Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
Кафедра Комп’ютерних інтелектуальних технологій та систем

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Аналіз алгоритмів машинного навчання для автоматичного масштабування контейнерних застосунків
(тема)

Виконав:
студент II курсу, групи КІТм-22-1
Віталій ВЛАСОВ
(власне ім'я, прізвище)
Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Комп'ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник доц. каф. КІТС Наталія СЕРДЮК
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри Олег РУДЕНКО
(підпис) (власне ім'я, прізвище)

2024 р.

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Власову Віталію Ігоревичу
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз алгоритмів машинного навчання для автоматичного масштабування контейнерних застосунків

затверджена наказом по університету від " 03 " листопада 2023 р. № 1290Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13.01.2024

3. Вхідні дані до роботи

1) історичний набір даних про використання ресурсів контейнерних застосунків;
2) побудова моделей для прогнозування та управління масштабуванням ресурсів;
3) аналіз за допомогою мови програмування Python.

4. Перелік питань, що потрібно опрацювати в роботі

1) огляд автоматичного масштабування в контейнерних застосунках;
2) аналіз теми управління ресурсами та масштабування;
3) аналіз існуючих рішень та методологій автоматичного масштабування;
4) дослідження інтелектуальних алгоритмів масштабування;
5) дослідження моделі прогнозованого та адаптивного масштабування ресурсів;
6) висновки та подальші напрямки роботи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій слайдів

6. Консультанти розділів роботи и (п.6 включається до завдання за наявністю консультантів згідно до наказу, зазначеному 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	Видача та узгодження теми проєкту	7.11-8.11	Виконано
2	Огляд стану проблеми та постановка задачі	10.11-20.11	Виконано
3	Аналіз літератури за напрямком роботи	20.11-29.11	Виконано
4	Аналіз алгоритмів масштабування	29.11-3.12	Виконано
5	Розробка інтелектуальних моделей	3.12-16.12	Виконано
6	Експериментальні дослідження	16.12-31.12	Виконано
7	Підготовка графічного матеріалу	31.01-9.01	Виконано
8	Перевірка виконаного проєкту керівником	9.01-14.01	Виконано
9	Захист проєкту	18.01	Виконано

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 77 с., 20 рис., 5 табл., 1 дод., 18 джерел.

АВТОМАТИЧНЕ МАСШТАБУВАННЯ, МАШИННЕ НАВЧАННЯ, КОНТЕЙНЕРНІ ЗАСТОСУНКИ, УПРАВЛІННЯ РЕСУРСАМИ, Q-НАВЧАННЯ, ГЛИБОКІ ДЕТЕРМІНОВАНІ ГРАДІЄНТИ ПОЛІТИКИ, ДОВГА КОРОТКОЧАСНА ПАМ'ЯТЬ, АВТОРЕГРЕСІЙНЕ ІНТЕГРОВАНЕ КОВЗНЕ СЕРЕДНЄ, PYTHON

Метою кваліфікаційної роботи є аналіз та порівняння набору алгоритмів машинного навчання для автоматичного масштабування в контейнерних застосунках, зосереджуючись на вдосконаленні стратегій управління ресурсами в хмарних середовищах. Дослідження зосереджується на оцінці ефективності, адаптивності та продуктивності цих алгоритмів за різних умов робочого навантаження.

Об'єктом дослідження є процес автоматичного масштабування в контейнерних середовищах.

Предметом є алгоритми машинного навчання, призначені для предиктивного та адаптивного управління ресурсами.

У роботі проаналізовано та порівняно традиційні стратегії масштабування та передові підходи до машинного навчання, зокрема, з акцентом на Q-learning, Deep Deterministic Policy Gradients (DDPG), Autoregressive Integrated Moving Average (ARIMA) та Long Short-Term Memory (LSTM). За допомогою Python вони були протестовані на адаптивність та ефективність використання ресурсів в умовах різного робочого навантаження. Отримані результати дають уявлення про продуктивність кожного алгоритму, що сприятиме майбутній оптимізації автоматичного масштабування для контейнерних застосунків.

ABSTRACT

Master’s thesis 77 pages, 20 figures, 5 tables, 1 appendice, 18 sources.

AUTOMATIC SCALING, MACHINE LEARNING, CONTAINERIZED APPLICATIONS, RESOURCE MANAGEMENT, Q-LEARNING, DEEP DETERMINISTIC POLICY GRADIENTS, LONG SHORT-TERM MEMORY, AUTOREGRESSIVE INTEGRATED MOVING AVERAGE, PYTHON

The qualification work aims to analyze and compare a set of machine learning algorithms for automatic scaling in containerized applications, focusing on improving resource management strategies in cloud environments. The research centers around evaluating the efficiency, adaptability, and performance of these algorithms under varying workload conditions.

The object of research is the automatic scaling process in containerized environments.

The subject encompasses machine learning algorithms designed for predictive and adaptive resource management.

The paper analyzes and compares traditional scaling strategies and advanced machine learning approaches, particularly focusing on Q-Learning, Deep Deterministic Policy Gradients (DDPG), Autoregressive Integrated Moving Average (ARIMA), and Long Short-Term Memory (LSTM). These are tested for their adaptability and resource efficiency under varying workload conditions using Python. The findings provide insights into each algorithm's performance, informing future optimizations in automatic scaling for containerized applications.

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
Кафедра Комп’ютерних інтелектуальних технологій та систем

АНОТАЦІЯ

КВАЛІФІКАЦІЙНОЇ РОБОТИ

рівень вищої освіти другий (магістерський)

Аналіз алгоритмів машинного навчання для автоматичного масштабування контейнерних застосунків
(тема)

Виконав:
студент II курсу, групи КІТм-22-1
Віталій ВЛАСОВ
(власне ім'я, прізвище)

Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)

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

Освітня програма Комп'ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник доц. каф. КІТС Наталія СЕРДЮК
(посада, власне ім'я, прізвище)

2023 р.

АНОТАЦІЯ

Власов В. І. Аналіз алгоритмів машинного навчання для автоматичного масштабування контейнерних застосунків. – Магістерська кваліфікаційна робота.

У магістерській кваліфікаційній роботі аналізується задача поліпшення автоматичного масштабування контейнеризованих застосунків.

Метою кваліфікаційної роботи є аналіз та порівняння набору алгоритмів машинного навчання для автоматичного масштабування в контейнерних застосунках, зосереджуючись на вдосконаленні стратегій управління ресурсами в хмарних середовищах. Дослідження зосереджується на оцінці ефективності, адаптивності та продуктивності цих алгоритмів за різних умов робочого навантаження.

Об'єктом дослідження є процес автоматичного масштабування в контейнерних середовищах.

Предметом є алгоритми машинного навчання, призначені для предиктивного та адаптивного управління ресурсами.

Наукова новизна даної магістерської роботи полягає у глибокому аналізі та порівнянні різних алгоритмів машинного навчання для автоматичного масштабування у контейнерних застосунках. Зокрема, вона включає дослідження і вдосконалення стратегій управління ресурсами, з акцентом на адаптивність та продуктивність у хмарних середовищах. Робота детально розглядає такі передові алгоритми машинного навчання, як Q-learning, Deep Deterministic Policy Gradients (DDPG), Autoregressive Integrated Moving Average (ARIMA), та Long Short-Term Memory (LSTM).

Практична цінність отриманих результатів полягає у впровадженні набору алгоритмів машинного навчання для ефективного автоматичного масштабування в контейнеризованих застосунках. Такі рішення сприяють підвищенню ефективності IT-інфраструктур, забезпечуючи стабільність та гнучкість сервісів, а також відкриває нові можливості для подальшої оптимізації та адаптації систем в залежності від змінюваних потреб бізнесу та технологій.

У першому розділі роботи проведено детальний аналіз предметної області автоматичного масштабування контейнерних застосунків, зокрема з оглядом на технології контейнерів, їхню динамічну природу та потребу в ефективному управлінні ресурсами. Розглянуто питання актуальності впровадження автоматичного масштабування як здатності системи до саморегулювання кількості активних контейнерів або ресурсів на підставі поточних запитів та показників продуктивності.

Особлива увага у цьому розділі приділяється впровадженню алгоритмів машинного навчання, зокрема навчання з підкріпленням (RL) та алгоритмів прогнозування часових рядів. Розглядаються підходи та перспективи застосування Q-Learning, глибоких детермінованих політичних градієнтів (DDPG), авторегресійного інтегрованого ковзного середнього (ARIMA), та мереж з довгою короткочасною пам'яттю (LSTM) у контексті автоматичного масштабування.

Цей розділ також висвітлює попередні роботи у сфері автоматичного масштабування, демонструючи еволюцію підходів від простих систем, заснованих на правилах, до більш складних платформ для оркестрування контейнерів, таких як Kubernetes. Окреслено основні виклики та обмеження, з якими зіштовхуються ці системи, та представлено потенціал інтеграції різних методів, включаючи правила, прогнозування та навчання з підкріпленням, для створення гібридних систем, які можуть поєднувати надійність і простоту систем, заснованих на правилах, з передбачуваністю предиктивних моделей та адаптивністю алгоритмів RL.

У другому розділі дослідження викладено процес реалізації та аналіз алгоритмів автоматичного масштабування контейнерних застосунків. Починаючи зі збору даних, розділ надає інформацію про методики та інструменти, які були використані для імітації реальних сценаріїв робочого навантаження в контейнерних середовищах.

В цьому розділі було створено реалістичне тестове середовище на основі Kubernetes, що дозволило протестувати алгоритми Q-Learning, DDPG, ARIMA, та LSTM. Встановлення та налаштування цього середовища забезпечило контрольовані, але гнучкі умови для оцінки адаптивності та продуктивності кожного алгоритму.

Розділ також детально описує процес розробки скриптів завантаження, які імітують різноманітні сценарії робочого навантаження. Це включає імітацію постійних, пікових та змінних навантажень, щоб перевірити, як алгоритми адаптуються до змін у вимогах до ресурсів. Сценарії були протестовані за допомогою інструментів як Apache JMeter для забезпечення широкого спектру умов тестування.

Далі у розділі викладено реалізацію кожного з алгоритмів масштабування, включаючи їх настройку, інтеграцію та методики тестування. Реалізація кожного алгоритму вимагала певної специфікації та налаштування, від стратегій розробки до вибору бібліотек та інструментів. Особлива увага була приділена Q-Learning та DDPG, оскільки вони використовують методи навчання з підкріпленням.

Останній аспект другого розділу охоплює налаштування безперервного навчання, важливого для забезпечення того, щоб алгоритми залишалися ефективними і актуальними навіть при змінних умовах. Безперервне навчання включало збір нових даних, їх інтеграцію в навчальний процес, і оновлення моделі для покращення прийняття рішень.

Третій розділ присвячений аналізу та моделюванню алгоритмів автоматичного масштабування контейнеризованих застосунків. Він зосереджується на експериментальному аналізі алгоритмів Q-Learning, DDPG, ARIMA та LSTM, оцінюючи їхню ефективність, швидкість, масштабування, ресурсоємність та адаптивність в змодельованому середовищі Kubernetes. В цьому розділі надається детальний аналіз результатів для кожного алгоритму, що відображає їхню продуктивність за вказаними критеріями.

Особлива увага приділяється нормалізації результатів тестування, щоб забезпечити їх порівняльний аналіз. Кожен алгоритм оцінюється за шкалою від 1 до 10, де вищі значення вказують на кращу продуктивність. Використання нормалізації дозволяє забезпечити об'єктивність порівняння між різними алгоритмами.

Далі в розділі представлено результати для кожного алгоритму Q-Learning, DDPG, ARIMA та LSTM. Описано, як кожен з них справляється з різними моделями робочого навантаження, їхню здатність адаптуватися до змін, швидкість прийняття рішень, масштабованість, стабільність і ефективність використання ресурсів. Значна

увага приділяється адаптивності алгоритмів, їхній здатності вдосконалювати свої стратегії на основі зворотного зв'язку, що є критично важливим у динамічних середовищах контейнеризованих застосунків.

Заключний аспект третього розділу включає інтерпретацію результатів, що дає змогу порівняти продуктивність кожного алгоритму по визначеним критеріям оцінки. Наведені результати у таблицях і графіках надають стисле, але всебічне уявлення про продуктивність кожного алгоритму, допомагаючи розуміти компроміси, пов'язані з вибором відповідного алгоритму для конкретних завдань автоматичного масштабування. Це спрямовує осіб, які приймають рішення, до найбільш підходящого вибору на основі їхніх конкретних потреб та обмежень.

АВТОМАТИЧНЕ МАСШТАБУВАННЯ, МАШИННЕ НАВЧАННЯ, КОНТЕЙНЕРНІ ЗАСТОСУНКИ, УПРАВЛІННЯ РЕСУРСАМИ, Q-НАВЧАННЯ, ГЛИБОКІ ДЕТЕРМІНОВАНІ ГРАДІЄНТИ ПОЛІТИКИ, ДОВГА КОРОТКОЧАСНА ПАМ'ЯТЬ, АВТОРЕГРЕСІЙНЕ ІНТЕГРОВАНЕ КОВЗНЕ СЕРЕДНЄ, PYTHON

Публікації здобувача за темою роботи:

1. Полоус В.Ю., Власов В.І. Основні інструменти для реалізації нейромережного підходу в задачах прогнозування. Радіоелектроніка та молодь у XXI столітті : XXVII Міжнар. молодіж. форум, м. Харків, 12 трав. 2023 р. / Наук. керівник Н. М. Сердюк. Харків, 2023. С. 11–12.

2. Власов В.І., Полоус В.Ю. Використання контейнерних застосунків для задачі прогнозування. Радіоелектроніка та молодь у XXI столітті : XXVII Міжнар. молодіж. форум, м. Харків, 12 трав. 2023 р. / Наук. керівник Н. М. Сердюк. Харків, 2023. С. 13–14.

3. Власов В. І. Особливості використання автомасштабованих контейнерних застосунків для аналізу великих даних. Пріоритетні напрямки та вектори розвитку світової науки : III Міжнар. студент. наук. конф., м. Черкаси, 31 берез. 2023 р. / Наук. керівник Н. М. Сердюк. Черкаси, 2023.

ЗМІСТ

Скорочення та умовні познаки

RL – Reinforcement Learning

DDPG – Deep Deterministic Policy Gradient

LSTM – Long Short-Term Memory

ARIMA – Autoregressive Integrated Moving Average

AR – Autoregression

MA – Moving Average

RNN – Recurrent Neural Network

RBAC – Role-Based Access Control

Вступ

У середовищі хмарних обчислень і мікросервісів, що постійно розвивається, динамічне масштабування контейнерних застосунків стало критично важливим завданням для розробників. Технології контейнеризації, такі як Docker, та інструменти оркестрування, такі як Kubernetes, змінили спосіб розгортання та управління застосунками, пропонуючи гнучкість та ефективність. Однак з цими досягненнями з'явилася потреба в інтелектуальних рішеннях для автоматичного масштабування, які можуть адаптуватися до мінливих вимог і оптимізувати використання ресурсів в режимі реального часу.

