МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Кваліфікаційна наукова
праця на правах
рукопису

ХОМЕНЧУК ВЛАДИСЛАВ ОЛЕГОВИЧ

УДК 004.67

ДИСЕРТАЦІЯ

МЕТОДИКА ПОБУДОВИ АДАПТИВНИХ КЛАСТЕРІВ КОМП'ЮТЕРНИХ СИСТЕМ ДЛЯ ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ З ПРИСТРОЇВ ІОТ

Спеціальність 123 «Комп'ютерна інженерія»
Галузь знань 12 «Інформаційні технології»
Подається на здобуття наукового ступеня
доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ В.О. Хоменчук

Науковий керівник: СТОРЧАК Каміла Павлівна, доктор технічних наук, професор

Електронний підпис
ХОМЕНЧУК
ВЛАДИСЛАВ
ОЛЕГОВИЧ
Ідентифікаційний
код
3485714550
Підписано у Вчасно

Київ – 2023

АНОТАЦІЯ

Хоменчук В.О. Методика побудови адаптивних кластерів комп’ютерних систем для збору та обробки інформації з пристроїв IoT. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії в галузі знань 12 – Інформаційні технології, спеціальність 123 – Комп'ютерна інженерія. – Державний університет інформаційно-комунікаційних технологій. – Київ, 2023.

Дисертаційна робота присвячена розробці методики побудови адаптивних кластерів комп’ютерних систем для збору та обробки інформації з пристроїв IoT.

Формування передових підходів до побудови та функціонування кластерів комп’ютерних систем, та розвиток на їх основі розподілених баз даних привело до виникнення та розвитку методів розподілення навантаження між елементами кластеру. Системи Інтернету речей (IoT) зазвичай збирають великі обсяги даних в реальному часі від сенсорів та пристроїв. Використання баз даних часових рядів (TSDB) може бути дуже ефективним для зберігання та аналізу цих даних. TSDB спеціально розроблені для ефективного зберігання та опрацювання великих обсягів часових рядів даних, що робить їх ідеальним вибором для систем IoT, де дані часто представлені у вигляді часових рядів. Багато TSDB оптимізовані для швидкого читання та запису даних, що важливо для систем, які працюють в реальному часі, таких як системи IoT. TSDB можуть надавати можливості для виконання агрегацій та опрацювання даних безпосередньо в базі даних, що зменшує навантаження на додаткові аналітичні інструменти.

Зростаюча популярність IoT за останні роки різко зростає та широко застосовується в різних галузях. З кожним роком кількість підключених пристроїв продовжує зростати. Від домашніх пристроїв до промислових систем, IoT надає можливість зберігати, обробляти та обмінюватися даними.

Зниження вартості сенсорів та їх покращення дозволяють вбудовувати їх у різні пристрої та об'єкти, що раніше не були підключені до Інтернету. Зростання швидкості та доступності бездротових мереж, таких як 5G, сприяє покращенню комунікації між підключеними пристроями. Впровадження IoT в промисловості призводить до оптимізації виробничих процесів, підвищення ефективності та зменшення витрат. Хмарні технології надають можливість зберігання та обробки великих обсягів даних з IoT, забезпечуючи доступ до них з будь-якого місця. Різко зростаюча популярність IoT призводить до стрімкого збільшення запитів до бази даних, що створює проблему нерівномірного навантаження.

Проблема нерівномірного навантаження в кластері розподіленої бази даних виникає, коли різні частини бази даних або вузли кластера отримують нерівномірну кількість запитів. Це може мати ряд наслідків, які впливають на ефективність та продуктивність системи. Основні аспекти цієї проблеми включають:

- Вузли або сервери в кластері можуть бути перевантажені внаслідок великої кількості запитів або завдань, тоді як інші можуть залишатися непоміченими. Це може впливати на швидкість виконання запитів та час відповіді.
- Нерівномірне навантаження може викликати неефективне використання ресурсів кластера, таких як обчислювальна потужність, пам'ять та мережеві ресурси. Деякі вузли можуть просто працювати на повну потужність, тоді як інші можуть залишатися невикористаними.

Дисертаційна роботи розкриває сутність розподілення навантаження в кластері розподіленої бази даних часових рядів та спрямована на вирішення задачі перенавантаження вузлів в кластері бази даних часових рядів. Для цього необхідно вирішити наступні наукові завдання:

1) Провести аналіз вимог до систем Інтернету речей та обробки даних.

2) Провести аналіз кластерних систем для створення розподіленої бази даних для систем IoT.

3) Розробити методику навчання нейронної мережі для розподілу навантаження баз даних часових рядів.

4) Розробити алгоритм роботи агенту бази даних часових рядів для маршрутизації запитів.

5) Розробити програмну модель нейронної мережі для аналізу навантаження на базу даних часових рядів.

В дисертаційній роботі описано методику побудови нейронної мережі для адаптивного розподілу навантаження в кластері розподіленої бази даних часових рядів. Отриманий показник точності для валідаційної вибірки нейронної мережі складає 86%. Модель з використанням нейронної мережі продемонструвала підвищення ефективності на 20% у порівняні з класичним методом розподілу навантаження. Отримані наступні наукові результати:

1) Розроблена методика навчання нейронної мережі для розподілу навантаження для баз даних часових рядів, що дозволяє забезпечити рівномірне навантаження на вузли кластера.

2) Розроблений алгоритм роботи агента бази даних часових рядів для забезпечення маршрутизації запитів.

3) Розроблена програмна модель нейронної мережі для аналізу навантаження на базу даних часових рядів, що дало змогу збільшити ефективність роботи кластера у 1,2 рази.

Розроблено стратегію розподілу запитів до бази даних часових рядів на основі рекурентної нейронної мережі. Для спрощення навчання рекурентної нейронної мережі, було запропоновано метод оптимізації структури вхідних даних за допомоги розподілу часових рядів у відокремлені hash bucket.

Ключові слова: IoT, Інтернет речей, Internet of Things, TSDB, граничні обчислення, бази даних часових рядів, бази даних, розподілені бази даних, розподілене зберігання даних, СУБД, нейронна мережа, tensorflow, neural network, машинне навчання, штучний інтелект, адаптивність.

ABSTRACT

Khomenchuk V.O. Methodology for Constructing Adaptive Clusters of Computer Systems for Collection and Processing of IoT Device Data. – Qualifying scientific work as a manuscript.

Dissertation for the degree of Doctor of Philosophy in the field of knowledge 12 – Information Technologies, specialization 123 – Computer Engineering. – State University of Information and Communication Technologies. – Kyiv, 2023.

The dissertation work is dedicated to the development of a methodology for constructing adaptive clusters of computer systems for the collection and processing of data from IoT devices. The advancement in the construction and functioning of computer system clusters, as well as the development of distributed databases based on them, has led to the emergence and evolution of methods for load distribution among cluster elements. Internet of Things (IoT) systems typically gather large volumes of real-time data from sensors and devices. The use of Time Series Databases (TSDB) can be highly effective for storing and analyzing this data. TSDBs are specifically designed for efficient storage and processing of large volumes of time series data, making them an ideal choice for IoT systems where data is often represented as time series. Many TSDBs are optimized for fast reading and writing of data, which is crucial for real-time systems like IoT. TSDBs can provide capabilities for performing aggregations and data processing directly in the database, reducing the load on additional analytical tools.

The growing popularity of IoT in recent years has led to a sharp increase in its widespread use in various fields. With each passing year, the number of connected devices continues to rise. From home devices to industrial systems, IoT enables the storage, processing, and exchange of data. The decrease in sensor costs and their improvement allows embedding them in various devices and objects that were not previously connected to the Internet. The increasing speed and availability of wireless networks, such as 5G, contribute to improved communication between

connected devices. The implementation of IoT in industry leads to the optimization of production processes, increased efficiency, and cost reduction. Cloud technologies provide the ability to store and process large volumes of IoT data, providing access to them from anywhere. The surging popularity of IoT results in a rapid increase in database queries, creating a problem of uneven load distribution.

The problem of uneven load distribution in a distributed database cluster arises when different parts of the database or cluster nodes receive an unequal number of queries. This can have several consequences affecting the efficiency and performance of the system. Key aspects of this problem include:

- Nodes or servers in the cluster may be overloaded due to a large number of queries or tasks, while others may remain unnoticed. This can affect the speed of query execution and response time.

- Uneven load distribution can lead to inefficient use of cluster resources, such as computational power, memory, and network resources. Some nodes may operate at full capacity, while others may remain unused.

The main task of the dissertation is to develop a methodology for constructing a separate agent for load distribution in a time series distributed database cluster. To achieve this, the following steps are necessary:

1) Analyze cluster requirements for Internet of Things (IoT) systems and data processing.

2) Analyze cluster systems for creating a distributed database for IoT systems.

3) Develop a neural network training methodology for load distribution in time series database clusters.

4) Develop an algorithm for the operation of the time series database agent to route queries.

5) Develop a software model of a neural network for analyzing the load on time series databases.

The dissertation describes the methodology for constructing a neural network for adaptive load distribution in a cluster of distributed time series databases. The accuracy index obtained for the neural network's validation sample is 86%. The

model, using the neural network, demonstrated a 20% increase in efficiency compared to the classical load distribution method. The following scientific results were obtained:

1) A training methodology for a neural network for load distribution for time series databases has been developed, ensuring even load distribution across cluster nodes.

2) An algorithm for the operation of the time series database agent to ensure query routing has been developed.

3) A software model of a neural network for analyzing the load on time series databases has been developed, resulting in a 1.2 increase in cluster efficiency.

A query distribution strategy for time series databases based on a recurrent neural network has been developed. To simplify the training of the recurrent neural network, an optimization method for the structure of input data has been proposed, distributing time series into separate hash buckets.

Keywords: IoT, Internet of Things, TSDB, Edge Computing, Time Series Databases, Databases, Distributed Databases, Distributed Data Storage, DBMS, Neural Network, TensorFlow, Machine Learning, Artificial Intelligence, Neural network, Machine learning, Artificial Intelligence, adaptability.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

1. Хоменчук В. Аналіз впливу хмарних обчислень і граничних обчислень на продуктивність мережі / В. Хоменчук // Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу, Київ, 28 листоп. 2023 р. – С. 361-362.

2. Хоменчук В. Використання баз даних часових рядів в архітектурі граничних обчислень. / В. Хоменчук // Зв’язок. – 2023. – № 1. – С. 39–42.

3. Хоменчук В. Переваги використання apache iotdb для адаптивних систем IoT / В. Хоменчук // Сучасний стан та перспективи розвитку IoT, Київ, 9 квіт. 2021 р. – С. 94-95.

4. Хоменчук В. Переваги та недоліки алгоритмів розподілу даних в кластері баз даних / В. Хоменчук // Сучасний стан та перспективи розвитку IoT, Київ, 7 квіт. 2023 р. – С. 95-97.

5. Хоменчук В. Стратегій розподілу навантаження децентралізованих баз даних / В. Хоменчук // Зв’язок. – 2023. – № 5. – С. 21–32.

6. Шефкін Б. В., Красюк І. В., Хоменчук В. О. та ін. Дослідження та впровадження нейронної мережі на основі TENSORFLOW // Зв’язок. – 2020. – № 6. – С. 18–20.

7. Свердлюк Б. І., Каграманова Ю. К., Хоменчук В. О. та ін. Керування давачами Розумного будинку за допомогою голосового помічника Google Assistant // Зв’язок. – 2021. – № 3. – С. 54–57.

ЗМІСТ

ВСТУП 11
РОЗДІЛ 1 ДОСЛІДЖЕННЯ МЕТОДІВ ПОБУДОВИ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ 15
1.1. Огляд технології Інтернету речей 15
1.2. Вимоги до систем Інтернету речей. 23
1.3. Аналіз даних IoT за допомоги штучного інтелекту 26
Висновки до розділу 1 37
РОЗДІЛ 2 ДОСЛІДЖЕННЯ МЕТОДІВ КЛАСТЕРІЗАЦІЇ ТА МАСШТАБУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ 38
2.1. Кластеризація серверних ресурсів 38
2.2. Архітектура побудови кластерних систем 43
2.3. Розподілена архітектура Інтернет речей 50
2.4. Управління ресурсами систем Інтернету речей 52
2.5. Використання баз даних часових рядів в архітектурі граничних обчислень для систем IoT 56
Висновки до розділу 2 62
РОЗДІЛ 3 РОЗРОБКА МЕТОДИКИ РОЗПОДІЛУ ЗБЕРІГАННЯ ДАНИХ В РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ 64
3.1. Алгоритми розподілу навантаження баз даних 64
3.2. Постановка задачі 74
3.3. Визначення типу нейронної мережі 78
3.4. Структура RNN та надбудови LSTM 89
3.5. Агент для бази даних часових рядів 102
3.6. Оптимізація структури вхідних даних за допомоги hash bucket 115
Висновок до розділу 3 116
ВИСНОВКИ 120
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 123
Додаток А 140
Додаток Б 141
Додаток В 146
Додаток Г 150

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

DDBMS - distributed database management system. Система управління розподіленими базами даних.

DTW - dynamic time warping. Динамічне вирівнювання часу.

GRU - gated recurrent units. Вентильні рекурентні блоки.

IoT - internet of things. Інтернет речей.

ITU - International Telecommunication Union. Міжнародний телекомунікаційний союз.

LSTM - long short-term memory. Довготермінова короткочасна пам'ять.

MQTT - message queuing telemetry transport. Транспорт для телеметрії чергових повідомлень.

NSPF - no single point of failure. Відсутність єдиного точкового відмови.

RFID - radio frequency identification. Ідентифікація за допомогою радіочастот.

RNN - recurrent neural network. Рекурентна нейронна мережа.

SMP - симетрична багатопроцесорність. Симетрична багатопроцесорність.

SQL - structured query language. Мова структурованих запитань.

TSDB - time series database. База даних часових рядів.

URLLC - ultra-reliable low latency communication. Надзвичайно надійний зв'язок з низькою затримкою.

МН - машинне навчання.

СУБД - система управління базами даних.

ШІ - штучний інтелект.

ВСТУП

Актуальність теми.

Актуальність адаптивних кластерів комп’ютерних систем баз даних часових рядів для систем Інтернету речей (IoT) зумовлене динамікою збільшення кількості підключених пристроїв та величезним обсягом даних, що генеруються цими пристроями. В сучасному світі зростає інтеграція IoT у різні галузі, що призводить до необхідності ефективного управління цими даними для забезпечення оптимальної продуктивності та функціональності систем.

Пристрої IoT є важливим джерелом постійного потоку часових рядів, що може суттєво збільшувати обсяг інформації, яку необхідно зберігати та обробляти. У цьому контексті, шардинг баз даних виявляється ефективним рішенням для розподілу цих даних між різними серверами або кластерами, забезпечуючи тим самим можливість розподілення робочого навантаження та збільшення продуктивності системи.

Потреба у масштабованості є ще однією важливою обставиною, оскільки системи IoT можуть зазнавати швидкого росту кількості підключених пристроїв. Адаптивний розподіл дозволяє динамічно додавати нові часові ряди в кластер, забезпечуючи системі можливість адаптації до змін у масштабах та забезпечення стабільної роботи навіть при інтенсивному трафіку даних.

Таким чином, адаптивний розподіл навантаження баз даних часових рядів в системах Інтернету речей виявляється необхідною стратегією для забезпечення ефективного управління обсягами даних, оптимізації продуктивності та забезпечення надійності систем у зростаючому і динамічному середовищі Інтернету речей.

Однієї з поточних проблем Інтернету речей є нестабільний характер обсягів даних, які генеруються, оскільки вони можуть значно змінюватися в залежності від конкретного застосування та контексту використання. У зв'язку з цим, ефективне управління цими даними вимагає гнучкості та здатності

адаптуватися до змін у робочому навантаженні. Використання адаптивного розподілу навантаження баз даних часових рядів дозволяє розглядати дані в маленьких, самостійних частинах, що полегшує масштабованість та забезпечує відмінну адаптивність систем IoT до змін навантаження.

Адаптивний розподіл навантаження баз даних часових рядів дозволяє ефективно розділити запити для обробки та збереження даних, забезпечуючи ефективне використання ресурсів та прискорення відповіді на запити. Отже, з точки зору оптимізації продуктивності та забезпечення швидкодії систем, використання адаптивного розподілу навантаження баз даних часових рядів має значущий вплив на ці параметри.

Використання машинного навчання (МН) є ключовим елементом для підвищення ефективності та функціональності систем шардингу баз даних часових рядів в контексті Інтернету речей (IoT). Введення алгоритмів МН дозволяє системі автоматично адаптуватися до змін та оптимізувати розподіл даних та прогнозувати майбутні обсяги навантаження.

Однією з ключових областей використання МН є прогнозування обсягів даних та динаміки їх змін. Навчані моделі можуть аналізувати історичні дані часових рядів, враховуючи патерни та тенденції, і прогнозувати майбутні обсяги навантаження. Це дозволяє системі розподілу адаптуватися до змін, розподіляючи ресурси відповідно до прогнозованих потреб.

В заключення, адаптивний розподіл навантаження баз даних часових рядів в системах Інтернету речей стає ключовим елементом для забезпечення ефективного та стійкого управління даними, оптимізації робочого процесу, та підтримки високої продуктивності в умовах постійних змін та зростаючого обсягу інформації. Використання машинного навчання в контексті розподілу навантаження баз даних часових рядів для систем Інтернету речей покращує адаптивність та ефективність системи, роблячи її більш гнучкою та здатною пристосовуватися до навантаження яке динамічно змінюється.

Зв'язок роботи з науковими програмами, планами, темами

Тематика дисертаційної роботи і отримані результати безпосередньо відповідають пріоритетності розвитку інформаційних та комунікаційних технологій в Україні до 2020 р. згідно із Законом України «Про пріоритетні напрями розвитку науки і техніки», від 11.07.2001 № 2623-III, зі змінами внесеними згідно із Законом України «Про наукову та науково-технічну діяльність» від 26.11.2015 № 848-VIII. Дисертаційна робота виконана відповідно до планів наукової і науково-технічної діяльності Державного університету інформаційно-комунікаційних технологій.

Мета і завдання дослідження.

Метою даної дисертації є оптимізація процесів розподілу навантаження в кластері бази даних часових рядів.

Наукове завдання полягає в необхідності створення нейронної мережі для ефективного розподілу навантаження в кластері розподіленої бази даних часових рядів. Для цього необхідно вирішити наступні наукові завдання:

1) Провести аналіз кластерних систем для створення розподіленої бази даних для систем IoT.

2) Розробити методику навчання нейронної мережі для розподілу навантаження баз даних часових рядів.

3) Розробити алгоритм роботи агенту бази даних часових рядів для маршрутизації запитів.

4) Розробити програмну модель нейронної мережі для аналізу навантаження на базу даних часових рядів.

Об'єктом дослідження є розподілені системи IoT.

Предметом дослідження є процес розподілу навантаження в кластері бази даних часових рядів.

Методи дослідження. Результати проведених і представлених в дисертації досліджень отримані з використанням методів системного аналізу, теорії інформації, нейромережного моделювання, оптимізації, формалізації та методів динамічного програмування.

Наукова новизна одержаних результатів полягає у наступному:

Основні результати, що мають наукову новизну та отримані у ході вирішення завдань, поставлених у дисертаційному дослідженні, полягають у наступному:

1) Розроблена методика навчання нейронної мережі для розподілу навантаження для баз даних часових рядів, що дозволяє забезпечити рівномірне навантаження на вузли кластера.

2) Розроблений алгоритм роботи агенту бази даних часових рядів для забезпечення маршрутизації запитів.

3) Розроблена програмна модель нейронної мережі для аналізу навантаження на базу даних часових рядів, що дало змогу збільшити ефективність роботи кластера у 1,2 рази.

Практичне значення отриманих результатів. Отримані результати дисертаційної роботи нададуть змогу покращити ефективність аналітики та прогнозування на основі даних IoT. Це може бути корисним в таких областях, як підтримка прийняття рішень на основі даних датчиків IoT, передбачення витрат ресурсів та оптимізація роботи баз даних часових рядів.

Результати дисертаційних досліджень впроваджені у виробничий процес ТОВ «ХУАВЕЙ Україна», ПрАТ «Київстар» (додаток В) та в навчальному процесі Державний університет інформаційно-комунікаційних технологій (Київ) (додаток Г).

РОЗДІЛ 1

ДОСЛІДЖЕННЯ МЕТОДІВ ПОБУДОВИ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ

1.1. Огляд технології Інтернету речей

Інтернет речей (Internet of Things) - це термін, який відноситься до вбудованого підключення до Інтернету будь-яких нетипових фізичних обчислювальних пристроїв, у яких вбудовані сенсори та програмне забезпечення, які можуть збирати, обробляти та передавати дані по бездротовим мережам. Існує багато реалізацій та протоколів мережевого зв'язку для підключення, інтеграції та взаємодії з розумними пристроями, але швидкий ріст попиту та використання пристроїв Інтернету речей змусив багато галузей покращити свою архітектуру систем та мереж з метою підтримки швидкого та надійного зв'язку між пристроями протягом останнього року. Телекомунікаційні компанії роблять великі кроки в напрямку інтеграції та забезпечення зв'язку між розумними пристроями (Інтернету речей), особливо з новими сервісними категоріями URLLC [63, 87] (надзвичайно надійний зв'язок з низькою затримкою) та eMB (покращене мобільне широкосмугове з'єднання) мережі 5G. Нова архітектура мережі 5G, яка включає малими сотами, міліметровими хвилями та гігабітними швидкостями передачі вгору і вниз, дозволяє швидше і надійніше встановлювати з'єднання та передачу даних для обробки в реальному часі.

Завдяки значним перевагам, які пропонують застосунки Інтернету речей, промисловість, науково-дослідні лабораторії та уряди з усього світу інвестують великі кошти в технології Інтернету речей.

Це призвело до фрагментації і сильної конкуренції на ринку Інтернету речей, який страждає від несумісності через наявність кількох стандартів. Міжнародний телекомунікаційний союз (ITU-T) визнав різні виклики, з якими

стикаються мережі Інтернету речей у порівнянні з традиційними мережами. ITU-T запропонував нову референцну модель для середовища Інтернету речей [32]. Вона складається з чотирьох рівнів: рівня пристроїв, рівня мережі, рівня підтримки сервісів та рівня підтримки застосувань. Також враховуються можливості управління та безпеки. Оскільки можливості управління та безпеки застосовуються до всіх чотирьох рівнів, вони представлені вертикально в референцній моделі ITU [67].

Нова парадигма підключених розумних пристроїв створює ряд дослідницьких викликів, пов'язаних переважно з взаємодією в різнорідних мережах, комунікацією між машинами, відкритою структурою для Інтернету речей, масштабним розгортанням інфраструктури, динамічним, автономним та адаптивним управлінням ресурсами, можливістю розгортання та розподіленою енергоефективною обробкою даних, механізмами захисту від атак, схемами контролю доступу та обліку для Інтернету речей.

Традиційні архітектури комп'ютерних мереж та референцні моделі можуть бути непридатними для вирішення цих викликів і проблем. Багато дослідників пропонували збільшити кількість рівнів порівняно з TCP/IP, але жоден з них не є стандартизованим у середовищі Інтернету речей. Традиційна трьохрівнева архітектура [109] має рівень додатків, мережі та сенсорів, тоді як у п'ятирівневій архітектурі додаються рівні бізнесу та обробки (рис. 1.1.).

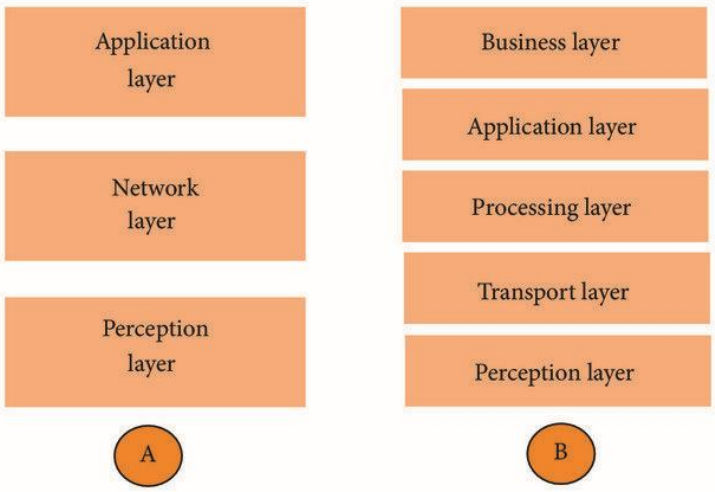

Рисунок 1.1. Трьохрівнева та п'ятирівнева архітектура систем IoT

Трьохрівнева архітектура систем IoT

- Рівень сенсорний - рівень також називається фізичним рівнем, його основна функціональність полягає у зборі даних, інформації та визнанні використання даних у фізичному світі. Тут всі виконавчі пристрої працюють відповідно до інформації, яка збирається датчиками різних об'єктів для виконання конкретних операцій відповідними об'єктами за допомогою 2D штрих-кодових міток та зчитачів RFID-міток, камер, GPS, датчиків, терміналів та мережі датчиків.
- Мережевий рівень – рівень виконує функцію інтерфейсу між рівнем застосування та рівнем сприйняття. Його функціональність полягає в обробці даних та інформації, отриманих з рівня сприйняття, а також в передачі даних та підключенні пристроїв у мережі. Він працює подібно до нервової системи, де включає мережу передачу сигналів або пакетів та Інтернет-мережу, центр управління мережею, інформаційний центр та центр інтелектуальної обробки тощо.
- Рівень додатків - це рівень, на якому фактично розгортаються різні додатки з використанням послуг Інтернету речей. Він надає персоналізовані послуги відповідно до потреб користувача, також створює зв'язок між користувачами та застосунками, що поєднує галузь для надання високорівневих рішень інтелектуальних застосунків, таких як моніторинг навколишнього середовища, моніторинг здоров'я, аналіз отриманих даних, звітність та інформування. Цей рівень має ключові компоненти при створенні серверних застосунків для обробки даних з IoT-пристроїв. При проектуванні рівня застосування використовуються такі поширені компоненти: брокери повідомлень, споживачі (які можуть бути складним набором декількох компонентів, що обробляють частину запитів), виробники (пристрої IoT або джерела даних) та системи збереження (нереляційні або реляційні бази даних) в залежності від вимог системи.

Декілька джерел даних (пристроїв Інтернету речей) публікують дані в IoT-брокер повідомлень у черзі або темі в залежності від використання. Клієнт підключається до черги або теми і починає приймати її повідомлення. Припускаючи, що всі дані коректні, споживач виконує перевірку даних, виявлення помилок, обробку помилок, обробку і шифрування даних. У частині обробки агреговані дані не тільки обробляються, стискаються, фільтруються та перевіряються, але також можуть аналізуватися та шифруватися. Після завершення частини обробки оброблені дані можна зберігати на власному обладнанні або з будь-яким постачальником хмарних послуг та використовувати для додаткового аналізу для бізнес-цілей[16].

Брокери повідомлень перевіряють, маршрутизують, зберігають та доставляють повідомлення до конкретних споживачів/призначень. Використання брокера повідомлень у системі Інтернету речей або будь-яких інших систем значно полегшує роботу розробника. Його функції, пов'язані з стандартизованими засобами керування потоком даних у додатках, сервісах та комунікаційних системах, здійснюються шляхом перекладу повідомлень між формальними протоколами передачі повідомлень. Великою перевагою використання брокерів повідомлень, коли система призначена для обслуговування великих обсягів даних надійним способом, є асинхронна передача повідомлень і можливість підтверджень. Повідомлення з черг або тем зберігаються у брокері до тих пір, поки клієнт не надішле підтвердження брокеру. Залежно від використання можуть бути теми або черги, а також модель передачі повідомлень від одного до одного чи модель публікації/підписки. Наприклад, коли пристрій Інтернету речей публікує повідомлення в тему, це повідомлення може бути отримано кількома споживачами, тоді як у випадку, коли повідомлення надсилається в чергу, його може отримати лише один споживач.

Архітектура клієнта може варіюватися від простого запиту до складних наборів компонентів, які виконують обробку даних. Складність системи залежить від типу даних, які потрібно обробляти, типу обчислень/обробки, які

необхідно виконати, а також, за потреби, від застосувань механізмів шифрування та застосування виявлення/обробки помилок у створених та оброблених даних. Основні операції в частині обробки включають збір/агрегацію даних, валідацію, запити, обробку, виявлення та обробку помилок, за потреби, шифрування.

Збір даних: У цьому кроці дані отримуються з брокера повідомлень і, за потреби, можуть бути збережені для майбутніх обчислень або обробки, коли використовується машинне навчання або штучний інтелект. За потреби агреговані дані можуть бути стиснуті та відфільтровані, що дозволяє системам працювати лише з цінною інформацією. Це дозволяє забезпечити ефективну комунікацію в умовах збільшення швидкості потоку даних та обмеженої пропускної здатності внутрішньої системи.

Валідація даних: Одна з ключових функцій, яку повинні мати компоненти обробки даних, - це процес валідації даних. При роботі з реальними даними та різними типами пристроїв Інтернету речей не завжди можна розраховувати на те, що згенеровані дані завжди будуть точними і надійними. Застосування певних правил, логіки, семантичних та валідаційних перевірок перед обробкою та збереженням оброблених даних є ключовим для наявності системи з показником валідації даних на рівні 99%.

Запити: Агреговані дані, що використовуються для обчислень та обробки до реальних даних на основі раніше обчислених даних, повинні бути доступні з правами лише для читання для доступу до обчислень та правами на читання та запис для оновлення обчислень. В такому випадку найкраще використовувати пам'ять з вказівником на дані для швидшого отримання.

Обробка: Частина обробки залежить від типу обчислень або обробки, які необхідно виконати над агрегованими даними. В основі обробки лежить перетворення згенерованих даних від пристроїв Інтернету речей в цінну інформацію, яка була зібрана та оброблена протягом певного періоду часу та використовується компаніями/постачальниками послуг у комерційних цілях. Сирих агрегованих даних, які не були оброблені, відфільтровані або піддані

будь-якому аналізу, не мають конкретного використання або значення для бізнесу [113].

Механізми виявлення та обробки помилок при проектуванні повністю автоматизованої системи з рівнем відкидання пошкоджених даних 0-1% важливо реалізувати компонент, який має не лише механізми валідації даних, але й механізми виявлення та обробки помилок для виправлення пошкоджених даних. Наявність компонента, який обробляє пошкоджені дані, збільшить ефективність системи з точки зору швидкості обробки даних, оскільки пошкоджені дані, які були відкинуті на етапі валідації даних, будуть відновлені в системі для подальшої обробки після виправлення. Залежно від даних та того, яка інформація з даних є критичною для частини обробки та валідації, компоненти обробки помилок можуть мати різний рівень складності, поки цінна інформація, що використовується в частині обробки, може бути виявлена, виправлена та відновлена в системі для подальшої обробки. Найпоширеніші механізми виявлення помилок включають виправлення форматів даних, типів, структур, перевірку джерел даних, додавання ймовірностей на різних рівнях для зменшення неоднозначності під час проходження валідації.

Візуалізація даних та аналітика при роботі з системами та пристроями Інтернету речей мають свої особливості. Організації мають справу з величезними обсягами даних, що надходять з різноманітних пристроїв, таких як сенсори, розумні пристрої та інтегровані платформи. Як зазначено в частині обробки, необроблена та непрофільтрована інформація фактично непридатна для обробки та аналізу. Інструменти візуалізації даних дає можливість для представлення відповідної інформації у зручному для користувачів та, що ще важливіше, зрозумілому форматі. Існує багато застосувань для моніторингу систем, агрегації статусу систем та журналів застосунків та представлення агрегованих даних у вигляді інформаційних панелей, графіків, діаграм та таблиць. Загальною архітектурою для агрегації журналів є компонент інтерфейсу користувача для візуалізації даних, агрегатор журналів та агент

журналу, який розгортається на кожному комп'ютері, і те саме стосується архітектури моніторингу. Багато інструментів візуалізації даних мають вбудовані сповіщення та оповіщення, що дозволяє користувачам або спеціалістам з IT безперервно отримувати оновлення та повідомлення, якщо досягнуто певний поріг [65].

П'ятирівнева архітектура систем IoT

П'ятирівнева архітектура систем IoT доповнює трьохрівневу, що враховує додатково бізнес-рівень та розділяє рівень додатків на рівень додатків та рівень обробки. Така архітектура підходить для великих IoT операторів, що дозволяє розподіляти та ізолювати рівні доступу до певних компонентів для клієнтів з різними потребами.

Рівень сенсорів: Основне завдання цього шару - сприймати фізичні властивості об'єктів (такі як температура, місцезнаходження тощо) за допомогою різних датчиків і перетворювати цю інформацію на цифрові сигнали, що полегшує їх передачу по мережі. Різноманітні датчики та обладнання, що використовуються у шарі сприйняття, можна порівняти з "елементами мережі" в Мережевому управлінні телекомунікаційної мережі. Техніки, що використовуються у цьому шарі, включають технологію сприйняття, технологію RFID, GPS та ін. Тим часом, багато об'єктів не можуть бути прямо сприйняті, тому для виконання цієї функціональності до їхніх тіл встановлюють мікрочіпи. Ці чіпи можуть відчувати температуру, швидкість тощо, а також обробляти зібрану інформацію за допомогою нанотехнологій, що дозволяє зробити чіпи достатньо малими, щоб їх можна було встановити у будь-який об'єкт, навіть у пісок. Це ключові технології цього шару.

Транспортний рівень: Транспортний шар, також відомий як мережевий шар, відповідає за передачу даних, отриманих зі рівня сенсорів, до центру обробки через різні мережі, такі як бездротові або кабельні мережі, включаючи локальну область підприємства (LAN). Основні техніки, що використовуються на цьому шарі, включають FTTx, 3G, Wifi, Bluetooth, Zigbee, інфрачервону технологію тощо. На цьому шарі доступні багато протоколів, таких як IPv6

(Internet Protocol version 6), який є необхідним для адресації мільярдів речей. Тому комунікація між різними мережами та сутностями є дуже важливою, оскільки вона з'єднує мільярди речей і включає в себе великі обсяги різноманітних мереж.

Рівень обробки та зберігання: рівень обробки в основному зберігає, аналізує, обробляє та передає інформацію про об'єкти з транспортного шару. Він виконує основну функцію, оскільки обробляє всю інформацію, зібрану сенсорним рівнем. Ця величезна кількість даних вимагатиме місця для зберігання, яке може бути забезпечено за допомогою таких технологій, як хмарні обчислення або системи управління базами даних. Після збереження, виконується процес аналізу отриманих даних і видає звітність та інформування в потрібний момент для виконання бажаного завдання, оскільки через велику кількість речей і великий обсяг інформації, яку вони несуть, дуже важливо зберігати та обробляти ці масові дані. Технології, які використовуються, включають бази даних, інтелектуальну обробку, хмарні обчислення, всезагальне обчислення тощо. Хмарні обчислення та великі дані є основними технологіями на цьому рівні.

Рівень додатків: Функціональність цього шару базується на даних, оброблених на рівні обробки, і розробляє різноманітні застосунки Інтернету речей, такі як аутентифікація, розумний транспорт, управління логістикою, сервіси на основі місцезнаходження та безпека. Він надає всілякі застосунки для кожної галузі, оскільки різноманітні застосунки сприяють розвитку Інтернету речей в широкому масштабі.

Бізнес рівень: подібний до менеджера Інтернету речей і включає управління застосунками, відповідною бізнес-моделі та іншими бізнес-аспектами. Він не лише керує розгортанням і оплатою різних застосунків, але й проводить дослідження щодо бізнес-моделей і моделей отримання прибутку. Як відомо, успіх технології залежить від пріоритету технології, а також від інноваційності та раціональності бізнес-моделі. Водночас, цей рівень повинен керувати конфіденційністю користувачів, що є не менш важливим у Інтернеті

речей.

