструктур і ранжує їх.

Крок 4: Після генерації ймовірнісної моделі отриманий результат перевіряється з використанням методу перевірки моделі (model checking). Цей крок, використовуючи формальні критерії, перевіряє, наскільки нефункціональні вимоги задовольняються доступними інфраструктурами у кожному класі еквівалентності. Розраховане значення перевірки моделі є виходом імовірнісної моделі та надає формальну гарантію.

Крок 5: Інфраструктура першого рангу автоматично вибирається та мікросервіс розгортається за допомогою інструмента оркестрування, такого як Kubernetes. Цей етап виконується у припущенні, що отримана формальна гарантія для інфраструктури першого рангу, тобто тієї, що має найвищу оцінку, прийнятна для розробника програмного забезпечення.

3.2 Архітектура програмного комплексу

IoT середовища використовують величезну кількість інтелектуальних додатків практично у всіх областях. Через різні формати та розмір неструктурованих даних, що генеруються в IoT середовищах, традиційні

хмарні інфраструктури не можуть досягти бажаної якості обслуговування (QoS) і можуть призвести до недоступності даних, проблем з дотриманням нормативних вимог та збільшення витрат на зберігання або обчислення. Вирішенням цих проблем є використання датацентричної архітектури, де розробляються системи, орієнтовані дані.

Філософія датацентричних систем наступна: у міру збільшення розміру даних вартість переміщення даних стає непомірно високою, а продуктивність погіршується. Тому необхідно переносити обчислення до даних, а чи не навпаки. Цей розділ описує багаторівневу датацентричну архітектуру. програмного комплексу та механізм для розгортання мікросервісів на оптимальної хмарної інфраструктури.

Системна архітектура розроблена у цій роботі, відповідає стандартам сумісності, встановленим організаціями, Cloud Native Computing Foundation (CNCF), Edge Computing Consortium Europe (ECCE)53 та OpenFog Consortium. Розроблений програмний комплекс складається з графічного інтерфейсу, модулів прийняття рішень та «Edge-to-Cloud» обчислень для розгортання контейнерних мікросервісів.

Графічний інтерфейс користувача – це вхідна точка для розробника програмного забезпечення, що реалізована з використанням технології EmberJS, що включає кілька уявлень, таких як подання створення компонента та подання композиції програми.

У графічному користувальницькому інтерфейсі програмний інженер розробляє приложение з контейнерних мікросервісів, керує вимогами до якості та вхідними параметрами для процесу розгортання та ініціює процес розгортання. Модуль ухвалення рішення відповідає за визначення оптимальної інфраструктури для розгортання контейнерних мікросервісів на основі Марковського процесу ухвалення рішення.

У цьому модулі також відбувається верифікація рішення про розгортання та аналіз можливих сценаріїв перерозподілу мікросервісів з однієї інфраструктури в іншу в деякий час у майбутньому. Крім того, модуль

також включає в себе оркестратор, який запускається для розгортання програми після отримання оптимальної інфраструктури та верифікації рішення. Модуль обчислення «Edge-to-Cloud» відбудеться з інфраструктур, розгортання контейнерних мікросервісів та даних, компонентів моніторингу та IoT пристроїв. Інфраструктури в цьому модулі використовуються для зберігання та обробки даних у обчислювальному континуумі. Залежно від мети та вимог програмних компонентів, що розгортаються, запропонована архітектура дозволяє розгортати контейнери програмних компонентів на периферії, інфраструктурах у тумані та в інфраструктури в хмарі.

3.3 Механізм прийняття рішень про розгортку

Модуль ухвалення рішень розбивається на дві групи компонентів: система Марковського випадкового процесу прийняття рішень та верифікації та система моніторингу (рисунок 1.2).