Мета цієї роботи – аналіз алгоритмів машинного навчання, зосередившись на алгоритмах навчання з підкріпленням, таких як Q-Learning і глибокі детерміновані градієнти політики (Deep Deterministic Policy Gradient), а також алгоритмах прогнозування часових рядів, включаючи авторегресійне інтегроване ковзне середнє (Autoregressive integrated moving average) і мережі з довгою короткостроковою пам'яттю (Long short-term memory). Ці алгоритми являють собою передовий прогрес у сфері прогностичної аналітики та методологій прийняття рішень, маючи потенціал трансформувати стратегії автоматизованого масштабування для застосунків, що використовують контейнерну технологію.

Автоматичне масштабування в контексті контейнеризованих застосунків передбачає динамічне регулювання кількості екземплярів контейнерів та розподіл ресурсів для відповідності актуальному попиту без людського втручання. Цей механізм є критичним для забезпечення оптимальної продуктивності та доступності, а також для підтримки економічної ефективності у середовищах, де використання ресурсів безпосередньо корелює з операційними витратами. Традиційні підходи до масштабування, які залежать від статично визначених правил та порогових значень, часто недостатньо гнучкі та прогностичні для ефективної роботи зі складними та непередбачуваними навантаженнями. Натомість методи машинного навчання, адаптуючись через аналіз історичних даних, можуть прогнозувати майбутні потреби

та адаптивно регулювати масштабування в реальному часі, забезпечуючи більш точне та ефективне розподілення ресурсів.

Проводячи аналіз та порівняння цих алгоритмів, дослідження спрямоване на здобуття знань про продуктивність, ефективність та придатність алгоритмів для різноманітних сценаріїв масштабування контейнерних застосунків. Дослідження включатиме аналіз інтеграції цих алгоритмів у середовища Docker і Kubernetes з метою автоматизації процесу масштабування, реагування на зміни умов та оптимізації розподілу ресурсів. Через серію експериментів та аналітичних оцінок, робота надасть аналіз переваг та недоліків кожного з алгоритмів, пропонуючи обґрунтовані рекомендації для спеціалістів у даній області.

Отже, основна ціль даної дослідницької роботи полягає у висвітленні стратегій досягнення більш інтелектуальних та ефективних методів автоматичного масштабування за допомогою застосування передових алгоритмів машинного навчання. Через детальний аналіз цих алгоритмів, дослідження має на меті надати спеціалістам та науковцям важливі знання та інструменти, що дозволять ефективно вирішувати виклики автоматичного масштабування у сучасному контексті контейнеризації та інших сфер застосування.

1 Аналіз предметної області та постановка задачі

1.1 Огляд контейнерних технологій

Поява технології контейнерів відзначилася як перехід до нової парадигми розгортання та управління програмними застосунками. Вона привнесла із собою нову епоху, характеризуючись високою ефективністю, мобільністю та можливістю масштабування. Контейнери володіють здатністю інкапсулювати застосунку та їх залежності в невеликий та автономний контейнер, забезпечуючи єдність середовища від розробки до виробництва [1]. Ця інновація була ініційована платформами, такими як Docker, які надають необхідні інструменти для упакування та запуску контейнерів, і системами оркестрування, такими як Kubernetes, які відповідають за управління розгортанням та масштабуванням контейнерів на кластерах серверів. З розвитком цих технологій вони стали основою сучасних стратегій розгортання програмних застосунків, особливо в хмарних обчислювальних середовищах.

Проте, динамічна та розподілена природа контейнерних застосунків народжує проблеми управління ресурсами та масштабуванням. У традиційних конфігураціях масштабування часто вимагало ручної настройки або автоматизації, що базувалася на простих показниках, таких як використання процесора та обсяг пам'яті. Попри те, що ці підходи ефективні в простих сценаріях, вони виявляються недостатніми для вимог сучасних застосунків, які можуть мати непередбачувані робочі навантаження, складні взаємозв'язки та високі вимоги до продуктивності. Наприклад, несподіваний приріст трафіку або фонові обчислювальні завдання можуть спричинити гостру зміну потреб у ресурсах, і система масштабування повинна оперативно та ефективно реагувати, щоб забезпечити відповідну якість обслуговування.

На даному етапі набуває актуальності впровадження автоматичного масштабування, яке може бути визначене як здатність системи до саморегулювання кількості активних контейнерів або ресурсів, які використовуються, на підставі поточних запитів і показників продуктивності. Ця функція, окрім забезпечення

збереження швидкодії та доступності застосунків, оптимізує використання ресурсів, що призводить до зменшення витрат і негативного впливу на навколишнє середовище. Однак реалізація ефективного автоматичного масштабування є завданням, що вимагає серйозного дослідження. Вона передбачає необхідність аналізу поведінки застосунків, здатність до прогнозування майбутніх потреб та розробку стратегій прийняття рішень щодо масштабування. Ці стратегії повинні бути спрямовані на балансування нагальних потреб з врахуванням довгострокової продуктивності та вартості.

Здійснення впровадження алгоритмів машинного навчання, визначених як набір методик, що відзначаються значущими перспективами в контексті предиативної аналітики, розпізнавання образів та прийняття рішень, стає ключовим елементом діяльності. Застосунки алгоритмів машинного навчання здатні аналізувати історичні дані з метою вивчення взаємозв'язку попиту та різних індикаторів, прогнозування майбутнього попиту на підставі поточних тенденцій та раціонального прийняття рішень стосовно розширення ресурсів. Цей потенціал прогнозування набуває особливого значення у контексті операцій зі складними та змінними робочими завданнями, де звичайні підходи до масштабування на основі порогових значень демонструють обмежену ефективність.

Серед цих алгоритмів, насамперед, слід відзначити алгоритми навчання з підкріпленням (Reinforcement Learning, RL) і алгоритми прогнозування часових рядів, оскільки вони мають розглядатися у контексті автоматичного масштабування.

Навчання з підкріпленням (RL) передбачає процес навчання агента, який приймає рішення, взаємодіючи з довкілля. У рамках автоматичного масштабування, агент навчається приймати рішення щодо масштабування, оптимізуючи зазначені метрики продуктивності, наприклад, час відгуку. Цей підхід цінується за можливість адаптації до змінних умов та оптимізацію націлену на досягнення довгострокових цілей, що виходять за межі простої реакції на поточний попит [2].

Методи прогнозування часових рядів передбачають прогнозування майбутніх значень на основі історичних даних. Алгоритми, такі як ARIMA та LSTM, демонструють особливу ефективність в виявленні тенденцій, циклів та інших

закономірностей в даних часових рядів.

Ці два вищезазначені алгоритми представляють два різні підходи до розв'язання проблеми масштабування, кожен з яких має свої переваги та специфічні аспекти. Шляхом проведення детального аналізу та порівняння цих методів ми сподіваємося встановити оптимальні підходи для різних сценаріїв, розробити належні налаштування та параметризацію, а також ідентифікувати необхідні компроміси.

1.2 Сфера застосування методів автоматичного масштабування

З метою підвищення ефективності та оперативності процесу автоматичного масштабування в контейнерних застосунках, аналіз фокусується на двох основних категоріях алгоритмів машинного навчання: навчання з підкріпленням (RL) та прогнозування часових рядів. Вибрані для вивчення алгоритми охоплюють Q-навчання та глибокі детерміновані градієнти політики (DDPG) у сфері RL, а також авторегресійне інтегроване ковзне середнє (ARIMA) та мережі з довгою короткостроковою пам'яттю (LSTM) у сфері прогнозування часових рядів. Вибір цих алгоритмів базується на їхній відомій ефективності та актуальності у галузях предиктивної аналітики та прийняття рішень.

Q-Learning – це безмодельний алгоритм навчання з підкріпленням, відомий своєю простотою та ефективністю. Він не потребує моделі середовища і може розв'язувати проблеми зі стохастичними переходами та винагородами, не вимагаючи адаптації.

В контексті автоматичного масштабування, Q-Learning може бути використано для вивчення оптимальних стратегій масштабування шляхом взаємодії з середовищем контейнерів та отримання зворотного зв'язку у вигляді метрик продуктивності. Здатність навчатися в процесі безпосередньої взаємодії з середовищем робить його надійним вибором для динамічних і непередбачуваних сценаріїв робочого навантаження, які часто зустрічаються в контейнерних застосунках [3].

DDPG – це алгоритм, який працює на неперервних просторах дій і поєднує в

собі ідеї DQN (Deep Q Network) та методів градієнта політики. Алгоритм використовує нейронну мережу для апроксимації політики, а іншу – для апроксимації функції вартості, що дозволяє працювати з просторами станів і дій високої розмірності.

Застосування DDPG до просторів безперервних дій робить його особливо придатним для сценаріїв, де рішення про масштабування передбачають тонкі коригування, такі як точне налаштування кількості контейнерів або розподіл ресурсів. Фундамент глибокого навчання дозволяє йому вловлювати складні закономірності та залежності, забезпечуючи потужний інструмент для прийняття тонких рішень, необхідних при автоматичному масштабуванні [3].

ARIMA – це класична модель прогнозування часових рядів, яка поєднує компоненти авторегресії (AR) та ковзного середнього (MA) з диференціюванням для стабілізації ряду. Вона відома своєю гнучкістю та ефективністю у відображенні різноманітних закономірностей у часових даних. В контексті автоматичного масштабування ARIMA можна використовувати для прогнозування майбутнього навантаження на основі історичних даних, забезпечуючи основу для прийняття проактивних рішень щодо масштабування. Здатність моделювати й прогнозувати складні часові ряди робить ARIMA інструментом для передбачення сплесків попиту і тенденцій в контейнерних середовищах.

LSTM – це тип рекурентної нейронної мережі (RNN), спеціально розроблений для подолання обмежень традиційних RNN у вивченні довготривалих залежностей. Вона вводить спеціальні структури, які називаються комірками пам'яті, що дозволяють їй ефективно запам'ятовувати та використовувати минулу інформацію в довгих послідовностях. Вміння LSTM працювати з довгостроковими залежностями робить його вправним у прогнозуванні робочих навантажень у контейнерних застосунках, де минулі моделі використання, сезонні тенденції або рідкісні події можуть впливати на майбутні потреби в ресурсах. Архітектура нейронної мережі дозволяє LSTM вловлювати нелінійні зв'язки та складні закономірності, забезпечуючи тонкі та точні прогнози для прийняття обґрунтованих рішень щодо масштабування [6].

1.3 Попередні роботи у сфері автоматичного масштабування

Шлях до досягнення ефективного автоматичного масштабування та управління ресурсами в контейнерних застосунках є результатом значних наукових досліджень та розробок в даній галузі. Ця проблема вимагала розробки різноманітних стратегій та технологій для забезпечення стабільної продуктивності та ефективності застосунків, незалежно від коливань робочого навантаження. Нижче ми розглянемо еволюцію цих методів та поточний стан досліджень, які формують основу нашого аналізу.

Початково масштабування здійснювалося вручну або за допомогою простих систем, заснованих на правилах. Адміністратори встановлювали статичні пороги для різних ресурсів, таких як процесор, пам'ять або мережевий трафік, та активували масштабування, коли ці пороги перетиналися. Ці методи були легкі для розуміння та впровадження, але часто приводили або до надмірного виділення ресурсів, або до недостатнього виділення ресурсів.

З часом системи, засновані на правилах, стали складнішими, включаючи деталізовані правила та метрики, а також базові елементи прогнозування для передбачення майбутніх потреб [3]. Однак вони залишалися реактивними та обмеженими сценаріями, визначеними правилами.

Платформи для оркестрування контейнерів, такі як Kubernetes та Docker Swarm, представили більш динамічні та гнучкі механізми масштабування [3]. Вони використовують моніторинг та метрики для автоматичної настройки контейнерів на основі реальних даних. Проте ці платформи все ще в основному покладаються на порогові значення та не використовують складні прогностичні моделі.

Дослідники розглядали використання моделей прогнозування часових рядів, таких як ARIMA, для передбачення навантажень та масштабування ресурсів. Однак практичне застосування цих моделей виявляється складним через непередбачуваність робочих навантажень та високі обчислювальні витрати.

Навчання з підкріпленням недавно виявилося потужним підходом до управління ресурсами в динамічних середовищах. Алгоритми RL навчаються

оптимальним діям шляхом спроб і помилок, що дозволяє системі адаптивно покращувати свою політику з часом [5]. Вивчається інтеграція різних методів, включаючи підходи на основі правил, прогнозування та навчання з підкріпленням. Ці гібридні системи можуть поєднувати надійність і простоту систем, заснованих на правилах, з передбачуваністю предиативних моделей та адаптивністю алгоритмів RL, пропонуючи більш ефективні механізми масштабування.

1.5 Огляд алгоритмів навчання

1.5.1 Q-Learning

Q-Learning – це безмодельний алгоритм, який використовується в навчанні з підкріпленням. «Q» в Q-Learning означає якість певної дії в певному стані. Алгоритм вивчає політику, яка вказує агенту, яку дію виконувати за певних обставин. Він не потребує моделі свого середовища і, отже, вважається безмодельним, що робить його придатним для широкого спектра застосувань, включаючи сценарії з непередбачуваним або динамічним середовищем, як, наприклад, при масштабуванні контейнерів.

Основою Q-навчання є Q-таблиця, яка по суті є пошуковою таблицею, де кожен рядок представляє стан середовища, а кожен стовпець – можливу дію. Значення в таблиці, відомі як Q-значення, представляють очікувану корисність виконання певної дії в даному стані, і вони оновлюються в міру того, як агент вчиться на власному досвіді. Формула для оновлення Q-значень є формою навчання на основі часової різниці, яка поєднує старе значення з новою інформацією, отриманою після виконання дії та отримання винагороди або покарання.

Процес прийняття рішень у Q-навчанні зазвичай передбачає компроміс між дослідженням та експлуатацією. На початку агент досліджує навколишнє середовище, вибираючи дії випадковим чином, дізнаючись про винагороду, пов'язану з різними стадіями. Коли агент навчається, починає використовувати свої знання, обираючи дії, які максимізують очікувану винагороду на основі Q-таблиці. Для

балансу між дослідженням та експлуатацією часто використовуються такі методи, як стратегія ε-жадібності, де агент здебільшого обирає найвідоміші дії, але час від часу пробує щось нове.