1.2. Вимоги до систем Інтернету речей

Кожний додаток або сервіс Інтернету речей, наприклад, стрімінг відео в режимі реального часу або прості повідомлення про події, може мати різні вимоги. Розгортання, підтримка та моніторинг великої кількості пристроїв є складним завданням у реальному середовищі. Для вирішення гетерогенності та складностей мережі ITU-T запропонувала референцну модель Інтернету речей, яка показана на рис. 1.2. [66].

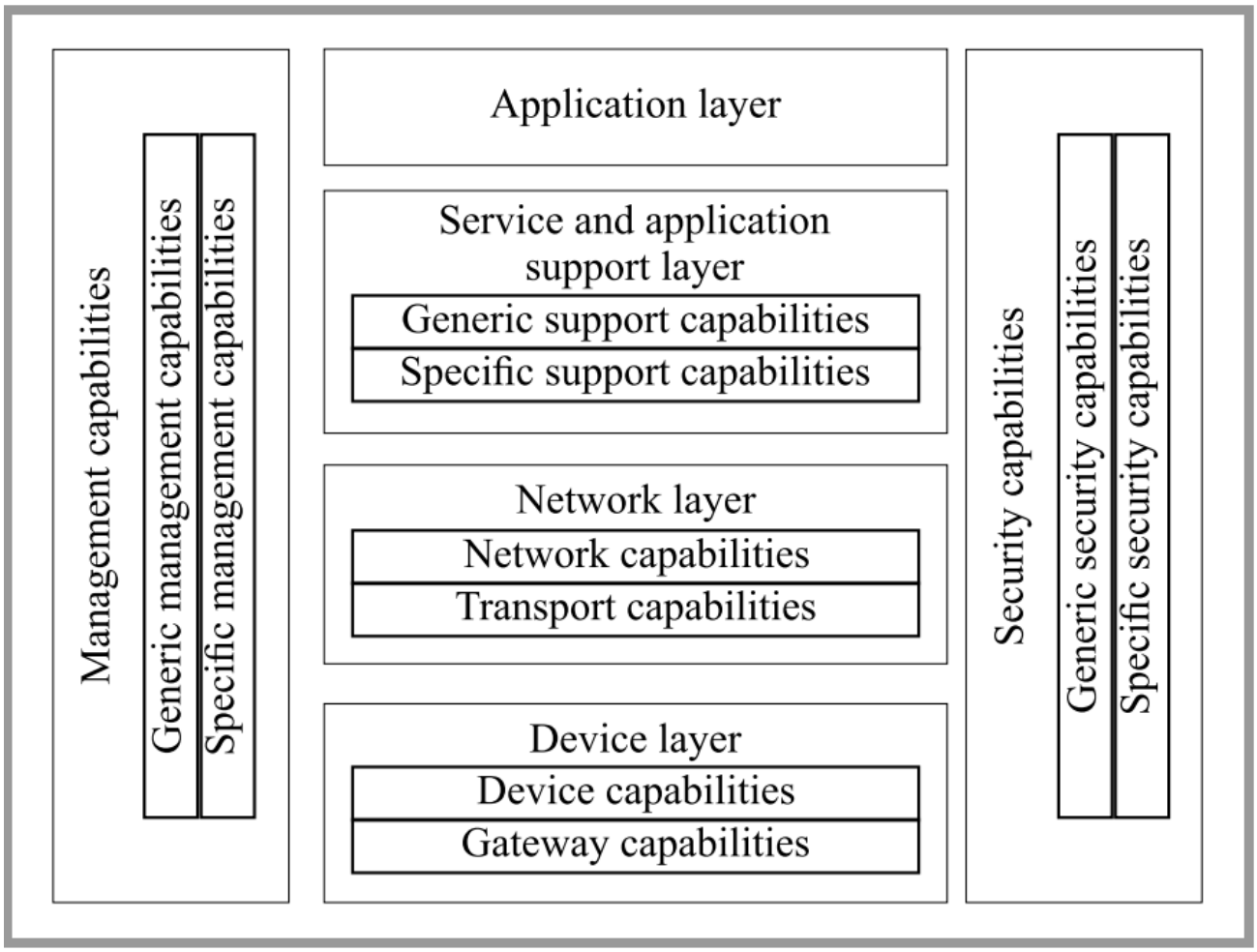

Рисунок 1.2. Еталонна модель ITU-T для IoT

Опис кожного рівня наведений нижче:

• Рівень додатків - компоненти Інтернет-рішень для кінцевих користувачів.

• Рівні підтримки сервісів та підтримки додатків - надають загальні та конкретні можливості підтримки. Загальні можливості використовуються для надання загальних функціональностей, які можуть бути використані будь-якими конкретними застосуваннями. Специфічні можливості підтримки надають сервіси, які використовуються конкретними застосуваннями.

• Рівень мережі - забезпечує можливості мережевого транспорту. З точки зору мережевих можливостей, він підтримує підключення пристроїв, керування мобільністю, маршрутизацію, авторизацію та аутентифікацію для різноманітних протоколів. З точки зору можливостей транспорту, він забезпечує передачу специфічних даних застосування, послуг Інтернету речей, управління та керування інформацією.

• Рівень пристрою - цей рівень підтримує можливості пристрою та шлюзу. Можливості пристрою включають пряму взаємодію з мережею зв'язку, опосередковану взаємодію з мережею зв'язку, спонтанну мережеву взаємодію між пристроями, а також цикли сну і прокидання для енергозбереження. Можливості шлюзу включають конвертацію протоколу та підтримку багатьох інтерфейсів.

• Можливості безпеки - підтримка конкретних та загальних можливостей. Загальні можливості включають авторизацію, аутентифікацію, захист приватності користувача, аудит безпеки на рівні застосування, безпечну комунікацію, передачу даних та сигналів на рівні мережі, а також аутентифікацію, авторизацію та контроль доступу на рівні пристрою.

• Можливості управління - тут розрізняють загальні та конкретні можливості. Специфічні можливості враховують вимоги конкретних застосувань. Загальні можливості, в свою чергу, виконують функції для конкретних застосувань, включаючи активацію пристроїв, деактивацію, віддалений доступ, стан пристроїв, оновлення програмного забезпечення,

управління топологією та трафіком. Рівень управління відповідає за виявлення несправностей, облік, конфігурацію, безпеку та продуктивність пристроїв та програмного забезпечення.

Кожного року до екосистеми Інтернету речей інтегрується велика кількість розумних об'єктів. У зв'язку з масштабом і гетерогенністю цього явища, потрібні динамічні, масштабовані і надійні рішення, що дозволять управляти ресурсами Інтернету речей. Після проведення широкого огляду існуючих рішень було виявили наступні основні проблеми, пов'язані з управлінням ресурсами[55, 64, 86, 101, 119]:

• Інтероперабельність ресурсів Інтернету речей, що використовуються на різних платформах - відсутня загальна описова модель для ресурсів, їх можливостей та формату обміну даними між різними організаціями з розробки стандартів. Інтероперабельність між кількома організаціями з розробки стандартів спрощує процес управління ресурсами в різних областях;

• Пріоритетизація виявлених ресурсів допомагає модулю управління ресурсами вибрати відповідний ресурс;

• Безпека - стандартний механізм безпеки для виявлення ресурсів;

• Балансування навантаження - для оптимального використання ресурсів необхідний розподілений модуль управління ресурсами, який може користуватися перевагами туманних обчислень і обчислень на краю мережі;

• Механізми динамічного розподілу завдань - на основі можливостей задіяних пристроїв. Існує великий потенціал для новаторських модулів динамічного управління завданнями, які базуються на ресурсах, доступних у системі;

• Оцінка ресурсів - для ефективного управління ресурсами;

• Механізми та алгоритми прогнозування, які використовуються в розподілі ресурсів;

• Розподіл ресурсів в реальному часу додатків Інтернету речей;

• Модель ціноутворення - для ресурсів, використовуваних в хмарному, туманному, граничному середовищі та Інтернеті речей;

• Стійкість - для забезпечення безперебійного надання послуг користувачеві під час відмови вузла.

Більшість запропонованих аналітичних моделей не вдаються точно відобразити реальні проблеми розподілу ресурсів, з якими стикаються в Інтернеті речей. Також багато припущень, зроблених при моделюванні проблеми, не вирішують основні виклики, і більшість досліджень не розглядає способи інтеграції локального оптимального рішення, виявленого в контексті глобальних рішень, пов'язаних з розподілом ресурсів.

1.3. Аналіз даних IoT за допомогою штучного інтелекту

Управління вимогами на основі ШІ — це застосування технологій штучного інтелекту (ШІ) у процесі збору, аналізу, документування та управління вимогами. Метою управління вимогами на основі ШІ є підвищення ефективності, точності та якості процесу управління вимогами, а також забезпечення більш ефективного прийняття рішень та співпраці між зацікавленими сторонами проекту. Технології штучного інтелекту можуть застосовуватися до різних аспектів процесу управління вимогами, включаючи виявлення вимог, аналіз, документування та управління.

Штучний інтелект грає важливу роль в системах Інтернету речей, надаючи їм додаткові можливості та покращуючи їх функціональність. Використання штучного інтелекту в системах IoT [1, 4, 5, 6]:

Аналітика даних: ШІ може використовуватись для аналізу великих обсягів даних, що генеруються пристроями IoT. Він може застосовувати розумові алгоритми, машинне навчання та глибоке навчання для виявлення залежностей, патернів та трендів у даних. Це дозволяє отримувати цінні інсайти, робити передбачення, виявляти аномалії та приймати ефективні рішення на основі цих даних.

Оптимізація процесів: ШІ може автоматизувати та оптимізувати різні процеси в системах IoT. Наприклад, використовуючи дані з сенсорів та

пристроїв, ШІ може автоматично керувати режимами роботи, розподіляти ресурси, планувати та оптимізувати робочі процеси. В результаті досягається ефективніше використання ресурсів, зниження витрат та покращення продуктивності.

Розпізнавання та інтерпретація даних з сенсорів: ШІ може бути використаний для розпізнавання та інтерпретації даних з різних сенсорів в системах IoT. Наприклад, він може аналізувати дані з відеокамер для розпізнавання об'єктів, виявлення руху або оцінки стану. Це відкриває можливості для розумних систем відеоспостереження, контролю якості або виявлення небезпек.

Розумні системи керування: ШІ дозволяє створювати розумні системи керування в системах IoT. Він може використовувати аналіз даних та машинне навчання для виявлення контексту, навчання та прийняття рішень. Наприклад, розумна система керування будинком може автоматично регулювати температуру, освітлення та енергію на основі пріоритетів та звичок мешканців.

Покращена безпека та моніторинг: ШІ може бути використаний для покращення безпеки в системах IoT. Він може виявляти незвичайну або підозрілу активність, моніторити мережеві з'єднання, аналізувати дані з різних джерел для виявлення загроз та вчасно реагувати на них. Це дозволяє запобігати кібератакам, втручанню у систему або несправностям.

Покращений користувацький досвід: ШІ може покращити користувацький досвід в системах IoT, забезпечуючи персоналізовані рекомендації, адаптуючись до потреб користувача та надаючи інтуїтивно зрозумілий інтерфейс. Наприклад, голосові асистенти або інтелектуальні системи домашнього асистента можуть спрощувати взаємодію користувачів зі смарт-пристроями.

Переваги використання аналізу IoT на основі штучного інтелекту (ШІ)

У сучасному швидкозмінному та технологічному світі аналіз даних IoT стало невід'ємною частиною розробки програмного забезпечення. З розвитком штучного інтелекту (ШІ) управління вимогами стало більш

ефективним, надійним і точним, ніж будь-коли раніше. Аналіз даних IoT на основі ШІ може забезпечити безліч переваг:

1) Автоматизація та оптимізація: ШІ може аналізувати великі обсяги даних з сенсорів та пристроїв IoT, що дозволяє автоматично приймати рішення, контролювати пристрої та виконувати операції без необхідності втручання людини. Це дозволяє оптимізувати роботу систем, зменшити людські помилки та збільшити продуктивність.
2) Розширені можливості аналізу даних: ШІ може застосовувати розумові алгоритми та машинне навчання для аналізу великих обсягів даних, зібраних з сенсорів IoT. Це дозволяє виявляти залежності, робити передбачення та видавати рекомендації на основі цих даних. Наприклад, ШІ може виявити аномальну поведінку пристроїв, передбачити витоки або несправності та запропонувати оптимальні рішення.
3) Аналітика та реагування в реальному часі: ШІ дозволяє проводити аналіз даних в реальному часі, що дозволяє системам IoT виявляти та реагувати на події швидше та ефективніше. Наприклад, система IoT з ШІ може миттєво реагувати на зміни умов навколишнього середовища, включати або вимикати пристрої, запускати аварійні процедури тощо.
4) Покращена безпека: ШІ може виявляти та реагувати на потенційні загрози безпеці в системах IoT. Він може аналізувати активність пристроїв, виявляти незвичайну поведінку та вчасно вживати заходів для запобігання кібератакам або несанкціонованому доступу до систем.
5) Зниження витрат та енергозбереження: ШІ може допомогти ефективно керувати ресурсами та оптимізувати використання енергії в системах IoT. Він може розподіляти навантаження, контролювати режими споживання енергії пристроїв та вживати заходів для

зниження витрат, що призводить до збереження коштів та енергії.

Ці переваги використання ШІ в системах IoT сприяють покращенню ефективності, продуктивності та безпеки таких систем, що робить їх більш корисними та конкурентоспроможними у сучасному світі.

ШІ вимагає великих обчислювальних ресурсів для обробки та аналізу даних. Накопичення величезного обсягу даних з пристроїв Інтернету речей (IoT) стає значущою проблемою, оскільки обробка таких масивних обсягів вимагає великої обчислювальної потужності та може стати викликом для існуючих систем обробки даних. Крім того, необхідно мати відповідні ресурси для зберігання великого обсягу даних, зокрема для тренування моделей машинного навчання та глибокого навчання.

1.4. Використання баз даних часових рядів в системах IoT

Класично, часовий ряд є послідовністю даних, які, як правило, відповідають певним вимірюванням протягом певного проміжку часу. Такого роду моделі зустрічаються у різних прикладних галузях. Найпростішим прикладом є системи взаємодії між пристроями (M2M) та Інтернет речей (IoT). Будь-які вимірювання з датчиків також формують часовий ряд. Завдяки їх загальності, завдання обробки часових рядів є одним з традиційних завдань для баз даних.

Розглянемо традиційні реляційні бази даних. На перший погляд, проблема зберігання часових рядів здається простою. Можемо створити таблицю, вказавши стовпчик з часом як ключ. Інші стовпці таблиці будуть представляти значення, виміряні в певний момент часу. Кожне нове вимірювання просто додає рядок до таблиці. Один вимір (один відлік) - один рядок. Звичайно, в будь-якій практичній системі кількість вимірюваних значень обмежена. Кожен датчик в системі IoT має певну кількість вимірюваних значень. Але на практиці, в заданий момент часу ми будемо

мати, ймовірно, лише частину з потенційно можливих атрибутів. Пристрої в системі, як правило, асинхронні, працюють за власними циклами зі своїми власними часовими інтервалами. Це означає, що кожен рядок в таблиці буде містити лише частину з потенційно можливих атрибутів. Інші атрибути будуть порожніми (значення null). Це призводить до неефективного використання дискового простору і ускладнює подальшу обробку.

Якщо звертатись до операцій з базою даних для часових рядів, то очевидно, що операція додавання (INSERT) буде переважати. Оновлення і видалення - незвичайні операції для часових рядів. Але найцікавішим моментом є те, що операція читання (SELECT) має свою специфіку. Операція читання прямо пов'язана з методами обробки. Обробка неперервних даних завжди пов'язана з певним "вікном" - частиною даних протягом короткого проміжку часу. Наприклад, розглянемо популярний алгоритм динамічного згладжування часу (Dynamic Time Warping - DTW). DTW описує наслідування двох сигналів, яке мінімізує відстань між ними. Припустимо, що у нас є два часових ряди: P довжиною n і Q довжиною m:

$$\mathrm{P} = \mathrm{p}_1, \mathrm{p}_2, \dots \mathrm{pn} \tag{1.1}$$

$$\mathrm{Q} = \mathrm{q}_1, \mathrm{q}_2, \dots \mathrm{qm} \tag{1.2}$$

Для вирівнювання двох послідовностей алгоритм конструює матрицю розміром n x m, де елемент на позиції (i, j) відповідає відстані:

$$\mathrm{d}(\mathrm{p}_i, \mathrm{q}_j) = (\mathrm{p}_i - \mathrm{q}_j)^2 \tag{1.3}$$

Ця формула описує відстань між двома точками часових рядів. Алгоритм DTW шукає найкраще відповідність між двома послідовностями, мінімізуючи цю відстань:

$$\mathrm{DTW}\,(\mathrm{P}, \mathrm{Q}) = \min \sqrt{\sum_{k-1}^{k} \mathrm{W}_k}\,, \tag{1.4}$$

де W - так званий шлях трансформації. Це набір сусідніх елементів матриці, які встановлюють відповідність між P і Q. Він представляє собою шлях, який мінімізує загальну відстань між P і Q.

DTW шукає відповідність між двома фрагментами, а не між усіма послідовностями. Повертаючись до баз даних, це означає, що запити на читання з складними умовами не є типовими. Просто потрібно читати наступну порцію даних (послідовно). Так, можуть бути задачі, які вимагають обробки всього ряду, наприклад, задачі білінгу, але навіть у цьому випадку це буде переважно пряме послідовне читання. Порівнюючи з оператором SELECT у мові SQL, реалізація конструкції SELECT TOP N для вибірки останніх N записів є важливішою, ніж той самий SELECT з умовою. Обробка часових рядів, з точки зору роботи з даними, є модельним прикладом лямбда-архітектури [47] (рисунок 1.3).

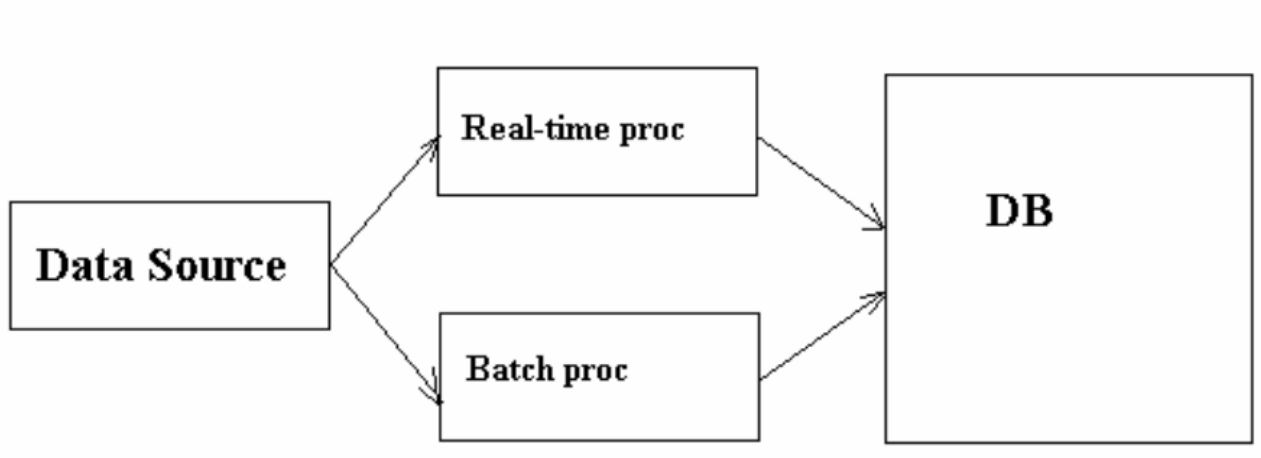

Рисунок 1.3. Лямбда-архітектура TSDB

Лямбда-архітектура - це архітектурний підхід, при якому довільна функція застосовується до довільного набору даних в найкоротший час. Універсальність цієї концепції забезпечує швидкість вирішення будь-яких завдань, включаючи складні. Якщо формалізувати цей підхід у вигляді функціонального рівняння, яке визначає будь-який запит в широкій області даних, воно буде виглядати так:

$$\text{Query} = \lambda\,(\text{Complete data}) = \lambda\,(\text{live streaming data}) * \lambda\,(\text{Stored data}) \quad (1.5)$$

Це рівняння означає, що всі пов'язані з даними запити можуть бути оброблені шляхом об'єднання результатів з історичного сховища у вигляді пакетів і потокової передачі в режимі реального часу за допомогою швидкісного рівня. Лямбда-архітектура дозволяє обробляти великі обсяги даних у режимі, близькому до реального часу. При цьому цей підхід є стійким до відмов і масштабованим. Завдяки функціям пакетного та швидкісного рівнів можна додавати нові дані до основного сховища, забезпечуючи при цьому збереження існуючих даних.

Обробка даних розподіляється на дві гілки - обробку поступаючих даних у режимі реального часу та пакетну обробку. Більша частина поступаючих даних обробляється саме у реальному часі. Це особливо важливо в Інтернеті речей (IoT) та міжмашинного взаємодії. Створення реакцій на сенсорні вимірювання (що, в основному, пов'язано з обробкою часових рядів) належить до задач реального часу. У той же час, ніщо не заважає існуванню окремої гілки для завдань без жорстких обмежень часу.

З точки зору обробки даних можна відзначити розширення SQL, призначені для полегшення роботи з часовими рядами. Наприклад, аналітика для часових рядів від Vertica. Тут є два основних аспекти: інтерполяція для обчислення пропущених значень та групування даних за часом. Ось типовий приклад:

```
SELECT time_bucket('1 година', стовпець_часу) AS годинна_група,
    AVG(значення) AS середнє_значення
FROM ваша_таблиця
WHERE стовпець_часу BETWEEN '2020-01-01 00:00:00' AND '2020-02-02
00:00:00'
GROUP BY годинна_група;
```

У цьому прикладі використовується функція time_bucket, щоб згрупувати дані за вказаними інтервалами часу (у цьому випадку - одна година). Потім обчислюється середнє значення стовпця значення для кожного інтервалу часу. Це дозволяє аналізувати часові ряди з урахуванням грануляції часу та виконувати агрегаційні операції над ними.

Такі розширення SQL спрощують роботу з часовими рядами, дозволяють виконувати різноманітні аналітичні операції та зменшують потребу в додаткових етапах попередньої обробки даних, таких як інтерполяція та групування.

Іншим схожим прикладом є віконні функції в PostgreSQL. Ці функції дозволяють виконувати обчислення над набором рядків у таблиці, які пов'язані з поточним рядком. В певній мірі вони схожі на агрегатні функції. Але на відміну від звичайних агрегатних функцій, використання віконних функцій не групує рядки в один. Це просто спосіб отримати доступ до кількох рядків таблиці під час обчислень, а не лише до поточного рядка запиту. Типовий приклад - ковзне середнє по трьом рядкам:

```
SELECT val, AVG(val) OVER (ORDER BY id ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) AS ковзне_середнє
FROM ваша_таблиця;
```

На концептуальному рівні NoSQL [33, 37, 49] як підхід оголошує відмову від єдиної моделі даних (наприклад, реляційної) і пропонує вибирати моделі даних в залежності від завдання. У цьому відношенні рішення, запропоновані для роботи з часовими рядами у NoSQL, можна описати як кращі рішення з представлення часових рядів поза реляційною моделлю. Одним із таких часто використовуваних рішень є OpenTSDB [98]. На рисунку 1.4. показана загальна архітектура системи.

OpenTSDB складається з набору так званих Time Series Daemon (TSD) та командних утиліт. Демони є повністю незалежними один від одного. Кожен

демон є просто оболонкою для доступу до HBase. Демони використовують HBase для зберігання даних і підтримують відкриті протоколи доступу до даних. Іншими словами, OpenTSDB - це схема (архітектура) використання HBase для зберігання часових рядів. Ця схема включає інтерфейсні елементи (TSD) та модель представлення даних в HBase. Незалежність демонів потрібна для горизонтального масштабування при збільшенні кількості потоків даних.

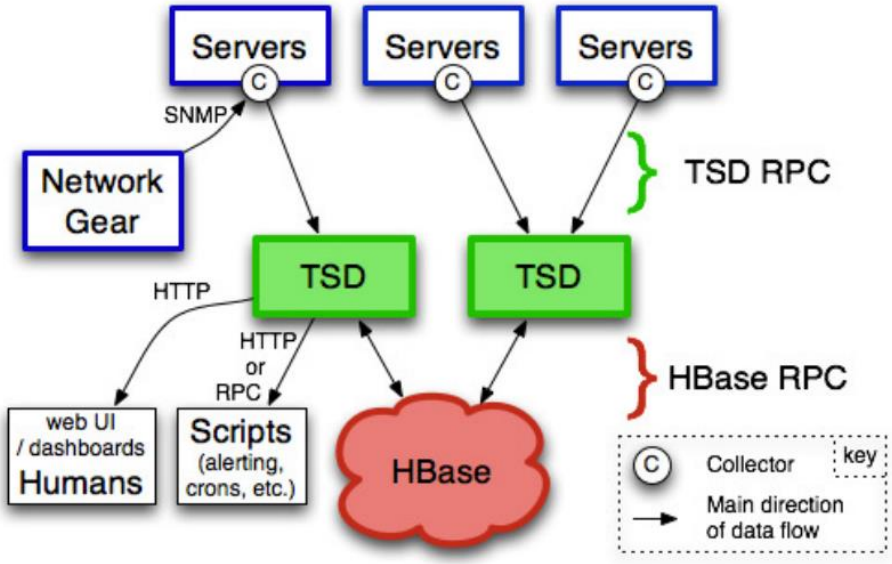

Рисунок 1.4. Архітектура openTSDB

Схема даних, що застосовується в OpenTSDB, також є фактичним стандартом для представлення часових рядів у моделі даних BigTable (яка є основою HBase). Її основною особливістю є можливість швидкої агрегації подібних часових рядів. У OpenTSDB елемент часового ряду (вимірювання) складається з назви метрики (імені вимірювання), мітки часу, значення та набору тегів (пари ключ-значення). Наприклад (часовий ряд, в якому збираються дані від датчиків температури), назва метрики - data.test, ключ - sensor_id, значення ключа - певний ідентифікатор (адреса) датчика.

Зазвичай, в системах NoSQL не передбачено окремого рівня API (принаймні, це не є обов'язковим), і описана вище модель визначає формат, у якому додаток буде записувати дані. Відповідно, всі можливі додатки будуть використовувати саме такий формат. Не потрібні ніякі перетворення схем (як

у реляційній моделі). Це, звичайно, дозволяє говорити про ефективну реалізацію операції додавання, яка, як було зазначено раніше, є основною при роботі з часовими рядами.

В OpenTSDB основна ідея полягає у введенні тегів при побудові схеми опису даних. Хоча метрики мають назву, вони можуть бути однаковими для декількох рядів. Унікальність вимірювання забезпечується комбінацією ключів (пари ключ-значення). Кожен часовий ряд повинен мати принаймні один тег. Вимірювання для одного тега зберігаються послідовно, що забезпечує швидку агрегацію даних. Множинні теги використовуються для реалізації багатозначних часових рядів.

Оскільки подібна модель представлення часових рядів у системах NoSQL стала певним фактичним стандартом, існують рішення для інших реалізацій моделі BigTable. Один з найбільш відомих та часто використовуваних прикладів такої реалізації - Cassandra. Для часових рядів KairosDB запропонував власну реалізацію OpenTSDB, використовуючи Cassandra як сховище даних.