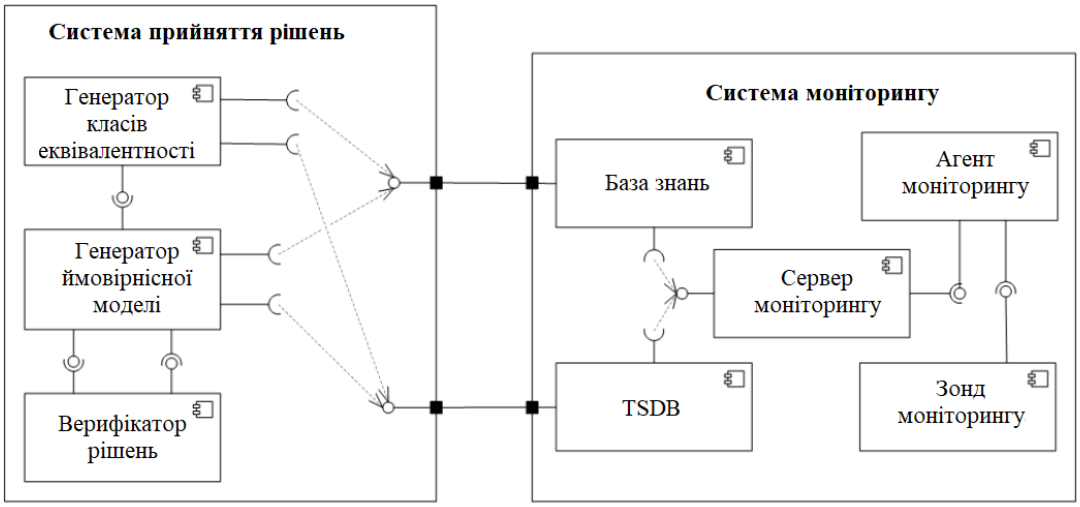

Рисунок 3.2 – Діаграма компонентів

Модуль ухвалення рішень розбивається на дві групи компонентів: система Марківського випадкового процесу прийняття рішень та верифікації та система моніторингу. Перший компонент – Генератор Класів

Еквівалентності (ГКЕ), ініціює роботу механізму. Цей компонент розподіляє всі доступні інфраструктури та створює початкову модель, яка використовується як вхідний параметр до створення класів еквівалентності. Потім він перевіряє, які інфраструктури задовольняють жорстким обмеженням, та призначає їх елементам класу еквівалентності.

Отриманий клас еквівалентності спрямовується в компонент Генератор Імовірнісної Моделі (ГІМ). ГІМ генерує кінцевий ймовірнісний автомат, розмір якого дорівнює розміру набору, який був надано з ГКЕ. ГІМ розраховує значення ймовірностей переходів, нагороди станів та корисності станів. Обидва компоненти, ГКЕ і ГІМ, розроблені з використанням технологій розробки на основі Java, таких як Java Jersey для RESTful веб-сервісів та Apache Maven для управління програмного забезпечення.

На основі імовірнісної логіки часу, що гілкується (PCTL), Верифікатор Рішень (ВР) перевіряє модель та вихідні параметри компоненти ГВМ. Таким чином, ВР перевіряє ступінь, в якому оптимальна інфраструктура буде задовольняти нефункціональним вимогам мікросервісу відповідно до конкретними обмеженнями, які були встановлені програмним інженером. Отже, цей компонент забезпечує інженеру впевненість у тому, що система виділяє оптимальну інфраструктуру для роботи мікросервісу.

Цей компонент використовує засіб перевірки моделі PRISM, яке застосовується для аналізу імовірнісних моделей. Для перевірки моделей, PRISM імпортує моделі, які перевіряють конфігурації з конфігураційних скриптів. Як тільки інфраструктура розгортання визначена та перевірена, механізм прийняття рішення про розгортання генерує YAML скрипт з інструкціями з розгортання системи оркестрування. Інструкції YAML містять інформацію про інфраструктуру розгортання, розгорнутих додатках та політиках резервного копіювання та реплікації. Правильне функціонування ймовірнісної обчислювальної логіки сильно залежить від системи моніторингу. Система моніторингу є набір компонентів, таких як зонди для

метрик різних атрибутів, агентів та сервери, які динамічно збирають метрики під час роботи програми.