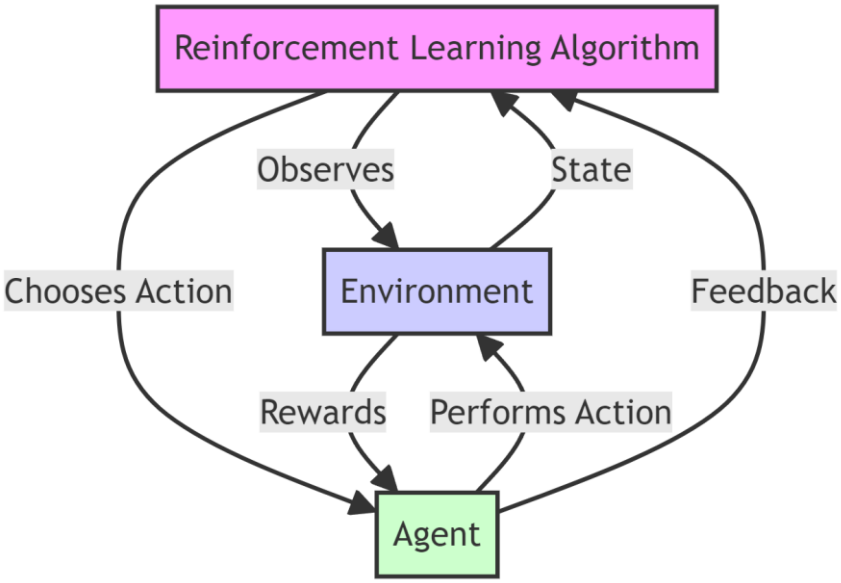

Рисунок 1.1 – Блок-схема алгоритмів алгоритмі навчання з підкріпленням

На рисунку 1.1 блок-схема ілюструє ключові компоненти та їх взаємодію в алгоритмі навчання з підкріпленням (RL), який використовується для автоматизації управління контейнерними застосунками. Алгоритм навчання з підкріпленням є центральним компонентом даної системи. Він виконує функцію організації навчального процесу, спостерігаючи за навколишнім середовищем з метою аналізу поточного стану. На основі цієї інформації алгоритм приймає рішення щодо оптимальних дій. З часом алгоритм навчається та вдосконалюється завдяки зворотному зв'язку, який він отримує від навколишнього середовища через винагороди, які залежать від результатів його дій. Середовище є фундаментальним компонентом системи та являють собою віртуальний світ, в якому агент (алгоритм навчання з підкріпленням) здійснює взаємодію. В контексті автоматизації контейнерних застосунків, це середовище може включати стан системи, параметри продуктивності, а також інші важливі дані. Середовище не лише надає поточний стан алгоритму RL, але й впливає на його дії та надає винагороду агенту на основі

успішності або невдачі здійснених дій. Агент виступає як виконавчий орган системи, реалізуючи дії в середовищі відповідно до алгоритму навчання з підкріпленням. У контексті автоматизації управління контейнерними застосунками, це може включати операції, такі як масштабування ресурсів та налаштування параметрів. Після виконання дії агент отримує винагороду від середовища, що свідчить про успішність або невдачу здійснених дій. Ця винагорода служить як сигнал для алгоритму RL, який використовує її для навчання та вдосконалення свого рішення з плином часу.

1.5.2 Глибокі детерміновані політичні градієнти (DDPG)

Глибокий детермінований градієнт лінії поведінки DDPG – це алгоритм, який поєднує ідеї DQN (Deep Q-Networks) та методів градієнта політики. Це алгоритм, який поширює ідею Q-Learning на безперервні простори дій. З практичної точки зору, в той час, як Q-Learning підходить для сценаріїв, де можливі дії є дискретними та обмеженими, DDPG показує кращі результати, коли дії є безперервними, наприклад, коригування ресурсів для контейнера на точний відсоток.

DDPG використовує дві нейронні мережі: актора та критика. Мережа актора відповідає за вибір дій, по суті, вивчаючи функцію політики, яка відображає стани в найкращі безперервні дії. Мережа критика оцінює функцію Q-значення, яка оцінює дії, що виконуються актором. Обидві мережі навчаються одночасно за допомогою зворотного поширення і техніки, яка називається відтворенням досвіду, де в буфері зберігається колекція кортежів досвіду (стан, дія, винагорода, наступний стан), з яких алгоритм робить вибірки для навчання. Це допомагає стабілізувати процес навчання і розірвати кореляцію між послідовними кроками навчання.

У DDPG мережа акторів безпосередньо обчислює найкращу дію, а не вибирає із заздалегідь визначених варіантів. Мережа критиків допомагає актору приймати рішення, надаючи зворотний зв'язок щодо очікуваної віддачі від дій. DDPG також включає шум (наприклад, процес Орнштейна-Уленбека) у вибір дії, щоб полегшити дослідження простору дій [12]. З часом, коли мережі навчаються і зближуються, алгоритм стає кращим у виборі дій, які максимізують довгострокову вигоду, що

робить його особливо ефективним для проблем, які потребують тонкого балансу між негайними й майбутніми міркуваннями.

1.5.3 Авторегресійне інтегроване ковзне середнє (ARIMA)

ARIMA – це популярна статистична модель, яка використовується для прогнозування та розуміння даних часових рядів. Вона є узагальненням простіших моделей, таких як авторегресія (AR) та ковзаюче середнє (MA), і призначена для опису автокорельованих часових рядів. Модель зазвичай позначається як:

$$\mathrm{ARIMA(p, d, q)}, \tag{1.1}$$

де n – кількість членів авторегресії;

d – кількість несезонних різниць, необхідних для стаціонарності;

q – кількість вагових помилок прогнозу в рівнянні прогнозу.

У моделюванні ARIMA першим кроком, як правило, є забезпечення стаціонарності часового ряду. Це передбачає диференціювання даних (віднімання спостереження від спостереження на попередньому часовому кроці) один або кілька разів, щоб усунути тенденції або сезонні структури. Компонента $AR\ (p)$ моделює поточне значення часового ряду як лінійну комбінацію попередніх значень. Компонента $MA\ (q)$ моделює його як лінійну комбінацію членів помилок минулих прогнозів. Параметри моделі зазвичай оцінюються за допомогою таких методів, як оцінка максимальної правдоподібності або нелінійний метод найменших квадратів, підганяючи модель під історичні дані.

Після успішного навчання моделі ARIMA, отриманої з використанням історичних даних, вона стає здатною до прогнозування майбутніх значень ряду. Основним принципом функціонування моделі ARIMA є лінійна комбінація попередніх спостережень та попередніх помилок. Цей підхід дозволяє моделі генерувати прогнози на основі історичних даних.

Отримані прогнози від моделі ARIMA можуть бути використані для прийняття обґрунтованих рішень щодо масштабування ресурсів в очікуванні майбутнього попиту. Важливою особливістю моделі є її здатність адаптуватися до змін у базовому

процесі з часом. Це досягається коштом оновлення моделі з використанням нових даних, що надходять у процесі роботи застосунку. Таким чином, модель ARIMA стає адаптивною до змін у вхідних даних і може надавати актуальні прогнози у реальному часі.

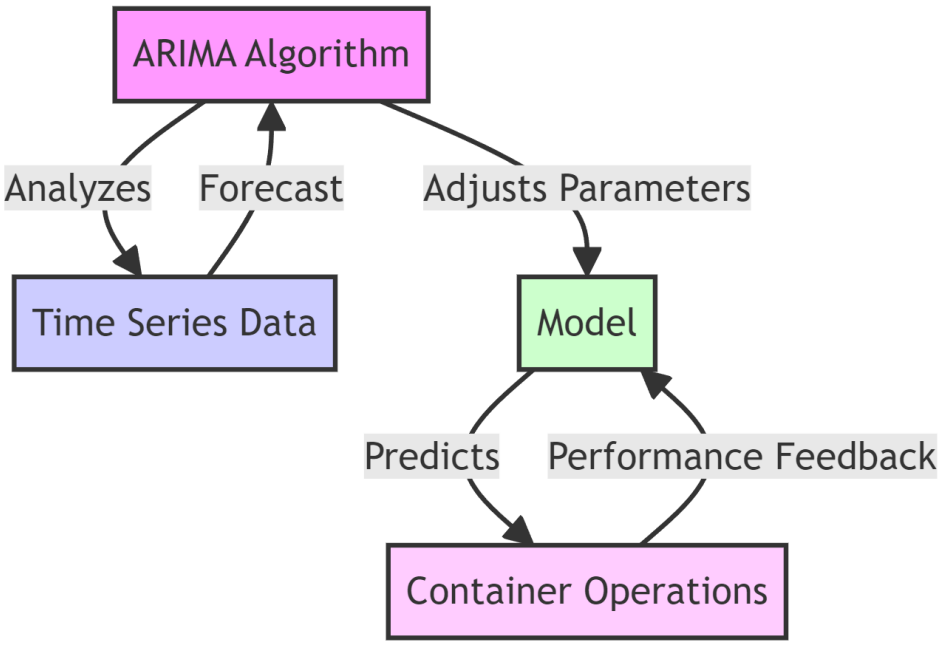

Рисунок 1.2 – Блок-схема алгоритму ARIMA

Представлена блок-схема на рисунку 1.2 ілюструє використання алгоритму ARIMA у сфері автоматизації контейнерних перевантажень. Розглянемо кожен компонент цієї системи та їх взаємодію.

Алгоритм ARIMA є центральним компонентом системи. Ця статистична модель спеціалізується на аналізі часових рядів даних. Використовуючи дані часових рядів, алгоритм ARIMA розглядає закономірності та тенденції, що містяться в цих даних з метою прогнозування.

Дані часових рядів є історичними даними, що охоплюють певний період часу. У контексті автоматизації контейнерних перевантажень ці дані можуть охоплювати інформацію про використання ресурсів, продуктивність або інші важливі операційні статистичні показники. Алгоритм ARIMA використовує ці дані для ретельного аналізу та розробки точних прогнозів.

Модель ARIMA – це математична модель, яка автоматично налаштовує свої параметри на основі аналізу даних часових рядів. Ці параметри постійно вдосконалюються і коригуються з часом, що покращує здатність моделі до точного прогнозування.

Операції з контейнерами становлять дії, які виконуються в середовищі контейнера, такі як масштабування ресурсів або коригування конфігурацій. Модель ARIMA прогнозує результати цих операцій і на основі фактичних даних про продуктивність уточнює свої прогнози.

На діаграмі показані наступні взаємодії:

- алгоритм ARIMA аналізує дані часових рядів, для зрозуміння закономірностей;
- прогнозує майбутні тенденції та інформує модель;
- модель коригує свої параметри на основі прогнозу та прогнозує результати контейнерних операцій;
- виконуються контейнерні операції, і модель отримує зворотний зв'язок про їх результати;
- зворотний зв'язок допомагає уточнити прогнози моделі, що роблять їх більш точними з часом.

1.5.4 Мережі довготривалої короткочасної пам'яті (LSTM)

LSTM-мережі – це особливий різновид рекурентних нейронних мереж (RNN), які здатні вивчати довгострокові залежності. Традиційні рекурентні нейронні мережі (RNN) виявляються недостатньо ефективними в моделюванні довгострокових залежностей в послідовних даних через певні технічні проблеми, такі як проблеми зникаючих та вибухових градієнтів. Відмінність між RNN і LSTM полягає у впровадженні унікальної структури механізмів воріт, що дозволяють останнім долати ці технічні обмеження.

Механізми воріт, властиві LSTM, включають регулюючі елементи, які керують потоком інформації в мережі. Ці ворота дозволяють LSTM зберігати чи забувати

інформацію протягом довгих часових інтервалів. Завдяки цим механізмам, LSTM може успішно моделювати складні залежності в послідовних даних і здатний здійснювати передбачення на довгій відстані [11].

LSTM обробляють дані часових рядів покроково, зберігаючи прихований вектор стану, який діє як «пам'ять» про минулі спостереженнях [15]. Цей стан оновлюється на кожному кроці, враховуючи поточні вхідні дані та попередній стан. Ключем до ефективності LSTM є три вентилі: вхідний вентиль, який контролює, скільки нового входу використовується для оновлення стану пам'яті; вентиль забуття, який контролює, скільки існуючої пам'яті забувається; і вихідний вентиль, який контролює внесок пам'яті у кінцевий вихід.

Після навчання LSTM можна використовувати для прогнозування майбутніх значень часового ряду. Маючи послідовність минулих спостережень, LSTM використовує вивчену структуру і параметри для прогнозування наступних значень у послідовності. Ці прогнози потім можуть бути використані для прийняття рішень про збільшення або зменшення ресурсів відповідно до очікуваного попиту. Як і у випадку з ARIMA, модель можна постійно оновлювати новими даними, щоб підтримувати її точність у часі.

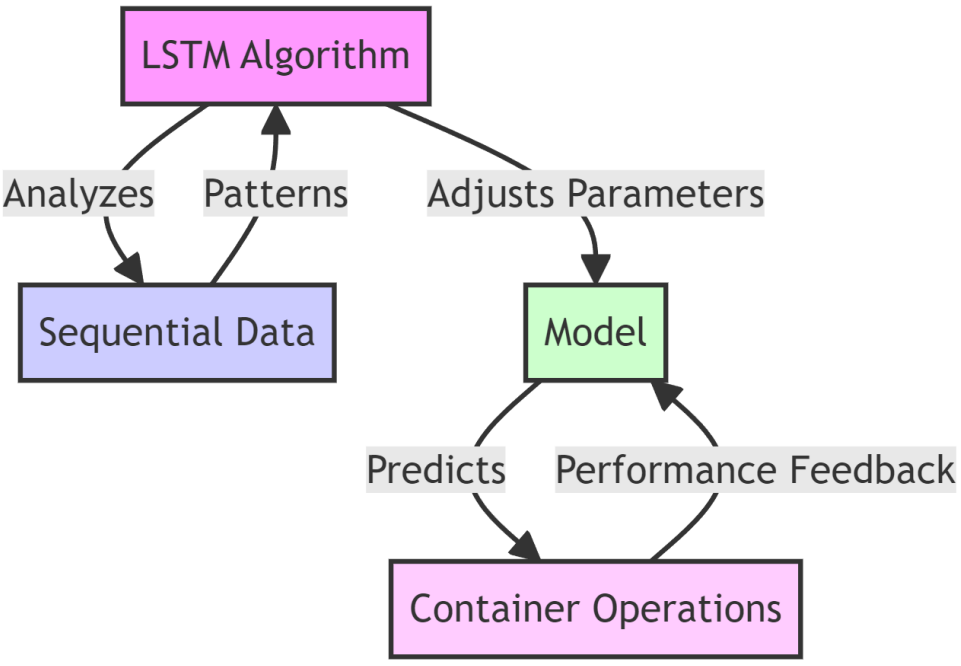

Рисунок 1.3 – Блок-схема алгоритму LSTM

Представлена блок-схема на рисунку 1.3 ілюструє використання алгоритму LSTM в галузі автоматизації контейнерів. Розглянемо кожен компонент цієї системи та їх взаємодію.

Алгоритм LSTM є центральним компонентом цієї системи. Це тип рекурентної нейронної мережі (RNN), відомий своєю здатністю запам'ятовувати інформацію протягом тривалого часу. Ця особливість робить пригідним для аналізу послідовних даних або часових рядів.

Послідовні дані становлять дані, які впорядковані в часі. У контексті автоматизації контейнерів, ці дані можуть містити інформацію про використання ресурсів, дані про продуктивність або іншу важливу операційну статистику. Алгоритм LSTM аналізує ці дані з метою розуміння складних закономірностей та залежностей, що містяться в них.

Модель LSTM налаштовує свої параметри, використовуючи аналіз послідовних даних, з метою здійснення точних прогнозів. Ці параметри постійно уточнюються при аналізі більшої кількості даних, що підвищує прогностичну здатність моделі.

Операції з контейнерами включають дії, що виконуються в середовищі контейнера, такі як масштабування ресурсів або коригування конфігурацій. Модель LSTM прогнозує результати цих операцій і на основі фактичних даних про продуктивність покращує свої прогнози.