Ще одна цікава деталь щодо поєднання NoSQL та часових рядів полягає в зростаючій підтримці виробниками пристроїв для вимірювання мови (системи) SenML. SenML визначає модель даних для вимірювань і набір метаданих для опису вимірювань. Одиницею представлення в SenML є об'єкт з набором атрибутів. Об'єкт містить масив сутностей (вимірювань). Кожна сутність (вимірювання) має атрибути, такі як унікальний ідентифікатор, час та виміряне значення. Документ SenML може бути представлений у форматі JSON або XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
bn="urn:dev:ow:10e2073a01080063" >
<e n="voltage" t="0" v="120.1" u="V" />
<e n="current" t="0" v="1.2" u="A" />
</senml>
```

NoSQL-рішення для представлення часових рядів Geras DB використовує SenML як формат представлення даних.

Ще один важливий момент для систем NoSQL [41, 121], які працюють з часовими рядами, - це підтримка MQTT. MQTT є популярним комунікаційним протоколом для міжмашинних комунікацій (M2M) та Інтернету речей (IoT). Цей протокол був спеціально розроблений для пристроїв з обмеженими обчислювальними можливостями. MQTT використовується як транспортний протокол для реалізації моделей публікації/підписки. Датчики (вимірювальні пристрої) публікують свої дані, а база даних виступає в ролі підписчика і повинна читати опубліковані дані для їх подальшого збереження. Це відповідає згаданій вище лямбда-архітектурі [115]. У контексті загальної схеми роботи з великими обсягами даних, таких як шина повідомлень може використовуватись, наприклад, Kafka, а з огляду на часові ряди виникає потреба саме у підтримці шини MQTT.

Поза контекстом OpenTSDB, Cassandra [74, 76] може бути найкращим вибором для зберігання часових рядів в системах NoSQL. Основний принцип представлення часових рядів в системах з колоночним зберіганням даних є досить очевидним. Можливо створити окрему групу стовпців з типом TimeUUID (це унікальний ідентифікатор та мітка часу). Ключ рядка відповідає назві нашого вимірювання, ім'я стовпця фактично є часом вимірювання, а значення стовпця містить виміряне значення. У такому випадку буде легко вибрати як окреме значення певного вимірювання, так і агрегувати дані для вказаного інтервалу. Важливо також зазначити, що упорядкування даних (сортування за часом) автоматично підтримується самою системою Cassandra. Технічно, Cassandra дозволяє зберігати до 2 мільярдів стовпців у одному рядку (це відповідає 2 мільярдам точок даних для часового ряду). Однак дані, що надходять з високою частотою (а саме такі системи розглядаються в NoSQL рішеннях), можуть перевищувати це значення. Проте, в Cassandra це питання можна вирішити, групуючи дані в різних рядках [85]. Наприклад, замість одного рядка для "sensor1", можна зберігати вимірювання по днях з ключами

"sensor1_20150402", "sensor1_20150403" і так далі (суфікс відповідає даті). За классичним визначенням, модель використання великих даних описується як 3V: різноманітність (Variety), швидкість надходження (Velocity) і обсяг (Volume) даних. У випадку часових рядів дані є однорідними, обсяг даних впливає лише на архівне зберігання, оскільки фактично обробляється лише невелика частина. Відповідно, швидкість надходження даних є визначальним фактором при виборі між традиційними системами та NoSQL рішеннями. З класу NoSQL рішень, на нашу думку, більш переваг має Cassandra.

Висновки до розділу 1

У даному розділі викладено вимоги, що ставляться до сучасних систем Інтернету речей (IoT). Здійснено глибинний аналіз функціональності та характеристик цих систем, розглядаючи їх ефективність та відповідність сучасним стандартам та вимогам ринку. Зосереджено увагу на ролі штучного інтелекту в контексті систем IoT, розкриваючи внесок цієї технології у вдосконалення функціональності, передбачення подій та оптимізацію ресурсного використання.

Детально розглянуто роль баз даних часових рядів у системах Інтернету речей. Виділено їх важливість у забезпеченні точного та ефективного збору, зберігання та аналізу часово залежних даних, які є ключовим елементом в інтегрованих системах IoT. Проаналізовано методи та стратегії роботи з такими базами даних з метою забезпечення їх оптимальної функціональності та надійності.

РОЗДІЛ 2

ДОСЛІДЖЕННЯ МЕТОДІВ КЛАСТЕРІЗАЦІЇ ТА МАСШТАБУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ

2.1. Кластеризація серверних ресурсів

Кластер серверів (Server Cluster) — це певна кількість серверів, об'єднаних у групу та створюючих єдиний ресурс. Таке рішення дозволяє суттєво збільшити надійність та продуктивність системи.

Група об'єднаних в локальну мережу комп'ютерів може називатися апаратним кластером, проте суть цього об'єднання полягає в підвищенні стабільності та працездатності системи завдяки єдиному програмному забезпеченню, яке управляється модулем (Менеджером кластера).

Загальна логіка кластера серверів створюється на рівні програмних протоколів та надає можливість [7, 8, 9]:

- керувати довільною кількістю апаратних ресурсів за допомогою одного програмного модуля;
- додавати та вдосконалювати програмні та апаратні ресурси без зупинки системи та масштабних архітектурних перетворень;
- забезпечувати безперебійну роботу системи при виході з ладу одного або декількох серверів;
- синхронізувати дані між серверами — одиницями кластера;
- ефективно розподіляти клієнтські запити по серверах;
- використовувати загальну базу даних кластера.

Фактично, головною метою кластера серверів є уникнення простою системи. В ідеалі будь-який інцидент, пов'язаний з зовнішнім втручанням або внутрішнім збоєм в роботі ресурсу, повинен залишатися непомітним для користувача.

При проектуванні систем з використанням серверного кластера необхідно враховувати можливості клієнтського програмного забезпечення щодо ідентифікації кластера та спільної роботи з командним модулем (Менеджером кластера). У протилежному випадку, існує ризик того, що спроба програми-клієнта за допомогою модуля отримати доступ до даних ресурсу через інші сервери може бути відхилена (конкретні механізми в даному випадку залежать від можливостей та налаштувань кластера та клієнтського обладнання).

Звичайно вважається, що кластери серверів поділяються на дві моделі:

Перша - це використання єдиного масиву зберігання інформації, що дає можливість більш швидкого перепідключення при збої. Проте, у випадку з об'ємною базою даних та великою кількістю апаратних одиниць у системі, можливе зниження продуктивності.

Друга - це модель, при якій сервери незалежні, так само як і їхні периферійні пристрої. У разі відмови перерозподіл відбувається між серверами. Тут ситуація зворотна - трафік у системі більш вільний, проте ускладнюється та обмежується використання загальної бази даних [69, 79].

У обох випадках існують певні ефективні інструменти для вирішення проблем, тому вибір конкретної моделі кластера обмежений лише вимогами до архітектури системи.

Кластер - група комп'ютерів, об'єднаних високошвидкісними каналами зв'язку, що представляє собою для користувача єдиний апаратний ресурс. Кластер - слабко пов'язана сукупність декількох обчислювальних систем, що працюють спільно для виконання загальних додатків та представляються користувачу єдиним комп'ютером. Зазвичай розрізняють наступні основні типи кластерів:

- відмовостійкі кластери (High-availability clusters, HAC);
- кластери з балансуванням навантаження (Load balancing clusters);
- високопродуктивні обчислювальні кластери (High performance

computing clusters, HPC);

- системи розподіленого обчислення.

Кластери високої доступності

Ці кластери позначаються абревіатурою HA (висока доступність - англ. High Availability). Вони створюються для забезпечення високої доступності сервісу, що надається кластером. Зайве число вузлів, що входять до кластера, гарантує надання сервісу в разі відмови одного або декількох серверів. Звичайна кількість вузлів - два, що є мінімальною кількістю для підвищення доступності. Розроблено безліч програмних рішень для побудови таких кластерів.

Кластери відмовостійкості та системи розділяються на 3 основних типи [46, 50, 60, 61]:

- Кластери з холодним резервом або активний/пасивний. Активний вузол виконує запити, а пасивний чекає на його відмову і включається в роботу, коли така ситуація виникає. Приклад - резервні мережеві з'єднання, зокрема Алгоритм сполученого дерева (Spanning Tree Algorithm).
- Кластери з гарячим резервом або активний/активний. Усі вузли виконують запити, і в разі відмови одного, навантаження перерозподіляється між залишками. Це кластер розподілу навантаження з підтримкою перерозподілу запитів при відмові [106].
- Кластери з модульною надлишковістю. Застосовується тільки в разі, коли простій системи абсолютно неприпустимий. Всі вузли одночасно виконують один і той же запит (або його частини), і з результатів вибирається будь-який. Необхідно гарантувати, що результати різних вузлів завжди будуть однаковими (або що різниці гарантовано не вплинуть на подальшу роботу) [99, 107]. Приклади - RAID і Triple modular redundancy.

Конкретні технології можуть поєднувати ці принципи у будь-якій комбінації. Наприклад, Linux-HA підтримує режим взаємного поглинання конфігурації (takeover), в якому критичні запити виконуються всіма вузлами разом, а інші рівномірно розподіляються між ними [15, 20, 21].

Кластери розподілу навантаження (Network Load Balancing, NLB) засновані на розподілі запитів через один або кілька вхідних вузлів, які перенаправляють їх на обробку в інші, обчислювальні вузли. Первинна мета таких кластерів - продуктивність, проте, в них часто також використовують методи, що підвищують надійність. Подібні конструкції називаються серверними фермами [62]. Програмне забезпечення може бути комерційним (OpenVMS, MOSIX, Platform LSF HPC, Solaris Cluster, Moab Cluster Suite, Maui Cluster Scheduler), або безкоштовним (OpenMosix, Sun Grid Engine, Linux Virtual Server [17]).

Обчислювальні кластери

Кластери використовуються з метою обчислювальних завдань, зокрема у наукових дослідженнях. Для обчислювальних кластерів важливими характеристиками є висока продуктивність процесора в операціях над числами з плаваючою точкою і низька затримка з'єднувальної мережі, а менш важливою є швидкість введення-виведення, яка більш важлива для баз даних та веб-сервісів. Обчислювальні кластери дозволяють зменшити час обчислень порівняно з одиночним комп'ютером, розбиваючи завдання на паралельно-виконувані гілки, які обмінюються даними через з'єднуючу мережу. Одна з типових конфігурацій - набір комп'ютерів, зібраних з загальнодоступних компонентів, з встановленою на них операційною системою Linux, і з'єднаних мережею Ethernet, Myrinet, InfiniBand або іншими. Спеціально виділяють обчислювальні кластери високої продуктивності (HPC Cluster - High-performance computing cluster).

Системи розподілених обчислень (grid)

Такі системи не вважаються кластерами, але їх принципи значною мірою схожі з кластерною технологією. Їх також називають grid-системами. Головна

відмінність - низька доступність кожного вузла, тобто неможливість гарантувати його роботу в заданий момент часу (вузли підключаються і відключаються в процесі роботи), тому задачу слід розбити на ряд незалежних один від одного процесів. Така система, на відміну від кластерів, не схожа на єдиний комп'ютер, а служить спрощеним засобом розподілу обчислень. Нестабільність конфігурації, в такому випадку, компенсується великою кількістю вузлів.

Програмно-орієнтований кластер - це група серверів, що логічно об'єднані, здатні обробляти ідентичні запити та використовуються як єдиний ресурс. Зазвичай сервери групуються за допомогою локальної мережі. Група серверів має більшу надійність та продуктивність, ніж окремий сервер. Об'єднання серверів в один ресурс відбувається на рівні програмних протоколів.

У відмінність від апаратного кластера комп'ютерів, програмні кластери вимагають [2, 31, 35]:

- Наявності спеціального програмного модуля (Cluster Manager), основною функцією якого є підтримка взаємодії між усіма серверами-членами кластера.
- Синхронізації даних між усіма серверами-членами кластера.
- Розподілу навантаження (клієнтських запитів) між серверами-членами кластера.
- Вміння клієнтського програмного забезпечення розпізнавати сервер, що представляє собою кластер серверів, та відповідно обробляти команди від Cluster Manager.

Якщо клієнтська програма не вміє розпізнавати кластер, вона буде працювати лише з тим сервером, до якого звернулася спочатку, а при спробі Cluster Manager перерозподілити запит на інші сервери, клієнтська програма може взагалі втратити доступ до цього сервера (результат залежить від конкретної реалізації кластера).

У більшості випадків кластери серверів функціонують на окремих комп'ютерах. Це дозволяє підвищити продуктивність за рахунок розподілу навантаження на апаратні ресурси та забезпечує стійкість до відмов на апаратному рівні.

Однак, принцип організації кластера серверів (на рівні програмного протоколу) дозволяє виконувати кілька програмних серверів на одному апаратному. Таке використання може бути популярним у таких випадках:

- при розробці та тестуванні кластерних рішень;
- при необхідності забезпечити доступність кластера з урахуванням частих змін конфігурації серверів - членів кластера, що потребують їх перезавантаження (перезавантаження відбувається по черзі) в умовах обмежених апаратних ресурсів.

2.2. Архітектура побудови кластерних систем

У функціональній класифікації кластери можна розділити на "Високошвидкісні" (High Performance, HP), "Системи Високої Готовності" (High Availability, HA), а також "Змішані Системи" [27, 29, 89, 95].

High Performance архітектура

Високошвидкісні кластери використовуються для завдань, які вимагають значної обчислювальної потужності. High Performance кластерна система визначається продуктивністю вузлів та зв'язками між ними. При цьому вплив швидкісних характеристик цих зв'язків на загальну продуктивність системи залежить від характеру виконуваного завдання. Якщо завдання вимагає частого обміну даними з підзадачами, то швидкодії комунікаційного інтерфейсу слід приділяти максимальну увагу. Чим менше взаємодіють частини паралельного завдання між собою, тим менше часу потрібно для його виконання. Це також визначає певні вимоги до програмування паралельних завдань.

Основні проблеми при необхідності обміну даними між підзадачами виникають у зв'язку з тим, що швидкодія передачі даних між центральним процесором та оперативною пам'яттю вузла значно перевищує швидкісні характеристики систем міжкомп'ютерної взаємодії. Крім того, сильно впливає на зміну функціонування системи, порівняно зі звичайними SMP системами, різниця в швидкодії кеш-пам'яті процесорів та міжвузлових комунікацій. Швидкодія інтерфейсів характеризується двома параметрами: пропускною здатністю неперервного потоку даних та максимальною кількістю найменших пакетів, які можна передати за одиницю часу [40, 52, 57, 58].

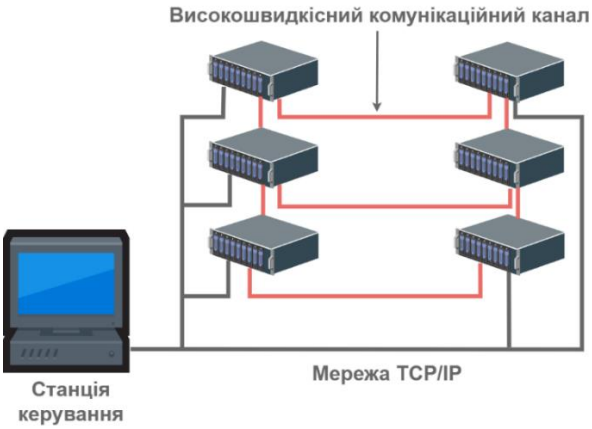

Рисунок 2.1 – High Performance кластер

High Availability архітектура

Кластери, що відносяться до систем високої готовності, використовуються там, де вартість можливого простою перевищує вартість витрат, необхідних для побудови кластерної системи.

Стійкість до відмов у кластері забезпечується подвійним забезпеченням усіх життєво-важливих компонент. Максимально стійка до відмов система повинна не мати жодної точки відмови, тобто активного елемента, відмова якого може призвести до втрати функціональності системи. Цю характеристику, як правило, називають — NSPF (No Single Point of Failure, — англ., відсутність єдиної точки відмови) [25].

При побудові систем високої готовності, головною метою є забезпечення мінімального часу простою.

Для того, щоб система мала високі показники готовності, необхідно:

- щоб її компоненти були максимально надійними;
- щоб вона була стійкою до відмов, бажано, щоб не мала точок відмов;
- а також важливо, щоб вона була зручною в обслуговуванні і дозволяла проводити заміну компонент без зупинки.

Знехтуванням будь-яким із зазначених параметрів може призвести до втрати функціональності системи.

Що стосується забезпечення максимальної надійності, вона здійснюється за допомогою використання електронних компонент високої та надвисокої інтеграції, підтримки нормальних режимів роботи, зокрема, теплових.

Стійкість до відмов забезпечується за допомогою використання спеціалізованих компонент (ECC, Chip Kill модулі пам'яті, стійкі блоки живлення та ін.), а також за допомогою технологій кластеризації. Завдяки кластеризації досягається така схема функціонування, коли при відмові одного з комп'ютерів, завдання перерозподіляються між іншими вузлами кластера, які працюють без збоїв. Причому одним із найважливіших завдань виробників кластерного програмного забезпечення є забезпечення мінімального часу відновлення системи в разі збою, оскільки стійкість до відмов системи потрібна саме для мінімізації так званого позапланового простою [42, 71].

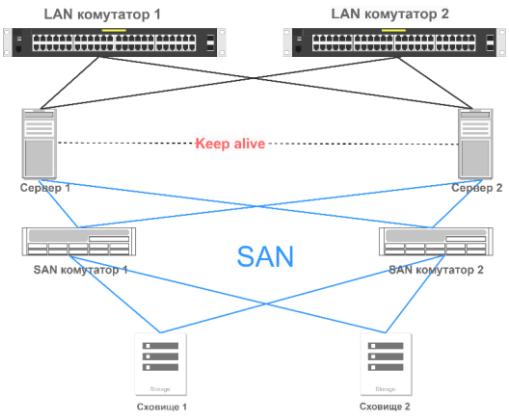

Рисунок 2.2 – High Availability кластер

Системи високої доступності можна розподілити на два типи:

1) Архітектура з розділеними ресурсами (Shared Nothing Architecture):

Цей тип кластерів використовує підхід, при якому кожен вузол має власні незалежні ресурси, такі як процесор, пам'ять та дисковий простір. Вузли в такій архітектурі не діляться спільними ресурсами і фізично не знаходяться в одній локальній сеті. Кожен вузол обробляє свої завдання та зберігає свої дані. Це забезпечує високий рівень безпеки та стійкості до відмов, оскільки проблеми в одному вузлі не впливають на решту системи. Такий підхід часто використовується для розподілених баз даних та обчислювальних кластерів [108, 111].

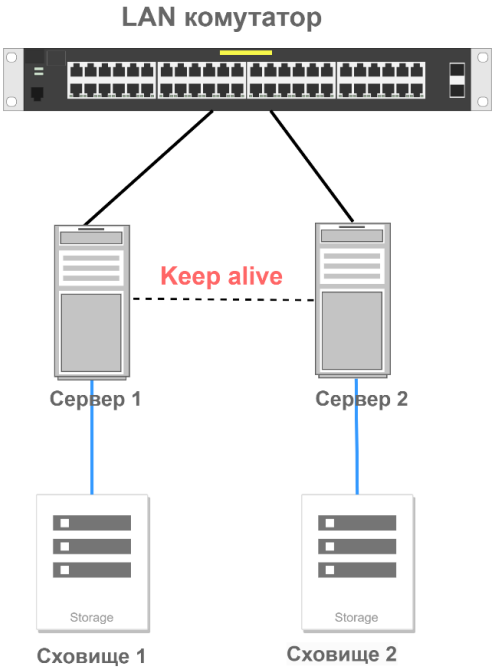

Рисунок 2.3 – High Availability архітектура з розділеними ресурсами

2) Архітектура зі спільними дисками (Shared Disk Architecture):

У цьому типі кластерів кілька вузлів мають спільний доступ до одного або декількох дисків. Це означає, що дані можуть бути збережені на спільному диску та доступні для всіх вузлів. Кластери з такою архітектурою можуть швидко реагувати на відмови, оскільки інші вузли можуть продовжувати роботу зі спільного диску, якщо один з вузлів вийшов із ладу. Цей тип

кластерів часто використовується для високодоступних серверних систем, баз даних та файлових сховищ [51, 100].

Архітектура зі спільними дисками класично використовується для побудови високодоступних кластерних систем, спрямованих на обробку великих обсягів даних. Така система складається зі спільного сховища даних та вузлів кластера, які розподіляють доступ до спільних даних. При високій потужності системи зберігання даних, при роботі з завданнями, спрямованими на їх обробку, архітектура зі спільними дисками є більш ефективною. У цьому випадку не потрібно зберігати декілька копій даних і в той же час, при виході з ладу вузла, завдання можуть бути миттєво доступні для інших вузлів.

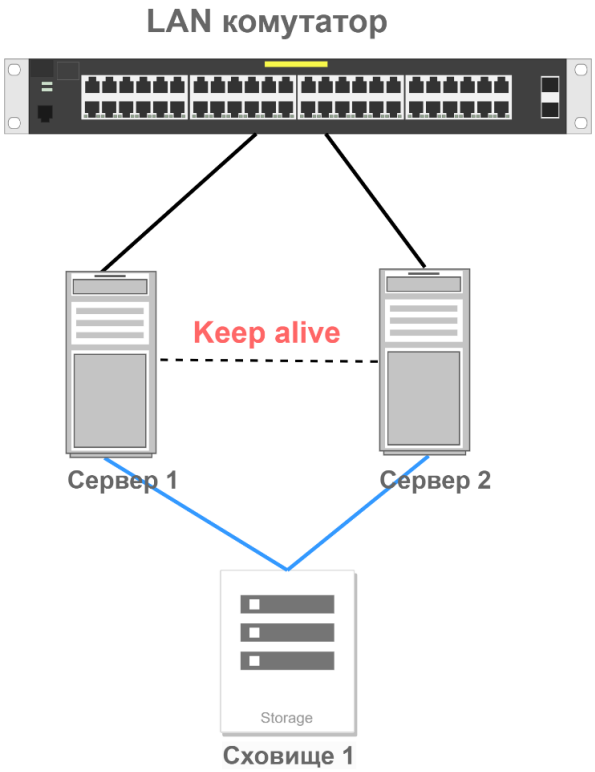

Рисунок 2.4 – High Availability архітектура зі спільними дисками

Система комунікацій у відмовостійких кластерних системах може бути побудована на такому самому обладнанні, як і у високошвидкісних кластерах. Проте, у випадку реалізації архітектури зі спільним диском, виникає необхідність забезпечення високошвидкісного доступу до спільної системи зберігання даних [80, 120]. Ця задача має сьогодні багато варіантів рішень.

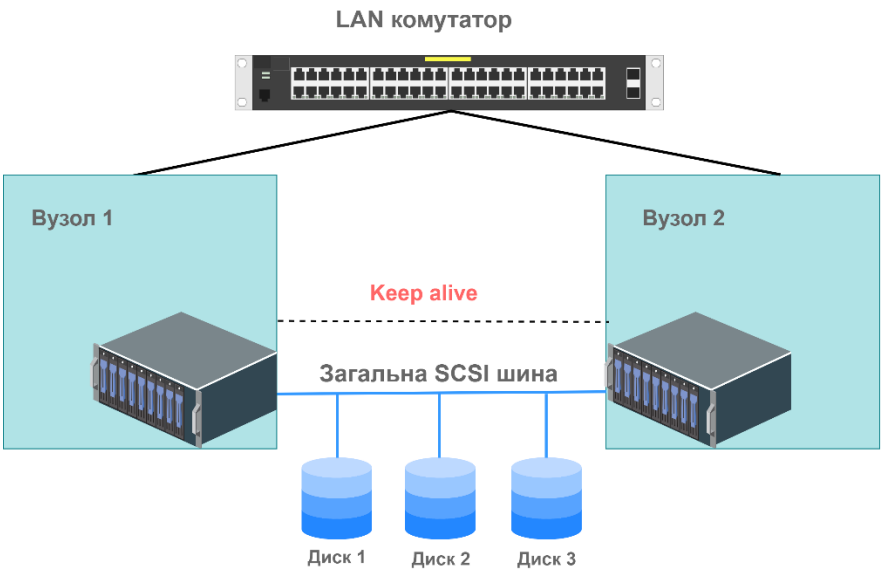

Рисунок 2.5 – High Availability архітектура зі спільною шиною SCSI

Розподілена система управління баз даних в розподілених системах.

Розподілена база даних - це сукупність декількох взаємопов'язаних баз даних, які фізично розподілені по різних місцях і обмінюються даними через комп'ютерну мережу. Бази даних у кластері логічно взаємопов'язані одна з одною. Часто вони представляють собою єдину логічну базу даних. Дані фізично зберігаються на декількох об’єктах. Дані на кожному сайті можуть управлятися СУБД незалежно від інших об’єктів. Розподілена база даних не є слабкопов'язаною файловою системою.

Розподілена система управління базами даних (DDBMS) - це децентралізована програмна система, яка управляє розподіленою базою даних, забезпечуючи спрощений доступ і обробку даних на різних вузлах (сайтах) мережі. Вона дозволяє об'єднати окремі бази даних, розташовані на різних фізичних місцях, в єдину логічну базу даних. І хоча вона може надавати уявлення про централізований доступ до даних, всі дії здійснюються в розподіленому середовищі. Така система дозволяє досягти більшої масштабованості та доступності, забезпечуючи одночасний доступ до даних з різних частин мережі [38, 59, 93, 102, 118]:

- DDBMS використовується для створення, пошуку, оновлення та

видалення розподілених баз даних.

- DDBMS періодично синхронізує базу даних і надає механізми доступу, завдяки яким розподілення стає прозорим для користувачів.
- DDBMS гарантує, що дані, змінені на будь-якому сайті, будуть постійно оновлюватися.
- DDBMS використовується в прикладних областях, де великі обсяги даних обробляються та доступні багатьом користувачам одночасно.
- DDBMS призначений для гетерогенних платформ баз даних.
- DDBMS підтримує конфіденційність та цілісність даних баз даних.

Системи керування розподіленими транзакціями – один з важливих елементів DDBMS. Із зростанням обсягу та складності транзакцій, системи керування розподіленими транзакціями мають стати більш потужними та ефективними. Вони допоможуть управляти транзакціями в різних додатках одночасно та забезпечать їх координацію та цілісність.

Розподілена обробка даних в реальному часі: З розвитком Інтернету речей (IoT) та інших додатків в реальному часі з'явиться потреба у розподіленій базі даних, яка може швидко обробляти великий обсяг даних у режимі реального часу.

Використання DDBMS для штучного інтелекту. Штучний інтелект обробляє велику кількість даних і потребує безвідмовного доступу до баз даних.

Загалом, розподілена база даних є важливим інструментом для організацій, які працюють з великим обсягом даних та потребують надійної, гнучкої та доступної інформаційної інфраструктури.

2.3. Розподілена архітектура Інтернет речей

Дані, зібрані датчиками, потребують обробки перед використанням для надання послуг користувачам або додатками. Обробка вимагає використання кількох ресурсів, які розподілені географічно. Вона включає в себе різноманітне апаратне забезпечення, програмне забезпечення та технології зв'язку. Пристрої можуть працювати як по протоколу IP, так і non-IP. Non-IP пристрої Інтернету речей зазвичай є пристроями з низьким споживанням енергії, які використовують такі протоколи, як Bluetooth, ZigBee та LwM2M. Ці пристрої передають зібрану інформацію до найближчого налаштованого шлюзу. Дані, зібрані кожним шлюзом, фільтруються та обробляються перед передачею на центральний сервер або в хмарне сховище. IP-засновані пристрої Інтернету речей можуть пересилати дані або через шлюз, або безпосередньо завантажувати їх на центральний сервер. Розподілена архітектура пов'язана з обчислювальними механізмами, які впливають на ресурси в середовищі Інтернету речей [43]:

• *Ресурси хмарних обчислень в системах Інтернету речей.*

Хмарні обчислення покладаються на величезні обчислювальні ресурси та ресурси зберігання. Пристрої Інтернету речей можуть використовуватись в інфраструктурах хмарного обчислення для зберігання та проведення складного аналізу даних Інтернету речей. Хмарні обчислення дають можливість знизити витрати на зберігання, обчислення та експлуатацію систем IoT. Однак інтеграція Інтернету речей та хмарних обчислень викликає багато складнощів. Для цього необхідно безвідмовне підключення до мережі, високі затримки між хмарою та пристроями Інтернету речей можуть бути неприпустимими для застосунків реального часу. Крім того, безпека та конфіденційність також становитимуть великі виклики, оскільки дані зберігаються віддалено [45].

• *Мережеві ресурси в системах Інтернету речей.*

Граничні обчислення зменшують вимоги до ресурсів кінцевих пристроїв та хмари. Пристрої на границі провайдерів можуть включати шлюз або контролер пристрів, які беруть на себе частину обчислень або конвертують данні в потрібний формат. Додатки, які вимагають низької затримки та високої швидкодії, отримують переваги від обчислення на границі.

• *Ресурси туманного обчислення в екосистемах Інтернету речей.*

Туманні обчислення схожі на граничні обчислення, які частково наближають обчислення, зберігання та мережеві служби до джерела даних, замість їх передачі до хмарного обчислення. Це дає змогу за допомоги одного або декілька пристроїв в локальній мережі, розподілити робоче навантаження хмари. Цей підхід пропонує переваги, схожі на обчислення на границі, але з набагато більшими потужностями обробки та сховища.

Вимоги до ресурсів розподілених систем Інтернету речей залежать виключно від функціональних та нефункціональних вимог системи. Детальні специфікації надаються ITU-T [3]. Різні вимоги та їх класифікація наведені нижче:

• Формалізовані вимоги бізнесу - високорівневий документ, який визначає цілі для розробки нової системи.

• Формалізовані вимоги зацікавлених сторін - включають перспективу та вимоги різних зацікавлених сторін, таких як користувачі/оператори/технічний персонал, пов'язані з використанням системи для досягнення бізнес-цілей.

• Формалізовані вимоги до системи - визначають технічні вимоги обраної системи, включаючи функціональні та нефункціональні характеристики:

1) Формалізовані вимоги до функціональності - визначають поведінку системи з точки зору сервісів/функцій/операцій. Важливі функціональні вимоги, визначені ITU для мереж Інтернету речей, включають вимоги до підтримки додатків, вимоги до сервісів, вимоги до зв'язку, вимоги до

пристроїв, вимоги до управління даними, вимоги до безпеки та конфіденційності.