Вона відіграє важливу роль у запропонованій системі, тому що використовується для вимірювання метрик (наприклад, пропускну здатність, затримки, завантаження процесора та пам'яті) і це гарантує, що будь-які мікросервіси задовольняють необхідним вимогам щодо якості обслуговування під час виконання. У як системи моніторингу використовуються Jcatascopia, Netdata та Prometheus.

3.3 Система моніторингу

Система моніторингу відбудеться з трьох компонентів: агенти моніторингу, зонди моніторингу та сервер моніторингу. Агенти моніторингу (Monitoring Agent) є легкими компонентами, які управляють збором метрик з віртуальних екземплярів машин та контейнерів. Зонди моніторингу (Monitoring Probes) є колекторами метрик, керовані агентами моніторингу. Вони створені для збору низькорівневих та високорівневих метрик.

Зонди моніторингу повністю контролюють поширення метрики. Вони пересилають метрики відповідному агенту моніторингу або періодично, або при наступі певної події. Сервер моніторингу (Monitoring Server) використовується для збору відстежуваних даних від агента моніторингу та передачі в базу даних.

Сервер моніторингу має бути встановлений на хості, який задовольняє вимог до обладнання Apache Cassandra. Іншими словами, хост повинен надавати достатньо пам'яті, ресурсів процесора та диска. База даних тимчасових рядів – це база даних, яка використовується для зберігання QoS метрик. В експериментах цієї роботи використовується база даних Apache Cassandra, яка є програмним забезпеченням з відкритим вихідним кодом. База знань використовується для збору комплексної інформації, яка потрібно ГКЕ і ГВМ як вхідні параметри.

База знань Apache Jena Fuseki збирає інформацію про обрану інфраструктуру, таку як: місце розташування інфраструктури, інформація про тип програми, оцінка якості досвіду та інформація про розгортання у вигляді RDF семантичних трійок. Використання такої бази знань дозволяє аналізувати довгострокові тренди або проводити різноманітні стратегічні аналізи, такі як, наприклад, тенденції використання.

3.4 Автоматичне розгортання застосунків

Процес автоматичного розгортання програм починається після того, як інфраструктура розгортання визначено та перевірено. У цій роботі для автоматичного розгортання додатків реалізується механізм оркестрування ресурсів Kubernetes. Щоб автоматично регулювати стан кластера, Kubernetes реалізує програмні компоненти, які називають контролерами. Таким чином, контролери Kubernetes є контурами управління, які постійно відстежують стан кластера та намагаються наблизити поточний стан до бажаного стану. Стан кластера є об'єктом Kubernetes. Зокрема, об'єкт описує: додатки, що працюють у кластері; вузли, на які налаштовані додатки; інформацію про доступність ресурсів та політику поведінки додатків (наприклад, політики перезапуску, оновлення, відмовостійкості, реплікації).

У Kubernetes, об'єкти є постійними компонентами, до яким можна звертатися (тобто створювати, видаляти, змінювати) через сервер API Kubernetes. При створенні об'єкта Kubernetes до специфікації потрібна деяка базова інформація, наприклад, ім'я об'єкта. Зокрема, коли Kubernetes об'єкт створюється за допомогою Kubernetes API, запит повинен включати все обов'язкові поля із тіла запиту. Хоча kubectl дозволяє отримувати запити конфігурації YAML, вони автоматично перетворюються на JSON під час виконання запитів API.

Однак дозволу конфігурації за замовчуванням немає обмежень на те, що користувачі можуть вимагати у API Kubernetes. У в результаті користувач

може запросити необмежену кількість реплік, будь-який Docker образ, довільні ресурси ЦП та пам'яті, що може призвести до неконтрольованого обчислювального середовища і викликати нестабільність, так як59 модулі, швидше за все, конкуруватимуть за ресурси.