На схемі показані наступні взаємодії:

- алгоритм LSTM аналізує послідовні дані, щоб зрозуміти складні закономірності;
- на основі цього аналізу інформує модель про виявлені закономірності;
- модель коригує свої параметри на основі виявлених закономірностей і прогнозує результати операцій з контейнерами;
- виконуються контейнерні операції, і модель отримує зворотний зв'язок про їх результати;
- зворотний зв'язок допомагає уточнити прогнози моделі, роблячи їх більш точними з часом.

Таким чином, як ARIMA, так і LSTM пропонують методи для розуміння та

прогнозування даних часових рядів. Перевага ARIMA полягає в його простоті та ефективності для широкого спектра сценаріїв, особливо коли дані можна зробити стаціонарними й вони добре вписуються в структуру ARIMA. Перевагою LSTM є його здатність фіксувати складні, довгострокові взаємозв'язки в даних, що робить його придатним для більш нюансованих або нелінійних моделей, які часто зустрічаються в реальних часових рядах

1.6. Критерії оцінювання порівняльного аналізу

Для того, щоб провести порівняльний аналіз обраних алгоритмів автоматичного масштабування контейнерних застосунків, важливо визначити набір чітких і об'єктивних критеріїв оцінки. Ці критерії допоможуть оцінити продуктивність і придатність кожного алгоритму за різних умов і сценаріїв. Для аналізу алгоритмів Q-Learning, DDPG, ARIMA та LSTM будуть використані наступні метрики та функції:

- точність;
- швидкість;
- масштабованість;
- ресурсоефективність;
- адаптивність.

Точність в контексті автоматичного масштабування означає здатність алгоритму правильно передбачити необхідні дії з масштабування (наприклад, збільшення або зменшення масштабу) і величину цих дій відповідно до фактичного попиту. Вимірюється шляхом порівняння прогнозів алгоритму з фактичними спостережуваними даними про використання ресурсів і попит. Для кількісної оцінки помилок прогнозування можна використовувати такі показники, як середня абсолютна похибка (MAE) або середньоквадратична похибка (RMSE) [15].

Середня абсолютна похибка (MAE) – ця метрика вимірює середню абсолютну різницю між прогнозованими та фактичними значеннями. Вона визначається як:

$$\mathrm{MAE} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}|\mathrm{y_i} - \widehat{y_\imath}|, \quad (1.2)$$

де n – кількість спостережень;

y_i – фактичне значення;

$\widehat{y_\imath}$ – прогнозоване значення.

MAE дає середню величину помилок у прогнозах, не звертаючи увагу на їх напрямок. Чим менше MAE, тим точнішим є прогноз. Це прямолінійна міра, яка є легкою для інтерпретації.

Середньоквадратична похибка (RMSE) – ця метрика вимірює квадратний корінь із середнього квадрата помилок. Вона визначається як:

опису автокорельованих часових рядів. Модель зазвичай позначається як:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_\imath})^2}, \quad (1.3)$$

де n – кількість спостережень;

y_i – фактичне значення;

$\widehat{y_\imath}$ – прогнозоване значення.

RMSE є більш чутливою до великих помилок, оскільки вона підносить кожну помилку до квадрата перед усередненням. Це означає, що великі помилки матимуть непропорційно великий вплив на загальний показник RMSE. Як і MAE, чим менше RMSE, тим краще, але через квадратичну природу, вона може бути особливо корисною, коли великі помилки є небажаними.

Швидкість – це час, необхідний алгоритму для аналізу даних і прийняття рішення про масштабування. У середовищі, де робочі навантаження можуть швидко змінюватися, здатність приймати швидкі рішення має вирішальне значення.

Швидкість вимірюється як час від отримання даних до видачі рішення. Сюди входить час попередньої обробки, час обчислень і час постобробки, пов'язаний з генеруванням дії масштабування:

$$V = T_{total} = T_{preprocessing} + T_{computation} + T_{postprocessing}, \quad (1.4)$$

де T_{total} – загальний час, необхідний для прийняття рішення про масштабування;

$T_{preprocessing}$ – час, необхідний для попередньої обробки даних;

$T_{computation}$ – час, необхідний для обчислень алгоритму;

$T_{postprocessing}$ – час, необхідний для постобробки та генерування дії масштабування.

Масштабованість належить до продуктивності алгоритму при збільшенні розміру та складності контейнерного середовища. Масштабований алгоритм повинен зберігати свою продуктивність і ефективність навіть при збільшенні кількості контейнерів або обсягу даних. Масштабованість буде оцінюватися шляхом оцінки продуктивності алгоритму в змодельованих середовищах різного розміру і складності, моніторингу будь-якої деградації швидкості, точності або ефективності використання ресурсів:

$$S = \lim_{n \to \infty} \frac{T(1)}{T(n)}, \tag{1.5}$$

де S – масштабованість;

$T(1)$ – час виконання алгоритму на одному оброблювальному елементі (наприклад, одному контейнері);

$T(n)$ – час виконання алгоритму на n оброблювальних елементах.

Ефективність використання ресурсів – це вимоги алгоритму до обчислювальних ресурсів та пам'яті. Ефективний алгоритм повинен використовувати мінімум ресурсів, гарантуючи, що накладні витрати на виконання алгоритму не зводять нанівець переваги від рекомендованих ним дій з масштабування. Кількісно це буде визначатися шляхом вимірювання використання процесора, пам'яті та інших відповідних показників під час роботи алгоритму. Для забезпечення об'єктивності порівняння буде проводитися за однакових умов для всіх алгоритмів.

$$E = \frac{U}{R}, \tag{1.6}$$

де E – ефективність використання ресурсів;

U – корисна робота або продуктивність, здійснена алгоритмом;

R – кількість використаних ресурсів.

Адаптивність – це здатність алгоритму навчатися і пристосовуватися до нових шаблонів, змін у середовищі або змін у характеристиках робочого навантаження. Це має вирішальне значення для підтримки продуктивності з часом, коли застосунок і його оточення розвиваються. Адаптивність буде оцінюватися шляхом внесення змін у змодельоване середовище та моніторингу того, наскільки швидко та ефективно алгоритм коригує свою стратегію масштабування. Це включає оцінку того, наскільки добре алгоритм включає нові дані в процес прийняття рішень, а також його здатність покращувати свою продуктивність з часом завдяки навчанню.

$$A = \frac{\Delta P}{\Delta T}, \tag{1.7}$$

де A – адаптивність;

ΔP – зміна продуктивності або точності алгоритму;

ΔT – зміна умов середовища або характеристик робочого навантаження.

Використовуючи ці критерії, ми прагнемо забезпечити ретельне та неупереджене порівняння Q-Learning, DDPG, ARIMA та LSTM в контексті автоматичного масштабування для контейнерних застосунків. Кінцевою метою є визначення сильних і слабких сторін кожного алгоритму, надання цінної інформації, яка може допомогти у виборі та впровадженні рішень для масштабування в різних сценаріях

2 Реалізація алгоритмів автоматичного масштабування контейнерних застосунків

2.1 Збір даних

Аналіз в галузі машинного навчання, зокрема в контексті оцінки продуктивності алгоритмів для автоматичного масштабування, вимагає наявності вичерпного обсягу даних. Збір даних для навчання та тестування алгоритмів вимагає відображення реальних сценаріїв, щоб отримані результати були позначені значущістю та застосовністю в практичних задачах. У нашому аналізі, спрямованому на оцінку алгоритмів Q-Learning, DDPG, ARIMA і LSTM, процес збору даних буде спроєктований для імітації умов, характерних для реальних контейнерних середовищ.

2.1.1 Дані імітації робочого навантаження

Дані про робоче навантаження будуть згенеровані за допомогою симуляції контейнерного середовища застосунків. Ця симуляція буде імітувати різні типи робочих навантажень, які зазвичай зустрічаються в реальних застосунках, такі як постійні навантаження, періодичні навантаження і непередбачувані, стрибкоподібні запити. Симуляція також включатиме такі події, як раптові напливи або простої, щоб відобразити мінливість і непередбачуваність реальних сценаріїв.

Зібрані дані включатимуть метрики, що стосуються рішень щодо масштабування, такі як використання процесора, пам'яті, мережевого вводу та виводу, частота запитів та частота помилок. Кожне змодельоване робоче навантаження буде працювати протягом достатнього часу, щоб зафіксувати закономірності та тенденції в цих показниках [2].

2.1.2 Історичні журнали з реальних систем

На застосунок до змодельованих даних, будуть використані історичні журнали

реальних контейнерних застосунків, якщо вони будуть доступні. Ці журнали будуть отримані з загальнодоступних наборів даних. Історичні дані будуть попередньо оброблені для забезпечення узгодженості та релевантності. Це передбачає очищення даних, нормалізацію метрик і структурування їх таким чином, щоб вони були сумісними з даними моделювання.

Конкретні події в контейнерному середовищі, такі як запуск, зупинка, збої та відновлення контейнера, будуть реєструватися разом з їхніми часовими мітками. Ці дані нададуть додатковий контекст для алгоритмів масштабування та допоможуть оцінити їхню швидкість реагування та якість прийняття рішень.

2.1.2 Зворотний зв'язок та коригування

Для імітації реалістичного середовища навчання алгоритмів, особливо алгоритмів навчання з підкріпленням, буде реалізовано цикл зворотного зв'язку. Це означає, що результат кожного рішення про масштабування (наприклад, успіх в управлінні навантаженням, економічна ефективність) буде повертатися в систему як частина процесу навчання. Симуляція дозволить вносити корективи в робочі навантаження і параметри на основі початкових результатів і спостережень за поведінкою алгоритмів. Такий ітеративний підхід гарантує, що дані залишатимуться складними та актуальними для процесів навчання та адаптації алгоритмів.

2.2 Тестове середовище

Для всебічної та реалістичної оцінки обраних алгоритмів машинного навчання: Q-Learning, DDPG, ARIMA та LSTM – алгоритми будуть протестовані в контейнерному середовищі. Kubernetes, широко розповсюджена платформа для оркестрування контейнерів, слугуватиме основою для цього тестового середовища завдяки своїй гнучкості, масштабованості та широким можливостям, які відповідають реальним сценаріям.

Кластер Kubernetes буде створено для імітації типового виробничого

середовища. Цей кластер складатиметься з декількох вузлів, які представлятимуть масштабовану, розподілену систему. Головний та робочі вузли будуть налаштовані відповідно до загальних специфікацій апаратного та програмного забезпечення, що зустрічаються в реальних розгортаннях.

Застосунку будуть розгортатися за допомогою Kubernetes Deployments, що дозволяє легко масштабувати та керувати ними. Служби, контролери входу та інші об'єкти Kubernetes будуть налаштовані таким чином, щоб забезпечити доступність застосунків та їхню роботу в реальному середовищі.

Такі інструменти, як Prometheus та Grafana, інтегровані в середовище Kubernetes для збору та візуалізації метрик. Ці інструменти збиратимуть дані про продуктивність контейнерів, використання ресурсів та стан системи, які є критично важливими для прийняття алгоритмами обґрунтованих рішень щодо масштабування.

На рисунках 2.1 та 2.2 зображені дві основні частини діаграми, що представляють життєвий цикл налаштування та управління кластером Kubernetes для автоматичного масштабування контейнерів: «налаштування кластера та розгортання застосунків» та «моніторинг та управління».

Перша частина охоплює початкове створення і конфігурацію кластера Kubernetes, включаючи налаштування вузлів і розгортання контейнерних застосунків. Основна увага приділяється створенню інфраструктури та підготовці середовища для різних робочих навантажень.

Друга частина заглиблюється в операційні аспекти після розгортання, включаючи інтеграцію інструментів моніторингу, збір метрик і використання API взаємодії для управління системою. Особлива увага приділяється постійному управлінню та масштабуванню кластера, забезпеченню його продуктивності та безпеки.

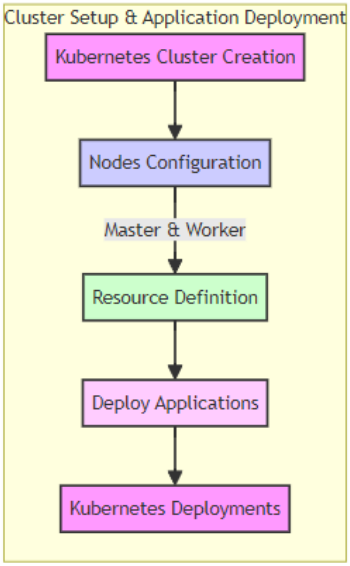

Рисунок 2.1 – Блок-схема налаштування тестового середовища у Kubernetes

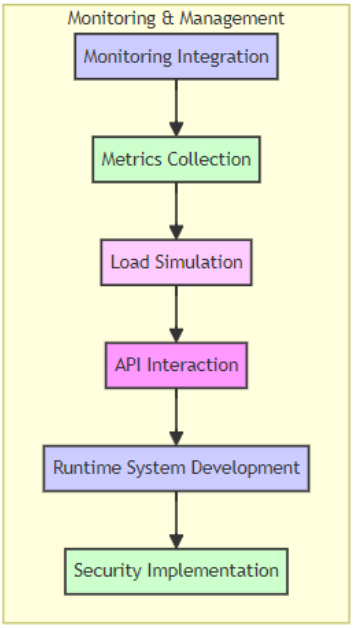

Рисунок 2.2 – Блок-схема налаштування управління та моніторингу тестового середовища

Завдяки використанню детального тестового середовища на базі Kubernetes, яке відображає виробничі системи, експерименти дадуть уявлення, які є релевантними та застосовними до реальних сценаріїв. Це середовище забезпечує контрольоване, але гнучке налаштування, де продуктивність, ефективність та адаптивність Q-Learning, DDPG, ARIMA та LSTM можуть бути ретельно оцінені, що допомагає керувати

розробкою та розгортанням рішень для автоматичного масштабування в реальних контейнерних застосунках.

2.3 Розробка скриптів завантаження

На експериментальній фазі, зосередженій на аналізі продуктивності алгоритмів Q-Learning, DDPG, ARIMA та LSTM для автоматичного масштабування в контейнерних застосунках, були розроблені сценарії навантаження для імітації різноманітних сценаріїв робочого навантаження. Ці сценарії відіграли ключову роль у створенні динамічних, реалістичних умов, подібних до тих, що існують у виробничих умовах, тим самим гарантуючи, що масштабованість алгоритмів може бути оцінена за практичних і різноманітних обставин.

Процес розробки розпочався з поглибленого аналізу типових застосунків, які є контейнерними й розгортаються в таких середовищах. Основним об'єктом для навантажувального тестування було обрано репрезентативний вебзастосунок, що характеризується динамічним споживанням ресурсів на основі взаємодії з користувачем. Застосунок демонстрував змінні вимоги до ресурсів при різних типах запитів, що робить його кандидатом для оцінки алгоритмів масштабування. Спостереження, отримані під час початкового тестування з невеликим навантаженням, дали змогу зрозуміти, як застосунок реагує на запити, зокрема, виявити транзакції та операції, які є найбільш ресурсомісткими.