2) Формалізовані вимоги до нефункціональних характеристик - визначають додаткові/якісні характеристики системи. Важливі нефункціональні вимоги, визначені ITU-T для мереж Інтернету речей, включають взаємопов'язаність, масштабованість, надійність, високу доступність, адаптивність та керованість. Вимоги до нефункціональних ресурсів складно обчислити, оскільки вони мають суб'єктивний та відносний характер. Спроба покращити один параметр може прямо вплинути на інші параметри тієї ж системи. У середовищі Інтернету речей управління ресурсами є важливим завданням з точки зору забезпечення надійності та безпеки. Функціональність управління ресурсами розподілена по різних рівнях референсної моделі. Будь-яке обладнання, програмне забезпечення та вузли зв'язку можуть бути змодельовані як ресурси.

2.4. Управління ресурсами систем Інтернету речей

Моделювання ресурсів є важливим етапом управління ресурсами. Це процес абстрагування та побудови метаданих для ресурсів Інтернету речей. Ефективне моделювання ресурсів є ключем до ефективних механізмів виявлення та розподілу ресурсів в мережах Інтернету речей. Ресурси моделюються на основі семантичних, віртуалізаційних та атрибутивних методів моделювання. Для моделювання програмних систем, моделювання на основі віртуалізації є єдиним рішення для відтворення максимальної подібності: У цій моделі сенсорне обладнання, програмне забезпечення або мережеві ресурси представлені віртуально в хмарі або шлюзі для полегшення управління та оптимізації використання ресурсів. Різні типи віртуалізації, на яких ґрунтується Інтернет речей, перераховані нижче [34]:

- Моделювання мережі, що використовує програмно-визначений підхід (SDN). SDN абстрагує фізичну мережу, що допомагає у централізованому керуванні та налаштуванні поведінки мережі. SDN має дві площини - пряму площину для передачі пакетів та контрольну площину для налаштування та управління мережею. Роз'єднання цих площин допомагає враховувати елементи різноманітних пристроїв;
- Віртуалізація мережевих функцій (NFV). NFV усуває потребу у спеціальному обладнанні та використовує стандартний сервер для мережевих функцій. Різні мережеві функції, такі як маршрутизація, брандмауер, балансування навантаження та запобігання вторгнення, можуть бути виконані стандартним сервером завдяки впровадженню NFV;
- Контейнери. Цей підхід є віртуалізацією на рівні операційної системи, де програмні додатки упаковуються зі своїми залежностями, бібліотеками та файлами конфігурації. Оскільки кілька контейнерів виконуються безпосередньо на хост-операційній системі, рішення є більш легким, ніж віртуальні машини. В системах Інтернету речей багато провайдерів хмарних послуг підтримують розробку та управління контейнерами. Контейнери, як правило, розгортаються в хмарному або туманному середовищі, що допомагає у реальному часі аналізувати ресурси, логи, помилки. Вони також діють можливість балансувати навантаження, зменшуючи навантаження на хмару;
- Віртуалізація сенсорів. У цьому підході абстрактна модель пристроїв Інтернету речей розміщується на центральному сервері. Користувацькі додатки надсилають запити на центральний сервер з абстрактним представленням пристрою Інтернету речей, а не фізичними пристроями Інтернету речей.

Оцінка та розподіл ресурсів.

У більшості застосунків Інтернету речей, запити користувачів та генерація даних є спонтанними процесами, які становлять великий виклик у передбаченні ресурсів Інтернету речей. Однак для оптимальної продуктивності системи та ефективного використання ресурсів необхідно певний рівень оцінки ресурсів. Зазвичай оцінка ресурсів здійснюється на основі історичних записів, збережених у хмарі. Туманні обчислення також можуть допомогти у оцінці ресурсів, оскільки вони розгорнуті близько до датчиків та мають краще знання про розташування.

Оцінка ресурсів Інтернету речей залежить від типу пристрою, рухомості пристрою, стану енергопостачання, типу генерованих або оброблюваних даних, методу комунікації, застосованих заходів безпеки та поведінки користувача. Надлишкове прогнозування ресурсів може призвести до їх простою. Неправильний розподіл ресурсів також безпосередньо впливає на продуктивність системи. Тому система управління ресурсами повинна точно передбачати ресурси, необхідні для оптимальної продуктивності системи. У цьому відношенні можуть використовуватися інструменти аналізу даних для прогнозування на основі історичних записів, збережених у хмарі [36, 44,114].

У гетерогенній системі IoT досягнення задовільного рівня обслуговування (SLA), пов’язаного з додатками кінцевого користувача, є серйозною проблемою. Наприклад, системи моніторингу і міські системи спостереження вимагають абсолютно різних стратегій управління ресурсами. Перед тим як програма відправляє запит до пристроїв Інтернету речей, виникає процес виділення ресурсів, що виконується кілька кроків. Автентифікація клієнта потребує певних ресурсів, тому пул доступних ресурсів, дозволи на доступ і обмеження використання повинні бути перевірені в структурі проміжного програмного забезпечення. Для вирішення складних проблем розподілу ресурсів потрібні математичні моделі, які працюють у реальному середовищі. У таких моделях конкретний сценарій

зіставляється зі стандартним алгоритмом оптимізації. Важливі категорії алгоритмів розподілу ресурсів включають наступне:

• Багатоцільові алгоритми оптимізації. Ці алгоритми визначають найкращий варіант з набору припустимих рішень. Проблеми розподілу ресурсів Інтернету речей часто моделюються як задачі багатоцільової оптимізації, оскільки вони включають конфлікт кількох критеріїв. Більшість проблем розподілу ресурсів намагаються застосовувати правило Парето, де зміна будь-якого критерію погіршує щонайменше один з решти критеріїв. Оскільки ці алгоритми знаходять точні рішення, вони мають високу обчислювальну складність і вимагають більше обчислювальних ресурсів;

• Евристичні алгоритми. Ці алгоритми швидше знаходять приблизне рішення, ніж точне рішення, яке вимагає більших обчислювальних витрат. Вони також використовуються для отримання майже оптимальних рішень, коли інші алгоритми оптимізації не вдаються згенерувати точні дані. Всі ці переваги досягаються за рахунок компромісу між оптимальністю, точністю, точністю і повнотою рішення. Ці алгоритми не дуже гнучкі і чутливі до ключових параметрів прийняття рішень. Дрібні зміни можуть суттєво вплинути на кінцеве рішення. До прикладів таких алгоритмів відносяться алгоритми жадібного пошуку та пошуку за забороною;

• Метаевристичні алгоритми. Це процедури вищого рівня, які використовуються для знаходження достатнього рішення на основі неповної або неідеальної інформації. Ці алгоритми не надають глобально оптимального рішення.

Керування ресурсами IoT, що використовуються в кожному рівні середовища, поставляє перед нами низку викликів. Більшість робіт проводиться для розробки математичних моделей розподілу ресурсів. Проте ці запропоновані моделі тестуються лише за допомогою інструментів моделювання або програмного забезпечення, розробленого власноруч [13].

Моделювання на основі семантичної мережі, хоч і покращує сумісність, також стикається з проблемами несумісності між різними SDOs та SSOs.

Технології, засновані на семантиці, зустрічають ряд інших проблем, таких як необхідність легкої семантичної технології, покращений механізм безпеки для реєстрації та виявлення ресурсів, велика потужність індексації та запитів, динамічне перетворення між стандартами, ефективні моделі потокової передачі та відкриті стандарти або інтерфейси для розробників додатків.

2.5. Використання баз даних часових рядів в архітектурі граничних обчислень для систем IoT

Системи на основі IoT, як правило, вимагають великої кількості даних. Великий обсяг даних збирається з великої кількості датчиків, а потоки цих даних постійні. Таким чином, зберігання даних та їх аналіз - це головні проблеми, які необхідно вирішувати. В останні роки широко використовують аналіз великих даних для аналізу даних, зібраних з систем IoT, з метою поліпшення виявлення та діагностики несправностей. Через інтенсивну природу обробки великих даних аналіз зазвичай потрібно проводити в хмарі. Однак, направлення даних до хмари може мати значну затримку комунікації. Крім того, багато з завдань виявлення та діагностики несправностей повинні виконуватися в режимі реального часу. У відповідь на таке балансування між часом та обмеженнями ресурсів, була розроблена нова обчислювальна інфраструктура, яка поєднує IoT, Edge та Cloud. Попередні обчислення, необхідні для своєчасних рішень, можуть бути зроблені на близьких Edge-шлюзах та локальних мережах. Інтегрований аналіз може бути залишений у хмарі, а більш складні та/або точніші результати аналізу можуть бути направлені назад до Edge-пристроїв та IoT-пристроїв для покращення процесу прийняття рішень або підтримки змінних тенденцій [82, 104].

Дані часових рядів – це послідовність точок даних, зібраних за проміжки часу, що дозволяє нам відстежувати зміни з часом. Дані часових рядів можуть відстежувати зміни протягом мілісекунд, днів або навіть років. Доступ до детальних, багатофункціональних часових рядів даних став одним із

найцінніших товарів у нашому світі. Підприємства, уряди, школи та громади, великі й малі, знаходять способи отримання цінності від аналізу даних часових рядів. Моделі використання розробниками програмного забезпечення вже відображають ту саму тенденцію. Насправді, протягом останніх двох років бази даних часових рядів (TSDB, або СУБД часових рядів — системи керування базами даних) стабільно залишаються категорією баз даних, що розвивається найшвидше.

Звичайні бази даних показують низьку ефективність роботи з часовими рядами. TSDB має спеціалізовані алгоритми роботи з часовими рядами, і існують принаймні дві причини, чому бази даних часових рядів є найшвидше зростаючою категорією баз даних сьогодні: масштаб та зручність використання.

Масштаб

Дані часових рядів накопичуються дуже швидко, а звичайні бази даних не призначені для роботи з таким масштабом. Традиційно, реляційні бази даних показують погані результати при роботі з дуже великими наборами даних, тоді як NoSQL-бази даних краще підходять для масштабування. З іншого боку, бази даних часових рядів, незалежно від того, чи вони реляційні, чи NoSQL, використовують ефективність, яку можна досягти лише тоді, коли час розглядається як пріоритетний показник, що дозволить надавати нові можливості, починаючи від покращень продуктивності, також краще стиснення даних

Зручність використання

Бази даних часових рядів також зазвичай містять вбудовані функції та операції, що є типовими для аналізу даних часових рядів, такі як політики збереження даних, постійні запити на агрегацію, гнучка категоризація за часом та ін. Навіть якщо ви тільки починаєте збирати цей тип даних і масштаб на даний момент не є проблемою, ці функції все ще можуть забезпечити кращий досвід користувача та спростити задачі аналізу даних. Мати вбудовані функції та можливості для аналізу тенденцій, які легко доступні на рівні даних, часто

дозволяє виявляти можливості, про які ви не знали, що вони існують, незалежно від того, наскільки великий чи малий ваш набір даних.

Отже, розробники все більше віддають перевагу використанню баз даних часових рядів та використовують їх для різноманітних випадків застосування [68]:

- Моніторинг програмних систем: віртуальні машини, контейнери, сервіси, додатки.
- Моніторинг фізичних систем: обладнання, машини, підключені пристрої, навколишнє середовище, наші домівки, наші тіла.
- Додатки для відстеження майна: транспортні засоби, вантажівки, фізичні контейнери, піддони.
- Системи фінансової торгівлі: класичні цінні папери, нові криптовалюти.
- Додатки, які відслідковують дані взаємодії з користувачем/клієнтом.
- Інструменти бізнес-аналітики: відстеження ключових показників та загального стану бізнесу.

Останніми роками було розроблено багато баз даних часових рядів (TSDB), щоб керувати зберіганням постійних потоків даних. Проте, через новизну досліджень TSDB, є ще місце для покращення їх конструкції.

Перша проблема полягає в тому, як ефективно зменшити розмір старих даних, щоб зберегти простір зберігання, але корисну інформацію в історичних даних зберегти. Хоча всі TSDB надають підтримку дискретизації, отримання ефективних функцій дискретизації.

Друга проблема із існуючими базами даних часових рядів полягає в тому, що вони не мають семантичної підтримки. Більшість таких баз даних призначені для підтримки окремої системи із збереженням продуктивності зберігання даних, але вони не мають підтримки семантики даних. Тому, дані у базі даних можуть інтерпретуватися тільки тими, хто проектує схему бази

даних. Це призводить до значних недоліків у ефективному пошуку та обробці даних. Наприклад, багато сучасних кіберфізичних систем (КФС) є високо складними. Коли система моніторингу на основі Інтернету речей використовується для виявлення несправностей у складній системі, поточні бази даних часових рядів не можуть зафіксувати зв'язки між потоками даних, що ускладнює аналіз потенційних зв'язків між потоками даних. Інтегрований аналіз даних, тобто аналіз потоків даних з подібних систем, може допомогти покращити точність аналізів. Без належної семантики даних буде дуже складно посилатися на подібні потоки даних та досягати інтегрованого аналізу. Проблема зберігання даних стає більш серйозною, коли ми розглядаємо інфраструктуру IoT Edge Cloud [48, 103, 116]. Потік даних може бути переміщений та оброблений в величезній інфраструктурі, що робить складним визначення місцезнаходження даних та ускладнює їх відновлення.

Третя проблема із існуючими системами зберігання часових рядів даних (TSDB) полягає в тому, що вони не моделюють події. Усі TSDB надають дуже хорошу підтримку для зберігання потоків даних, які збираються з відносно сталою швидкістю, тобто дані надходять через певний інтервал часу. З іншого боку, дані подій (тригерів), надходять до системи випадковим чином. Зберігання потоків даних та підтримка ефективного пошуку для потоків даних на основі подій є критичним так само, як для потоків даних з відносно фіксованими інтервалами збору.

У системі моніторингу на основі Інтернету речей нам потрібно позначити потоки даних моніторингу за кількома важливими атрибутами, включаючи те, який датчик створив потік даних моніторингу, яку метрику та якої цільової сутності моніториться датчиком, яке є співвідношення цільової сутності з цільовою системою, яка одиниця виміру використовується для потоку даних тощо. Багато з цих атрибутів не моделюються існуючими базами даних часових рядів через складність їх специфікації. Наприклад, для вказання, яка сутність у цільовій системі моніториться, нам потрібно мати можливість вказати архітектуру системи та скласти модель для специфікації

сутностей системи. Нам також потрібно вказати датчики та їх відношення до сутностей у цільовій системі. Для вказання метрики, яка моніториться, нам потрібно правильно визначити метрику та її відношення до інших метрик. Іншими словами, нам потрібно визначити онтологію метрики, щоб кожна метрика, на яку посилається система моніторингу, можна було належним чином розуміти.

Онтологія системи визначає ієрархію сутностей в системах домену застосування, включаючи сенсори та сутності, що моніторяться. За допомогою онтології системи ми можемо визначити архітектуру системи, яка правильно описує цільову систему та використовувати її для визначення, який сенсор моніторить яку сутність у системі та кореляцій між сутностями в цільовій системі. Онтологія метрики визначає різні метрики, які можуть використовуватися в домені застосування та їх взаємозв'язки. Онтологія одиниць вимірювання визначає одиниці, які можуть використовуватися для різних метричних показників у домені застосування та їх взаємозв'язки. Відповідно, потоки даних можуть бути визначені, зв'язавши їх з конкретними метриками, конкретними сутностями та конкретними одиницями вимірювання.

TSDB у Інфраструктурі IoT-Edge-Cloud

IoT-пристрої розгортаються зі зростаючою швидкістю і оцінюється, що є десятки мільярдів фізичних об'єктів, які підключені до Інтернету, і ця кількість продовжує швидко зростати. Багато з цих IoT-пристроїв - це датчики, які генерують великий обсяг даних. Загалом, IoT-пристрої мають обмежену обчислювальну потужність та місце зберігання і, таким чином, не підходять для постійного зберігання та обробки даних, які вони самі генерують. Хмара надає величезні обчислювальні та зберігальні можливості, які можна використовувати для обробки та зберігання даних IoT. Однак, багато IoT-систем збирають дані для прийняття рішень в реальному часі, а затримки зв'язку для передачі даних до централізованих хмарних дата-центрів може бути завеликою.

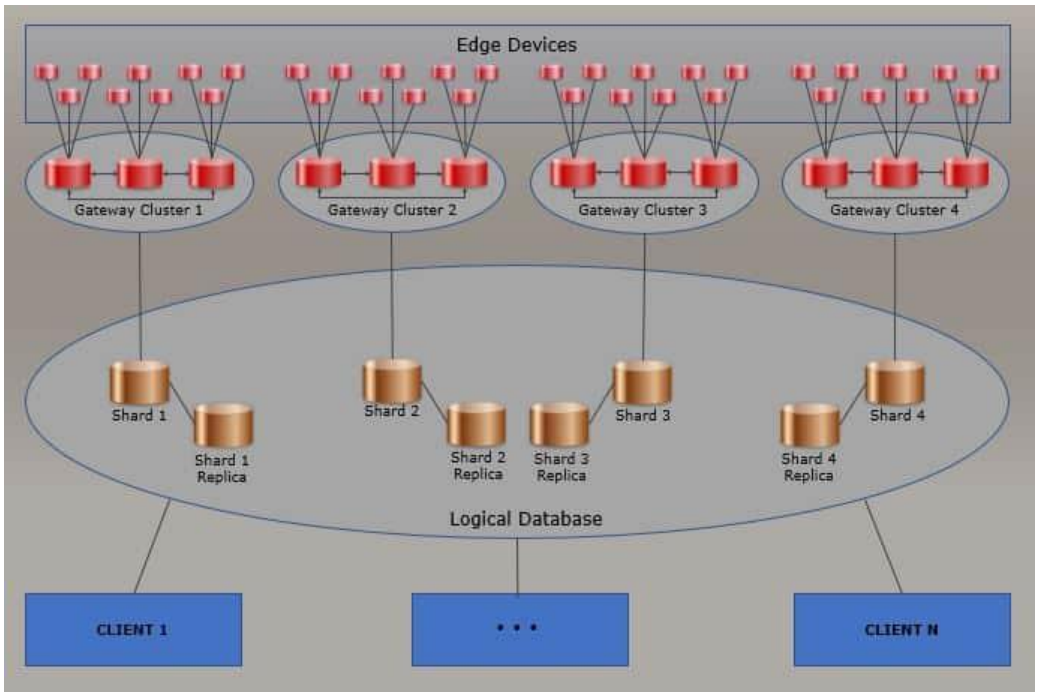

Рисунок 2.6 – Архітектура розподіленої хмари з вузлами граничних обчислень

Для вирішення цих компромісів були запропоновані рішення для розподіленого та туманного обчислення [23, 110]. Крайові вузли, як правило, служать шлюзом для IoT-пристроїв та можуть виконувати певні попередні обчислення. Інфраструктура IoT-Edge-Cloud проілюстрована на рис. 2.6.

Інфраструктура IoT-Edge-Cloud становить велику масштабну розподілену платформу для обробки та зберігання даних IoT. Для забезпечення балансу між своєчасністю та потужностями зберігання, дозволитить пристроям IoT зберігати тимчасові дані в граничних вузлах, які потрібні для швидкого прийняття рішень, а хмара - постійні дані. Потоки даних від пристроїв IoT можуть направлятися до граничних вузлів для попередньої обробки. Оригінальні або оброблені дані можуть далі перетікати до хмари для інтегрованої обробки та можливого зберігання. У такій інфраструктурі для зберігання даних можуть використовуватись TSDB. Кожен граничний вузол може запускати TSDB-екземпляр для одного вузла та керувати даними як рівноправний партнер.

Основна проблема з розподіленим сховищем в інфраструктурі полягає в тому, як керувати запитами для отримання необхідних даних. Щоб полегшити

інфраструктурне виявлення, будь то виявлення конкретного потоку даних або даних часового ряду, нам потрібно скласти структуровану інформацію для керування маршрутизацією запитів. Якщо явно не прописати джерело отримання даних, потрібен окремий агент який буде виконувати функцію пошуку даних та виконувати обчислювальні операції передбачені функціоналом баз даних часових рядів.

З розвитком інтернету речей з'являється все більше і більше підключених до Інтернету пристроїв, що генерують великі обсяги даних. Хоча хмарні технології забезпечують велику обчислювальну та зберігальну потужність, вони не завжди підходять для обробки та зберігання даних IoT, особливо для рішень, які потрібно приймати в реальному часі. Тому були запропоновані інші рішення для розподіленого обчислення з використанням граничних вузлів, які можуть виконувати певні попередні обчислення та забезпечувати баланс між своєчасністю та потужностями зберігання.

Інфраструктура IoT-Edge-Cloud є великою масштабною розподіленою платформою для обробки та зберігання даних IoT. Для керування запитами на отримання необхідних даних потрібна структурована інформація для керування маршрутизацією запитів, щоб полегшити інфраструктурне виявлення. В такій інфраструктурі кожен граничний вузол може запускати TSDB-екземпляр для одного вузла та керувати даними як рівноправний партнер.

Висновки до розділу 2

У даному розділі розглянуто класифікації та групування серверних ресурсів шляхом їх кластеризації, що є важливою стратегією оптимізації роботи серверних інфраструктур. Здійснено аналіз принципів формування кластерів та їх взаємодії для забезпечення ефективного розподілу завдань та ресурсів, що сприяє підвищенню продуктивності та надійності систем.

Детально розглянуто архітектурні аспекти побудови кластерних систем, враховуючи взаємодію серверів, мережі та зберігання даних. Акцентовано увагу на впровадженні резервування та масштабованості для забезпечення оптимальної продуктивності та гнучкості в умовах зростаючих обчислювальних завдань.

Досліджено розподілену архітектуру Інтернету речей (IoT), яка є ключовим аспектом сучасних технологій зв'язку та обчислень. Зосереджено увагу на взаємодії та інтеграції різноманітних пристроїв та компонентів, розподілених у глобальній мережі. Визначено проблематику управління ресурсами в системах Інтернету речей, враховуючи велику різноманітність пристроїв та їх обмежені характеристики. Особливу увагу приділено використанню баз даних часових рядів у контексті архітектури граничних обчислень для систем IoT. З'ясовано значення цих баз даних у відстеженні та аналізі часово залежних даних в режимі реального часу. Розглянуто специфіку використання цих баз даних у граничних обчисленнях, де обробка даних відбувається на межі мережі, надаючи можливість ефективного контролю та оптимізації ресурсів пристроїв IoT.

РОЗДІЛ 3

РОЗРОБКА МЕТОДИКИ РОЗПОДІЛУ ЗБЕРІГАННЯ ДАНИХ В РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

3.1. Алгоритми розподілу навантаження баз даних

Для досягнення високої ефективності децентралізованої системи необхідно забезпечити оптимальну продуктивність всіх її компонентів.

Вертикальне масштабування, наприклад, за допомогою збільшення ємності дисків, обчислювальної потужності, об'єму пам'яті або кількості мережевих адаптерів, може тимчасово обійти деякі з обмежень, що виникають у системі. Однак для комерційних додатків, які підтримують велику кількість користувачів та операцій з величезними обсягами даних, необхідно досягти масштабованості без значних обмежень, тому вертикальне масштабування не завжди є оптимальним рішенням.

У процесі розробки децентралізованої системи доцільно ретельно розглянути альтернативні підходи до масштабування, такі як горизонтальне масштабування та інші методи оптимізації. Це дозволить забезпечити ефективну роботу системи, яка відповідає вимогам великої кількості користувачів та оптимальному управлінню великими об'ємами даних.

Горизонтальне масштабування баз даних має вирішувати одразу декілька задач [11, 12, 28]:

1) Масштабування дискового простору. Сховища даних для великих хмарних додатків зазвичай містять велику кількість інформації, обсяг якої з часом збільшується. Зазвичай обсяг дискового простору на сервері обмежений. Хоча ви можете замінювати існуючі диски на

більш ємні або додавати додаткові диски по мірі накопичення даних, рано чи пізно система досягне певної межі, понад яку ємність сховища на цьому сервері збільшити буде неможливо.

2) Обчислювальні ресурси. Хмарні додатки повинні підтримувати велику кількість одночасно працюючих користувачів, кожен з яких виконує запити для отримання даних із сховища даних. Один сервер, на якому розміщене сховище даних, може не забезпечити необхідної обчислювальної потужності для підтримки цього навантаження, що призведе до довгого часу відгуку для користувачів та частим збоям, оскільки додатки намагаються зберігати та отримувати дані. Можливо, можна додати пам'ять або оновити процесори, але система досягне обмеження, якщо більше не вдасться збільшити обчислювальні ресурси.
3) Пропускна спроможність мережі. Продуктивність сховища даних, що працює на ізольованому сервері, залежить також від швидкості, з якою сервер може отримувати запити та надсилати відповіді [53]. Коли обсяг мережевого трафіку перевищує пропускну спроможність мережі, до якої підключений сервер, виконання деяких запитів призводить до збоїв - це реальна ситуація.
4) Географія. Іноді необхідно, щоб створювані користувачами дані зберігалися в тому ж регіоні, де знаходяться самі користувачі. Це може бути пов'язано з законодавчими обмеженнями, стандартами або вимогами до продуктивності та припустимої затримки при доступі до даних. Якщо при цьому користувачі розподілені по різних країнах та регіонах, ви не зможете зберігати всі дані додатка в одному сховищі даних.

В рамках даного дослідження можливе розділення сховища даних на горизонтальні секції, що дозволяє створювати сегментовану структуру даних. Кожен сегмент має ідентичну структуру даних, проте зберігає різні підмножини інформації. Кожен сегмент є самостійним сховищем даних та

може включати інформацію про декілька типів об'єктів. Реалізація сегментованої структури передбачає виконання кожного сегмента на окремому сервері, який використовується як вузол зберігання.

Такий підхід пропонує деякі переваги [10, 19]:

- Масштабованість системи через можливість додавання нових сегментів, що працюють на додаткових серверах зберігання.
- Зниження вартості і обмежень щодо обладнання завдяки використанню стандартних серверів замість спеціалізованих та витратних комп'ютерів для кожного вузла зберігання.
- Мінімізація конфліктів та підвищення продуктивності завдяки раціональному розподілу робочого навантаження між сегментами.
- Оптимізація доступу до даних завдяки фізичному розміщенню сегментів у хмарних областях ближче до кінцевих користувачів, які взаємодіють з цими даними.

Для досягнення оптимального балансу між продуктивністю та масштабованістю вкрай важливо належним чином розподілити дані відповідно до типів запитів, що виконує додаток. У багатьох випадках схеми сегментування не зможуть точно відповідати вимогам кожного окремого запиту. Наприклад, в мультитенантній системі додатка може потрібно отримати дані клієнта за його ідентифікатором, а також здійснювати пошук цих даних за іншим атрибутом, таким як ім'я чи місцезнаходження клієнта. У таких випадках бажано використовувати стратегію сегментування, де ключ сегмента відповідає найчастіше виконуваним запитам.

Якщо потрібно регулярно отримувати дані, використовуючи комбінацію значень деяких атрибутів, можна визначити композитний ключ сегмента, що зв'язує необхідні атрибути. Також можна застосовувати інший підхід - використовувати шаблон таблиці індексів, який дозволяє швидко знаходити дані за значеннями атрибутів, які не входять до ключа сегмента.