У Kubernetes основним об'єктом є Pod, який є групу з одного чи кількох контейнерів. Однак якщо програма розгортається з використанням об'єктів Pod, для забезпечення кількості запущених модулів та їх стану працездатності необхідно буде зробити додаткові кроки. Наприклад, Kubernetes пропонує використовувати більш складні об'єкти, такі як ReplicaSets і Deployments, керують життєвим циклом модулів. Крім того, існують об'єкти для виявлення сервісів, такі як Service та Ingress, та об'єкти для налаштування конфігурації такі як ConfigMap та Secrets.

У рамках роботи, конфігурація розгортання була виражена реалізацією Deployment об'єктів. Deployment є об'єктом, який є додатком, що працює на кластері. Він описує бажане стан програми в кластері, а контролер розгортання змінює фактичний стан.

Лістинг 3.1 – Конфігурація розгортання

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: FileUpload-v1-deployment
labels:
app: fileUpload-v1
spec:
containers:
- name: fileUpload-v1
image: localrepo.org/fileupload/fileupload-v1
ports:
- containerPort: 8082
volumeMounts:
- mountPath: /data/assets
name: fileupload-volume
volumes:
- name: fileupload-volume
hostPath:
path: /mnt/kube-data/imported-volume/app-data
```

```
type: Directory
imagePullSecrets:
- name: regcred
nodeSelector:
id: 753167ef-b0af-41f6-8014-ee43991df4f8
name: a1.xlarge
location: Kharkiv
```

Kubernetes зчитує інструкції з поля spec та запускає1 потрібний додаток на запитаному вузлі.

ВИСНОВКИ

У ході підготовки кваліфікаційної роботи розроблено формалізований метод для автоматизації процесу розгортання застосунків у хмарних інфраструктурах, що дозволяє використовувати нефункціональні вимоги для досягнення високої якості обслуговування. Розроблена концепція ухвалення рішень для розгортання мікросервісів. Отримана ймовірнісна модель, яка використовує метод класифікації еквівалентності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lamia Youseff, Keith Seymour, Haihang You, Jack Dongarra, Rich Wolski: The Impact of Paravirtualized Memory Hierarchy on Linear Algebra Computational Kernels and Software. //HPDC’08, Boston, Massachusetts, USA, June 23-27, 2008.

2. А.В.Богданов, Е.Н.Станкова: Распределенные Linuxкластеры как основы сетей науки и образования будущего. // Труды IX Всероссийской научно-методической конференции Телематика-2002.

3. John L. Hennessy, David A. Patterson: "Computer Architecture: A Quantitative Approach, 4thEdition", Morgan Kaufmann Publishers, ISBN 10: 0-12-370490-1, ISBN 13: 978-0-12-370490-0, 2007.

4. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing onlarge clusters. // OSDI’04: Sixth Symposium on Operating System Design and Implementation, December 2004.

5. А.В.Богданов, В.В.Корхов, В.В.Мареев, Е.Н.Станкова: Архитектуры и топологии многопроцессорных вычислительных систем. «Интернет-Университет ИнформационныхТехнологий»,2004.

6. А.В.Богданов, В.В.Корхов, В.В.Мареев, Е.Н.Станкова; Издательство: Интернет-университет информационных технологий - ИНТУИТ.ру, Архитектуры итопологии многопроцессорных вычислительных систем. (учебник), Серия: Основы информационных технологий , 2004.

7. А.В.Богданов, М.И.Павлова, Е.Н.Станкова, Л.С.Юденич: Высокопроизводительные вычислительные алгоритмы (учебное пособие)// URL: http://www.csa.ru/old/analitik/distant/q_start.html.

8. А.И. Аветисян, О.И. Самоваров, Д.А. Грушин: Архитектура и системное программное обеспечение вычислительных кластерных систем// Институт системного программирования Российской академии наук, Москва, 2005.