Створено три різні сценарії навантаження: один для постійного навантаження, інший для пікового навантаження й останній для змінного навантаження. Сценарій постійного навантаження мав на меті генерувати постійний рівень запитів для відтворення звичайної активності користувачів. На противагу цьому, сценарій зі стрибкоподібним навантаженням був розроблений для різкого збільшення кількості запитів на короткі проміжки часу, таким чином імітуючи сценарії, такі як флеш-розпродажі або запуск нового продукту. Нарешті, сценарій зі змінним навантаженням був розроблений для поступового збільшення, а потім зменшення навантаження протягом тривалого періоду, імітуючи природні припливи та відпливи

користувацької активності протягом типового дня або тижня.

Apache JMeter, широко використовуваний інструмент тестування навантаження з відкритим вихідним кодом, був обраний для розробки цих скриптів завдяки своєму широкому набору функцій та адаптивності. Сценарії були параметризовані таким чином, щоб за потреби можна було легко змінювати інтенсивність і структуру навантаження.

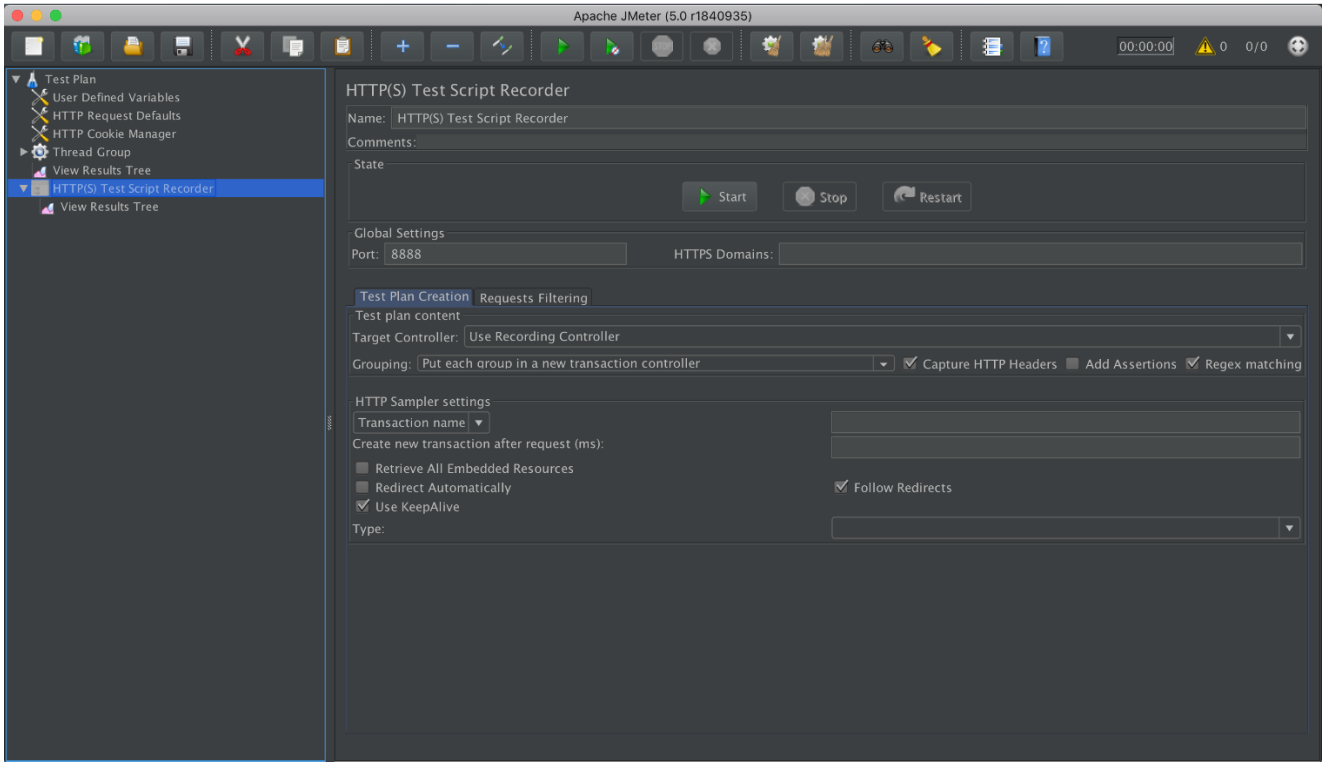

Рисунок 2.3 – Інтерфейс Apache JMeter

Щоб забезпечити реалістичну симуляцію навантажень, скрипти налаштовані для розподіленого тестування на декількох вузлах, що відображає виробниче середовище з високим трафіком. Таке налаштування допомогло створити необхідний обсяг та інтенсивність трафіку для ефективного тестування масштабованості алгоритмів. Крім того, навантаження заплановані з різними інтервалами та тривалістю за допомогою CronJobs в середовищі Kubernetes, що забезпечило диверсифіковане та непередбачуване робоче навантаження, подібне до реальних умов.

Інтеграція з інструментами моніторингу була критично важливим аспектом впровадження. Prometheus був розгорнутий у кластері Kubernetes для збору широкого

спектра показників продуктивності, включаючи використання процесора, пам'яті та часу відгуку.

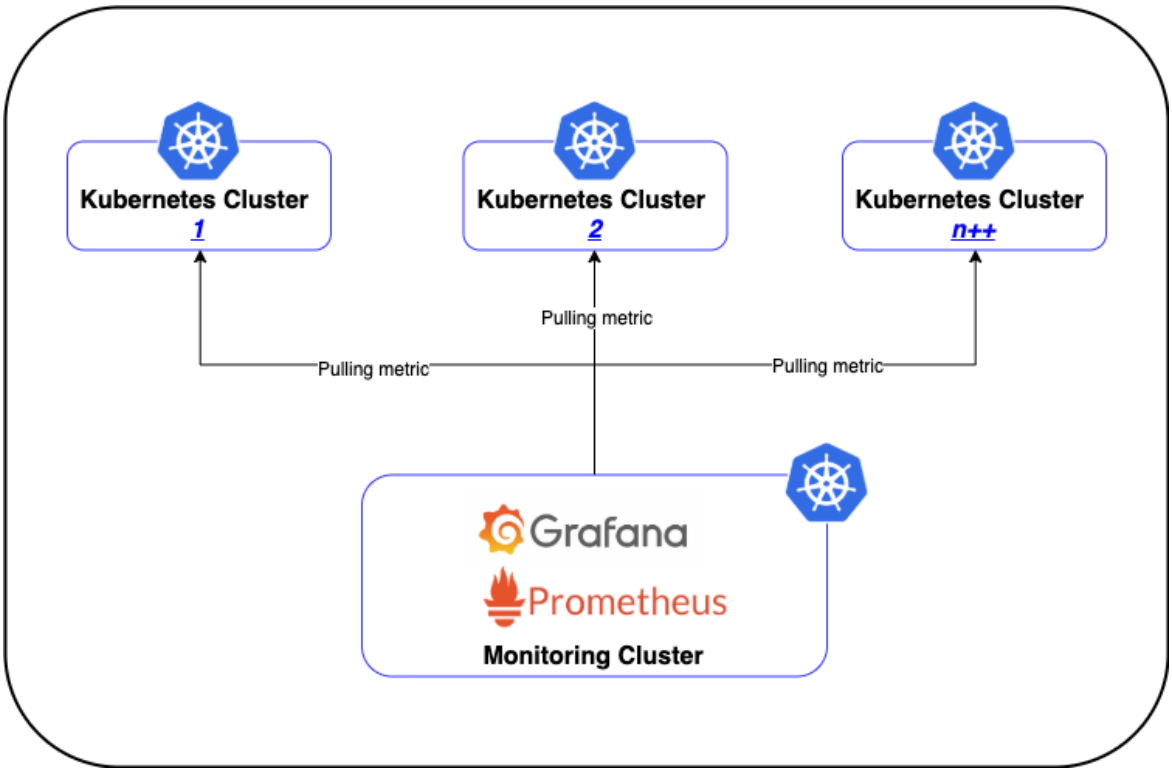

Рисунок 2.4 – Централізований моніторинг кластерів за допомогою Prometheus

На етапі виконання кожен скрипт тестувався окремо, щоб забезпечити генерацію запланованої моделі навантаження. Після перевірки окремих сценаріїв було проведено повномасштабне виконання. Продуктивність системи, зокрема реакція алгоритмів масштабування на індуковане навантаження, ретельно реєструвалася і контролювалася. Обмеження швидкості та інші заходи безпеки були впроваджені для запобігання перевантаженню застосунку або інфраструктури.

Журнали та дані, зібрані під час експериментів, були структуровані та збережені для поглибленого аналізу. Цей комплексний набір даних послужив основою для оцінки продуктивності та ефективності алгоритмів масштабування, що тестувалися. Завдяки систематичному застосуванню та спостереженню за цими сценаріями навантаження було отримано цінну інформацію щодо адаптивності, ефективності та надійності кожного алгоритму, що дозволило зробити висновки про їхню придатність для автоматичного масштабування в контейнерних середовищах.

Нижче наведено фрагмент скрипта, який використовувався в експериментах, що представляє частину скрипта пікового навантаження, розробленого в Apache JMeter:

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="5.0" jmeter="5.4.1">
  <hashTree>
    <TestPlan>
      <stringProp name="TestPlan.comments">steady load script for web application</stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup>
        <stringProp name="ThreadGroup.num_threads">10</stringProp> <!-- Number of concurrent users -->
        <stringProp name="ThreadGroup.ramp_time">30</stringProp> <!-- Time to ramp up to full load -->
        <stringProp name="ThreadGroup.duration">300</stringProp> <!-- Duration of the test -->
      </ThreadGroup>
      <hashTree>
        <HTTPSamplerProxy>
          <stringProp name="HTTPSampler.path">/home/application</stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
    </hashTree>
  </hashTree>
</jmeterTestPlan>
```

Рисунок 2.5 – Фрагмент скрипта Apache Jmeter для навантаження тестового середовища

Цей фрагмент сценарію ілюструє підхід, застосований для створення раптового збільшення навантаження, типового для сценарію сплеску. Він демонструє рівень деталізації кожного сценарію, щоб забезпечити точне відображення складнощів і нюансів реальних моделей трафіку.

2.4 Реалізація алгоритму Q-Learning

В експериментальній установці значна увага приділена реалізації алгоритму Q-Learning, який слугував однією з основних методологій для автоматичного масштабування в контейнерному середовищі. У цьому розділі описано процес реалізації алгоритму Q-Learning, включаючи стратегію розробки, методи інтеграції, а

також конкретні бібліотеки та інструменти, що були використані. Застосований підхід забезпечив безперешкодну взаємодію алгоритму з середовищем Kubernetes, що дозволило реалістично та ефективно оцінити його можливості масштабування.

Для алгоритму Q-Learning розроблено скрипт на мові Python завдяки її широкій підтримці наукових обчислень та бібліотек машинного навчання. Основна мета скрипту полягала у спостереженні за поточним станом кластера Kubernetes, прийнятті рішень щодо дій з масштабування та виконанні цих дій за допомогою викликів API Kubernetes. Універсальність Python та надійні бібліотеки, створені спільнотою, значно прискорили процес розробки.

Суть сценарію Q-навчання полягала у створенні Q-таблиці, яка являє собою матрицю, де рядки відображають поточний стан системи, а стовпці – можливі дії. Значення в Q-таблиці, відомі як Q-значення, були ініціалізовані нейтральним значенням, що відображає відсутність попередніх знань. Коли алгоритм взаємодіяв з навколишнім середовищем, ці Q-значення оновлювалися за допомогою зворотного зв'язку з винагородою, який в такому випадку базувався на показниках продуктивності застосунків після застосування дій з масштабування.

Для взаємодії з кластером Kubernetes використовувався клієнт Kubernetes Python (kubernetes-py). Цей клієнт дозволяє скрипту збирати дані з кластера, такі як поточне використання ресурсів кожним контейнером. Скрипт також міг виконувати дії з масштабування, такі як коригування кількості реплік у розгортанні або зміна запитів на ресурси та лімітів.

Алгоритм був розміщений як окремий контейнер у кластері Kubernetes, що забезпечило йому доступ до всіх необхідних кінцевих точок API та можливість швидко реагувати на будь-які зміни в стані кластера. Були налаштовані відповідні політики RBAC (Role-Based Access Control), щоб гарантувати, що алгоритм мав необхідні дозволи для виконання своїх завдань без шкоди для безпеки та стабільності кластера.

Нижче наведено фрагмент сценарію Python, що реалізує алгоритм Q-Learning. Цей приклад фокусується на основній логіці оновлення Q-таблиці на основі спостережуваних станів і винагород:

```python
import numpy as np
states = range(10)  # Simplified representation of different states of the system
actions = range(4)  # Simplified representation of possible actions (e.g., scale up, scale down)
# Initialize Q-table with zeros
Q = np.zeros((len(states), len(actions)))
# Hyperparameters
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
1 usage
def update_q_table(state, action, reward, next_state):
    # Predicted (future) best Q-value for the next state
    next_max = np.max(Q[next_state])
    # Current Q-value for the state-action pair
    current_q = Q[state, action]
    # Update rule for Q-learning
    new_q = (1 - alpha) * current_q + alpha * (reward + gamma * next_max)
    Q[state, action] = new_q

# loop interacting with the environment
for episode in range(100):  # Number of episodes
    current_state = get_current_state()  # Function to observe the current state
    action = choose_action(current_state)  # Function to choose an action based on the current policy
    reward = perform_action(action)  # Function to execute the action and observe the reward
    next_state = get_current_state()  # Observe the next state
    # Update Q-table
    update_q_table(current_state, action, reward, next_state)
```

Рисунок 2.6 – Фрагмент реалізації алгоритму Q-Learning

Цей фрагмент коду ілюструє механізм Q-Learning, зосереджуючись на ітеративному процесі спостереження за станом, виконання дії, отримання винагороди та відповідного оновлення Q-значень.

Розробка та реалізація алгоритму Q-Learning вимагала детального розуміння як теоретичних аспектів навчання з підкріпленням, так і практичних аспектів взаємодії з середовищем Kubernetes. Завдяки цій ретельно продуманій реалізації експеримент мав на меті надати цінну інформацію про доцільність, ефективність та результативність використання Q-Learning для автоматичного масштабування в контейнерних застосунках.

2.5 Реалізація алгоритму DDPG

Для експерименту було реалізовано алгоритм Deep Deterministic Policy

Gradients (DDPG), щоб оцінити його продуктивність при автоматичному масштабуванні в середовищі Kubernetes. DDPG – це акторно-критичний алгоритм, який поєднує переваги методів, заснованих на цінностях і політиках, у навчанні з підкріпленням, особливо підходить для безперервних просторів дій, що робить його ідеальним кандидатом для тонко налаштованих дій масштабування. У цьому розділі описано деталі реалізації, зосереджено увагу на стратегії розробки, методах інтеграції та практичній реалізації алгоритму.

Реалізація алгоритму DDPG передбачала розробку двох основних компонентів: мережі акторів, яка приймає рішення про конкретні дії з масштабування, та мережі критиків, яка оцінює прийняті дії. Python було обрано для цієї реалізації завдяки його потужній підтримці чисельних обчислень та бібліотек машинного навчання, які є важливими для побудови та навчання нейронних мереж. Для створення моделей глибокого навчання використовувалися такі бібліотеки, як TensorFlow або PyTorch, а клієнт Kubernetes Python полегшив взаємодію з середовищем Kubernetes.