При виборі ключа сегмента та правил розподілу даних по сегментах, зазвичай використовується одна з трьох стратегій. Варто зауважити, що між сегментами та серверами, на яких вони розміщені, не обов'язково повинно бути відповідність "один до одного" - один сервер може містити кілька сегментів.

Стратегія пошуку. В цьому варіанті логіка сегментування використовує карту відповідності за ключем сегмента та направляє запити до того сегмента, що містить необхідні дані. Наприклад, у багатокористувацькому додатку всі дані кожного клієнта можна зберігати в одному сегменті, тоді ідентифікатор клієнта можна використовувати як ключ сегмента. Дані декількох клієнтів можуть зберігатися в одному сегменті, але дані одного клієнта не можуть опинитися в декількох сегментах. На рисунку 3.1. показаний варіант сегментування даних за ідентифікаторами клієнтів.

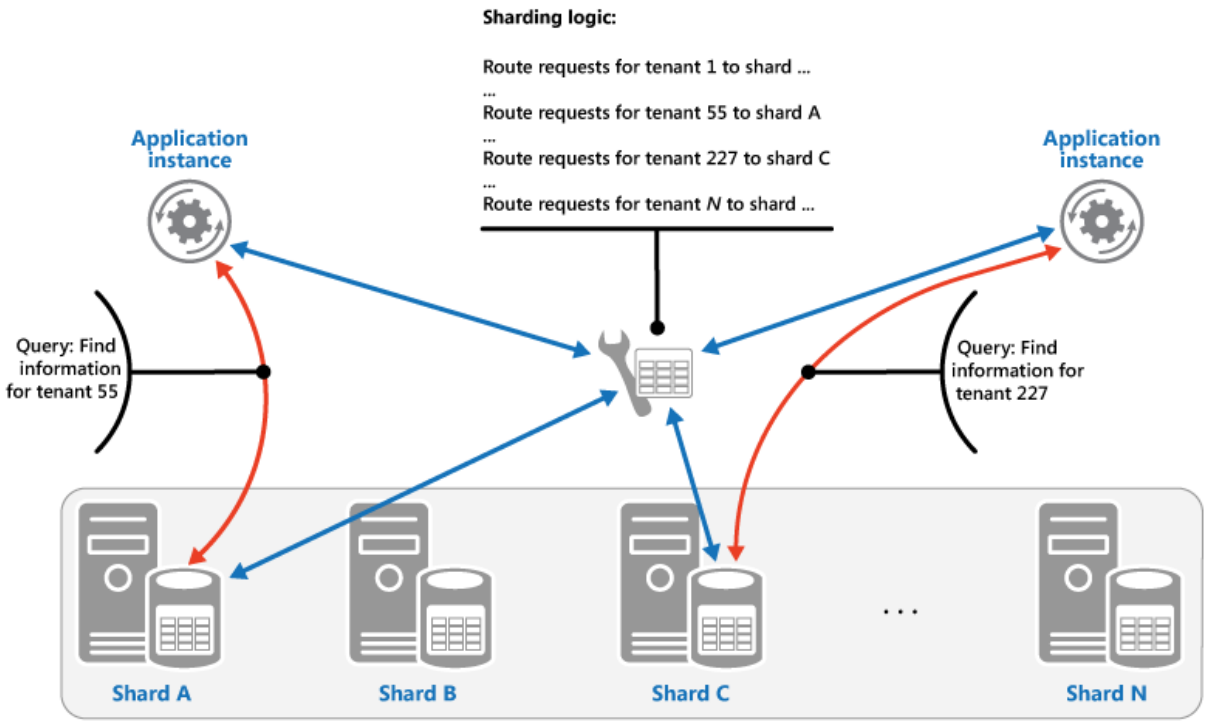

Рисунок 3.1 – Сегментування даних за ідентифікаторами клієнтів

Для зіставлення значень ключа сегмента та фізичного сховища можна використовувати фізичні сегменти, тоді кожен ключ сегмента буде

зіставлений з фізичною секцією. Існує і більш гнучкий спосіб розподілу сегментів - віртуальне секціонування. В цьому випадку ключі сегмента зіставляються з такою самою кількістю віртуальних сегментів, які в свою чергу зіставляються з меншою кількістю фізичних секцій. У цьому варіанті додаток знаходить дані за допомогою ключа сегмента, який посилається на віртуальний сегмент і система автоматично зіставляє віртуальні сегменти з фізичними секціями. Таке зіставлення між віртуальними сегментами та фізичними секціями можна змінити в будь-який момент, не застосовуючи програмні методи, а просто застосувавши інший набір ключів сегмента.

Стратегія діапазонів. Ця стратегія полягає у групуванні пов'язаних елементів в одному сегменті та їх упорядкуванні за ключем сегмента - ключі сегментів є послідовними. Це корисно для додатків, які часто отримують набори елементів з деякого діапазону значень (запити до даних повертають набір елементів, для яких значення ключа сегмента потрапляють в заданий діапазон).

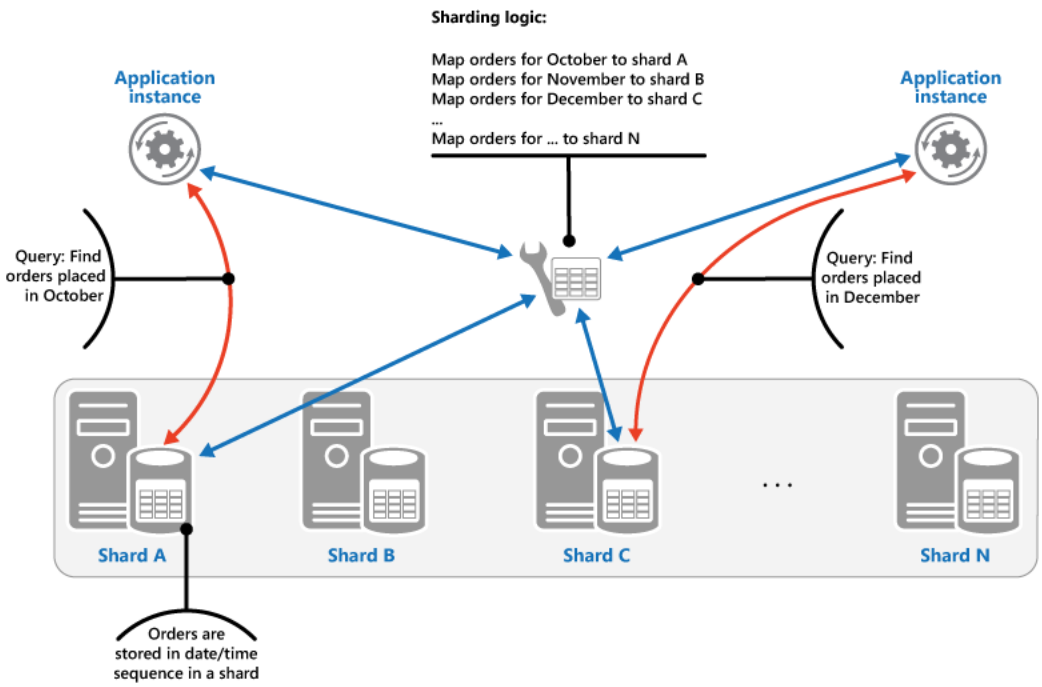

Рисунок 3.2 – Сегментування даних за діапазонами

Наприклад, якщо додатку регулярно потрібно отримувати список всіх замовлень, розміщених у певний місяць, ці дані можна отримувати швидше, розмістивши замовлення за кожний місяць в одному сегменті та упорядковуючи їх за датою та часом. Якщо замовлення розташовані в різних сегментах, їх доведеться витягати окремо, тобто виконувати велику кількість окремих запитів (повертаючих один елемент даних). На рисунку 3.2. показана схема зберігання послідовних наборів (діапазонів) даних в сегментах.

У цьому прикладі використовується складний ключ сегмента, в якому найбільш важливим елементом є місяць замовлення, а за ним слідують день та час замовлення. При створенні нових замовлень та розміщенні їх по сегментах дані натуральним чином сортуються. Деякі сховища даних підтримують ключі сегмента, які складаються з двох елементів - ключа секції, що визначає сегмент, і ключа рядка, що однозначно визначає елемент в межах сегмента. Зазвичай дані в сегменті зберігаються в порядку значень ключа рядка. Елементи, для яких застосовуються запити за діапазонами значень і які потрібно зберігати в одному сегменті, матимуть однакові значення ключа секції та унікальні значення ключа рядка.

Стратегія хешування. Ця стратегія дозволяє знизити ймовірність виникнення високоактивних сегментів (сегментів з надмірно високим навантаженням). В цьому варіанті дані розподіляються між сегментами так, щоб дотримувалась рівновага між розміром кожного сегмента та середнім навантаженням, що припадає на цей сегмент. Логіка сегментації визначає, в якому сегменті зберігати кожний елемент даних за значенням хеш-індексу за одним або кількома атрибутами даних.

Функція хешування повинна бути обрана так, щоб рівномірно розподілити дані між сегментами. Можливо, для цього доведеться додати до формули обчислення деякий випадковий елемент. На рисунку 3.3. зображено сегментацію даних за хешем ідентифікаторів клієнтів.

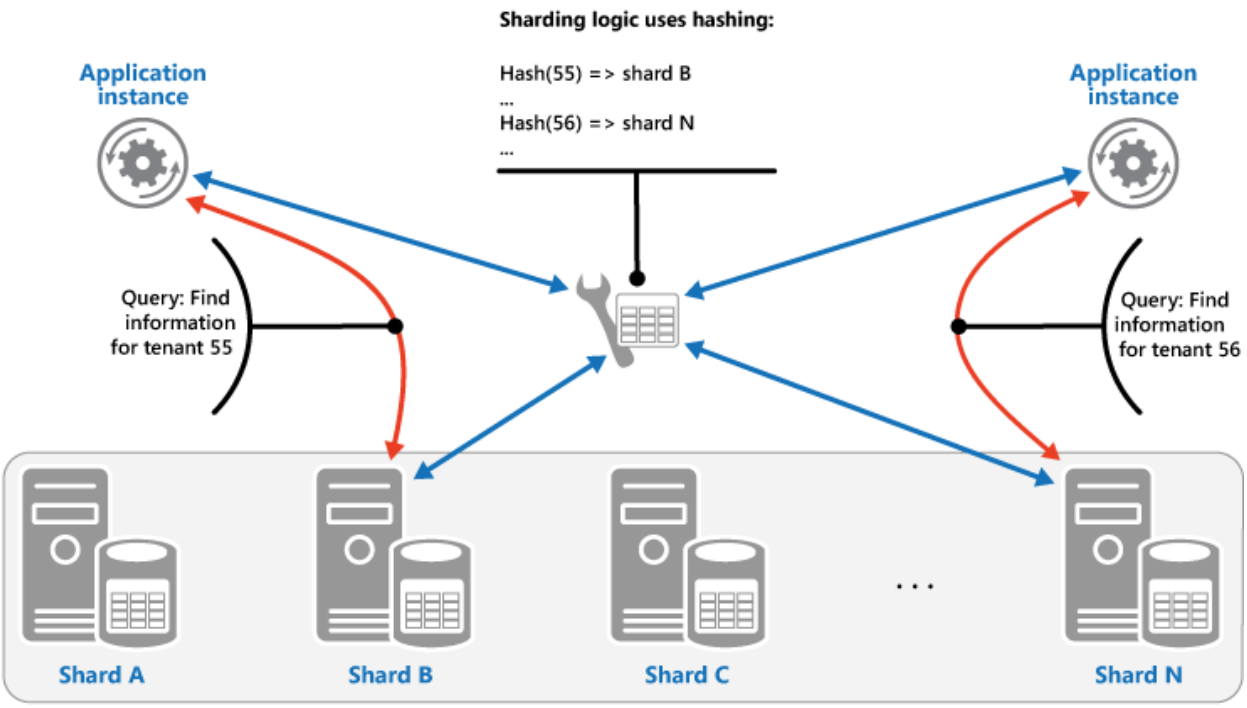

Рисунок 3.3 – Сегментування даних за діапазонами

Щоб зрозуміти переваги стратегії хешування порівняно з іншими стратегіями сегментації, розглянемо наступний приклад. Багатокористувацький додаток по черзі реєструє нових клієнтів та розподіляє отримані дані між сегментами у сховищі даних. Якщо використовувати стратегію діапазонів, всі дані про клієнтів з ідентифікаторами від 1 до n зберігаються в сегменті A, з ідентифікаторами від n+1 до m - в сегменті B та ін. Якщо припустити, що найбільшу активність проявляють недавно зареєстровані клієнти, найбільше навантаження з роботи з даними припаде на невелику кількість сегментів, що створить високоактивний сегмент. В цій же ситуації стратегія хешування буде розподіляти клієнтів по сегментах на основі хешу ідентифікатора. Це означає, що клієнти з послідовними ідентифікаторами, ймовірно, потраплять в різні сегменти, що дозволить рівномірно розподілити навантаження. На рисунку вище показано таку поведінку для клієнтів з ідентифікаторами 55 та 56.

Переваги та особливості кожної з трьох основних стратегій сегментування.

Стратегія пошуку. Ця стратегія надає максимальний контроль над налаштуванням та використанням сегментів. Використання віртуальних сегментів дозволяє усунути проблеми з балансуванням даних завдяки можливості додавати нові фізичні секції для розподілу робочого навантаження. Відповідність віртуальних сегментів та фізичних секцій, на яких вони реалізовані, можна змінювати, не застосовуючи програмні методи, тобто не змінюючи код додатка, який зберігає та отримує дані за значенням ключа сегмента. Пошук розташування сегмента може створювати додаткове навантаження.

Стратегія діапазонів. Легко реалізовується та добре працює з запитами за діапазонами, оскільки часто дозволяє за допомогою однієї операції отримати кілька елементів даних з одного сегмента. Ця стратегія спрощує процеси управління даними. Наприклад, якщо включити в один сегмент користувачів з одного регіону, можна запланувати оновлення окремо для кожного часового поясу на основі локальних особливостей використання навантаження та запитів. Але ця стратегія не дозволяє забезпечити оптимальне балансування між сегментами. Перерозподіл даних між сегментами буде складним і не вирішить проблему нерівномірного навантаження, якщо основна частина навантаження створюється для даних з близькими значеннями ключа сегмента.

Стратегія хешування. Дозволяє розподіляти дані та навантаження найбільш рівномірно. Маршрутизацію запитів виконує прямо з додатка за допомогою хеш-функції. Немає необхідності використовувати карту відповідності. Обчислення хешу може створити додаткове навантаження. Крім того, це ускладнить перерозподіл навантаження між сегментами.

Операції масштабування та переміщення даних. Кожна стратегія сегментації створює різні можливості та різні рівні складності у контексті зміни масштабу, переміщення даних та підтримки стану системи.

Стратегія пошуку дозволяє виконувати масштабування та переміщення даних на рівні користувачів, і для цього не потрібно вимикати систему. Достатньо тимчасово призупинити операції користувачів або їхню частину (це можна зробити в періоди низького навантаження), потім перемістити дані в інший віртуальний сегмент або інший фізичний сегмент, змінити карту відповідностей та оголосити недійсними або оновити всі кешовані дані. Після цього можна відновити роботу користувачів. Операції такого типу часто можна виконувати централізовано. Стратегія пошуку потребує узгодженості даних із кешуванням та реплікацією.

Стратегія діапазонів накладає деякі обмеження на операції масштабування та переміщення даних. Зазвичай їх слід виконувати, відключивши з мережі сховище даних або його частину, оскільки дані потрібно ділити та об'єднувати в багатьох сегментах. Переміщення даних для балансування сегментів не завжди дозволяє вирішити проблему нерівномірного навантаження, якщо основна частина навантаження створюється для даних з близькими значеннями ключа сегмента або ідентифікаторами із одного діапазону. Стратегія діапазонів також може включати дії по підтримці стану, щоб діапазони можна було відповідним чином зіставити з фізичними секціями.

Стратегія хешування значно ускладнює операції масштабування та переміщення даних, оскільки ключі секцій — це хеш-індекси ключів сегментів або ідентифікаторів даних. Потрібно визначити нове розташування для кожного сегмента на основі результатів виконання хеш-функції або змінити цю функцію для правильної відповідності. Однак стратегія хешування не потребує підтримки стану.

Проблеми шаблонів сегментації баз даних.

- Складність і ризики переміщення даних: При використанні деяких стратегій сегментації, таких як стратегія діапазонів або

хешування, переміщення даних в інший сегмент або сервер може бути складним та ризикованим. Це може призвести до тимчасової недоступності даних або втрати їх цілісності під час процесу переміщення.

- Нерівномірне навантаження: Деякі шаблони сегментації можуть призводити до нерівномірного розподілу навантаження між сегментами бази даних. Це може спричинити перевантаження деяких сегментів, тоді як інші залишатимуться недостатньо використаними, що може погіршити продуктивність та завдати шкоди користувачам.
- Поділ транзакцій: При використанні деяких стратегій сегментації, наприклад, коли дані розподілені між різними фізичними серверами, виникає проблема поділу транзакцій між сегментами. Це може збільшити складність управління цілісністю даних та збільшити ймовірність виникнення проблем з одночасним доступом до даних.
- Перевищення обсягу даних: Іноді стратегії сегментації можуть зазнавати проблем з обмеженням максимального обсягу даних, який може бути збережений в кожному сегменті. Це може створювати обмеження щодо розміру бази даних, що може змінюватись з часом і приводити до необхідності переструктуризації сегментів.
- Складність управління станом: При використанні деяких стратегій сегментації, зокрема стратегії діапазонів, необхідно відстежувати та управляти станом сегментів та їх відповідностями до фізичних серверів. Це може бути викликом при масштабуванні та підтримці системи.

Вибір конкретного шаблону сегментації баз даних повинен здійснюватися з урахуванням особливостей конкретної системи та вимог до

даних, а також має бути підтриманим належним контролем, щоб уникнути проблем у майбутньому.

3.2. Постановка задачі

Кожна з стратегій сегментації баз даних має спрямованість на конкретного користувача та враховує особливості навантаження, що унеможливлює створенню адаптивної системи на основі цих стратегій. Однак, незважаючи на переваги, кожна з цих стратегій виявляє окремі недоліки, які накопичуються з часом і потребують періодичного втручання людини для їх вирішення. Деякі з цих проблем включають:

1) Переміщення даних через нерівномірне заповнення сегментів. При використанні певних стратегій сегментації може виникати нерівномірність заповнення сегментів бази даних. Це може призвести до надмірного заповнення деяких сегментів, тоді як інші залишаються недостатньо заповненими. В таких випадках необхідне переміщення даних для забезпечення балансу між сегментами.
2) Переміщення даних через нерівномірність навантаження на сегментах. Деякі стратегії сегментації можуть створювати нерівномірне розподілення навантаження на різних сегментах бази даних. Це може призвести до перевантаження деяких сегментів, тоді як інші залишаються менше навантаженими. Для оптимізації продуктивності системи необхідне переміщення даних або перерозподіл навантаження.
3) Створення реплікацій для розподілення навантаження. Для підтримки збалансованого розподілу навантаження деякі стратегії сегментації баз даних вимагають створення реплікацій даних. Це може призводити до збільшення обсягу даних та збільшення вимог до

обчислювальних ресурсів.

4) Обчислення нових параметрів для алгоритмів розподілу збереження даних. При зміні стратегії сегментації або розмірів сегментів може виникнути необхідність в перерахунку параметрів для алгоритмів розподілу та збереження даних.

Застосування технологій штучного інтелекту (ШІ) для розподілу навантаження та управління збереженими даними визначає перспективний напрямок розвитку сучасних баз даних. ШІ може ефективно вирішити проблеми нерівномірного розподілу навантаження на сегментах бази даних, а також усунути нерівномірність збережених даних між ними.

Для розподілу навантаження, ШІ може використовувати аналіз поведінки користувачів, їхніх запитів до бази даних та обробляти цю інформацію з метою ефективного розподілу завдань між сегментами. За допомогою методів машинного навчання, ШІ може прогнозувати збільшення навантаження та адаптувати розподіл даних для оптимального використання ресурсів.

Для усунення нерівномірності збережених даних, ШІ може застосовувати алгоритми балансування даних, які враховують обсяг та типи інформації у кожному сегменті. Це дозволяє забезпечити рівномірний розподіл даних та уникнути перевантаження або недостатньої заповненості сегментів.

Застосування ШІ в розподілі навантаження та управлінні даними вимагає впровадження складних алгоритмів та систем управління. Однак, переваги цього підходу полягають у здатності адаптуватися до змінного навантаження та забезпечувати оптимальний доступ до даних для користувачів. Додатково, використання ШІ може підвищити продуктивність

та надійність баз даних, що робить його привабливим для впровадження в сучасних інформаційних системах.

Застосування баз даних часових рядів є важливим аспектом для систем Інтернету речей (IoT), які збирають, зберігають та аналізують великі обсяги даних, що залежать від часу. Ці системи IoT включають в себе різноманітні пристрої та сенсори, які постійно збирають дані про оточуюче середовище та діяльність користувачів.

Збір та аналіз даних у режимі реального часу є однією з головних вимог для систем IoT. Бази даних часових рядів спеціально розроблені для ефективного збереження та обробки даних, де кожен запис містить відмітку часу, що відображає момент отримання даних. Це дозволяє точно відслідковувати динаміку змін даних в часі, а також здійснювати аналіз тенденцій та прогнозування.

Завдяки базам даних часових рядів системи IoT можуть ефективно зберігати потокові дані великого обсягу та забезпечувати швидкий доступ до них для аналізу та прийняття рішень у режимі реального часу. Такі дані можуть бути зібрані від різних джерел, таких як сенсори з мережі розподіленого збору даних або пристрої IoT.

При використанні баз даних часових рядів в системах IoT, аналітичні можливості розширюються, що дозволяє використовувати різноманітні методи прогнозування, машинного навчання та статистичного аналізу для виявлення аномалій, попередження про небезпечні події та оптимізації процесів у реальному часі. Використання таких баз даних також сприяє зниженню затримок при доступі до даних та забезпечує високу доступність і надійність систем IoT.

Таким чином, використання баз даних часових рядів стає ключовим фактором для розвитку систем Інтернету речей, допомагаючи забезпечити збір, збереження та аналіз великого обсягу даних, що залежать від часу, з

метою забезпечення ефективного управління та використанням інформації в реальному часі.

Застосування штучного інтелекту (ШІ) для керування системою баз даних часових рядів в системах Інтернету речей (IoT) забезпечить створення інтелектуальних та автономних систем, що забезпечують ефективне та оптимізоване управління даними залежно від умов і потреб, що змінюються.

Основні аспекти застосування ШІ для керування базами даних часових рядів в системах IoT включають:

- Прогнозування та аналіз динаміки даних: Застосування ШІ дозволяє прогнозувати та аналізувати динаміку даних в базах часових рядів. ШІ може виявляти тенденції, зміни та аномалії в часових рядах, що допомагає забезпечити вчасне реагування на зміни у навколишньому середовищі та виконання важливих завдань.
- Оптимізація роботи системи: ШІ може визначати оптимальні параметри роботи баз даних часових рядів залежно від зміни навантаження та обсягу даних. Використовуючи методи машинного навчання, ШІ може адаптувати конфігурацію баз даних, забезпечуючи максимальну продуктивність та ефективність роботи системи.
- Управління ресурсами: ШІ може управляти розподілом ресурсів системи, забезпечуючи оптимальне використання обчислювальної потужності та забезпечення надійності баз даних.

Використання штучного інтелекту для керування базами даних часових рядів у системах Інтернету речей веде до створення автономних, ефективних та інтелектуальних систем, які здатні ефективно адаптуватися умов і навантаження, що змінюються. Це сприяє забезпеченню надійності,

ефективності та оптимізації функціонування систем IoT, що робить їх привабливими для впровадження в різноманітних галузях [112].

3.3. Визначення типу нейронної мережі

Нейронні мережі представляють собою глибоке навчання з використанням штучного інтелекту. Штучні нейронні мережі надихаються біологічними нейронами в людському організмі, які активуються в певних обставинах, що призводить до відповідних дій здійснених організмом у відповідь. Штучні нейронні мережі складаються з різних шарів взаємопов'язаних штучних нейронів, які працюють завдяки функціям активації, що допомагають їх вмикати або вимикати. Так само, як традиційні алгоритми машинного навчання, тут також є певні значення, які нейронні мережі вивчають під час фази навчання [18, 24, 30, 56, 73].

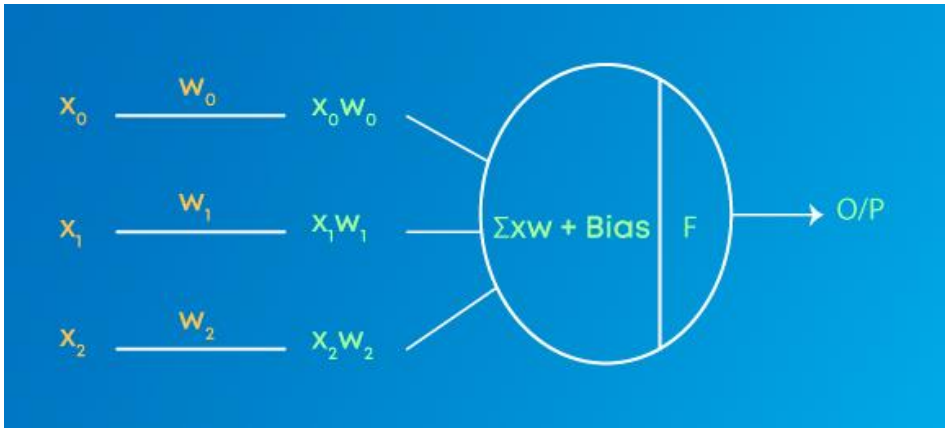

Рисунок 3.4 – Базова архітектура роботи нейронної мережі

Кожен нейрон отримує перемножену версію вхідних даних і випадкові ваги, які потім додаються до статичного значення зсуву (унікального для кожного шару нейронів); потім це значення передається відповідній функції активації, яка визначає остаточне значення, що видається нейроном. Існує різноманітність функцій активації, які використовуються залежно від природи вхідних значень. Після того, як виведення здійснюється з остаточного шару

нейронної мережі, обчислюється функція втрат (вхід проти виходу), і виконується процес зворотного поширення, де ваги коригуються для зниження помилки. Знаходження оптимальних значень ваг є головною метою цієї операції [72, 75, 81].

Ваги - це числові значення, які множаться на вхідні дані. Під час зворотнього поширення, вони модифікуються з метою зменшення втрат. Простими словами, ваги є значеннями, що вивчаються машинами з Нейронних Мереж. Вони самостійно коригуються в залежності від різниці між передбаченими виходами та навчальними входами.

Функція активації - це математична формула, яка допомагає нейрону ввімкнутися або вимкнутися.

Базові складові нейронних мереж [94]:

- Вхідний шар представляє розміри вхідного вектору.
- Прихований шар представляє проміжні вузли, які ділять простір вхідних даних на регіони з межами. Він отримує набір зважених вхідних даних та виробляє вихід через функцію активації.
- Вихідний шар представляє вихід нейронної мережі.

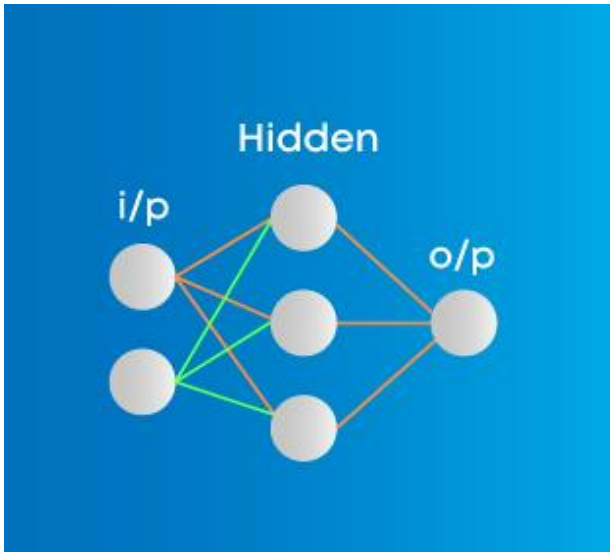

Рисунок 3.5 – Базова архітектура нейронної мережі

Архітектура Персептрона

Модель персептрона є однією з найпростіших і найдавніших моделей нейрона. Це найменша одиниця нейронної мережі, яка здійснює певні обчислення для виявлення ознак або бізнес-інтелекту по вхідних даних. Вона приймає зважені входи і застосовує функцію активації для отримання виходу як кінцевого результату. Персептрон також відомий як TLU (логічна одиниця з порогом).

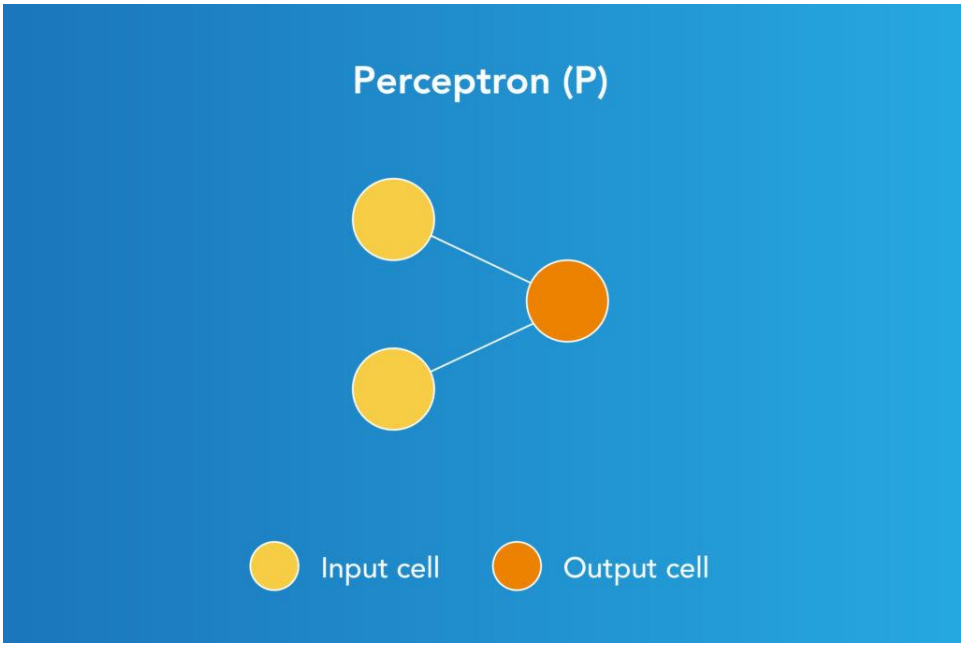

Рисунок 3.6 – Архітектура персептрона

Персептрон - це алгоритм навчання з учителем, який класифікує дані на дві категорії, тому він є бінарним класифікатором. Персептрон розділяє простір входів на дві категорії, яку представляє наступне рівняння:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \tag{3.1}$$

Переваги персептрону

Персептрони можуть реалізовувати логічні вентелі, такі як І (AND), АБО (OR) або ВІДМІННІСТЬ (NAND).

Недоліки персептрону

Персептрони можуть вивчати лише лінійно роздільні проблеми, такі як логічна операція AND. Для нелінійних проблем, таких як логічна операція XOR, вони не працюють.

Нейронні мережі прямого поширення

Це найпростіша форма нейронних мереж, де вхідні дані рухаються лише в одному напрямку, проходять через штучні нейронні вузли і виходять через вихідні вузли. Тут можуть або не можуть бути приховані шари, проте вхідний і вихідний шари завжди присутні. Залежно від цього, їх можна поділити на одношарові або багатошарові нейронні мережі прямого поширення [54, 90, 91].

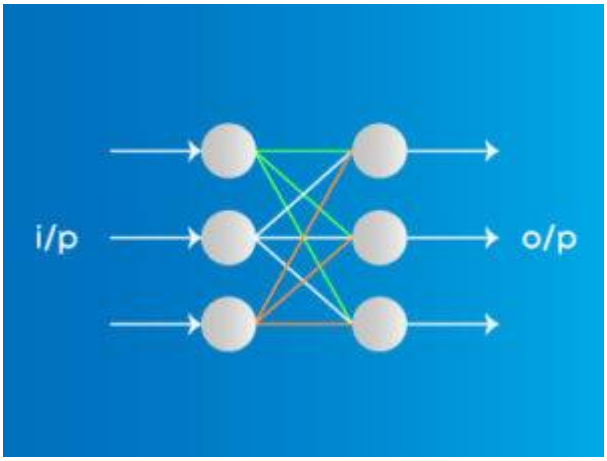

Рисунок 3.7 – Архітектура мережі прямого поширення

Кількість шарів залежить від складності функції. Тут присутній лише односторонній прямий шлях, і немає зворотнього поширення. Ваги тут є статичними. Функція активації отримує вхідні дані, які множаться на ваги. Для цього використовується функція активації класифікації або функція активації кроку. Наприклад: Нейрон активується, якщо його значення перевищує поріг

(зазвичай 0), і нейрон видає 1 як вихід. Нейрон не активується, якщо його значення менше порогового значення (зазвичай 0), що вважається -1. Вони досить прості у підтримці та добре впораються з даними, що містять багато шуму.

Переваги нейронних мереж прямого поширення:

- Швидкі та ефективні (Прямий односпрямований процес).
- Добре реагують на шумні дані.

Недоліки нейронних мереж прямого поширення:

- Не можуть використовуватися для глибокого навчання, через відсутність щільних шарів та зворотного поширення.

Багатошаровий персептрон

Це точка входу до складних нейронних мереж, де вхідні дані проходять через різні шари штучних нейронів. Кожен окремий вузол з'єднаний з усіма нейронами наступного шару, що робить його пов'язаною нейронною мережею. Присутній вхідний і вихідний шари, а також кілька прихованих шарів, тобто принаймні три або більше шарів загалом. Він має двостороннє поширення, тобто прямий та зворотний процеси поширення.

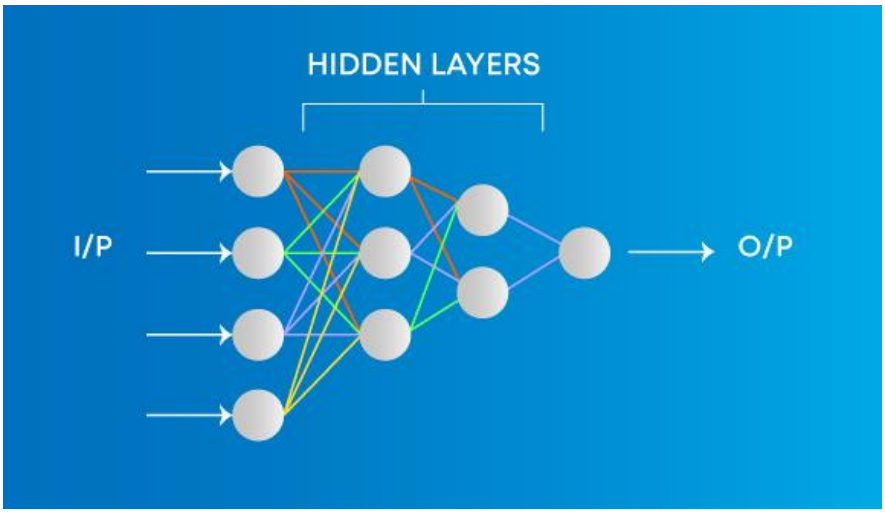

Рисунок 3.8 – Архітектура багатошарового персептрона

Входи множаться на ваги і передаються функції активації, а під час зворотного поширення вони модифікуються з метою зменшення втрат. Ваги є значеннями, які навчаються машинами з нейронних мереж. Вони самостійно коригуються в залежності від різниці між передбаченими виходами та навчальними входами. Використовуються нелінійні функції активації, за якими слідує функція активації softmax для вихідного шару.

Переваги багатошарового персептрона:

- Використовується для глибокого навчання, завдяки наявності щільних повністю з'єднаних шарів і зворотного поширення.

Недоліки багатошарового персептрона:

- В порівнянні складний у проектуванні та підтримці.
- Порівняно повільний (залежить від кількості прихованих шарів).

Згорточна нейронна мережа (Convolutional Neural Network - CNN) - це тип нейронної мережі, яка спеціалізується на обробці зображень та інших типів даних з високим розміром просторових структур. Вона була інспірована механізмами візуальної перцепції у мозку людини[117].

Основна ідея застосування згорткових шарів у мережі полягає у використанні фільтрів або ядер для здійснення згортки зображення, тобто поелементного перемноження пікселів вхідного зображення з відповідними елементами фільтра та підсумовування результатів. Це дозволяє виокремлювати різноманітні ознаки (наприклад, риси обличчя, краї, текстури тощо) з вхідних даних.

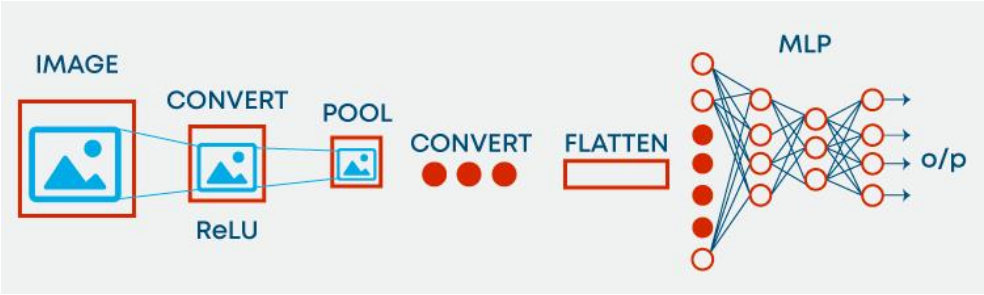

Рисунок 3.9 – Архітектура згорточної нейронної мережі

Згорткова нейронна мережа має також пулінгові шари, які зменшують розмір вихідних даних та роблять їх більш узагальненими. Ці шари здійснюють підсумовування або інші операції на певних областях даних, що дозволяє знизити кількість параметрів та обчислювальний витрат у мережі.

Переваги згорткової нейронної мережі:

- Використовується для глибокого навчання з невеликою кількістю параметрів.
- Менше параметрів для навчання порівняно з повністю з'єднаним шаром.

Недоліки згорткової нейронної мережі:

- Порівняно складний у проектуванні та підтримці.
- Порівняно повільний (залежить від кількості прихованих шарів).

Мережа з радіально-базисними функціями (Radial Basis Function Network) складається з вектору вхідних даних, за яким слідує шар нейронів та вихідний шар з одним вузлом на кожну категорію. Класифікація виконується шляхом вимірювання подібності вхідних даних до точок даних з навчального набору, де кожен нейрон зберігає прототип.

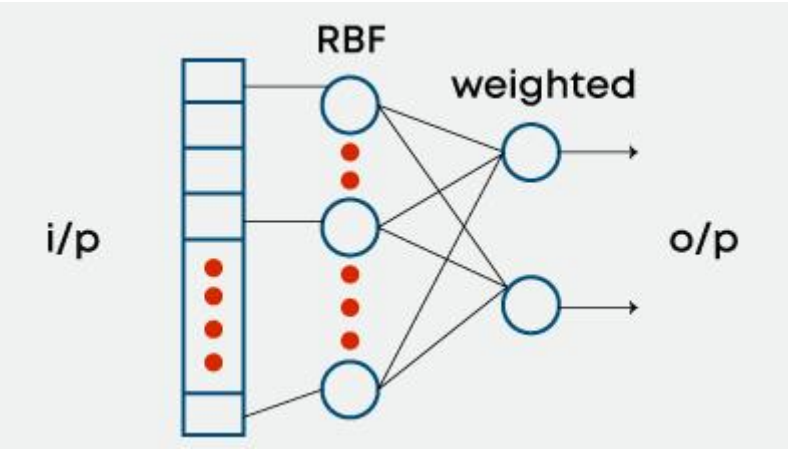

Рисунок 3.10 – Архітектура мережі з радіально-базисними функціями

Коли необхідно класифікувати новий вектор вхідних даних, кожен нейрон обчислює евклідову відстань між вхідними даними та його прототипом. Наприклад, якщо у нас є два класи, тобто клас A і клас B, тоді нові вхідні дані, які потрібно класифікувати, більше подібні до прототипів класу A, ніж до прототипів класу B. Тому вони можуть бути позначені або класифіковані як клас A.

Кожен нейрон порівнює вхідний вектор з його прототипом і виводить значення, яке знаходиться в діапазоні від 0 до 1, що вимірює подібність. Коли вхід дорівнює прототипу, вихід цього нейрона буде 1, а збільшення відстані між вхідними даними та прототипом призводить до експоненційного зниження відповіді до 0. Вихідний шар складається з набору нейронів (один на кожну категорію).

Рекурентні нейронні мережі