9. Антонов А. С: Параллельное программирование с использованием технологии MPI: Учебное пособие. -М: Изд-во МГУ, 2004. с-71.

10. Антонов А. С: Параллельное программирование с использованием технологии OpenMP: Учебное пособие. -М.: Изд-во МГУ, 2009. с-77.

11. Бажанов С.Е., Кутепов В.П., Шестаков Д.А: Разработка и реализация системы функционального параллельного программирования на вычислительных сред.

12. Данилов В.В: Архитектура процессоров SunUltraSparcT1 и T2 (Niagara). // Московский инженерно-физический институт. 2007 г. стр. 8.

13. Деревянко А.С., Солощук М.Н. –Харьков: Технологии и средства консолидации информации. // Учебное пособие. НТУ "ХПИ", 2008.

14. Дмитрий Кузьмин, Федор Казаков, Денис Привалихин, Александр Легалов: На пути к переносимым параллельным программам. // URL: http://citforum.ru/programming/theory/parall_prog/

15. Е.В. Адуцкевич: “Организация обмена данными на параллельных компьютерах с распределенной памятью”, Институт математики НАН Беларуси, Минск (Беларусь), 2005. Lamia Youseff, Keith Seymour, Haihang You, Jack Dongarra, Rich Wolski: The Impact of Paravirtualized Memory Hierarchy on Linear Algebra Computational Kernels and Software. //HPDC’08, Boston, Massachusetts, USA, June 23-27, 2008.

16. А.В.Богданов, Е.Н.Станкова: Распределенные Linuxкластеры как основы сетей науки и образования будущего. // Труды IX Всероссийской научно-методической конференции Телематика-2002.

17. John L. Hennessy, David A. Patterson: "Computer Architecture: A Quantitative Approach, 4thEdition", Morgan Kaufmann Publishers, ISBN 10: 0-12-370490-1, ISBN 13: 978-0-12-370490-0, 2007.

18. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing onlarge clusters. // OSDI’04: Sixth Symposium on Operating System Design and Implementation, December 2004.

19. А.В.Богданов, В.В.Корхов, В.В.Мареев, Е.Н.Станкова:

Архитектуры и топологии многопроцессорных вычислительных систем. «Интернет-Университет ИнформационныхТехнологий»,2004.

20. А.В.Богданов, В.В.Корхов, В.В.Мареев, Е.Н.Станкова; Издательство: Интернет-университет информационных технологий - ИНТУИТ.ру, Архитектуры итопологии многопроцессорных вычислительных систем. (учебник), Серия: Основы информационных технологий , 2004.

21. А.В.Богданов, М.И.Павлова, Е.Н.Станкова, Л.С.Юденич: Высокопроизводительные вычислительные алгоритмы (учебное пособие)// URL: http://www.csa.ru/old/analitik/distant/q_start.html.

22. А.И. Аветисян, О.И. Самоваров, Д.А. Грушин: Архитектура и системное программное обеспечение вычислительных кластерных систем// Институт системного программирования Российской академии наук, Москва, 2005.

23. Антонов А. С: Параллельное программирование с использованием технологии MPI: Учебное пособие. -М: Изд-во МГУ, 2004. с-71.

24. Антонов А. С: Параллельное программирование с использованием технологии OpenMP: Учебное пособие. -М.: Изд-во МГУ, 2009. с-77.

25. Бажанов С.Е., Кутепов В.П., Шестаков Д.А: Разработка и реализация системы функционального параллельного программирования на вычислительных сред.

26. Данилов В.В: Архитектура процессоров SunUltraSparcT1 и T2 (Niagara). // Московский инженерно-физический институт. 2007 г. стр. 8.

27. Дяченко В.О., Коваленко А.А., Стельмахова А.С. Метод використання ресурсів в хмарних системах // Проблеми інформатизації : десята міжнародна науково-технічна конференція. Черкаси – Баку – Бельсько-Бяла – Харків, 2022, т.2, с.95.