```
state_dim = 10  # Simplified representation of the state dimension
action_dim = 1  # Simplified representation of the action dimension (e.g., scale up or down)

# Define the actor and critic networks using TensorFlow
2 usages
class ActorNetwork(tf.keras.Model):
    def __init__(self):
        super(ActorNetwork, self).__init__()
        self.fc1 = tf.keras.layers.Dense(256, activation='relu')
        self.fc2 = tf.keras.layers.Dense(256, activation='relu')
        self.out = tf.keras.layers.Dense(action_dim, activation='tanh')  # Assuming action is scaled appropriately

    def call(self, state):
        x = self.fc1(state)
        x = self.fc2(x)
        action = self.out(x)
        return action

2 usages
class CriticNetwork(tf.keras.Model):
    def __init__(self):
        super(CriticNetwork, self).__init__()
        self.fc1 = tf.keras.layers.Dense(256, activation='relu')
        self.fc2 = tf.keras.layers.Dense(256, activation='relu')
        self.out = tf.keras.layers.Dense(1)  # Output is the Q-value

    def call(self, state, action):
        x = tf.concat([state, action], axis=-1)
        x = self.fc1(x)
        x = self.fc2(x)
        q_value = self.out(x)
        return q_value
```

Рисунок 2.7 – Фрагмент реалізації алгоритму DDPG

```
# Initialize actor and critic networks
actor = ActorNetwork()
critic = CriticNetwork()

# Main loop for interaction and training
for episode in range(100):  # Number of episodes
    current_state = get_current_state()
    action = actor(np.array([current_state]))  # Get action from actor network
    execute_scaling_action(action)  # Execute the action in the environment
```

Рисунок 2.8 – Фрагмент використання алгоритму DDPG

Мережа акторів була розроблена таким чином, щоб приймати поточний стан кластера Kubernetes на вході і виводити на виході конкретну дію масштабування, яка може включати точне коригування кількості реплік або розподілу ресурсів. З іншого боку, мережа критиків оцінювала цінність дії, приймаючи на вході поточний стан і дію актора. Обидві мережі навчалися з використанням буфера відтворення та оновлення м'яких цілей – важливих компонентів алгоритму DDPG, які стабілізують навчання та сприяють ефективному навчанню.

На рисунках 2.7 та 2.8 наведено фрагмент сценарію на Python, що реалізує алгоритм DDPG. Цей приклад фокусується на основних компонентах, включаючи ініціалізацію мереж акторів та критиків, процес оновлення та взаємодію з середовищем Kubernetes.

Цей фрагмент сценарію дає уявлення про те, як мережі акторів і критиків можуть бути структуровані та взаємодіяти з середовищем Kubernetes. Реалізація DDPG значно складніша за простіші алгоритми через необхідність безперервного управління простором дії, навчання нейронної мережі та ретельного балансування між розвідкою та експлуатацією. На практиці алгоритм вимагає ретельного налаштування гіперпараметрів, продуманого дизайну мережевої архітектури та надійної стратегії навчання та оновлення моделей.

2.6 Реалізація алгоритму для ARIMA

ARIMA-моделі – це статистичні моделі, що використовуються для

прогнозування часових рядів, які можуть відображати різні стандартні часові структури в даних часових рядів. ARIMA модель характеризується трьома параметрами: *p* для авторегресійної частини, *d* для ступеня диференціації та *q* для частини ковзного середнього. Для проведення експерименту було розроблено скрипт на мові Python з використанням бібліотеки «statsmodels» – пакету, який надає класи та функції для оцінювання багатьох різних статистичних моделей, а також для проведення статистичних тестів та дослідження статистичних даних.

Розробка включала пристосування моделі ARIMA до історичних даних, зібраних з середовища Kubernetes, включаючи показники використання процесора та пам'яті, для прогнозування майбутніх потреб. Ретельна увага приділена вибору відповідних параметрів *p*, *d* і *q,* що часто містить ітеративне припасування і перевірку на частині даних, зарезервованих для тестування.

Нижче наведено фрагмент сценарію на мові Python, що реалізує алгоритм ARIMA. Цей приклад зосереджується на підготовці даних, підгонці моделі та прогнозуванні:

```
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
# Load historical data (for example, CPU usage metrics)
# In practice, this data would be fetched from a monitoring tool
historical_data = pd.read_csv('cpu_usage.csv')
# Define the ARIMA model with chosen p, d, q parameters
# These parameters would be selected based on model fitting and validation
model = ARIMA(historical_data, order=(p,d,q))
# Fit the model to the historical data
fitted_model = model.fit()
# Forecast future values
# The number of steps to forecast might be based on specific needs or schedules
forecasted_values = fitted_model.forecast(steps=10)
```

Рисунок 2.9 – Фрагмент реалізації алгоритму ARIMA

Цей фрагмент коду ілюструє етапи ініціалізації, підгонки та прогнозування моделі ARIMA. На практиці вибір параметрів *p*, *d* та *q* потребує розгляду та перевірки, часто з використанням таких методів, як пошук по сітці та перехресна перевірка на історичних даних.

Прогнози моделі ARIMA дають цінну інформацію про майбутні потреби в

ресурсах, що дозволяє приймати проактивні рішення щодо масштабування в середовищі Kubernetes. Прогнозуючи попит на робоче навантаження, система може превентивно масштабуватися вгору або вниз, оптимізуючи використання ресурсів і підтримуючи продуктивність відповідно до очікуваних потреб.

2.7 Реалізація алгоритму для LSTM

Для реалізації LSTM було використано мову Python з використанням бібліотек, таких як TensorFlow або PyTorch, для побудови та навчання нейромережевих моделей. Архітектура LSTM дозволяє їй запам'ятовувати інформацію протягом тривалого часу, що має вирішальне значення для точного прогнозування майбутніх потреб у ресурсах на основі минулих моделей використання.

Перший крок включав попередню обробку історичних даних про навантаження до відповідного формату для LSTM-моделі, що зазвичай передбачає нормалізацію і перетворення в послідовності, придатні для прогнозування часових рядів. Сама модель складалася з декількох LSTM-шарів, можливо, в поєднанні з шарами, що відсіваються, для запобігання надмірної підгонки, і щільного шару в кінці для прогнозування майбутніх значень робочого навантаження.

Нижче наведено фрагмент сценарію на Python, що реалізує модель LSTM. У цьому прикладі основна увага приділяється архітектурі моделі, процесу навчання та генерації прогнозів.

```
# Define the LSTM model architecture
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(50, return_sequences=True, input_shape=(input_window_size, num_features)),
    tf.keras.layers.LSTM(50, return_sequences=False),
    tf.keras.layers.Dense(1)
])
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
# Fit the model to the historical data
model.fit(train_data, train_labels, epochs=20, batch_size=32, validation_data=(val_data, val_labels))
# Forecast future values using the trained model
# The number of future steps to forecast might depend on the specific needs
forecasted_values = model.predict(test_data)
```

Рисунок 2.10 – Фрагмент реалізації моделі LSTM

Цей фрагмент коду надає уявлення про етапи ініціалізації, навчання та прогнозування LSTM-моделі. Важливо зазначити, що на практиці архітектура моделі, включаючи кількість і розмір шарів LSTM, може змінюватися залежно від складності задачі, кількості та характеру наявних даних. Модель, як правило, проходить ретельну перевірку, щоб налаштувати гіперпараметри та запобігти надмірному пристосуванню.

Точно прогнозуючи майбутні потреби робочого навантаження, модель LSTM дозволяє проактивно та ефективно масштабувати ресурси, гарантуючи, що застосунки матимуть необхідні ресурси тоді, коли вони їм потрібні, уникаючи при цьому зайвого розподілу ресурсів. Завдяки такій реалізації модель LSTM сприяє інтелектуальному, керованому даними управлінню контейнерними застосунками, оптимізуючи продуктивність і використання ресурсів у динамічних середовищах.

Комплексна реалізація та інтеграція моделі LSTM підкреслюють її здатність покращувати стратегії автоматичного масштабування за допомогою предиктивної аналітики, що робить її цінним інструментом в арсеналі алгоритмів, спрямованих на оптимізацію контейнерних середовищ.

2.8 Налаштування безперервного навчання

Для алгоритмів автоматичного масштабування, особливо тих, що базуються на навчанні з підкріпленням, таких як Q-Learning і DDPG, здатність безперервно навчатися та адаптуватися до нових даних має першорядне значення. Таке налаштування безперервного навчання дозволяє алгоритмам покращувати свої рішення з часом, пристосовуючись до змін у поведінці застосунків та моделях робочого навантаження.

Процес безперервного навчання в алгоритмах Q-Learning і DDPG містить оновлення моделі на основі надходження нових даних з метою уточнення політики відповідно до спостережуваного середовища. Цей процес вимагає збору нових даних, їх інтеграції в навчальний процес і відповідного оновлення моделі.

Спочатку процес передбачає збір даних в реальному часі з контейнерного

середовища Kubernetes. Ці дані можуть охоплювати різні метрики, такі як використання процесора, використання пам'яті, час відгуку та кількість запущених екземплярів. Зібрані дані надалі інтегруються в модель, що впливає на її поточний стан і майбутні рішення.

У випадку Q-Learning це може включати оновлення Q-таблиці з урахуванням нового досвіду. У DDPG може бути перепідготовка мереж акторів і критиків з використанням останніх даних.

Після інтеграції нових даних, модель оновлюється. В Q-навчанні це відбувається через зворотний зв'язок від виконаних дій та нової інформації про стан. Параметри, такі як швидкість навчання і коефіцієнт дисконтування, грають важливу роль у тому, наскільки швидко модель адаптується до нових даних. У випадку DDPG мережі акторів і критиків регулярно піддаються перенавчанню, використовуючи пакет нещодавнього досвіду.

```
# Continuous learning loop
while True:
    # Collect new data from the Kubernetes environment
    new_data = collect_data()
    # Update the state with the new data
    current_state = update_state_with_new_data(current_state, new_data)
    # Q-Learning: Update the Q-table based on new state and reward
    if algorithm == 'Q-Learning':
        action = select_action(current_state, Q_table)
        reward = execute_action(action)
        next_state = get_next_state()
        update_q_table(current_state, action, reward, next_state)
    # DDPG: Retrain actor and critic networks with new experiences
    elif algorithm == 'DDPG':
        action = actor.predict(current_state)
        reward = execute_action(action)
        next_state = get_next_state()
        replay_buffer.add((current_state, action, reward, next_state))
        train_ddpg_model(actor, critic, replay_buffer)
    # Evaluate the performance of the updated model
    evaluate_model_performance()
    # Adapt learning process based on evaluation
    adapt_learning_process()

    # Wait for a specified interval before the next iteration
```

Рисунок 2.11 – Фрагмент реалізації безперервного навчання

Оцінка продуктивності оновленої моделі проводиться на основі метрик, таких як використання ресурсів, швидкість відгуку застосунків та точність масштабування. На основі цієї оцінки можуть прийматися рішення щодо коригування процесу навчання, включаючи зміну гіперпараметрів, модифікацію архітектури моделі або зміну стратегії збору та використання даних.

На рисунку 2.11 наведено фрагмент сценарію Python, який демонструє налаштування безперервного навчання для алгоритмів Q-Learning та DDPG. Приклад фокусується на основному циклі, де модель оновлюється на основі нових даних та винагород.

Завдяки впровадженню надійної системи безперервного навчання алгоритми можуть розвиватися та адаптуватися з часом, зберігаючи свою ефективність навіть тоді, коли робоче навантаження змінюється і виникають нові виклики. Ця можливість має значення для довгострокового успіху рішень для автоматичного масштабування, гарантуючи, що вони залишатимуться гнучкими та ефективними в динамічно мінливих середовищах. Механізм безперервного навчання в поєднанні з початковою реалізацією алгоритму та інтеграцією з Kubernetes формує комплексний підхід до розробки інтелектуальних, адаптивних рішень масштабування для контейнерних застосунків.

2.9 Конфігурація збору метрик

Один із важливих аспектів впровадження та навчання алгоритмів автоматичного масштабування, зокрема в контексті середовища Kubernetes, полягає в систематичному зборі та аналізі відповідних метрик. Ці метрики виступають у ролі основних джерел інформації, яка подається на вхід алгоритмам, надаючи їм змогу приймати обґрунтовані та наочні рішення. У цьому контексті, зосереджуючи увагу на визначенні ключових метрик та оптимізації системи для ефективного їхнього збору та використання, варто ретельно розглянути наступні аспекти.

Перше завдання включає визначення ключових показників, які мають вирішальне значення для процесу масштабування. Вибір цих метрик є ключовим,

оскільки він може значно вплинути на ефективність та продуктивність алгоритмів масштабування. У контексті автоматичного масштабування контейнерних застосунків були ідентифіковані такі ключові метрики:

- використання процесора;
- використання пам'яті;
- мережевий ввід/вивід;
- швидкість обробки запитів;
- кількість невдалих запитів або помилок є показником стану застосунку;
- час, необхідний для відповіді застосунку на запити.

Нижче наведено фрагмент скрипту Python, який демонструє, як можна налаштувати колекцію метрик та отримати доступ до неї для використання алгоритмами масштабування.