Рекурентна нейронна мережа має архітектуру, в якій вихідний результат шару повертається назад на вхід для допомоги в передбаченні результату цього шару [26, 77, 78]. Зазвичай перший шар - це звичайна з односторонньою передачею вперед (feed-forward) нейронна мережа, за яким слідує шар рекурентних нейронів, де певна інформація, яка була у попередньому часовому кроці, запам'ятовується за допомогою функції пам'яті. У цьому випадку реалізовано передачу вперед (forward propagation). Рекурентна нейронна мережа зберігає інформацію, необхідну для майбутнього використання. Якщо передбачення неправильне, застосовується швидкість навчання, щоб здійснити невеликі зміни. Таким чином, результативність з поступовим зростанням до правильного передбачення під час зворотного поширення (backpropagation).

Переваги рекурентних нейронних мереж (RNN):

- Моделює послідовні дані: Однією з переваг є здатність моделювати послідовні дані, де кожна вибірка можна вважати

залежна від попередніх.

Недоліки рекурентних нейронних мереж (RNN):

- Проблеми з зникненням градієнту, що ускладнює процес навчання.
- Важко обробляти довгі послідовні дані.

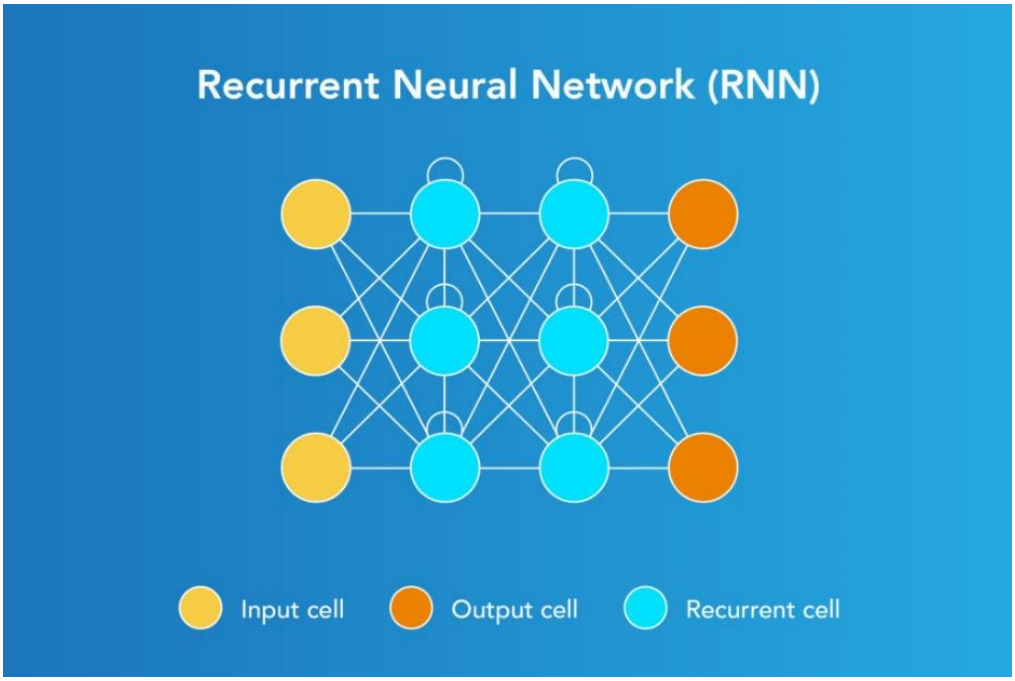

Рисунок 3.11 – Архітектура рекурентні нейронні мережі

Надбудова над RNN: Мережі LSTM (Long Short-Term Memory)

Мережі LSTM [88] - це тип RNN, що використовує спеціальні блоки (units) в додаток до стандартних блоків. Модуль LSTM включає «клітинку пам'яті», яка може зберігати інформацію у пам'яті на тривалий час. Для контролю над тим, коли інформація вводиться у пам'ять, коли вона виводиться та коли вона забувається, використовується набір вентилів (gates). Є три типи вентелів: во вводу, виводу та забування. Вони дозволяють моделі вивчати більш довгострокові залежності у послідовних даних.

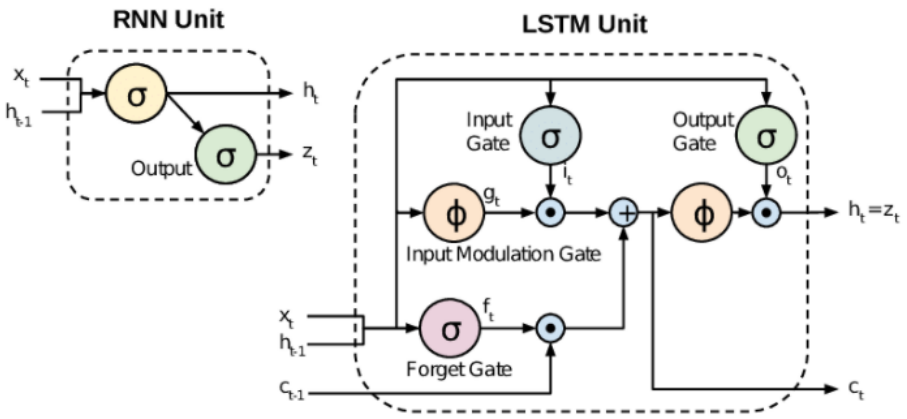

Рисунок 3.12 – Функціональна схема рекурентні нейронні мережі та LSTM

Моделі послідовності

Модель послідовності складається з двох рекурентних нейронних мереж. Тут існує кодер, який обробляє вхідні дані, і декодер, який обробляє вихідні дані. Кодер та декодер працюють одночасно - за допомогою одного набору параметрів або різних. У цій моделі, на відміну від звичайної рекурентної нейронної мережі (RNN), особливо застосовується тоді, коли довжина вхідних даних дорівнює довжині вихідних даних. Незважаючи на те, що вони мають схожі переваги та обмеження з RNN, ці моделі зазвичай застосовуються головним чином у чат-ботах, системах машинного перекладу та системах відповідей на запитання.

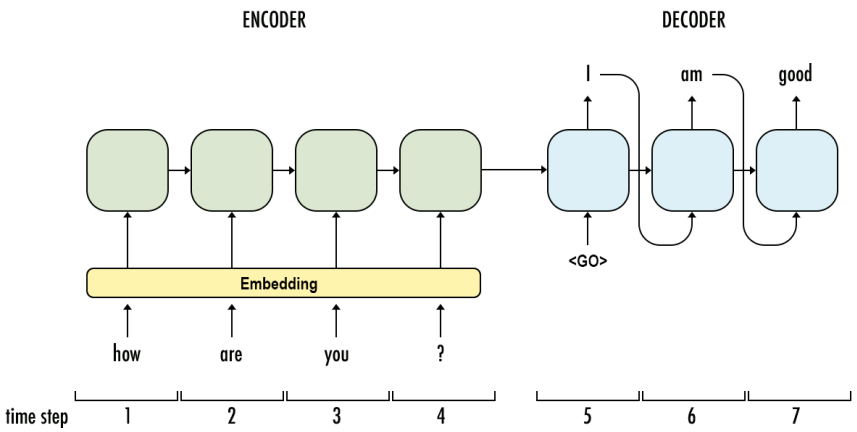

Рисунок 3.13 – Архітектура моделі послідовності

Модульна нейронна мережа

Модульна нейронна мережа має кілька різних мереж, які працюють незалежно одна від одної і виконують підзавдання. Різні мережі фактично не взаємодіють або не спілкуються одна з одною під час обчислювального процесу. Вони працюють незалежно одна від одної з метою досягнення вихідного результату [92, 96, 105].

Як результат, великий та складний обчислювальний процес виконується значно швидше завдяки розбиттю його на незалежні компоненти. Швидкість обчислення зростає, тому що мережі не взаємодіють або навіть не з'єднані між собою.

Переваги модульної нейронної мережі:

- Ефективність
- Незалежне навчання
- Надійність

Недоліки модульної нейронної мережі:

- Проблеми зі змінними цілями (Moving target Problems)

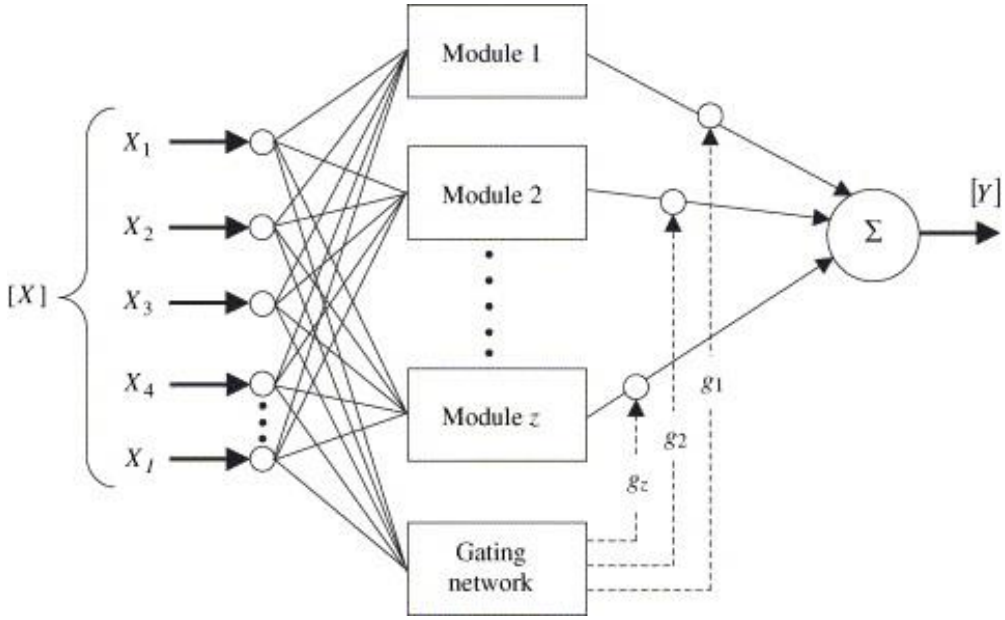

Рисунок 3.14 – Архітектура модульної нейронної мережі

Застосування архітектури довго-короткострокової пам'яті (LSTM) для вирішення завдань, пов'язаних із розподілом баз даних, де об'єднуються принципи машинного навчання та адміністрування баз даних з метою оптимізації розподілу ресурсів та підвищення продуктивності інфраструктури даних.

У цьому контексті, LSTM [22, 39, 97] - це вид рекурентної нейронної мережі, розробленої для обробки послідовних даних, яка, за відповідним використанням, може сприяти аналізу та оптимізації процесів розподілу баз даних. LSTM може дати наступні можливості використання для ефективного розподілу даних:

По-перше, LSTM може бути застосований для прогнозування навантаження на базу даних, зокрема, передбачення піків навантаження та часових інтервалів з підвищеною активністю. Це дозволяє системі адаптуватися до змін у запитах та ресурсних потребах, надаючи підґрунтя для прийняття рішень щодо розподілу даних.

По-друге, нейронні мережі LSTM можуть бути використані для аналізу структури та характеристик запитів до бази даних. Аналіз типів запитів, їх ресурсних вимог та інших параметрів може служити основою для групування схожих даних разом на одному сервері, забезпечуючи оптимізований розподіл.

По-третє, на основі аналізу прогнозованого навантаження та характеристик запитів, LSTM може сприяти прийняттю рішень щодо розподілу даних на різні елементи кластеру. Цей процес може бути автоматизованим та динамічно адаптованим до змін у навантаженні.

3.4. Структура RNN та надбудови LSTM

Рекурентні нейронні мережі є класом штучних нейронних мереж, призначених для роботи з послідовними даними, такими як текст, звук, часові

ряди тощо. Основна особливість RNN полягає в їх здатності зберігати та використовувати інформацію з попередніх кроків обробки.

Структура RNN:

Рекурентна нейронна мережа має структуру, що складається з повторюючись блоків, які приймають на вхід вхідні дані та попередній стан. Кожен такий блок має внутрішні зв'язки, які дозволяють інформації переноситися через часові кроки. Зазвичай, в цих блоках використовуються операції додавання та активації, такі як гіперболічний тангенс або ректифікована лінійна функція (ReLU).

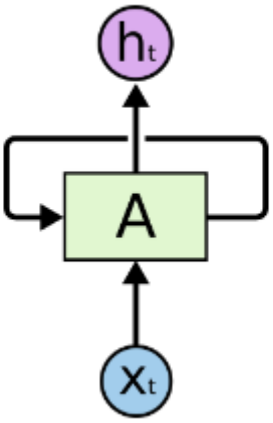

Рисунок 3.15 – Петля рекурентної мережі

На рисунку 3.16 фрагмент нейронної мережі, позначений як A, аналізує певний вхід x(t) та генерує значення h(t). Петля дозволяє передавати інформацію від одного кроку мережі до наступного.

Ці петлі надають рекурентним нейронним мережам не очевидний функціонал. Однак, якщо розгорнути петлю, виявляється, що вони не такі дуже відмінні від звичайних нейронних мереж. Рекурентну нейронну мережу можна уявити як кілька копій однієї і тієї ж мережі, кожна з яких передає

повідомлення наступнику. На рисунку 3.17 зображено розгорнуту петлю рекурентної мережі:

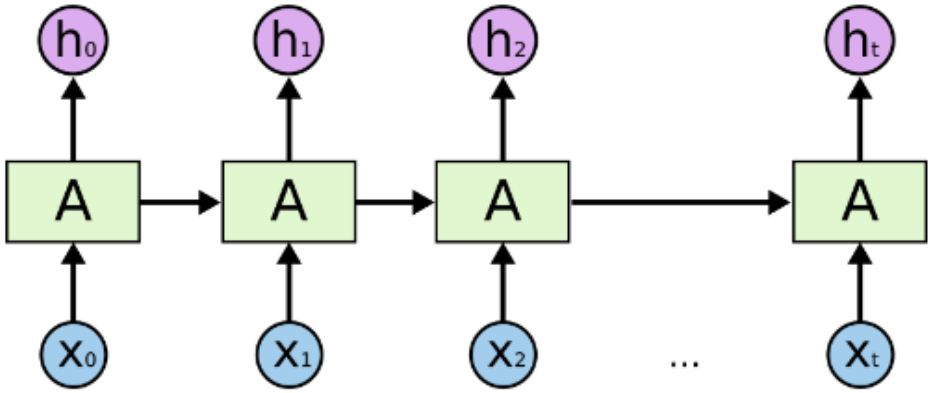

Рисунок 3.16 – Розгорнута петля рекурентної мережі

Перехід від звичайних рекурентних нейронних мереж до рекурентних нейронних мереж з довготривалою пам'яттю виник з метою подолання деяких важливих обмежень та проблем, які виникають при використанні RNN.

Важкість збереження довгострокових залежностей: Звичайні RNN мають обмежену здатність зберігати та передавати довгострокові залежності в даних, оскільки градієнти швидко зникають або прямують до нескінченності. Це може призводити до того, що RNN мають обмежені можливості для аналізу та генерації послідовностей.

Відсутність механізму вибору і забування інформації: Звичайні RNN не мають ефективного механізму для вибору, яку інформацію зберегти та яку забути в процесі обробки послідовностей. Це обмежує їх здатність до адаптації до різних завдань, де важливість різних частин послідовності може відрізнятися.

Відсутність механізму контролю градієнта: Звичайні RNN можуть стикатися з проблемами в управлінні градієнтами, особливо при обробці довгих послідовностей. Це може призводити до проблем в навчанні та впливати на ефективність моделі.

LSTM були вперше запропоновані з метою подолання цих обмежень та проблем RNN. Вони мають спеціальний механізм керування пам'яттю та градієнтами, що дозволяє їм ефективно зберігати та передавати інформацію протягом довгих послідовностей, вибирати, яку інформацію зберегти або забути, та контролювати градієнти під час навчання. Це робить їх набагато більш потужними та ефективними для багатьох завдань обробки послідовностей порівняно зі звичайними RNN.

LSTM - це особливий вид Рекурентних Нейронних Мереж, які здатні вивчати довгострокові залежності. Вони спеціально розроблені для уникнення проблеми довгострокової залежності. Зберігання інформації на довгий період часу практично є їхньою типовою поведінкою.

Усі рекурентні нейронні мережі мають форму ланцюга з модулів нейронних мереж, що повторюються [14]. У стандартних RNN цей модуль, що повторюється може мати дуже просту структуру, таку як один шар tanh.

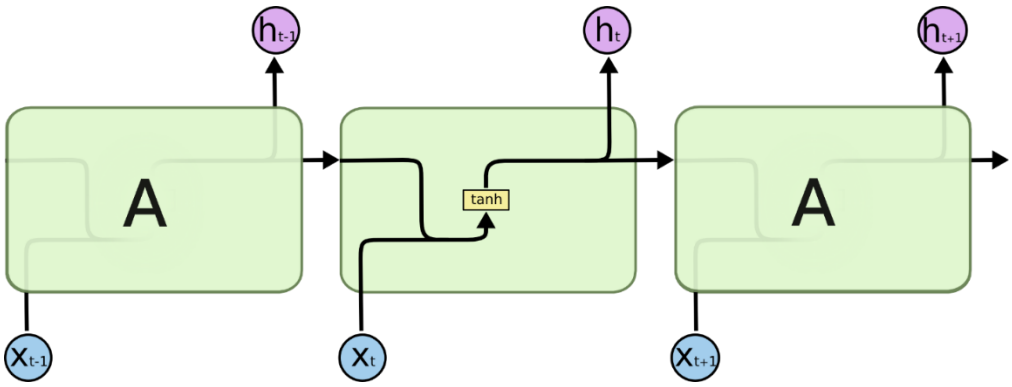

Рисунок 3.17 – Архітектура одношарової рекурентної мережі

Структура LSTM [70, 83] може бути розглянута як послідовність блоків, що повторюються кожен з яких виконує певні обчислення та має внутрішню пам'ять. Ключовою рисою LSTM є наявність чотирьох інтерактивних шарів в кожному блоку:

Шар забування (Forget Gate): Цей шар визначає, яка інформація з попереднього кроку потрібно зберегти або викинути. Він враховує попередньо стан пам'яті та поточний вхід і вирішує, яка інформація залишається в пам'яті LSTM.

Шар оновлення (Input Gate): Цей шар визначає, яка нова інформація повинна бути додана до пам'яті. Він враховує попередній стан пам'яті та поточний вхід, обчислює потенційний оновлений стан пам'яті та визначає, яка інформація повинна бути оновлена.

Вихідний шар (Output Gate): Цей шар визначає, яким чином пам'ять LSTM впливає на вихід моделі. Він враховує оновлений стан пам'яті та поточний вхід, обчислює вихідний сигнал та регулює його за допомогою функції активації.

Стан пам'яті (Cell State): Це внутрішня пам'ять LSTM, яка зберігає інформацію про довгострокові залежності. Шари забування та оновлення керують змінами у цьому стані, щоб відфільтрувати непотрібну інформацію та додати нові відомості.

Така комплексна структура дозволяє LSTM ефективно вивчати та використовувати довгострокові залежності в послідовностях даних. Це дає їм здатність розрізняти та обробляти інформацію з різних проміжків, що дозволяє вирішувати більш складні завдання в області обробки послідовностей.

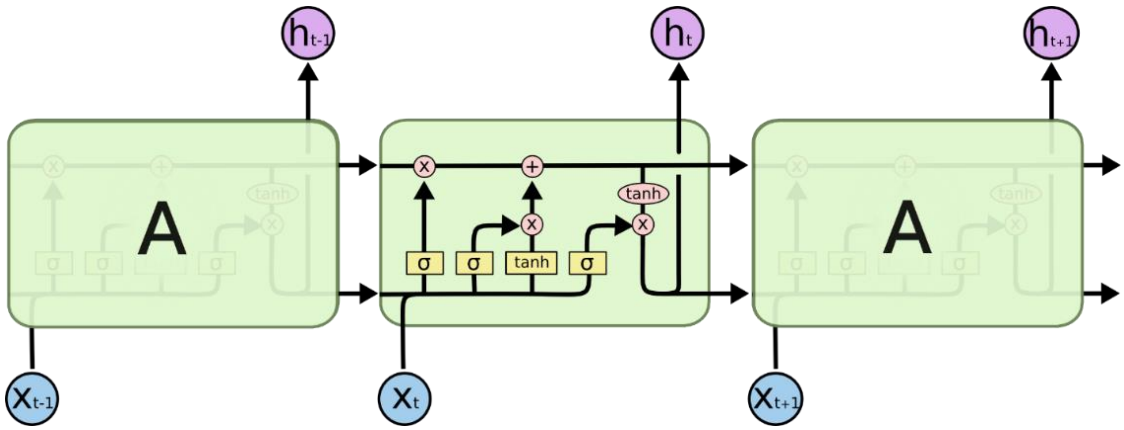

Рисунок 3.18 – Архітектура LSTM мережі

На рисунку 3.19 кожна лінія несе цілий вектор, від виходу одного вузла до входів інших. Рожеві круги представляють поелементні операції, такі як додавання векторів, тоді як жовті блоки - це навчені шари нейронної мережі. Злиті лінії позначають конкатенацію, тоді як розгалужена лінія позначає копіювання її вмісту і копії подаються у різні місця.

Ключовим елементом LSTMs є стан комірки, горизонтальна лінія, що пролягає згори по діаграмі.

Стан комірки подібний до конвеєрного транспортера. Він простягається вздовж всього ланцюга, з деякими незначними лінійними взаємодіями. Інформація може легко просто протікати вздовж нього без змін.

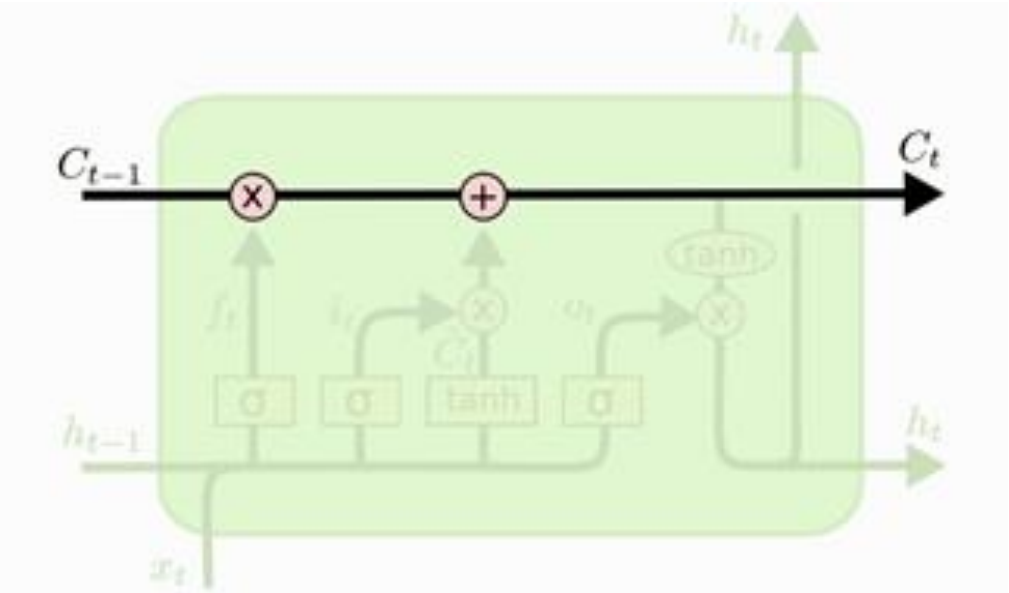

Рисунок 3.19 – Шлях зміни пам’яті відносно обчислених параметрів

LSTM дійсно має можливість видаляти або додавати інформацію до стану комірки, що ретельно регулюється структурами, відомими як вентилі.

Вентилі - це механізми, які регулюють потік інформації, контролюючи передачу та обробку сигналів у нейронах, зокрема в контексті входу,

забування та виведення. Вони складаються з шару сигмоїдальної нейронної мережі та операції поелементного множення.

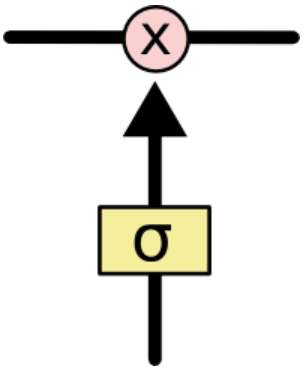

Рисунок 3.20 – Вентиль на архітектурі

Перший крок в LSTM - вирішення, яку інформацію необхідно видалити з комірки. Це рішення приймається за допомогою сигмоїдного шару, який називається "шаром забування" (forget gate layer). Він аналізує h(t−1) та x(t), і для кожного числа в комірці C(t−1) виводить число в діапазоні від 0 до 1. Значення 1 означає "повністю залишити це", тоді як значення 0 означає "повністю видалити це". В такому випадку комірка може містити інформацію про стан поточного признаку. Коли з’являється новий признак, необхідно забути інформацію про попередній.

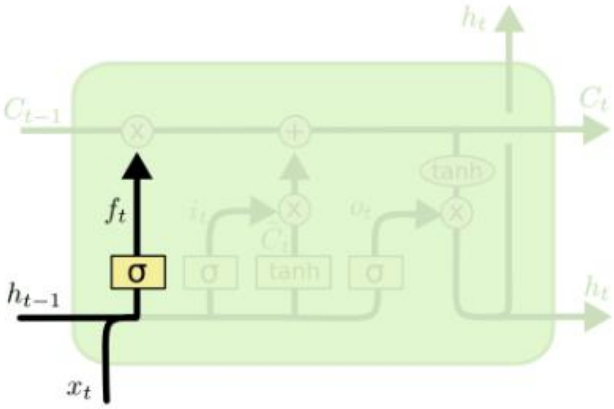

Рисунок 3.21 – Сигмоїдний шар забування для визначення актуальності параметрів

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{3.2}$$

Наступним кроком є вирішення, яку нову інформацію потрібно зберегти в комірці. Цей процес має дві частини. По-перше, сигмоїдний шар, який називається "шар входу" (input gate layer), вирішує, які значення буде оновлювати. Далі, гіперболічний тангенціальний (tanh) шар створює вектор нових потенційних значень, Č(t), які можуть бути додані до параметрів. У наступному кроці поєднання цих двох елементів, щоб створити оновлення параметрів.

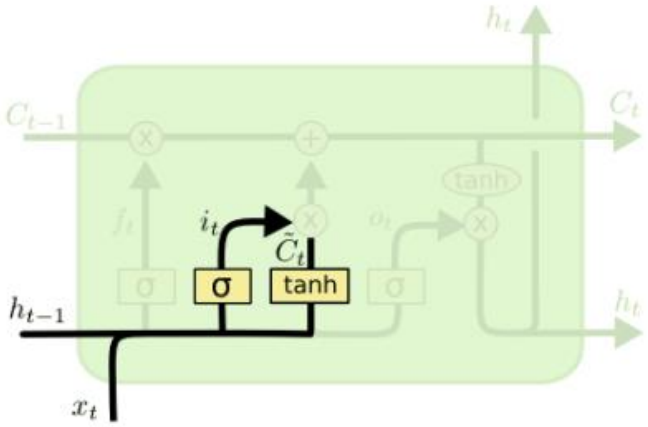

Рисунок 3.22 – Сигмоїдний та гіперболічний тангенціальний (tanh) шар для обчислення параметрів оновлення

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.3}$$

$$\tilde{C}_t = \tanh(Wc \cdot [h_{t-1}, x_t] + b_C) \tag{3.4}$$

Після визначення параметрів забування та оновлення інформації в комірках, потрібно замінити параметри основної комірки C(t−1), новими параметрами C(t). Помноживши параметри f(t), відкидаючи ті параметри, що було визначено забути раніше. Потім додається i(t) ∗ Č(t). Це нові потенційні

значення, масштабовані тим, наскільки було визначено оновити кожне значення параметрів.

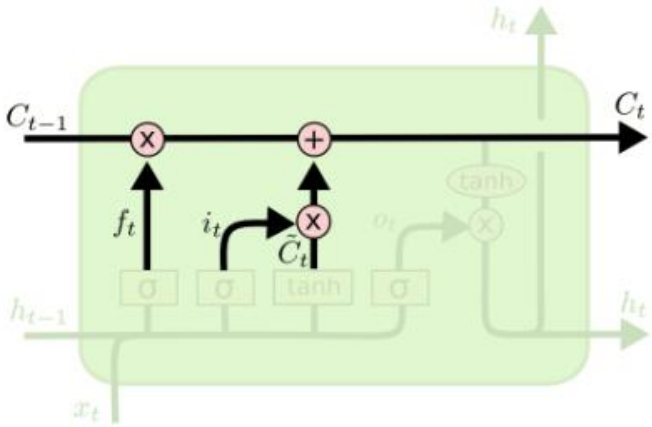

Рисунок 3.23 – Зміна параметрів комірок пам’яті

Комірки пам’яті формуються наступним чином:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.5}$$

Кінцевою дією буде обчислення вихідних даних. Вони будуть базуватися на параметрах комірок пам’яті з певними фільтрами. Першим кроком обчислення вихідних даних буде пропускання через сигмоїдний шар, який вирішує, які параметри будуть переходити на вихід. Наступним кроком це проходження через гіперболічний тангенціальний (tanh) шар (для приведення значень до діапазону від -1 до 1) та помножуємо його на вихід вентель сигмоїдного шару, щоб завдати впливу параметрів обчислених комірок в пам’яті.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.6}$$

$$h_t = o_t * \tanh(C_t) \tag{3.7}$$

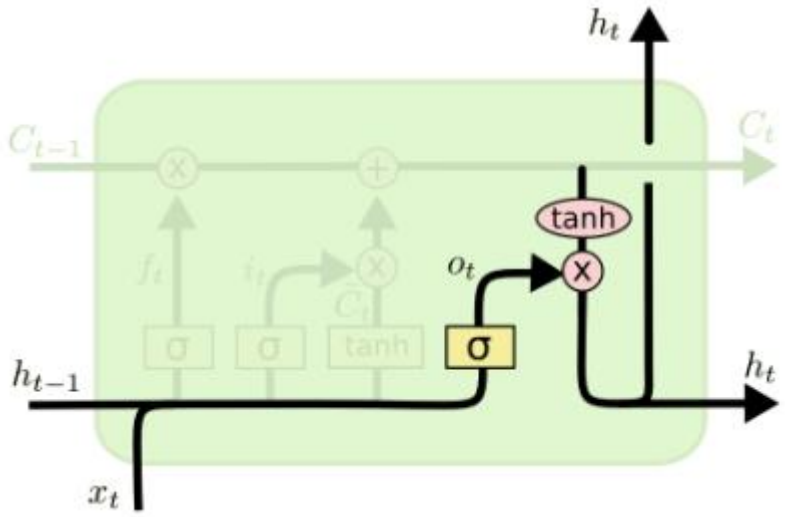

Рисунок 3.24 – Модифікація вхідних даних відносно даних в комірках пам’яті

Модифікації LSTM:

LSTM з перехідними з'єднаннями (LSTM with Peephole Connections): Ця модифікація включає додаткові перехідні з'єднання, які дозволяють шарам забуття, входу та виводу переглядати значення стану комірки. Це може допомогти вентилям краще регулювати процеси забуття та додавання інформації.

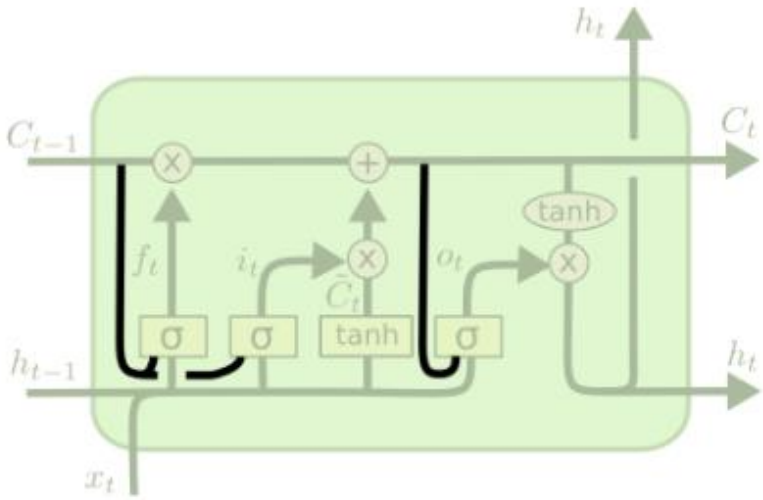

Рисунок 3.25 – LSTM з перехідними з'єднаннями

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C}_{t-1}, h_{t-1}, x_t] + b_f\right) \tag{3.8}$$

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C}_{t-1}, h_{t-1}, x_t] + b_f\right) \tag{3.9}$$

$$o_t = \sigma(W_o \cdot [\boldsymbol{C}_t, h_{t-1}, x_t] + b_o) \tag{3.10}$$

Об’єднана пара вентилів (Coupled Forget and Input Gates): В цій модифікації шари забуття та входу працюють разом, вирішуючи, яку інформацію слід забути та яку додати. Це може полегшити регулювання внутрішніх станів мережі.

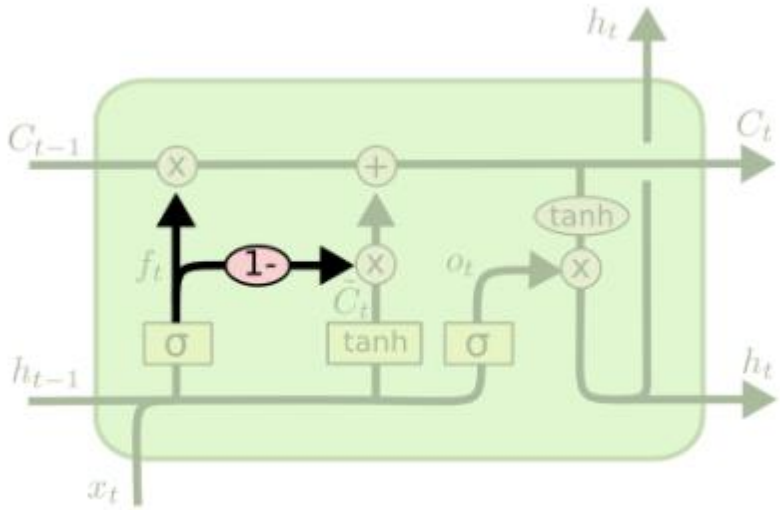

Рисунок 3.26 – LSTM з перехідними з'єднаннями

$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t \tag{3.11}$$

Gated Recurrent Unit (GRU): GRU об'єднує шари забуття та входу в один "шар оновлення". GRU також об'єднує стан комірки та стан прихованого шару, мінімізуючи обчислювальні ресурси та спрощуючи модель. Цей варіант зазвичай добре працює на практиці.

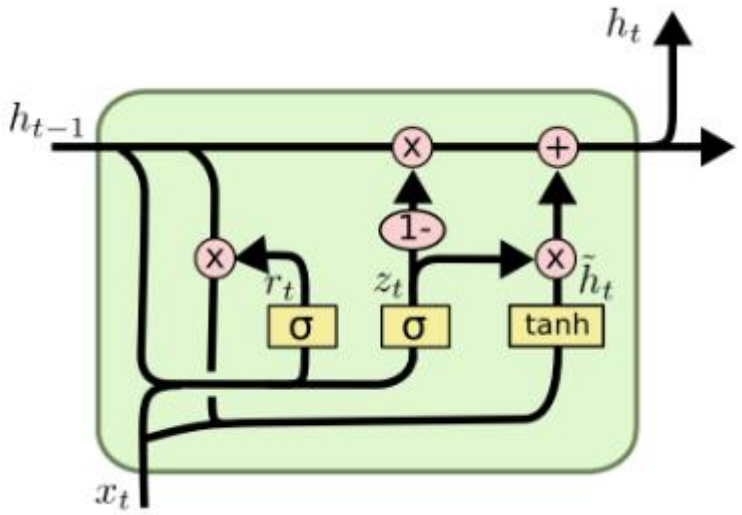

Рисунок 3.27 – Архітектура Gated Recurrent Unit

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{3.12}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{3.13}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{3.14}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{3.15}$$

Depth Gated RNNs: Варіант включає шари забуття на кожному рівні глибокої рекурентної мережі. Це дозволяє кожному рівню вільно вирішувати, яку інформацію слід забути на даному етапі обчислень.

Ці різноманітні варіанти дозволяють адаптувати архітектуру LSTM до різних завдань та специфічних вимог, що може покращити ефективність та рішення в конкретних додаткових сценаріях використання.

Процес впровадження LSTM для оптимізації шардингу баз даних передбачає комплексну послідовність кроків, яка включає наступні аспекти:

1) Представлення та трансформація даних: Перед використанням LSTM, необхідно відповідно структурувати та конвертувати дані. Прикладом може бути перетворення історичних даних про навантаження на базу

даних у хронологічні часові ряди, де кожна точка відображає зміну навантаження в часовому контексті. Інші аспекти, такі як типи запитів, характеристики та параметри ресурсів, також мають бути включені у вхідні дані для аналізу LSTM.

2) Вибір і конфігурація LSTM: Реалізація моделі LSTM вимагає обирання оптимальної архітектури. Серед параметрів, які потребують налаштування, можуть бути кількість шарів, кількість прихованих одиниць, функції активації та коефіцієнти регуляризації. Правильний вибір та налаштування цих параметрів можуть суттєво вплинути на продуктивність та адекватність моделі.
3) Підготовка даних для навчання: Ефективне навчання LSTM вимагає детальної підготовки даних. Це включає розбиття даних на тренувальні, валідаційні та тестувальні набори, а також нормалізацію даних та можливу регуляризацію. Ця фаза є критичною для якісного навчання та адекватної оцінки моделі.
4) Прогнозування та аналіз результатів: Після успішного навчання LSTM моделі можна проводити аналіз її здатності прогнозувати зміни навантаження на базу даних. Ретельний аналіз прогнозів дозволяє визначити ефективність методу та здатність вірно передбачати піки та зміни навантаження.
5) Адаптивне навчання та моніторинг: Зважаючи на динаміку навантаження на базу даних, варто розглянути можливість використання адаптивного навчання LSTM. Це може включати регулярне оновлення моделі з новими даними та періодичний моніторинг точності прогнозів.
6) Прийняття рішень на основі аналізу: Прогнози, отримані за допомогою LSTM, мають стати однією з основних основ для прийняття рішень щодо стратегії шардингу. Підтримуючи це аналізом інших чинників, таких як архітектура інфраструктури та вимоги до продуктивності, можна забезпечити обґрунтоване та ефективне розподілення даних.

Ця систематична послідовність кроків використання LSTM для оптимізації шардингу баз даних залежить від глибокого розуміння як методологічних аспектів машинного навчання, так і технічних особливостей баз даних. Впровадження та інтеграція цього підходу можуть сприяти покращенню продуктивності та ефективності розподіленої інфраструктури.

3.5. Агент для бази даних часових рядів

Для забезпечення контролю ресурсів та навантаження мережі доцільно буде створити агента керування навантаженням. Агент балансування навантаження для баз даних буде важливим компонентом у розподілених обчислювальних системах та середовищах IoT, де зберігаються та обробляються великі обсяги даних. Агент балансування навантаження для баз даних виконуватиме важливі функції для оптимізації роботи системи та забезпечення ефективної обробки запитів. Основний функціонал агенту може включати:

- Моніторинг навантаження: Агенти відстежують поточний стан обчислювальних вузлів, серверів або інших ресурсів, що використовуються для зберігання та обробки даних в базі даних. Це включає аналіз завантаження процесора, пам'яті, мережевого трафіку та інших показників продуктивності.
- Прийняття рішень: Базуючись на зібраних даних та аналізі, агент зможе приймати рішення щодо перерозподілу ресурсів. Наприклад, він зможе визначити, які сервери або вузли мають менше завантаження та здатні взяти на себе додаткові завдання.
- Динамічне перенесення навантаження: Агент зможе ініціювати процес переміщення даних або завдань з одного вузла до іншого для

рівномірного розподілу навантаження. Це може включати переміщення даних між серверами або навіть розподіл запитів до бази даних на різні вузли.

- Прогнозування навантаження: Агент зможе застосовувати аналітичні методи для прогнозування майбутнього навантаження на ресурси бази даних. Це дозволяє агентам попередньо приготувати ресурси для очікуваного піку обробки.
- Автоматична адаптація: На основі зібраних даних та аналізу агент зможе автоматично налаштовувати параметри роботи бази даних, такі як кількість паралельних обробок, розподіл завдань тощо.

Загалом, агент балансування навантаження для бази даних допоможе забезпечити ефективне використання ресурсів, зменшити час відповіді на запити.

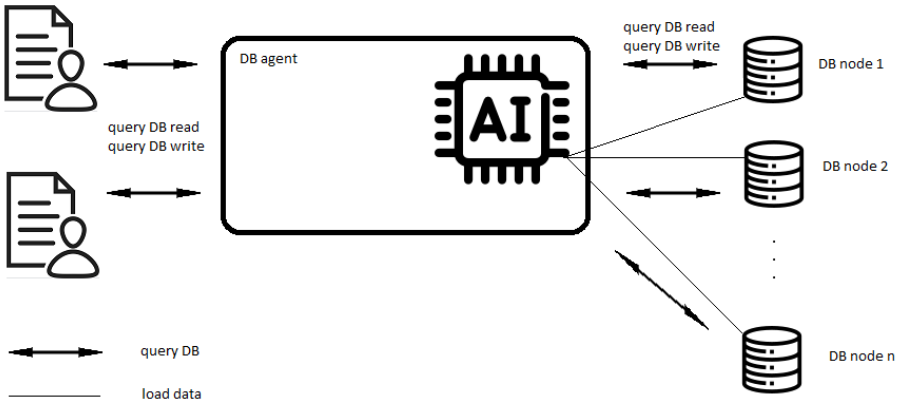

Рисунок 3.28 – Архітектура СУБД з агентом

Основними параметрами для відслідковування навантаженням є:

- Час відповіді сервера (Response Time): Ця метрика визначає, скільки часу сервер потребує для обробки запиту та відправлення відповіді. Короткий час відповіді вказує на швидку реакцію сервера.

- Пропускна здатність (Throughput): Ця метрика визначає кількість запитів, які сервер може обробити протягом одиниці часу. Вона вимірює, наскільки швидко сервер може обслуговувати користувачів або додатки.
- Використання CPU: Моніторинг використання центрального процесора дозволяє визначити, наскільки інтенсивно сервер використовує обчислювальні ресурси.
- Використання оперативної пам'яті (RAM): Ця метрика вказує на використання сервером оперативної пам'яті. Велике використання може вказувати на нестабільну роботу додатків або великі обсяги даних.
- Час завантаження (Load Times): Вимірювання часу завантаження сторінок або додатків допомагає визначити, наскільки швидко користувачі отримують доступ до вмісту.
- Споживання полоси пропускання (Bandwidth Usage): Ця метрика вказує на обсяг передачі даних через мережу. Вона важлива для оцінки використання мережевих ресурсів.
- Кількість запитів до бази даних: Моніторинг кількості запитів до бази даних допомагає визначити навантаження на базу даних та швидкість виконання запитів.
- Відсоток успішних запитів: Вимірювання відсотка успішних запитів допомагає визначити стабільність та надійність системи.
- Споживання ресурсів у контексті процесів або служб: Метрики, які вказують на використання ресурсів конкретними процесами чи службами, мають важливе значення для ідентифікації факторів, які призводять до значного витрати ресурсів.

- Індекси часу доступності (Availability Metrics): Вимірювання часу доступності системи допомагає визначити, наскільки система була доступною для користувачів протягом певного періоду.

Для юнікс-подібних операційних систем є окремий показник який поєднує безліч метрик – це Unix load average (LA). Це значення охоплює метрики навантаження на центральний процесор, навантаження читання/запису дисків, роботи мережі та навіть цикли блокувань. Параметру LA буде достатньо для оцінки навантаження на елементи кластера.

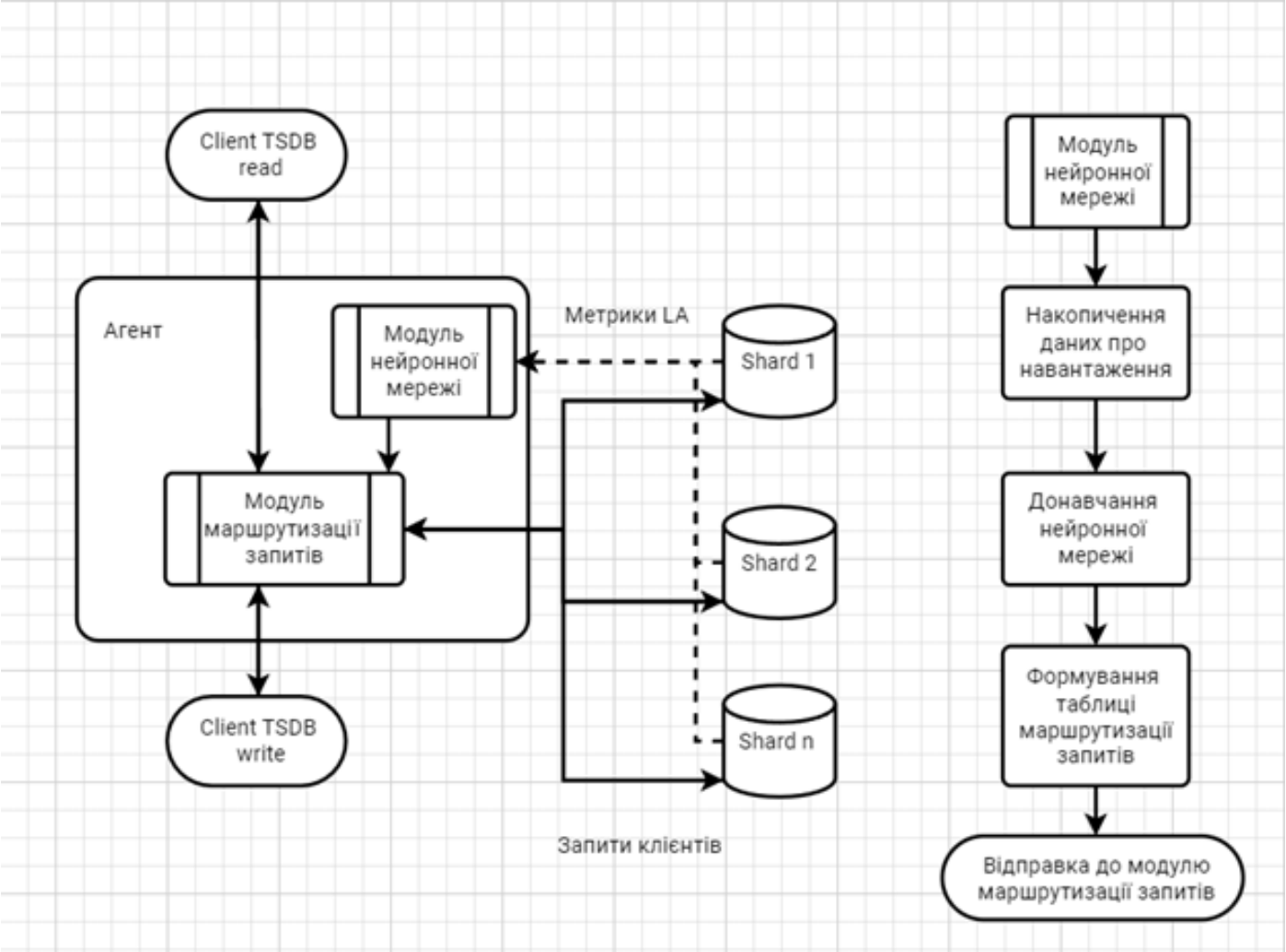

Рисунок 3.29 – Алгоритм роботи агенту бази даних часових рядів

3.5. Навчання моделі нейронної мережі для створення таблиці маршрутизації запитів агенту бази даних часових рядів

Процес навчання нейронної мережі - це ітеративний процес, під час якого модель здатна адаптуватися до вхідних даних та покращувати свої навички. Навчання нейронної мережі може бути поділено на кілька основних кроків [3]:

1) *Збір та підготовка даних.* Спершу необхідно зібрати дані, які використовуються для навчання мережі. Дані можуть бути розбиті на тренувальну, валідаційну та тестову вибірки. Дані можуть також бути нормалізовані або стандартизовані для полегшення навчання.
2) *Створення архітектури мережі.* Визначається архітектура мережі, включаючи кількість шарів, кількість нейронів у кожному шарі, функції активації тощо. Вибір архітектури залежить від конкретної задачі.
3) *Ініціалізація ваг.* Ваги мережі ініціалізуються випадковими значеннями або можуть бути встановлені заздалегідь.
4) *Визначення функції втрат*. Вибирається функція втрат, яка визначає, наскільки відповіді мережі відрізняються від очікуваних значень. Популярні функції втрат включають середньоквадратичну помилку (MSE) для задач регресії та функцію перехресної ентропії (Cross-Entropy) для задач класифікації.
5) *Вибір оптимізатора*. Вибирається алгоритм оптимізації, який відповідає за коригування ваг для зменшення втрат. Популярні оптимізатори включають стохастичний градієнтний спуск (SGD), Adam, RMSprop тощо.
6) *Навчання моделі*. В процесі навчання модель отримує тренувальні дані, робить передбачення та коригує ваги, щоб зменшити втрати. Градієнти визначаються шляхом обчислення похідних функції втрат по вагам.
7) *Завершення навчання.* Навчання може завершитися після фіксованої

кількості епох (повних проходів через навчальний набір) або на основі критерію зупинки, наприклад, досягнення певного рівня точності на валідаційному наборі.

8) *Тестування моделі.* Після завершення навчання модель тестується на тестовому наборі даних, щоб оцінити її продуктивність та здатність до узагальнення на нові дані.
9) *Оцінка результатів.* Модель оцінюється за допомогою різних метрик, таких як точність, середньоквадратична помилка, чутливість та інші, в залежності від задачі.
10) *Застосування та налаштування.* Остаточна модель може бути використана для прийняття передбачень на нових даних та, за необхідності, піддаватися налаштуванню та оптимізації.
11) *Моніторинг та підтримка.* Після впровадження моделі в експлуатацію важливо здійснювати моніторинг її продуктивності та здатності до узагальнення, виявляти проблеми та вносити необхідні корективи.

Шардинг бази даних часових рядів (TSDB) на основі розбиття на часові інтервали є ефективним способом оптимізації управління великими обсягами часових даних. Шардинг дозволяє розподілити навантаження на декілька фізичних серверів чи вузлів для забезпечення більшої масштабованості і швидкодії. Кожен запис часового ряду повинен бути маркований або призначений для конкретного шарду на основі його часової прив'язки. Це допомагає визначити, в який шард слід зберігати дані. Бази даних часових рядів найчастіше використовують теги і для спрощення операцій можна створити окремий hash ідентифікатор. Хеш-функція приймає вхідні дані (такі як текстовий рядок або послідовність байтів) і обчислює фіксований хеш-код, який є числовим значенням. Унікальність хеш-коду зазвичай залежить від властивостей самої хеш-функції та вхідних даних.

```
# HELP mugs_of_beverage_total The total number of mugs drank.
# TYPE mugs_of_beverage_total counter
mugs_of_beverage_total{content="coffee", temperature="hot"} 2
mugs_of_beverage_total{content="coffee", temperature="cold"} 1
mugs_of_beverage_total{content="tea", temperature="hot"} 1
```

↓

```
# HELP mugs_of_beverage_total The total number of mugs drank.
# TYPE mugs_of_beverage_total counter
{__name__="mugs_of_beverage_total", content="coffee", temperature="hot"} 2
{__name__="mugs_of_beverage_total", content="coffee", temperature="cold"} 1
{__name__="mugs_of_beverage_total", content="tea", temperature="hot"} 1
```

↓

```
hash({__name__="mugs_of_beverage_total", content="coffee", temperature="hot"})  ⇒ 423534
hash({__name__="mugs_of_beverage_total", content="coffee", temperature="cold"}) ⇒ 255112
hash({__name__="mugs_of_beverage_total", content="tea", temperature="hot"})     ⇒ 094212
```

Рисунок 3.30 – Отримання hash ідентифікатора

Однак важливо зауважити, що унікальність хеш-коду не може бути гарантованою в усіх випадках, оскільки можливі колізії, коли різні вхідні дані мають однаковий хеш-код. Зазвичай хеш-функції розроблені таким чином, щоб мінімізувати ризик колізій, але вони не можуть бути ідеальними. Тому за одним hash ідентифікатором може знаходитись декілька часових рядів.

Для забезпечення рівномірного розподілу часових рядів між вузлами бази даних (DB node) під час навчання нейронної мережі важливо створити структуру даних, що містить повну інформацію щодо кожного часового ряду. Ця інформація повинна включати:

Ідентифікатор часового ряду: Унікальний ідентифікатор кожного часового ряду для точного визначення та відстеження кожного об'єкта.

Навантаження за попередній період: Інформація про навантаження часового ряду або його параметри протягом попереднього періоду.

Очікуване навантаження на наступний період: Прогнозоване значення навантаження часового ряду на майбутній період. Це важливо для навчання мережі на передбачення майбутніх значень та оптимального розподілу ресурсів.

Таблиця 3.1. Вхідні дані для навчання нейронної мережі за часовий період

	cLoad1	cLoad2	TS 1	TS 2	TS 3	TS n	nLoad1	nLoad2
Shard 1	0,12	0,34	1	0	0	0	0,61	0,41
Shard 2	0,83	0,67	0	0	1	0	0,73	0,34
Shard 3	0,25	0,36	0	1	0	0	0,12	0,43
Shard 4	0,18	0,40	0	0	0	0	0,40	0,25
Shard 5	0,82	0,37	0	0	0	1	0,83	0,65

Shard – мінімальна одиниця поділу бази.

cLoad (current load) – навантаження кожного shard за певний часовий інтервал поточних налаштувань.

nLoad (next load) – очікуване навантаження кожного shard на наступні часові інтервали.

TS (time series) – часовий ряд.

Ця структура даних дозволить організувати інформацію про часові ряди та їх навантаження, щоб нейронна мережа могла навчитися моделювати та передбачати навантаження на наступні періоди з урахуванням попередніх значень відповідно їх ідентифікаторів ряду.

У даному дослідженні використовувався масив послідовних даних формату даних таблиці 3.1. Цей масив складений на основі даних, отриманих з вимірювань, проведених на випадково розподілених часових рядах.

Після навчання нейронної мережі, параметри очікуваного навантаження (nLoad) будуть інструментом для контролю навантаження. Для рівномірного навантаження потрібно підставити значення рівні середньому навантаженню за певний часовий інтервал.

Таблиця 3.2. Очікувані вихідні дані нейронної мережі

	TS 1	TS 2	TS 3	TS n	cLoad1	cLoad2
Shard 1	0	0	0	0	0,61	0,41
Shard 2	0	1	1	0	0,73	0,34
Shard 3	0	0	0	0	0,12	0,43
Shard 4	0	0	0	1	0,40	0,25
Shard 5	1	0	0	0	0,83	0,65

Значення часових рядів в масиві будуть відрізнятись від очікуваних. На виході ми отримаємо чисті дані від нейронної мережі і значення для кожного shard/ts буде в межах 0..1. Так як один часовий ряд в певний момент повинен бути тільки на одному з shard-ів, тому за одиницю приймаємо найбільший з запропонованих варіантів нейронної мережі, а всі інші приймаємо за нуль.

На основі цих даних агент формує таблицю відповідності (таблиця 3.3), що складається з назв часових рядів та часових інтервалів. Кожний запит на запис певного часового ряду отримає IP адресу відповідного shard-у для збереження. Кожний запит на отримання інформації з часового ряду, запустить ціпочку зчитування кожного з shard-ів за певний часовий проміжок.

Таблиця 3.3. Таблиця відповідності часових рядів в агенті бази даних

Timestamp / Time series	1645653600	1645740000	1645826400	N + 1 day
Time series 1	shard 1	shard 1	shard 3	shard 4
Time series 2	shard 5	shard 2	shard 1	shard 5
Time series 3	shard 3	shard 4	shard 4	shard 1
Time series 4	shard 2	shard 5	shard 1	shard 2
Time series 5	shard 4	shard 3	shard 3	shard 1

Основний шар нейронної мережі — це шар LSTM[84]. Цей шар використовується для зберігання інформації про попередні стани вхідних даних, що дозволяє моделі ефективно враховувати довгострокові залежності. Це дозволяє моделі ефективно враховувати довгострокові залежності між даними. Такий підхід дає можливість нейронні мережі здатність виявляти складні залежності в послідовностях даних.

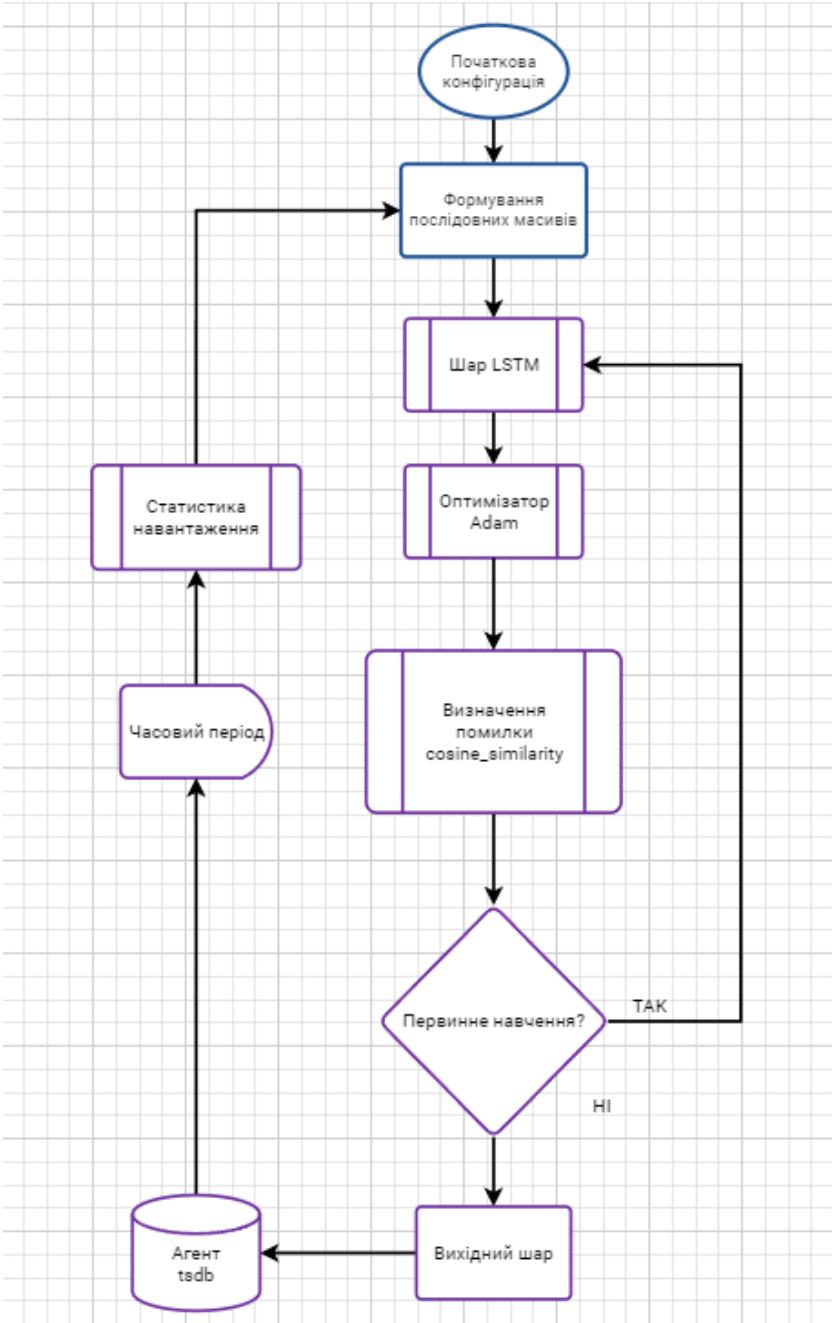

Рисунок 3.31 – Блок-схема розробленого алгоритму навчання нейронної мережі

LSTM(server_count*(day_interval_count+tag_count),input_shape=(lstm_count, server_count * (day_interval_count*2+tag_count))

Шар приймає послідовність даних в форматі (два параметри):

table_count, server_count * (day_interval_count*2+tag_count).

Де перший параметр (table_count) – кількість послідовних масивів.

Другий параметр (server_count * (day_interval_count*2+tag_count)) – переформатований масив даних таблиці 3.1., що складається з кількості серверів, кількості інтервалів в часовому періоді та кількості тегів.

Вихід з шару матиме один переформатований масив (формату таблиці 3.2.) розмірністю: server_count*(day_interval_count+tag_count)

Первинне навчання – процес навчання нейронної мережі на випадково-розподілений часових рядах та їх навантаженням. Виконується лише один раз з невеликим навантаженням для базової інціалізації ваг нейронної мережі.

Циклічне навчання – процес циклічного донавчання нейронної мережі для актуалізації значень ваг нейронної мережі. Виконується перед кожним часовим періодом з останніми показниками навантаження на компоненти кластеру.

За для перевірки можливості навчання нейронної мережі було створено модель на мові програмування Python, яка випадковим чином заповнювала наступні вхідні дані часових рядів:

1) Індекси часових інтервалів.
2) Глибина аналізу для операції зчитування.
3) Складність запиту.

Також прораховувались умовні load average (LA) на основі цих генерацій з врахуванням міграцій часових рядів. 80% отриманих даних подавались на вхід нейронної мережі, інші 20% формували валідаційну вибірку.

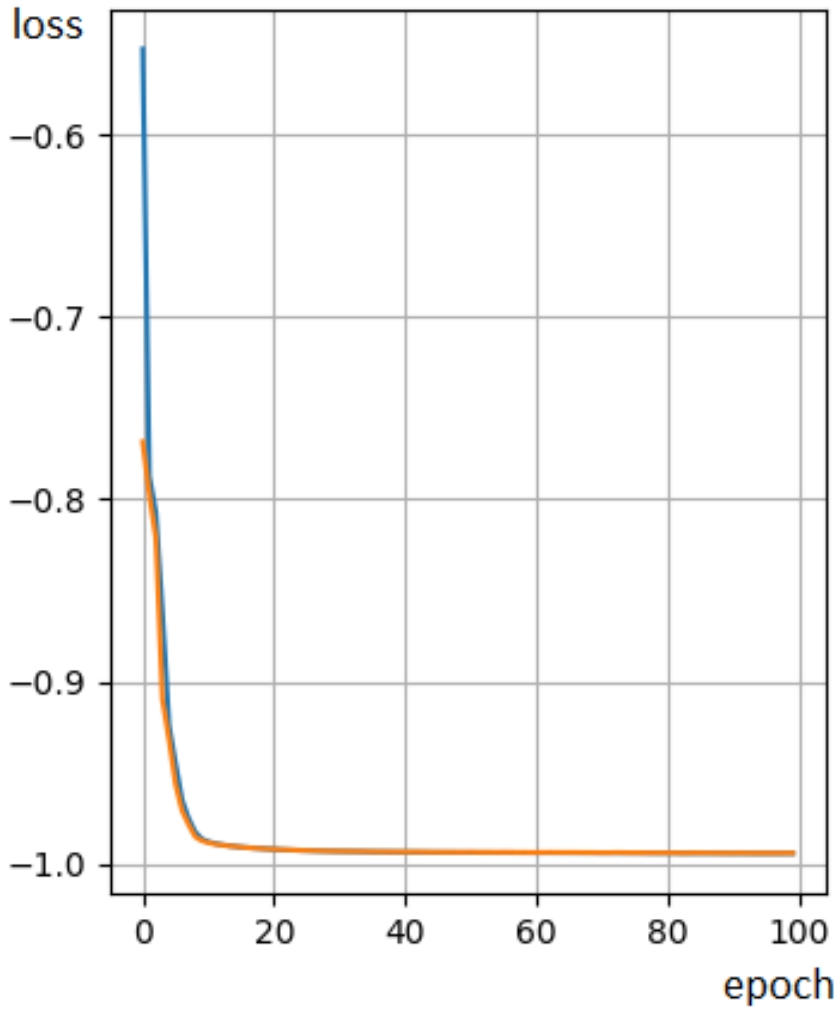

Рисунок 3.32 – Графік значень функції втрат. Синя крива – втрати навчальної вибірки, помаранчева – втрати варіаційної вибірки

Зменшення значень функції втрат при навчанні вказує на позитивну тенденцію навчання нейронної мережі. Функція втрат (cosine_similarity) повертає значення від -1 до 1, що показує близькість між прогнозом та ціллю. Коли це від’ємне число від -1 до 0, 0 вказує на ортогональність, а значення, ближчі до -1, вказують на більшу подібність. Значення, ближчі до 1, вказують на більшу відмінність.

Близькість значень функції втрат та значень втрат валідаційної вибірки – це може бути ознакою того, що модель навчилася адаптуватися до

навчальних даних, які повинні бути сформовані з початкових даних отриманих від клієнтів баз даних.

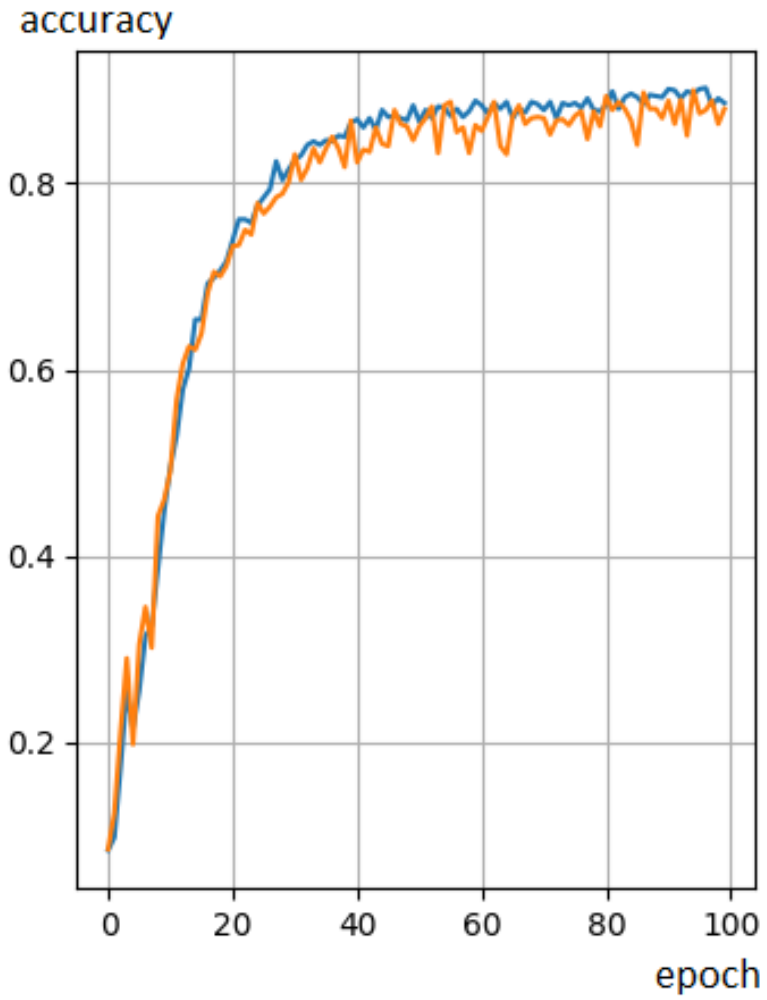

Рисунок 3.33 – Графік точності навчання нейронної мережі. Синя крива – точність навчальної вибірки, помаранчева – точність варіаційної вибірки

Точність це метрика, яка визначає, наскільки ефективно модель може робити правильні прогнози на вхідних даних і може приймати значення від 0 до 1. Високий рівень точності нейронної мережі свідчить про те, що модель ефективно вирішує завдання прогнозування на даному наборі даних.

3.6. Оптимізація структури вхідних даних за допомоги hash bucket

З часом кількість часових рядів може значно збільшитись, що неминуче привиде до значного зростання складності навчання нейронної мережі. Також є проблема необхідності перенавчання нейронної мережі при додаванні нового часового ряду, що потребуватиме все більше і більше ресурсів.

Основне неперіодичне навантаження в базах даних часових рядів створюють користувачі та системи аналітики. Більшість часових рядів можуть мати майже нульове навантаження в часовому інтервалі. Агрегація навантаження часових рядів дасть змогу значно спростити навчання нейронної мережі та зменшити розмірність таблиці агенту бази даних (таблиця 3.3.), що значно пришвидшить маршрутизацію запитів між елементами кластера.

Для цього потрібно розподілити всі часові ряди на окремі комірки (buckets). Їх кількість повинна бути значно більша ніж кількість серверів кластера, щоб забезпечити варіативність балансування для нейронної мережі. Так як кількість часових рядів і їх формат може бути будь-який, доцільніше буде використати стратегію хешування для розподілу часових рядів між комірками.

Hash table схожа на масив, індекси якого це результат виконання hash-функції. Він зазвичай використовується для зберігання даних типу "ключ-значення". Кожен ключ має бути унікальним і зіставлятися з певним індексом у масиві, а його значення зберігається в цьому місці. Кожен ключ має бути унікальним і зіставлятися з певним індексом у масиві, а його значення зберігається в цьому місці. Завдяки цьому операції вставки, пошуку та видалення виконуються зі складністю O(1). Більшість hash-функцій може віддавати однакове значення на виході при різних вихідних даних (хоча ймовірність такого дуже мала), що може створити колізію. Що б унеможливити колізію є два варіанти вирішення цієї проблеми.

1) Окремий ланцюжок. Кожен запис hash-таблиці відповідає сегменту, що містить змінний набір елементів. Як правило, це реалізується як зв'язаний список, тому кожен запис масиву (якщо він непорожній) містить вказівник на початок зв'язаного списку. Щоб перевірити, чи є елемент у хеш-таблиці, ключ спочатку хешується, щоб знайти зв'язаний список для пошуку. Потім зв'язаний список сканується, щоб побачити, чи присутній потрібний елемент. Якщо пов'язаний список короткий, це сканування буде дуже швидким.

2) Лінійна проба. Замість того, щоб поміщати конфліктуючі елементи в пов'язаний список, усі елементи зберігаються в самому масиві. Коли додавання нового елемента до хеш-таблиці створює колізію, хеш-таблиця знаходить інше місце в масиві, щоб розмістити його. Найпростіший спосіб знайти порожній індекс полягає в попередньому пошуку індексів масиву з фіксованим кроком (часто 1), шукаючи невикористаний запис; ця стратегія лінійної проби має тенденцію створювати велику кількість кластеризації елементів у таблиці, що призводить до зниження продуктивності.

Hash buckets формуються методом остачі від ділення. Зазвичай цей метод використовується для послідовного розподілу запитів. Кожен часовий ряд може мати різне навантаження, тому навантаження hash buckets може відрізнятись в рази один від одного.

Висновок до розділу 3

У розглянутому розділі проаналізовано класичні стратегії розподілу навантаження. Для створення методики був розроблений алгоритм роботи агента бази даних часових рядів для забезпечення маршрутизації запитів. Проведено аналіз для вибору типу нейронної мережі для прогнозування

навантаження. Розроблено методику навчання нейронної мережі для розподілу навантаження баз даних часових рядів. Данна методика стане основою для розробки новітніх систем розподілених баз даних часових рядів. Розроблена програмна модель нейронної мережі для аналізу навантаження на базу даних часових рядів.

Для доведення ефективності було зроблено по 100 програмних генерацій з випадковими параметрами навантаження на hash bucket. Навантаження на програмні генерації були розподілені по за двома методиками: почергове розподілення між елементами бази даних та за новою методикою після навчання нейронної мережі. Для оцінки було обрано два показники:

1. Максимальне навантаження Unix load average з всіх елементів бази даних.
2. Мінімальне середнє навантаження Unix load average за часовий інтервал з всіх елементів бази даних.

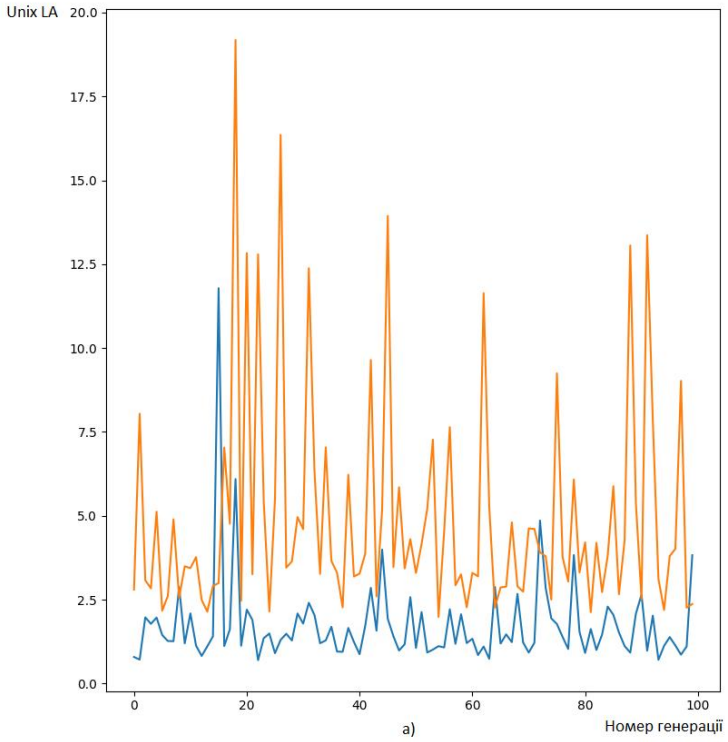

Рисунок 3.34 – Графік максимального навантаження Unix load average з всіх елементів бази даних

Синя крива – максимальне навантаження Unix LA з розподілених hash buckets за новою методикою після навчання нейронної мережі, помаранчева – максимальне навантаження Unix LA з розподіленими hash buckets почергово до кожного елементу.

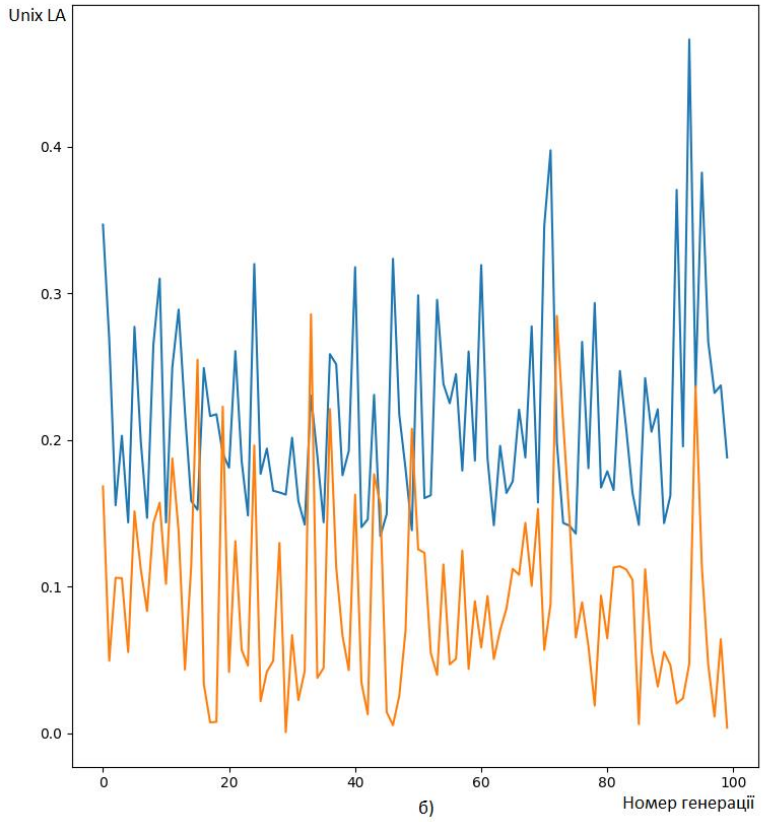

Рисунок 3.35 – Графік мінімального середнього навантаження Unix load average за часовий інтервал з всіх елементів бази даних

Синя крива – мінімальне середнє навантаження Unix LA з розподілених hash buckets за новою методикою після навчання нейронної мережі, помаранчева – мінімальне середнє Unix LA з розподіленими hash buckets почергово до кожного елементу.

Розроблена методика представляє собою комплекс інноваційних підходів та технічних рішень, спрямованих на оптимізацію та ефективне управління навантаженням баз даних часових рядів. Застосування цієї

методики виокремлюється високим ступенем автоматизації часових рядів, що динамічно змінюються.

В основі методики лежать передові алгоритмічні рішення, що дозволяють не лише забезпечити ефективний аналіз навантаження, а й оперативно реагувати на зміни в обсязі і характері навантаження. Застосування розробленої методики може значно підвищити ефективність роботи розподілених баз даних часових рядів, забезпечуючи їх оптимальну продуктивність в умовах навантаження, яке постійно змінюється.

ВИСНОВКИ

В дисертаційній роботі вирішувалось наукове завдання з розробки методики побудови адаптивних кластерів комп’ютерних систем для збору та обробки інформації з пристроїв IoT.

В роботі були розглянуті ключові моменти про стратегії розподілу баз даних часових рядів. Застосування розподіленої бази даних часових рядів вирізняється не лише своєю здатністю до оптимізації навантаження, але й рольовим впливом на сучасні концепції архітектури систем IoT. В контексті росту Інтернету речей та збільшення кількості розподілених даних, розподіл навантаження стає ключовою стратегією для забезпечення не лише продуктивності але й забезпечення надійності систем обробки часових рядів.

Основні виклики, які виникають при роботі з великими обсягами часових даних, включають не тільки швидкість обробки але і збереження цілісності та доступності. Стратегії розподілу навантаження дозволяють вирішити ці завдання, створюючи ефективні рішення для обробки великих обсягів інформації в реальному часі.

Розроблена методика навчання нейронної мережі в ролі аналізатора навантаження, що може адаптуватися до змін. Використовуючи механізми навчання, така система може постійно покращувати точність для максимальної ефективності в умовах зростаючої складності та об'єму даних. Застосування нейронних мереж для розподілу навантаження може призвести до автоматизації рішень щодо визначення оптимального розподілу даних, враховуючи динаміку часових рядів. Це важливо, особливо в областях, де штучний інтелект може швидко адаптуватися до змін в навколишньому середовищі та вимог бізнесу. Крім того, нейронна мережа може виявитися ефективною в роботі з нетиповими аномаліями та варіаціями в часових рядах, що важко передбачити та керувати за допомогою традиційних стратегій розподілу.

Розроблений алгоритм маршрутизації запитів для агенту бази даних часових рядів є перспективним рішенням для ефективного управління та обробки даних у великих та динамічних середовищах.

У рамках викладеної дисертаційної роботи докладно розглядається методика розробки та реалізації нейронної мережі з метою адаптивного розподілу навантаження в контексті кластеру розподіленої бази даних, орієнтованої на обробку часових рядів. Використовуючи передові підходи та методи в області штучного інтелекту та машинного навчання, дана методика пропонує інтегрований підхід для вдосконалення розподілу завдань у великих обсягах часових даних.

З метою перевірки запроплонованої методики було розроблено програмну модель нейронної мережі для аналізу навантаження на базу даних часових рядів для порівняння з класичними методами розподілу. Отримані результати підтверджують ефективність запропонованої нейронної мережі. Зокрема, показник точності для валідаційної вибірки нейронної мережі становить 88%, що свідчить про високу адаптивність та точність розподілу навантаження в контексті роботи з часовими рядами. В порівняні з рівномірним розподілом, методика дала змогу збільшити ефективність роботи кластера в 1,2 рази.

Важливо відзначити, що в порівнянні з класичним методом розподілу навантаження модель, реалізована за допомогою нейронної мережі, продемонструвала значущий приріст ефективності. Це свідчить про переваги та перспективи використання передових методів машинного навчання для оптимізації управління ресурсами в системах обробки часових рядів.

Результати дисертаційних досліджень впроваджені у виробничий процес ТОВ «ХУАВЕЙ Україна», ПрАТ «Київстар» (додаток В) та в навчальний процес Державного університету інформаційно-комунікаційних технологій (Київ) (додаток Г).

Усе зазначене підсилює вагу та актуальність поданої дисертаційної роботи, а отримані наукові результати стають важливим внеском у розвиток сучасних підходів до управління та обробки часових рядів у розподілених інформаційних системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алтинніков Д. Як штучний інтелект та IOT доповнюють один одного / Д. Алтинніков, А. Тушич // Всеукр. наук.-техн. конф. “Сучасний стан та перспективи розвитку IOT”, Київ. 2020. – С. 303–304.
2. Аналіз вимог до проектування хмарної платформи для інтернету речей / К. Сторчак [та ін.] // Зв’язок. – 2019. – № 6. – С. 26–34.
3. Вимоги до інтелектуальних систем аналізу даних та їх класифікацій / А. Тушич [та ін.] // Телекомунікаційні та інформаційні технології. – 2019. – № 1. – С. 31–36.
4. Шефкін Б. В., Красюк І. В., Хоменчук В. О. та ін. Дослідження та впровадження нейронної мережі на основі TENSORFLOW // Зв’язок. – 2020. – № 6. – С. 18–20.
5. Інтелектуальний аналіз даних з використанням нейронних мереж / К. Сторчак [та ін.] // Зв’язок. – 2018. – № 4. – С. 17–19.
6. Тушич А. Машинне навчання з використанням TENSORFLOW / А. Тушич // XII Міжнар. наук.-техн. конф. студентів та аспірантів “Перспективи розвитку інформаційно-телекомунікаційних технологій та систем”, Київ, 17 груд. 2020 р. – Київ, 2020. – С. 379.
7. Хоменчук В. Аналіз впливу хмарних обчислень і граничних обчислень на продуктивність мережі / В. Хоменчук // Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу, Київ, 28 листоп. 2023 р. – С. 361-362.
8. Хоменчук В. Використання баз даних часових рядів в архітектурі граничних обчислень. / В. Хоменчук // Звя'зок. – 2023. – № 1. – С. 39–42.
9. Хоменчук В. Переваги використання apache iotdb для адаптивних систем IoT / В. Хоменчук // Сучасний стан та перспективи розвитку IoT, Київ, 9 квіт. 2021 р. – С. 94-95.

10. Хоменчук В. Переваги та недоліки алгоритмів розподілу даних в кластері баз даних / В. Хоменчук // Сучасний стан та перспективи розвитку IoT, Київ, 7 квіт. 2023 р. – С. 95-97.

11. Хоменчук В. Стратегій розподілу навантаження децентралізованих баз даних / В. Хоменчук // Звя'зок. – 2023. – № 5. – С. 21–32.

12. Abdelhafiz B. M. Distributed database using sharding database architecture [Electronic resource] / Bahaa Mahmoud Abdelhafiz // 2020 IEEE asia-pacific conference on computer science and data engineering (CSDE), Gold Coast, Australia, 16–18 December 2020. – Mode of access: https://doi.org/10.1109/csde50874.2020.9411547 (date of access: 12.04.2023). – Title from screen.

13. Abu-Elkheir M. Data management for the internet of things: design primitives and solution [Electronic resource] / Mervat Abu-Elkheir, Mohammad Hayajneh, Najah Ali // Sensors. – 2013. – Vol. 13, no. 11. – P. 15582–15612. – Mode of access: https://doi.org/10.3390/s131115582 (date of access: 21.02.2020). – Title from screen.

14. A clockwork RNN [Electronic resource] / Jan Koutník [et al.] // arxiv.org. – Mode of access: https://arxiv.org/abs/1402.3511 (date of access: 03.07.2023). – Title from screen.

15. Active-Standby for High-Availability in FaaS [Electronic resource] / Yasmina Bouizem [et al.] // Middleware '20: 21st international middleware conference, Delft Netherlands. – New York, NY, USA, 2020. – Mode of access: https://doi.org/10.1145/3429880.3430097 (date of access: 25.05.2023). – Title from screen.

16. Adapa S. Indian smart cities and cleaner production initiatives–integrated framework and recommendations / S. Adapa // Journal of cleaner production. – 2018. – Vol. 172. – P. 3351–3366.

17. Agarwal A. QuicX - A protocol for effective communication in Linux clusters [Electronic resource] / Ankit Agarwal // 2012 international conference on

computing, electronics and electrical technologies (ICCEET), Nagercoil, Tamil Nadu, India, 21–22 March 2012. – Mode of access: https://doi.org/10.1109/icceet.2012.6203797 (date of access: 21.06.2023). – Title from screen.

18. Aggarwal C. Correction to: neural networks and deep learning [Electronic resource] / Charu Aggarwal // Neural networks and deep learning. – Cham, 2023. – Mode of access: https://doi.org/10.1007/978-3-031-29642-0_13 (date of access: 12.07.2023). – Title from screen.

19. Agrawal S. Integrating vertical and horizontal partitioning into automated physical database design [Electronic resource] / Sanjay Agrawal, Vivek Narasayya, Beverly Yang // The 2004 ACM SIGMOD international conference, Paris, France, 13–18 June 2004. – New York, New York, USA, 2004. – Mode of access: https://doi.org/10.1145/1007568.1007609 (date of access: 10.05.2023). – Title from screen.

20. A high availability application service platform for nuclear power enterprises [Electronic resource] / Zhe Sun [et al.] // 2016 IEEE 20th international conference on computer supported cooperative work in design (CSCWD), Nanchang, China, 4–6 May 2016. – Mode of access: https://doi.org/10.1109/cscwd.2016.7566060 (date of access: 24.05.2023). – Title from screen.

21. A high-availability cloud for research computing [Electronic resource] / Justin Riley [et al.] // Computer. – 2017. – Vol. 50, no. 6. – P. 92–95. – Mode of access: https://doi.org/10.1109/mc.2017.182 (date of access: 14.09.2022). – Title from screen.

22. Alhamid M. LSTM and bidirectional LSTM for regression [Electronic resource] / Mohammed Alhamid // towardsdatascience.com. – Mode of access: https://towardsdatascience.com/lstm-and-bidirectional-lstm-for-regression-4fddf910c655 (date of access: 06.09.2022). – Title from screen.

23. All one needs to know about fog computing and related edge computing paradigms: a complete survey [Electronic resource] / Ashkan Yousefpour [et al.] //

Journal of systems architecture. – 2019. – Vol. 98. – P. 289–330. – Mode of access: https://doi.org/10.1016/j.sysarc.2019.02.009 (date of access: 04.01.2023). – Title from screen.

24. Ansari S. Computer vision modeling on the cloud [Electronic resource] / Shamshad Ansari // Building computer vision applications using artificial neural networks. – Berkeley, CA, 2020. – P. 389–442. – Mode of access: https://doi.org/10.1007/978-1-4842-5887-3_10 (date of access: 07.09.2022). – Title from screen.

25. Application cluster service scheme for near-zero-downtime services [Electronic resource] / Fan-Tien Cheng [et al.] // 2005 IEEE international conference on robotics and automation, Barcelona, Spain. – Mode of access: https://doi.org/10.1109/robot.2005.1570743 (date of access: 04.04.2023). – Title from screen.

26. A recurrent latent variable model for sequential data [Electronic resource] / Junyoung Chung [et al.] // arxiv.org. – Mode of access: https://arxiv.org/pdf/1506.02216v3.pdf (date of access: 23.05.2023). – Title from screen.

27. Availability analysis of design configurations to compose virtual performance-optimized data center systems in next-generation cloud data centers [Electronic resource] / Daniel Rosendo [et al.] // Software: practice and experience. – 2020. – Vol. 50, no. 6. – P. 805–826. – Mode of access: https://doi.org/10.1002/spe.2833 (date of access: 14.12.2022). – Title from screen.

28. Bagui S. Database sharding [Electronic resource] / Sikha Bagui, Loi Tang Nguyen // International journal of cloud applications and computing. – 2015. – Vol. 5, no. 2. – P. 36–52. – Mode of access: https://doi.org/10.4018/ijcac.2015040103 (date of access: 06.12.2023). – Title from screen.

29. Barroso L. A. The datacenter as a computer: designing warehouse-scale machines, third edition [Electronic resource] / Luiz André Barroso, Urs Hölzle, Parthasarathy Ranganathan // Synthesis lectures on computer architecture. – 2018. –

Vol. 13, no. 3. – Mode of access: https://doi.org/10.2200/s00874ed3v01y201809cac046 (date of access: 14.03.2023). – Title from screen.

30. Bayer J. Learning stochastic recurrent networks [Electronic resource] / Justin Bayer, Christian Osendorfer // arxiv.org. – Mode of access: https://arxiv.org/abs/1411.7610 (date of access: 10.08.2023). – Title from screen.

31. Birman K. Adding high availability and autonomic behavior to Web services [Electronic resource] / K. Birman, R. van Renesse, W. Vogels // Proceedings. 26th international conference on software engineering, Edinburgh, UK. – Mode of access: https://doi.org/10.1109/icse.2004.1317410 (date of access: 07.03.2023). – Title from screen.

32. Botta A. Integration of cloud computing and internet of things: a survey / Alessio Botta, Walter de Donato, Valerio Persico // Future generation computer systems. – 2016. – Volume 56. – P. 684–700.

33. Cloud storage availability and performance assessment: a study based on NoSQL DBMS / C. Gomes [et al.] // The journal of supercomputing. – 2021. – P. 1–21.

34. Context aware computing for the Internet of Things / C. Perera [et al.] // IEEE communications surveys tutorials. – 2014. – Vol. 16. – P. 414–454.

35. Critchley T. High availability IT services / Terry Critchley: Auerbach Publications, 2014.

36. Data management in the cloud: limitations and opportunities [Electronic resource] // semanticscholar.org. – Mode of access: https://www.semanticscholar.org/paper/Data-Management-in-the-Cloud:-Limitations-and-Abadi/a3d1fb532bf4297ae9608bdc09ae5464ef58ee30 (date of access: 27.11.2023). – Title from screen.

37. Data modeling and nosql databases - A systematic mapping review [Electronic resource] / Harley Vera-Olivera [et al.] // ACM computing surveys. – 2021. – Vol. 54, no. 6. – P. 1–26. – Mode of access: https://doi.org/10.1145/3457608 (date of access: 16.07.2021). – Title from screen.

38. Davies A. MySQL clustering / Alex Davies, Harrison Fisk. Pearson Education, Limited, 2006. – 216 p.

39. Depth-Gated LSTM [Electronic resource] / Kaisheng Yao [et al.] // arxiv.org. – Mode of access: https://arxiv.org/abs/1508.03790 (date of access: 17.03.2023). – Title from screen.

40. Design and implementation of heartbeat in multi-machine environment [Electronic resource] / Zonghao Hou [et al.] // 17th international conference on advanced information networking and applications. AINA 2003, Xi'an, China. – Mode of access: https://doi.org/10.1109/aina.2003.1192949 (date of access: 07.06.2023). – Title from screen.

41. Dimitrievska J. Architecture and systems design for real-time IoT applications [Electronic resource] / Jasmina Dimitrievska // keitaro.com. – Mode of access: https://www.keitaro.com/2021/12/02/architecture-and-systems-design-for-real-time-iot-applications/ (date of access: 06.03.2023). – Title from screen.

42. Distefano S. Availability assessment of HA standby redundant clusters [Electronic resource] / Salvatore Distefano, Francesco Longo, Marco Scarpa // 2010 IEEE international symposium on reliable distributed systems (SRDS), New Delhi, Punjab India, 31 October – 3 November 2010. – Mode of access: https://doi.org/10.1109/srds.2010.37 (date of access: 03.07.2023). – Title from screen.

43. Distributed data service for data management in internet of things middleware [Electronic resource] / Ruben Cruz Huacarpuma [et al.] // Sensors. – 2017. – Vol. 17, no. 5. – P. 977. – Mode of access: https://doi.org/10.3390/s17050977 (date of access: 05.12.2023). – Title from screen.

44. Dukaric R. Towards a unified taxonomy and architecture of cloud frameworks [Electronic resource] / Robert Dukaric, Matjaz B. Juric // Future generation computer systems. – 2013. – Vol. 29, no. 5. – P. 1196–1210. – Mode of access: https://doi.org/10.1016/j.future.2012.09.006 (date of access: 21.02.2023). – Title from screen.

45. Edge computing in industrial internet of things: architecture, advances and challenges [Electronic resource] / Tie Qiu [et al.] // IEEE communications surveys & tutorials. – 2020. – Vol. 22, no. 4. – P. 2462–2488. – Mode of access: https://doi.org/10.1109/comst.2020.3009103 (date of access: 05.04.2023). – Title from screen.

46. Fan L. Research and implementation on model for high availability of enterprise information system [Electronic resource] / Li Fan, Wang Tiejun, Wang Liuyi // IERI procedia. – 2012. – Vol. 3. – P. 181–185. – Mode of access: https://doi.org/10.1016/j.ieri.2012.09.030 (date of access: 10.07.2023). – Title from screen.

47. Fan W. Mining big data: current status, and forecast to the future / W. Fan, A. Bifet // ACM SIGKDD explorations newsletter. – 2013. – Vol. 14, no. 2. – P. 1–5.

48. Cloud computing [Electronic resource] / Mine Omurgonulsen [et al.] // Journal of global information management. – 2021. – Vol. 29, no. 6. – P. 1–25. – Mode of access: https://doi.org/10.4018/jgim.20211101.oa40 (date of access: 01.10.2023). – Title from screen.

49. Fast compaction algorithms for nosql databases / M. Ghosh [et al.] // IEEE 35th international conference on distributed computing systems. – 2015. – P. 452–461.

50. Flex: high-availability datacenters with zero reserved power [Electronic resource] / Chaojie Zhang [et al.] // 2021 ACM/IEEE 48th annual international symposium on computer architecture (ISCA), Valencia, Spain, 14–18 June 2021. – Mode of access: https://doi.org/10.1109/isca52012.2021.00033 (date of access: 18.01.2023). – Title from screen.

51. Frampton M. The big data stack [Electronic resource] / Michael Frampton // Complete guide to open source big data stack. – Berkeley, CA, 2018. – P. 339–356. – Mode of access: https://doi.org/10.1007/978-1-4842-2149-5_10 (date of access: 14.09.2022). – Title from screen.

52. FT-Aurora: A highly available IaaS cloud manager based on replication [Electronic resource] / Gustavo B. Heimovski [et al.] // Computer networks. – 2020. – Vol. 168. – P. 107041. – Mode of access: https://doi.org/10.1016/j.comnet.2019.107041 (date of access: 14.06.2022). – Title from screen.

53. Gabriel G. Atomic transactions in distributed systems / G. Gabriel // Institute of informatics and applied mathematics university of berne. – 2002. – P. 7–10.

54. Gacovski Z. Natural language processing / Zoran Gacovski. Arcler Education Inc, 2020.

55. Gigli M. Internet of things: services and applications categorization [Electronic resource] / Matthew Gigli, Simon Koo // Advances in internet of things. – 2011. – Vol. 01, no. 02. – P. 27–31. – Mode of access: https://doi.org/10.4236/ait.2011.12004 (date of access: 05.10.2023). – Title from screen.

56. Great Learning Team. Types of neural networks and definition of neural network [Electronic resource] / Great Learning Team // mygreatlearning.com. – Mode of access: https://www.mygreatlearning.com/blog/types-of-neural-networks/. (date of access: 01.08.2023). – Title from screen.

57. Han Wang. Architectural design and implementation of highly available and scalable medical system with IBM Websphere middleware [Electronic resource] / Han Wang, Hao Wang, Jinmei Shen // Proceedings. 17th IEEE symposium on computer-based medical systems, Bethesda, MD, USA. – Mode of access: https://doi.org/10.1109/cbms.2004.1311710 (date of access: 14.09.2022). – Title from screen.

58. Hiep M. Q. Dynamic policy management system for high availability in a multi-site cloud [Electronic resource] / Mai Quang Hiep, Hyunsik Yang, Younghan Kim // 2020 international conference on information and communication technology convergence (ICTC), Jeju Island, 21–23 October 2020. – Mode of

access: https://doi.org/10.1109/ictc49870.2020.9289410 (date of access: 05.04.2023). – Title from screen.

59. High availability and scalability guide for DB2 on linux, unix, and windows. – [S. l. : s. n.], 2007. – 424 p.

60. High-availability deployment for large enterprises [Electronic resource] / Huahui Lyu [et al.] // 2016 international conference on progress in informatics and computing (PIC), Shanghai, China, 23–25 December 2016. – Mode of access: https://doi.org/10.1109/pic.2016.7949552 (date of access: 15.02.2023). – Title from screen.

61. High availability in clouds: systematic review and research challenges [Electronic resource] / Patricia T. Endo [et al.] // Journal of cloud computing. – 2016. – Vol. 5, no. 1. – Mode of access: https://doi.org/10.1186/s13677-016-0066-8 (date of access: 11.04.2023). – Title from screen.

62. High-Performance software load balancer for cloud-native architecture [Electronic resource] / Jung-Bok Lee [et al.] // IEEE access. – 2021. – Vol. 9. – P. 123704–123716. – Mode of access: https://doi.org/10.1109/access.2021.3108801 (date of access: 07.12.2023). – Title from screen.

63. Hirley A. Beyond 5g urllc evolution: new service modes and practical considerations / Alves Hirley, Do Jo Gweon, Shin JaeSheung // ITU journal on future and evolving technologies. – 2022. – Volume 3, Issue 3. – P. 545–554.

64. Internet of Things-enabled smart cities: state-of-the-art and future trends [Electronic resource] / Amir H. Alavi [et al.] // Measurement. – 2018. – Vol. 129. – P. 589–606. – Mode of access: https://doi.org/10.1016/j.measurement.2018.07.067 (date of access: 09.12.2020). – Title from screen.

65. Internet of Things (IoT): a vision, architectural elements, and future directions // Future generation computer systems. – 2013. – Vol. 29(7). – P. 1–19.

66. IoT reference model [Electronic resource] // ITU. – Mode of access: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060 (date of access: 01.07.2020). – Title from screen.

67. ITU common functional and non functional requirement of iot [Electronic resource] // ITU. – Mode of access: https://www.itu.int/rec/T-REC-Y.2066 (date of access: 16.06.2022). – Title from screen.

68. Jensen S. K. Time series management systems: a survey / S. K. Jensen, T. B. Pedersen, C. Thomsen // IEEE transactions on knowledge and data engineering. – 2017. – Vol. 29, no. 11. – P. 2581–2600.

69. Kaitovic I. Impact of failure prediction on availability: modeling and comparative analysis of predictive and reactive methods [Electronic resource] / Igor Kaitovic, Miroslaw Malek // IEEE transactions on dependable and secure computing. – 2018. – P. 1. – Mode of access: https://doi.org/10.1109/tdsc.2018.2806448 (date of access: 07.06.2023). – Title from screen.

70. Kalchbrenner N. Grid long short-term memory [Electronic resource] / Nal Kalchbrenner, Ivo Danihelka, Alex Graves // arxiv.org. – Mode of access: https://arxiv.org/abs/1507.01526 (date of access: 07.09.2022). – Title from screen.

71. Kanso A. Achieving high availability at the application level in the cloud [Electronic resource] / Ali Kanso, Yves Lemieux // 2013 IEEE 6th international conference on cloud computing (CLOUD), Santa Clara, CA, 28 June – 3 July 2013. – Mode of access: https://doi.org/10.1109/cloud.2013.24 (date of access: 07.06.2023). – Title from screen.

72. Katsaggelos A. Machine learning refined: foundations, algorithms, and applications / Aggelos Katsaggelos, Jeremy Watt, Reza Borhani. Cambridge University Press, 2020.

73. Koehn P. Neural machine translation / Philipp Koehn. Cambridge University Press, 2020. – 200 p.

74. Kona S. Compactions in apache cassandra. performance analysis of compaction strategies in apache cassandra / Srinand Kona // Msee-2016-21 : Corpus ID: 51813256, Karlskrona.

75. Kutz J. N. Data-Driven science and engineering: machine learning, dynamical systems, and control / J. Nathan Kutz, Steven L. Brunton. Cambridge University Press, 2019.

76. Lakshman A. Cassandra [Electronic resource] / Avinash Lakshman, Prashant Malik // ACM SIGOPS operating systems review. – 2010. – Vol. 44, no. 2. – P. 35–40. – Mode of access: https://doi.org/10.1145/1773912.1773922 (date of access: 12.08.2021). – Title from screen.

77. Learning phrase representations using RNN encoder–decoder for statistical machine translation [Electronic resource] / Kyunghyun Cho [et al.] // Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), Doha, Qatar. – Stroudsburg, PA, USA, 2014. – Mode of access: https://doi.org/10.3115/v1/d14-1179 (date of access: 07.09.2022). – Title from screen.

78. Lee G.-Y. Universal prediction system realization using RNN [Electronic resource] / Gwon-Yoon Lee, Sang-Boo Lee // The journal of korean institute of information technology. – 2018. – Vol. 16, no. 10. – P. 11–20. – Mode of access: https://doi.org/10.14801/jkiit.2018.16.10.11 (date of access: 27.01.2023). – Title from screen.

79. Lee Y.-J. A stochastic availability prediction model for head nodes in the HA cluster [Electronic resource] / Yong-Ju Lee, Hag-Young Kim, Cheol-Hoon Lee // 22nd international conference on advanced information networking and applications - workshops (aina workshops 2008), Gino-wan, Okinawa, Japan, 25–28 March 2008. – Mode of access: https://doi.org/10.1109/waina.2008.36 (date of access: 16.05.2023). – Title from screen.

80. Levitin G. Cold vs. hot standby mission operation cost minimization for 1-out-of-N systems [Electronic resource] / Gregory Levitin, Liudong Xing, Yuanshun Dai // European journal of operational research. – 2014. – Vol. 234, no. 1. – P. 155–162. – Mode of access: https://doi.org/10.1016/j.ejor.2013.10.051 (date of access: 17.04.2023). – Title from screen.

81. Lieberman D. A. Learning and memory / David A. Lieberman. Cambridge University Press, 2020.

82. Li W. Leveraging linux containers to achieve high availability for cloud services [Electronic resource] / Wubin Li, Ali Kanso, Abdelouahed Gherbi // 2015 IEEE international conference on cloud engineering (IC2E), Tempe, AZ, USA, 9–13 March 2015. – Mode of access: https://doi.org/10.1109/ic2e.2015.17 (date of access: 04.04.2023). – Title from screen.

83. LSTM: a search space odyssey [Electronic resource] / Klaus Greff [et al.] // IEEE transactions on neural networks and learning systems. – 2017. – Vol. 28, no. 10. – P. 2222–2232. – Mode of access: https://doi.org/10.1109/tnnls.2016.2582924 (date of access: 09.05.2023). – Title from screen.

84. LSTM layer [Electronic resource] // keras.io. – Mode of access: https://keras.io/api/layers/recurrent_layers/lstm/ (date of access: 03.10.2022). – Title from screen.

85. Lu B. Research on cassandra data compaction strategies for time-series data [Electronic resource] / Bai Lu, Yang Xiaohui // Journal of computers. – 2016. – P. 504–512. – Mode of access: https://doi.org/10.17706/jcp.11.6.504-512 (date of access: 15.12.2021). – Title from screen.

86. Lu Y. Internet of Things: a systematic review of the business literature from the user and organisational perspectives [Electronic resource] / Yang Lu, Savvas Papagiannidis, Eleftherios Alamanos // Technological forecasting and social change. – 2018. – Vol. 136. – P. 285–297. – Mode of access: https://doi.org/10.1016/j.techfore.2018.01.022 (date of access: 07.03.2023). – Title from screen.

87. Mahmood N. H. Uplink Grant-Free Access Solutions for URLLC services in 5G New Radio / N. H. Mahmood // 16th intern. symp. on wireless commun. syst. (ISWCS). – 2019. – P. 607–612.

88. Manaswi N. K. RNN and LSTM [Electronic resource] / Navin Kumar Manaswi // Deep learning with applications using python. – Berkeley, CA, 2018. –

P. 115–126. – Mode of access: https://doi.org/10.1007/978-1-4842-3516-4_9 (date of access: 06.03.2023). – Title from screen.

89. Mansouri Y. Data storage management in cloud environments [Electronic resource] / Yaser Mansouri, Adel Nadjaran Toosi, Rajkumar Buyya // ACM computing surveys. – 2018. – Vol. 50, no. 6. – P. 1–51. – Mode of access: https://doi.org/10.1145/3136623 (date of access: 12.10.2022). – Title from screen.

90. Michelucci U. Advanced applied deep learning: convolutional neural networks and object detection / Umberto Michelucci. Apress, 2019. – 303 p.

91. Michelucci U. Applied deep learning: a case-based approach to understanding deep neural networks / Umberto Michelucci. Apress, 2018. – 432 p.

92. Mici L. A self-organizing neural network architecture for learning human-object interactions [Electronic resource] / Luiza Mici, German I. Parisi, Stefan Wermter // Neurocomputing. – 2018. – Vol. 307. – P. 14–24. – Mode of access: https://doi.org/10.1016/j.neucom.2018.04.015 (date of access: 08.03.2023). – Title from screen.

93. Mysql 5.1 cluster dba certification study guide / Jon Stephens [et al.]. Lulu.Com, 2007.

94. DasGupta B. Neural Networks [Electronic resource] / Bhaskar DasGupta, Derong Liu, Hava T. Siegelmann // Handbook of Approximation Algorithms and Metaheuristics, Second Edition. – P. 345–359. – Mode of access: https://doi.org/10.1201/9781351236423-20 (date of access: 07.09.2021). – Title from screen.

95. Nguyen T. A. Availability modeling and analysis of a data center for disaster tolerance [Electronic resource] / Tuan Anh Nguyen, Dong Seong Kim, Jong Sou Park // Future generation computer systems. – 2016. – Vol. 56. – P. 27–50. – Mode of access: https://doi.org/10.1016/j.future.2015.08.017 (date of access: 12.07.2023). – Title from screen.

96. Oracle business intelligence with machine learning: artificial intelligence techniques in obiee for actionable bi. – [S. l. : s. n.].

97. Peterson C. Generating rhyming poetry using LSTM recurrent neural networks [Electronic resource] : thesis / Peterson Cole. – Mode of access: http://hdl.handle.net/1828/10801 (date of access: 06.02.2023). – Title from screen.

98. Prasad S. Smart meter data analytics using OpenTSDB and Hadoop / S. Prasad, S. Avinash // Innovative smart grid technologies-asia (ISGT asia). – 2013. – P. 1–6.

99. Rabbat R. A high-availability clustering architecture with data integrity guarantees [Electronic resource] / R. Rabbat, T. McNeal, T. Burke // Proceedings 2001 IEEE international conference on cluster computing, Las Vegas, Nevada, USA, 14–17 October 2001. – Mode of access: https://doi.org/10.1109/clustr.2001.959974 (date of access: 26.05.2023). – Title from screen.

100. Ranade D. Shared data clusters: scaleable, manageable, and highly available systems (VERITAS series) / Dilip Ranade. Wiley, 2002. – 430 p.

101. Reka S. S. Future effectual role of energy delivery: A comprehensive review of Internet of Things and smart grid [Electronic resource] / S. Sofana Reka, Tomislav Dragicevic // Renewable and sustainable energy reviews. – 2018. – Vol. 91. – P. 90–108. – Mode of access: https://doi.org/10.1016/j.rser.2018.03.089 (date of access: 14.07.2022). – Title from screen.

102. RemusDB: transparent high availability for database systems [Electronic resource] / Umar Farooq Minhas [et al.] // The VLDB Journal. – 2012. – Vol. 22, no. 1. – P. 29–45. – Mode of access: https://doi.org/10.1007/s00778-012-0294-6 (date of access: 06.06.2023). – Title from screen.

103. Remus: high availability via asynchronous virtual machine replication / Brendan Cully [et al.] // Proceedings of the 5th USENIX symposium on networked systems design and implementation. – 2008. – P. 161–174.

104. Resource allocation based on redundancy models for high availability cloud [Electronic resource] / Glauco Estácio Gonçalves [et al.] // Computing. – 2019. – Vol. 102, no. 1. – P. 43–63. – Mode of access:

https://doi.org/10.1007/s00607-019-00728-1 (date of access: 25.07.2023). – Title from screen.

105. Salvaris M. Deep learning with azure: building and deploying artificial intelligence solutions on the microsoft AI platform / Mathew Salvaris. Apress, 2018. – 312 p.

106. Sheghdara M. Automatic retrieval and analysis of high availability scenarios from system execution traces: a case study on hot standby router protocol [Electronic resource] / Maged Sheghdara, Jameleddine Hassine // Journal of systems and software. – 2020. – Vol. 161. – P. 110490. – Mode of access: https://doi.org/10.1016/j.jss.2019.110490 (date of access: 11.04.2023). – Title from screen.

107. Somasekaram P. High-availability clusters: a taxonomy, survey, and future directions [Electronic resource] / Premathas Somasekaram, Radu Calinescu, Rajkumar Buyya // Journal of systems and software. – 2022. – Vol. 187. – P. 111208. – Mode of access: https://doi.org/10.1016/j.jss.2021.111208 (date of access: 21.12.2022). – Title from screen.

108. Symmetric active/active high availability for high-performance computing system services: accomplishments and limitations [Electronic resource] / C. Engelmann [et al.] // 2008 8th IEEE international symposium on cluster computing and the grid (ccgrid), Lyon, France, 19–22 May 2008. – Mode of access: https://doi.org/10.1109/ccgrid.2008.78 (date of access: 11.05.2023). – Title from screen.

109. Thawani P. Understanding architecture of internet of things [Electronic resource] / Pragati Thawani // researchgate.net. – Mode of access: https://www.researchgate.net/publication/362490707_Understanding_Architecture_of_Internet_of_Things (date of access: 10.08.2023). – Title from screen.

110. The embedded iot time series database for hybrid solid-state storage system [Electronic resource] / Tao Cai [et al.] // Scientific programming. – 2021. – Vol. 2021. – P. 1–13. – Mode of access: https://doi.org/10.1155/2021/9948533 (date of access: 21.06.2022). – Title from screen.

111. Uhlemann K. JOSHUA: symmetric active/active replication for highly available HPC job and resource management [Electronic resource] / K. Uhlemann, C. Engelmann, S. Scott // 2006 IEEE international conference on cluster computing, Barcelona, Spain, 25–28 September 2006. – Mode of access: https://doi.org/10.1109/clustr.2006.311855 (date of access: 15.12.2022). – Title from screen.

112. Schelhaas W. Predicting network performancein IoT environments using LSTM [Electronic resource] : thesis / Schelhaas Wietze. – Mode of access: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-454062 (date of access: 27.06.2022). – Title from screen.

113. Vercellis C. Business intelligence: data mining and optimization for decision making / C. Vercellis. John Wiley & Sons, 2011.

114. Vongsingthong S. A review of data management in internet of things / Suwimon Vongsingthong, Sucha Smanchat // KKU research journal. – 2015. – P. 215–240.

115. Waddington D. G. A fast lightweight time-series store for iot data / D. G. Waddington, C. Lin // CoRR - computing research repository. – 2016.

116. Wen Z. Design and implementation of high-availability paas platform based on virtualization platform [Electronic resource] / Zepeng Wen, Yan Liang, Gongliang Li // 2020 IEEE 5th information technology and mechatronics engineering conference (ITOEC), Chongqing, China, 12–14 June 2020. – Mode of access: https://doi.org/10.1109/itoec49072.2020.9141564 (date of access: 24.05.2023). – Title from screen.

117. Wu J. Essentials of pattern recognition: an accessible approach / Jianxin Wu. : University of Cambridge ESOL Examinations, 2020. – 395 p.

118. Xiong H. A database-specific pattern for multi-cloud high availability and disaster recovery [Electronic resource] / Huanhuan Xiong, Frank Fowley, Claus Pahl // Communications in computer and information science. – Cham, 2016. – P. 374–388. – Mode of access: https://doi.org/10.1007/978-3-319-33313-7_29 (date of access: 11.10.2022). – Title from screen.

119. Zaidan A. A survey on communication components for iot-based technologies in smart homes / A. Zaidan, B. Zaidan, M. Qahtan // Telecommunication systems. – 2018. – Vol. 69. – P. 1–25.

120. Zhu L. A SCSI3 persistent reservation synchronization solution for iscsi targets cluster hosting ceph RBD with active/active connections [Electronic resource] / Lingshan Zhu, Jinlong Lin // 2020 IEEE 4th information technology, networking, electronic and automation control conference (ITNEC), Chongqing, China, 12–14 June 2020. – Mode of access: https://doi.org/10.1109/itnec48623.2020.9085061 (date of access: 04.10.2023). – Title from screen.

121. Zhu S. Creating a NoSQL database for the Internet of Things : Creating a key-value store on the SensibleThings platform / S. Zhu // PhD thesis, mid sweden university, faculty of science, technology and media, department of information and communication systems, Sundsvall, 2015.

Додаток А

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІ

1. Хоменчук В. Аналіз впливу хмарних обчислень і граничних обчислень на продуктивність мережі / В. Хоменчук // Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу, Київ, 28 листоп. 2023 р. – С. 361-362.

2. Хоменчук В. Використання баз даних часових рядів в архітектурі граничних обчислень. / В. Хоменчук // Звя'зок. – 2023. – № 1. – С. 39–42.

3. Хоменчук В. Переваги використання apache iotdb для адаптивних систем IoT / В. Хоменчук // Сучасний стан та перспективи розвитку IoT, Київ, 9 квіт. 2021 р. – С. 94-95.

4. Хоменчук В. Переваги та недоліки алгоритмів розподілу даних в кластері баз даних / В. Хоменчук // Сучасний стан та перспективи розвитку IoT, Київ, 7 квіт. 2023 р. – С. 95-97.

5. Хоменчук В. Стратегій розподілу навантаження децентралізованих баз даних / В. Хоменчук // Звя'зок. – 2023. – № 5. – С. 21–32.

6. Шефкін Б. В., Красюк І. В., Хоменчук В. О. та ін. Дослідження та впровадження нейронної мережі на основі TENSORFLOW // Зв’язок. – 2020. – № 6. – С. 18–20.

7. Свердлюк Б. І., Каграманова Ю. К., Хоменчук В. О. та ін. Керування давачами Розумного будинку за допомогою голосового помічника Google Assistant // Зв’язок. – 2021. – № 3. – С. 54–57.

Додаток Б

training.py