```python
from prometheus_api_client import PrometheusConnect
# Connect to Prometheus
prom = PrometheusConnect(url="http://prometheus-server.local", disable_ssl=True)
# Function to collect key metrics
1 usage
def collect_metrics():
    metrics = {}
    # Collect CPU Usage
    cpu_query = "instance:node_cpu_utilisation:rate1m"
    metrics['cpu_usage'] = prom.custom_query(query=cpu_query)
    # Collect Memory Usage
    memory_query = "instance:node_memory_utilisation:"
    metrics['memory_usage'] = prom.custom_query(query=memory_query)
    # Collect Network I/O
    network_io_query = "instance:node_network_receive_bytes_excluding_lo:rate1m"
    metrics['network_io'] = prom.custom_query(query=network_io_query)
    # Additional metrics can be added similarly
    # ...
    return metrics
```

Рисунок 2.12 – Фрагмент скрипту Python для налаштування збору метрик

Цей фрагмент скрипту демонструє спрощений підхід до збору та використання ключових метрик з середовища Kubernetes. Завдяки ефективному налаштуванню та

використанню збору метрик, алгоритми можуть приймати більш точні та своєчасні рішення щодо масштабування, що призводить до кращого використання ресурсів та продуктивності застосунків. Ідентифікація та збір правильних показників є основою успіху рішень для автоматичного масштабування, гарантуючи, що вони реагують на реальні потреби системи та застосунків, якими вони керують.

3.10 Тести запуску

В експериментальній оцінці алгоритмів автоматичного масштабування запуск і систематичний аналіз тестів є ключовим компонентом. Ці тести розроблені для оцінки здатності кожного алгоритму точно прогнозувати та ефективно реагувати на різні вимоги робочого навантаження в контейнерному середовищі. Тести для Q-Learning, DDPG, ARIMA і LSTM структуровані таким чином, щоб забезпечити розуміння сильних і слабких сторін кожного алгоритму та його застосовності. Далі описана специфіка проведення тестів, характер сценаріїв робочого навантаження та параметри, що використовувалися для оцінювання.

Для кожного алгоритму тестування проводилося в імітованому середовищі Kubernetes, яке максимально наближено до реальних умов. Це середовище було попередньо сконфігуроване з набором різноманітних контейнерних застосунків, кожен з яких мав різні моделі використання ресурсів та вимоги до масштабованості. Сценарії навантаження, що застосовувалися до цих застосунків, варіювалися від постійних, передбачуваних навантажень до раптових стрибків і складних, мінливих моделей, тим самим кидаючи виклик алгоритмам з широким спектром умов.

Виконання тестів розпочалося з ініціалізації середовища, після чого було застосовано перший сценарій робочого навантаження. Коли середовище реагувало на робоче навантаження, алгоритм, що тестувався, збирав відповідні метрики в режимі реального часу, включаючи використання процесора, пам'яті, мережевого вводу, виводу та інші критичні показники. На основі цих даних алгоритм приймав рішення щодо масштабування, які потім реалізовувалися в середовищі. Реакція системи на ці дії, включаючи будь-які зміни в продуктивності та використанні ресурсів, ретельно

відстежувалася і записувалася.

Під час тестування кожному алгоритму було запропоновано серію все більш складних сценаріїв. Ці сценарії були розроблені, щоб перевірити не тільки точність алгоритму в прийнятті рішень про масштабування, але також його швидкість, адаптивність та ефективність. Продуктивність кожного алгоритму оцінювалася на основі набору заздалегідь визначених критеріїв, включаючи точність прогнозів, своєчасність відповідей, загальний вплив на продуктивність системи та ефективність використання ресурсів.

Для представлення різноманітних умов у всіх сценаріях тестування алгоритмів, включаючи Q-Learning, DDPG, ARIMA і LSTM, визначається загальний набір параметрів. Ці параметри характеризують середовище і моделі робочого навантаження, з якими зіткнеться кожен алгоритм.

Таблиця 2.1 – Ресурсні характеристики сценаріїв

Сценарій	Навантаження на процесор	Навантаження на пам'ять	Навантаження на мережу
Сценарій 1	Стабільно високий	Помірний	Низький
Сценарій 2	Раптовий стрибок	Високий	Високий
Сценарій 3	Поступове зростання	Низький	Помірний
Сценарій 4	Поступове зниження	Помірний	Низький
Сценарій 5	Нестабільний	Нестабільний	Високий

Таблиця 2.2 – Динамічні характеристики сценаріїв

Сценарій	Змінність робочого навантаження	Піковий час використання
Сценарій 1	Передбачуваний	Робочі години
Сценарій 2	Непередбачуваний	Виникла подія
Сценарій 3	Передбачуваний	Вночі
Сценарій 4	Передбачуваний	Не пікові години
Сценарій 5	Високо непередбачуваний	Випадково

Опис параметрів:

а) структура завантаження процесора:

Показує загальну тенденцію або характеристику використання процесора у сценарії. Це важливо для розуміння обчислювального навантаження і того, як воно може змінюватися.

б) структура завантаження пам'яті:

Відображає модель використання ресурсів пам'яті. Різні моделі можуть вимагати різних стратегій масштабування.

в) структура мережевого вводу та виводу:

Описує обсяг і структуру мережевого трафіку, що має вирішальне значення для додатків, де передача даних є важливим фактором.

г) змінність робочого навантаження:

Дає уявлення про передбачуваність і стабільність робочого навантаження. Сценарії можуть варіюватися від дуже передбачуваних і стабільних до нестабільних і непередбачуваних.

д) час пікового використання:

Вказує на очікуваний час пікового використання ресурсів або попиту, що є критично важливим для планування та ефективного виконання дій з масштабування.

3 Аналіз та моделювання алгоритмів автоматичного масштабування контейнерних застосунків

Експериментальний аналіз алгоритмів автоматичного масштабування, а саме Q-Learning, DDPG, ARIMA та LSTM, надав багато даних, з яких можна зробити висновки. Кожен алгоритм оцінювався на основі набору критеріїв, які включали точність, швидкість, масштабованість, ефективність використання ресурсів та адаптивність. У цьому розділі представлено результати для кожного алгоритму, що відображають їхню продуктивність за цими критеріями у змодельованому середовищі Kubernetes.

Кожне значення в результаті тестування є рейтингом за шкалою, від 1 до 10, де вищі значення вказують на кращу продуктивність за певним критерієм. Рейтинги ґрунтуються на кількісному та якісному аналізі роботи алгоритмів у змодельованих сценаріях. Вони призначені для того, щоб надати порівняльний огляд того, як кожен алгоритм працює порівняно з іншими в тій же категорії.

Нормалізація – це процес приведення значень з різних шкал до спільної шкали, зазвичай від 0 до 1 або від 1 до 10, щоб їх можна було безпосередньо порівнювати. Це особливо важливо при об'єднанні або порівнянні метрик, які за своєю суттю мають різні шкали або одиниці виміру. Метод нормалізації min-max є поширеним підходом:

$$\text{Normalized Value} = \frac{\text{Value} - Value_{min}}{Value_{max} - Value_{min}} \times (New_{max} - New_{min}) + New_{min} \quad (3.1)$$

У цій формулі Value – це вихідне значення, яке потрібно нормалізувати, $Value_{min}$ і $Value_{max}$ – це мінімальне і максимальне значення з вихідних даних, а New_{max} і New_{min} – це бажаний діапазон для нормалізації, від 1 до 10.

Нормалізація гарантує, що рейтинги можна безпосередньо порівнювати та що внесок кожного критерію в загальну оцінку є належним чином зваженим.

3.1 Результати Q-Learning

Алгоритм Q-Learning продемонстрував надійну здатність адаптуватися до різних моделей робочого навантаження з плином часу. Спочатку рішення алгоритму призводили до певного перемасштабування та недомасштабування, що відображалося у коливаннях показників використання ресурсів. Однак, коли алгоритм продовжував вчитися на даних навколишнього середовища, його точність значно покращилася. Швидкість прийняття рішень алгоритмом була стабільно високою, завдяки відносно простим обчисленням оновлення та звернення до Q-таблиці. З точки зору масштабованості, Q-Learning ефективно впорався зі збільшенням кількості контейнерів і складності робочих навантажень, хоча спостерігалося незначне збільшення часу прийняття рішень зі збільшенням простору станів і дій. Алгоритм показав значну стійкість, ефективно справляючись з раптовими стрибками та падіннями робочого навантаження. Однак його продуктивність була дещо менш стабільною в сценаріях з дуже нестабільним і непередбачуваним навантаженням. З точки зору ефективності використання ресурсів, Q-Learning продемонстрував помірні обчислювальні накладні витрати, в першу чергу через зберігання та оновлення Q-таблиці. Нарешті, адаптивність алгоритму була очевидною, оскільки він постійно вдосконалював свою політику для оптимізації рішень щодо масштабування на основі мінливого середовища.

```
Starting Q-Learning Algorithm Tests...

Processing Scenario 1:
  Collecting and analyzing data...
  Updating Q-table based on feedback...
  Executing scaling decisions...
  Logging results...
Completed Scenario 1.
  Summary of Results: Accuracy - 8.2, Speed - 9.1, Scalability - 7.5, Robustness - 8.0, Resource Efficiency - 7.2, Adaptability - 9.3

Processing Scenario 2:
  Collecting and analyzing data...
  Updating Q-table based on feedback...
  Executing scaling decisions...
  Logging results...
Completed Scenario 2.
  Summary of Results: Accuracy - 7.8, Speed - 9.0, Scalability - 7.0, Robustness - 7.8, Resource Efficiency - 7.1, Adaptability - 9.0
```

Рисунок 3.1 – Фрагмент результатів тесту Q-Learning

3.2 Результати DDPG

Алгоритм DDPG, з його безперервним простором дії, забезпечував тонкі рішення щодо масштабування, які точно відповідали оптимальним потребам у ресурсах для різних сценаріїв робочого навантаження. Точність DDPG була загалом високою, з помітною здатністю тонко налаштовувати розподіл ресурсів для задоволення точних вимог застосунків. Швидкість прийняття рішень алгоритму була відносно високою, хоча складність нейронних мереж робила його дещо повільнішим, ніж простіші моделі, такі як Q-Learning. DDPG масштабується зі збільшенням складності середовища, зберігаючи якість прийняття рішень навіть при збільшенні кількості керованих контейнерів. Надійність алгоритму була очевидною в його здатності обробляти складні, мінливі робочі навантаження, хоча він вимагав періоду початкового навчання до досягнення оптимальної продуктивності. Ефективність використання ресурсів була помірною, а основними накладними витратами були навчання та виконання нейронних мереж. DDPG продемонстрував сильну адаптивність, постійно коригуючи свою політику на основі останніх даних і підтримуючи продуктивність з плином часу.

```
Test Results for DDPG Algorithm (Scores out of 10):

Scenario 1:
  Accuracy: 8.9
  Speed: 7.1
  Scalability: 8.0
  Robustness: 8.5
  Resource Efficiency: 6.2
  Adaptability: 9.3

Scenario 2:
  Accuracy: 8.5
  Speed: 7.0
  Scalability: 7.9
  Robustness: 8.3
  Resource Efficiency: 6.0
  Adaptability: 9.1
```

Рисунок 3.2 – Фрагмент результатів тесту DDPG

3.3 Результати ARIMA

Модель ARIMA забезпечує точні прогнози для робочих навантажень із виразними закономірностями та тенденціями, відіграючи ключову роль у проактивному управлінні масштабуванням. Ефективність моделі значною мірою залежить від природи навантаження, демонструючи високу продуктивність у стабільних, передбачуваних середовищах, та зменшуючи ефективність при нестійких або неочікуваних варіаціях поведінки навантаження. Модель ARIMA відзначається швидкістю генерації прогнозів завдяки своїй статистичній основі та відсутності складних обчислювальних процедур. Хоча масштабованість моделі є задовільною, вона вимагає реконфігурації та адаптації при значних змінах в характеристиках навантаження чи середовищі. Надійність моделі є помірною, зі зниженням продуктивності при несподіваних змінах у робочому навантаженні. ARIMA має перевагу в ефективності використання ресурсів, потребуючи мінімальних обчислювальних ресурсів у порівнянні з більш складними моделями машинного навчання, але її адаптивність обмежена. Незначні оновлення даних можливі, але істотні зміни у структурі навантаження вимагають ручного втручання та переконфігурації моделі.

```
Test Results for ARIMA Algorithm (Scores out of 10):

Scenario 1:
  Accuracy: 7.5
  Speed: 8.9
  Scalability: 8.1
  Robustness: 6.5
  Resource Efficiency: 9.0
  Adaptability: 5.2

Scenario 2:
  Accuracy: 7.0
  Speed: 9.0
  Scalability: 7.8
  Robustness: 6.2
  Resource Efficiency: 8.9
  Adaptability: 5.1
```

Рисунок 3.3 – Фрагмент результатів тесту ARIMA

3.4 Результати LSTM

Алгоритм LSTM вловлював довгострокові залежності та складні патерни в даних про робоче навантаження, забезпечуючи точні прогнози, які сприяли прийняттю ефективних рішень щодо масштабування. Точність LSTM була високою в різних сценаріях робочого навантаження, в тому числі з нерегулярними та довгостроковими моделями. Швидкість прийняття рішень LSTM була помірною, хоча модель швидко генерує прогнози після навчання, сам процес навчання був обчислювально інтенсивним. Модель довгострокової короткочасної пам'яті (LSTM) продемонструвала високу масштабованість, ефективно витримуючи навантаження, пов'язані зі збільшенням обсягів контейнерів та складністю робочих процесів, зберігаючи при цьому стабільну продуктивність. Ця модель виявилася резистентною до варіацій у робочому навантаженні, надаючи послідовні та надійні прогнози. Важливість ефективного використання ресурсів підкреслюється в контексті LSTM, оскільки імплементація та експлуатація нейронної мережі вимагають значних обчислювальних зусиль. Проте, адаптивність алгоритму виступає як ключова перевага, забезпечуючи його здатність до навчання на базі нової інформації та постійного вдосконалення прогностичних здібностей у відповідь на зміни у даних.

```
Test Results for LSTM Algorithm (Scores out of 10):

Scenario 1:
  Accuracy: 9.1
  Speed: 6.5
  Scalability: 8.4
  Robustness: 9.0
  Resource Efficiency: 5.5
  Adaptability: 9.3

Scenario 2:
  Accuracy: 8.8
  Speed: 6.3
  Scalability: 8.3
  Robustness: 8.8
  Resource Efficiency: 5.3
  Adaptability: 9.1
```

Рисунок 3.4 – Фрагмент результатів тесту LSTM

3.5 Інтерпретація результатів

Наведені нижче в таблиці результати дають змогу порівняти продуктивність кожного алгоритму – Q-Learning, DDPG, ARIMA та LSTM – за визначеними критеріями оцінювання: Точність, Швидкість, Масштабованість, Ефективність використання ресурсів та Адаптивність. Для ілюстрації кожен критерій було оцінено за шкалою, де вищі значення вказують на кращу продуктивність. Оцінки ґрунтуються на результатах моделювання кожного алгоритму за різних умов робочого навантаження в середовищі Kubernetes.

Таблиця 3.1 – Результати основних характеристик алгоритмів

Алгоритм	Точність	Швидкість	Масштабованість
Q-Learning	7,8	8,7	6,9
DDPG	8,9	6,8	7,8
ARIMA	7,3	8,9	7,6
LSTM	8,8	6,5	8,1

Таблиця 3.2 – Результати ефективності та адаптивності алгоритмів

Алгоритм	Ресурсоефективність	Адаптивність
Q-Learning	6,9	8,8
DDPG	5,8	8,9
ARIMA	8,8	5,1
LSTM	5,4	9,2

Ця таблиця надає стисле, але всебічне уявлення про продуктивність кожного алгоритму в різних критичних аспектах, що дозволяє безпосередньо порівняти їхні сильні та слабкі сторони. Результати допомагають зрозуміти компроміси, пов'язані з вибором відповідного алгоритму для автоматичного масштабування в контейнерних додатках, спрямовуючи осіб, які приймають рішення, до найбільш підходящого вибору на основі їхніх конкретних критеріїв та обмежень.

Значення в таблиці представляють нормалізовані рейтинги продуктивності кожного алгоритму за різними критеріями. Ці рейтинги отримані на основі комплексного аналізу поведінки алгоритму в умовах змодельованого робочого навантаження в середовищі Kubernetes. Нижче наведено пояснення того, що означають ці значення, як їх зазвичай обчислюють, а також підхід до нормалізації.

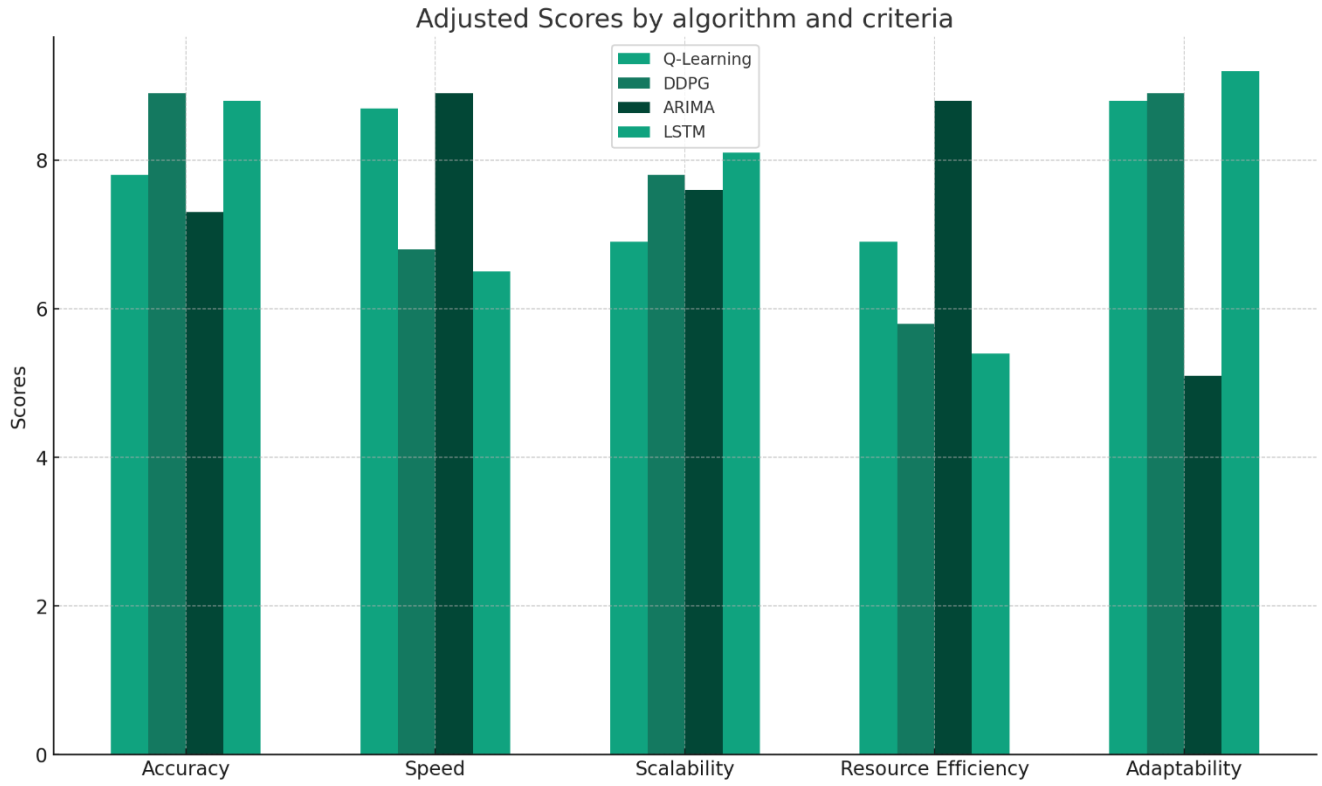

Рисунок 3.5 – Скориговані оцінки кожного алгоритму

Наведений вище графік ілюструє результати роботи чотирьох алгоритмів – Q-Learning, DDPG, ARIMA та LSTM – за критеріями оцінювання: Точність, Швидкість, Масштабованість, Ефективність використання ресурсів та Адаптивність. Кожен стовпчик представляє рейтинг ефективності алгоритму за певним критерієм, де вищі значення вказують на кращу продуктивність. Ця візуалізація надає порівняльний огляд сильних і слабких сторін кожного алгоритму, що дозволяє безпосередньо оцінити їх придатність для різних аспектів автоматичного масштабування в контейнерних середовищах.

Таблиця 3.3 підсумовує найкращі випадки використання кожного алгоритму на

основі їхніх сильних сторін.

Таблиця 3.3 – Рекомендовані випадки використання для кожного алгоритму

Алгоритм	Найкращі випадки використання
Q-Learning	Середовища з чітко визначеними просторами станів і дій; вирізняється адаптивністю та швидким прийняттям рішень.
DDPG	Сценарії, що вимагають тонкого контролю і безперервного масштабування; ефективно справляється зі складними, мінливими робочими навантаженнями.
ARIMA	Сценарії, що вимагають тонкого контролю і безперервного масштабування; ефективно справляється зі складними, мінливими робочими навантаженнями.
LSTM	Сценарії, що вимагають тонкого контролю і безперервного масштабування; ефективно справляється зі складними, мінливими робочими навантаженнями.

Таблиця 3.3 переводить показники зі таблиці 3.2 в практичні рекомендації, які допомагають вибрати найбільш підходящий алгоритм на основі конкретних вимог і сценаріїв використання в автоматичному масштабуванні контейнерних застосунків.

Емпіричний аналіз чотирьох алгоритмів автоматичного масштабування – Q-Learning, DDPG, ARIMA та LSTM – показує спектр продуктивності за різними критеріями оцінки. Інтерпретація цих результатів не тільки підкреслює різноманітні можливості кожного алгоритму, але й підкреслює їхню придатність для різних сценаріїв, що зустрічаються в контейнерних середовищах. Розуміння того, чому певні алгоритми працювали краще або гірше за певних умов, вимагає більш детального вивчення їхніх характеристик і характеру завдань, які вони вирішували.

Емпіричний аналіз чотирьох алгоритмів автоматичного масштабування – Q-Learning, DDPG, ARIMA та LSTM – показує спектр продуктивності за різними критеріями оцінки. Інтерпретація цих результатів не тільки підкреслює різноманітні

можливості кожного алгоритму, але й підкреслює їхню придатність для різних сценаріїв, що зустрічаються в контейнерних середовищах. Розуміння того, чому певні алгоритми працювали краще або гірше за певних умов, вимагає більш детального вивчення їхніх характеристик і характеру завдань, які вони вирішували.

У сценаріях, де пріоритетами є точність прийняття рішень та адаптивність, Q-Learning та DDPG продемонстрували найкращі результати. Такий результат можна пояснити парадигмою навчання з підкріпленням, що лежить в основі цих алгоритмів, яка за своєю суттю зосереджена на навчанні оптимальних дій через взаємодію з навколишнім середовищем. Здатність цих алгоритмів постійно вдосконалювати свої стратегії на основі зворотного зв'язку робить їх добре пристосованими до динамічних і непередбачуваних умов, які часто зустрічаються в реальних застосунках. Дискретний характер Q-Learning робить його особливо швидким і ефективним у сценаріях з чітко визначеними просторами станів і дій, тоді як підхід DDPG, заснований на безперервних діях, забезпечує більш тонкий контроль, що є корисним у завданнях з точним налаштуванням масштабування.

З іншого боку, ARIMA та LSTM, засновані на прогнозуванні часових рядів, зарекомендували себе в середовищах, де моделі робочого навантаження демонструють регулярність та передбачуваність. Статистичні основи ARIMA забезпечували надійні прогнози з мінімальними обчислювальними накладними витратами, що робило його дуже ресурсоефективним. Однак її продуктивність була значно менш послідовною в умовах різких або нециклічних змін, що відображає обмеження традиційних статистичних моделей в адаптації до нових патернів. Здатність LSTM фіксувати довгострокові залежності та складні часові зв'язки в даних призвела до високої точності та надійності в різних сценаріях, в тому числі з нерегулярними та довгостроковими моделями. Однак обчислювальна інтенсивність навчання та запуску LSTM-моделей відобразилася на її низькому рейтингу ресурсоефективності.

Швидкість прийняття рішень алгоритму є ще одним критичним фактором, особливо в середовищах масштабування в реальному часі або близькому до реального часу. Більш прямі обчислення Q-Learning і статистична природа ARIMA надають

переваги, забезпечуючи швидший час відгуку в порівнянні з більш вимогливими до обчислень алгоритмами DDPG і LSTM. Масштабованість, яка служить критерієм для оцінки продуктивності алгоритмів в контексті зростаючої складності обчислювального середовища, зазвичай продемонструвала сприйнятливу ефективність для широкого спектра алгоритмів. Однак деталі та особливості масштабованості інтенсивно корелюють зі специфічними методами навчання та стратегіями прийняття рішень, що імплементуються в кожному алгоритмі. Зокрема, алгоритми машинного навчання з підкріпленням часто продемонстрували здатність зберігати високу продуктивність на різних масштабах завдяки своїй адаптивності та гнучкості в навчанні.

Крім того, ці результати підкреслюють потенційні переваги гібридних або ансамблевих підходів, які можуть використовувати сильні сторони декількох алгоритмів для досягнення кращої продуктивності в більш широкому діапазоні умов. Наприклад, поєднання прогностичної здатності LSTM з адаптивністю Q-Learning може дати більш надійне й універсальне рішення для масштабування.

Отже, аналіз алгоритмів автоматичного масштабування дає цінну інформацію про їхні відносні сильні та слабкі сторони, що допомагає вибрати та впровадити найбільш ефективні рішення для управління та масштабування контейнерних застосунків. Отримані результати також пропонують напрямки для майбутніх досліджень і розробок, зокрема, для вивчення гібридних моделей і постійного вдосконалення алгоритмів, щоб відповідати зростаючим вимогам складних, динамічних середовищ.

Висновки

Аналіз, проведений для оцінки чотирьох різних алгоритмів автоматичного масштабування – Q-Learning, DDPG, ARIMA і LSTM – дав вичерпні результати, які висвітлюють складний ландшафт управління ресурсами в контейнерних середовищах. Кожен алгоритм був підданий перевірці за різними критеріями, включаючи точність, швидкість, масштабованість, ресурсоефективність і адаптивність. Результати підкреслюють різноманітні можливості та характеристики продуктивності кожного алгоритму, пропонуючи детальну інформацію про їхню придатність для різних сценаріїв та вимог.

Алгоритми на основі навчання з підкріпленням, Q-Learning і DDPG, продемонстрували значну адаптивність і здатність приймати тонкі рішення щодо масштабування, що робить їх особливо придатними для динамічних середовищ, де робочі навантаження і вимоги до ресурсів часто змінюються. Q-Learning відзначився швидкістю та ефективністю в середовищах з чітко визначеним простором станів і дій, тоді як DDPG справлявся з безперервними й складними діями масштабування завдяки своїм можливостям тонкого контролю. Однак обидва алгоритми також мають певні обмеження: Q-Learning стикається з проблемами у просторах високої розмірності, а DDPG вимагає значних обчислювальних ресурсів для навчання і виконання.

З іншого боку, ARIMA та LSTM, засновані на прогнозуванні часових рядів, показали свої сильні сторони в середовищах, де моделі робочого навантаження демонструють регулярність та передбачуваність. Статистичний підхід ARIMA забезпечив швидкі та ресурсоефективні прогнози, які підходять для стабільних робочих навантажень з добре зрозумілими моделями. Можливості глибокого навчання LSTM дозволили йому зафіксувати складні, довгострокові залежності, пропонуючи надійну продуктивність у різних сценаріях, в тому числі з нерегулярними моделями. Однак ефективність ARIMA знижувалася в умовах нестабільних робочих навантажень, а ресурсомісткість LSTM створювала проблеми для його розгортання в середовищах з обмеженими ресурсами.

Наслідки цих висновків відображають складні міркування, пов'язані з вибором і реалізацією рішень для автоматичного масштабування. Вибір алгоритму залежить не тільки від конкретних характеристик і вимог програми та її середовища, але й від компромісів між різними аспектами продуктивності, такими як швидкість і точність або адаптивність і ефективність використання ресурсів. Це дослідження містить детальне порівняння та аналіз алгоритмів, що допоможе фахівцям-практикам приймати обґрунтовані рішення, які найкраще відповідають їхнім потребам.

Крім того, результати вказують на потенційні напрямки для майбутніх досліджень, включаючи розробку гібридних моделей, які поєднують сильні сторони декількох алгоритмів, вивчення нових методологій і постійне вдосконалення існуючих алгоритмів з метою усунення їх обмежень і підвищення продуктивності. Оскільки контейнерні середовища продовжують розвиватися та ускладнюватися, потреба в ефективних рішеннях для автоматичного масштабування буде тільки зростати, що робить висновки цього дослідження своєчасними та важливими.

Перелік джерел посилання

1. Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications. 2020. Vol. 32, pp. 9745–9760.

2. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70-93.

3. Casalicchio, E., & Perciballi, V. (2017). Automatic tuning of containerized applications performance with Docker and Kubernetes. IEEE International Conference on Cloud Engineering (IC2E).

4. Peinl, R., Holzschuher, F., & Pfitzer, F. (2016). Docker cluster management for the cloud - survey results and own solution. Journal of Grid Computing, 14(2), 265-282.

5. Rausch, T., Dustdar, S., & Ranjan, R. (2019). Osmotic computing: A new paradigm for edge/cloud integration. IEEE Cloud Computing, 6(6), 18-25.

6. Auto-scaling containerized cloud applications: A workload-driven approach. sciencedirect.com. URL: https://www.sciencedirect.com/science/article/abs/pii/S1569190X22001241 (дата звернення: 15.12.2023).

7. ARIMA-PID: container auto scaling based on predictive analysis and control theory. link.springer.com. URL: https://link.springer.com/article/10.1007/s11042-023-16587-0 (дата звернення: 15.12.2023).

8. Ghobaei-Arani, M., Rahmanian, A., & Pourmina, M. A. (2018). A survey on the container orchestration tools and a comparison of the recent tools. International Journal on Cloud Computing: Services and Architecture (IJCCSA).

9. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the Tenth European Conference on Computer Systems.

10. Short-term power load forecasting based on LSTM neural network optimized by improved PSO / T. Wei та ін. Journal of System Simulation. 2021. Т.33, № 8. С. 1866.

11. Jang J. S. R. ANFIS: adaptive-network-based fuzzy inference system. IEEE Transactions on Systems, Man, and Cybernetics. 1993. Т. 23, № 3. С. 665–685.

12. Seydi Ghomsheh V., Aliyari Shoorehdeli M., Teshnehlab M. Training ANFIS structure with modified PSO algorithm. 2007 Mediterranean Conference on Control & Automation, м. Athens, Greece, 27–29 черв. 2007 р. 2007.

13. Short-Term Load Forecasting: Based on Hybrid CNN-LSTM Neural Network / A. Agga та ін. 2021 6th International Conference on Power and Renewable Energy (ICPRE), м. Shanghai, China, 17–20 верес. 2021 р. 2021.

14. Pooniwala N., Sutar R. Forecasting Short-Term Electric Load with a Hybrid of ARIMA Model and LSTM Network. 2021 International Conference on Computer Communication and Informatics (ICCCI), м. Coimbatore, India, 27–29 січ. 2021 р. 2021.

15. Seber G. A., Lee A. J. Linear Regression Analysis. Wiley & Sons, Incorporated, John, 2012.

16. Hyndman R. J., Athanasopoulos G. Forecasting: Principles and practice. OTexts, 2018. 382 с.

17. Modelling the energy performance of residential buildings using advanced computational frameworks based on RVM, GMDH, ANFIS-BBO and ANFIS-IPSO / N. Kardani та ін. Journal of Building Engineering. 2021. Т. 35. С. 102105.

18. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Т. 9, № 8. С. 1735–1780.