```
import data_generator
import numpy as np
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import LSTM

import config

tags = data_generator.tags(tag_count, day_interval_count)

servers_ed = []
for i in range(lstm_count):
    servers_ed.append( data_generator.servers(tags, server_count, tag_count) )

servers_ed_LA_old = data_generator.servers_LA(servers_ed, server_count,
day_interval_count)

servers_ed.pop(0)
servers_ed.append( data_generator.servers(tags, server_count, tag_count) )

servers_ed_LA = data_generator.servers_LA(servers_ed, server_count,
day_interval_count)
last_server_bool = data_generator.servers_to_bool(servers_ed)
```

```
#[day][server][LA+tag_bool_LA]
X_all_LA_bool_LA = []
Y_all_bool_LA = []
for i in range(data_count+lstm_count):
    print(i)
    #[server][LA+tag_bool_LA]
    LA_bool_LA = []
    for s in range(server_count):
        #LA_bool_LA.append(servers_ed_LA_old[s] + last_server_bool[s] +
servers_ed_LA[s])
        LA_bool_LA += servers_ed_LA_old[s] + last_server_bool[s] +
servers_ed_LA[s]

    X_all_LA_bool_LA.append(LA_bool_LA)

    servers_ed_LA_old = list(servers_ed_LA)
    servers_ed.pop(0)
    servers_ed.append( data_generator.servers(tags, server_count, tag_count) )

    servers_ed_LA = data_generator.servers_LA(servers_ed, server_count,
day_interval_count)
    last_server_bool = data_generator.servers_to_bool(servers_ed)

    #[server][tag_bool_LA]
    bool_LA = []
    for s in range(server_count):
        bool_LA += last_server_bool[s] + servers_ed_LA_old[s]

    Y_all_bool_LA.append( bool_LA )
```

```
X = []
Y = []
for data in range(data_count):
    X.append([X_all_LA_bool_LA[data+i] for i in range(lstm_count)])
    Y.append(Y_all_bool_LA[data+lstm_count-1])

model = Sequential( )

model.add(LSTM(server_count*(day_interval_count+tag_count),input_shape=(lst
m_count, server_count * (day_interval_count*2+tag_count) ),
return_sequences=False))

model.compile(loss='cosine_similarity',optimizer ='adam',metrics=['accuracy'])

model.summary()
his =
model.fit(X[:int(data_count*0.8)],Y[:int(data_count*0.8)],epochs=100,batch_size=
20,validation_data=(X[int(data_count*0.2):], Y[int(data_count*0.2):]),verbose=1);

model.save('models/tsdb_sharding.keras')

plt.subplot(1,2,1)
plt.plot(his.history['loss'])        #blue
plt.plot(his.history['val_loss'])   #orange
plt.grid(True)
plt.subplot(1,2,2)
plt.plot(his.history['accuracy'])
plt.plot(his.history['val_accuracy'])
plt.grid(True)
plt.show()
```

data_generator.py

```
from random import randint, random, lognormvariate

def tags(tag_count, day_interval_count):
    tags = []
    for i in range(tag_count):
        interval = []
        for j in range(randint(1, day_interval_count)):
            interval.append(randint(0, day_interval_count-1))
        interval = list(set(interval))
        interval.sort()
        tags.append( [interval, randint(1, 3)*2 ,  lognormvariate(0.5,4) / tag_count ]   )
    return tags

def servers(tags, server_count, tag_count):
    servers = []
    for i in range(server_count):
        servers.append([])
        for j in range(tag_count):
            servers[i].append([-1,-1,-1])
    for tag in range(tag_count):
        servers[randint(0, server_count-1)][tag] = tags[tag]
     return servers

def servers_LA_clear(server_count, day_interval_count):
    servers_ed_LA = []
    for i in range(server_count):
        interval = []
        for d in range(day_interval_count):
```

```
            interval.append(0)
        servers_ed_LA.append(interval)
    return servers_ed_LA

def servers_LA(servers_ed, server_count, day_interval_count):
    servers_ed_LA = servers_LA_clear(server_count, day_interval_count)

    day = 1
    for servers_day in servers_ed[::-1]:
        for serv_id in range(len(servers_day)):
            for tag in servers_day[serv_id]:
                if tag[2] >= 0 and tag[1] >= day:
                    for interval in tag[0]:
                        #print(str(serv_id)+' '+str(interval)+' '+str(tag[2]))
                        servers_ed_LA[serv_id][interval] += tag[2]

        day+=1
    return servers_ed_LA

def servers_to_bool(servers_ed):
    servers_last_day_bool = []
    for server in servers_ed[-1]:
        tags_bool = []
        for tag in server:
            if tag[2]>=0:
                tags_bool.append(True)
            else:
                tags_bool.append(False)
        servers_last_day_bool.append(tags_bool)
    return servers_last_day_bool
```

Додаток В

HUAWEI
ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «ХУАВЕЙ УКРАЇНА»
LIMITED LIABILITY COMPANY "HUAWEI UKRAINE"
ТОВ «ХУАВЕЙ УКРАЇНА» / LLC "HUAWEI UKRAINE"

03124, Київ, вул. Радищева, буд. 10/14, (ідентифікаційний код: 33501267), тел. 498-77-18, факс 498-77-19
10/14, Radischeva street, Kiev, 03124, Ukraine, (Company Code: 33501267), tel. 498-77-18, fax 498-77-19

ЗАТВЕРДЖУЮ
Заступник директора з питань персоналу
ТОВ "Хуавей Україна"
_____ В.Б. Білавка
(підпис, ініціали, прізвище)
«16» вересня 2023 р.

АКТ
про реалізацію результатів дисертаційної роботи
Хоменчука Владислава Олеговича
на тему: «Методика побудови адаптивних кластерів комп'ютерних систем для збору та обробки інформації з пристроїв IoT»

Матеріали роботи використано в науково-дослідних і дослідно-конструкторських роботах, що ведуться в ТОВ "Хуавей Україна", а саме, для вдосконалення платформи HUAWEI FusionSphere для розробки модуля контролю навантаження в кластері, яка надає інструментарій для впровадження можливостей машинного навчання контролю за навантаженням. Розроблені методи, що належать автору Хоменчуку В.О., показали свою ефективність, дозволяючи підвищити продуктивність використання серверних ресурсів спеціально для систем Інтернету речей (IoT). Застосування цих методів відкриває нові перспективи для оптимізації та управління серверними ресурсами, що є критично важливим у сучасних інформаційних технологіях, особливо у контексті розвитку IoT-систем. Такий підхід не лише сприяє підвищенню ефективності використання інфраструктури, але також створює підґрунтя для подальших досліджень та інновацій у сфері обчислювальної техніки та інформаційних технологій.

Заступник директора з питань
персоналу ТОВ "Хуавей Україна" В.Б. Білавка

КИЇВСТАР

ЗАТВЕРДЖУЮ
Начальник департаменту
Департамент транспортної мережі
_____ С.В. Кирилюк
«22» вересня 2023 р.
ПрАТ "Київстар"
ЕДРПОУ 21673832
03113, м. Київ, вул. Дегтярівська, 53,
www.kyivstar.ua

АКТ
про реалізацію результатів дисертаційної роботи
Хоменчука Владислава Олеговича
на тему: «Методика побудови адаптивних кластерів комп'ютерних систем для збору та обробки інформації з пристроїв IoT»

Розроблена методика є перспективним рішенням для побудови кластерних баз даних часових рядів. З використанням зазначених в методиці підходів та принципів були розроблені технічні рішення, які націлені на удосконалення архітектури кластерних систем Інтернету речей в компанії ПрАТ "Київстар". Нові технічні рішення, на основі розробленої методики прискорили роботу систем збору та аналітики даних отриманих від пристроїв IoT. Їх впровадження дозволило підвищити швидкість аналітичних процесів в середньому на 20%.

Начальник департаменту С.В. Кирилюк
Департамент транспортної мережі

Підпис нач. департаменту транспортної мережі С.В. Кирилюка завіряю /Орлова Л.М./ Г. інспектор кадрів

КИЇВСТАР

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «КИЇВСТАР»

03115, м. Київ, пр.德ержинського, 55
тел.: +380 44 207 10 75, факс: +380 44 2 17 21 03
www.kyivstar.ua

Від 23.10.20 р.
№ 26992/06

Щодо законодавчого регулювання використання печаток

Приватне акціонерне товариство «Київстар» (надалі – ПрАТ «Київстар») висловлює Вам свою повагу та у відповідь на Ваш запит щодо необхідності проставлення печатки підприємства повідомляє наступне.

Застосування печаток підприємством у своїй господарській діяльності визначається, серед іншого, положеннями Господарського кодексу України та іншими законодавчими актами. Аналізуючи норми чинного в Україні законодавства щодо регулювання застосування печаток випливає наступне:

1. Господарський кодекс у статті 58[1] визначає, що суб'єкт господарювання **має право** використовувати у своїй діяльності печатки. Використання суб'єктом господарювання печатки **не є обов'язковим**. Відбиток печатки **не може бути обов'язковим** реквізитом будь-якого документа, що подається суб'єктом господарювання до органу державної влади або органу місцевого самоврядування. **Наявність або відсутність відбитка печатки суб'єкта господарювання на документі не створює юридичних наслідків.**

Отже, законодавець визначив проставлення печатки як право суб'єкта господарювання, а не обов'язок та встановив неприпустимість вимоги проставлення такої печатки з боку державних органів та суб'єктів господарювання усіх форм власності.

2. У свою чергу, Цивільний кодекс України приписами статті 207 встановлює виключний перелік вимог до письмової форми правочину для визнання його вчиненим, серед яких:

- правочин **вважається таким, що вчинений** у письмовій формі, якщо він підписаний його стороною (сторонами).

Як вбачається з аналізу положень зазначених цивільних норм права, наявність або відсутність печатки підприємства не має юридичних наслідків для визнання договору вчиненим.

3. Закон України «Про бухгалтерський облік та фінансову звітність в Україні» в статті 9 визначає, що підставою для бухгалтерського обліку господарських операцій є первинні документи та встановлює **виключний перелік обов'язкових реквізитів** первинних документів:

- назва документа (форми);
- дата складання;
- назва підприємства, від імені якого складено документ;
- зміст й обсяг господарської операції, одиниця виміру господарської операції;
- посади осіб, відповідальних за здійснення господарської операції та правильність її оформлення;
- особистий підпис або інші дані, що надають змогу ідентифікувати особу, яка брала участь у здійсненні господарської операції.

Враховуючи зазначене, печатка не є обов'язковим реквізитом первинних документів та не може вимагатися контролюючими державними органами.

4. Одночасно, слід звернути увагу на позицію Міністерства юстиції України, висловлену в листі від 09.04.2015 р. N 265-0-2-15/8.1 «Щодо застосування положень Закону України від 15 квітня 2014 року N 1206-VII "Про внесення змін до деяких законодавчих актів України щодо спрощення порядку відкриття бізнесу". Аналізуючи положення чинного законодавства Міністерство юстиції зазначило, що з Господарського кодексу України виключені норми, які передбачали обов'язковість використання печаток. При цьому відповідно до Загальних положень Господарського кодексу України основним установчим документом, що регулює діяльність господарського товариства, є статут. До установчих документів можуть бути включені також відомості щодо інших умов діяльності господарського товариства, які не суперечать закону. Отже, якщо господарське товариство у процесі своєї діяльності не

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «КИЇВСТАР»

03117, м. Київ, вул. Достоєвського, 52,
тел.: +380 44 299 00 70, факс: +380 44 232 21 01
www.kyivstar.ua

КИЇВСТАР

має бажання надалі використовувати печатку, воно має право відмовитися від неї. При цьому з огляду на те, що статут є основним установчим документом, який регулює діяльність господарського товариства, положення щодо застосування або не застосування печатки господарським товариством має бути визначене в статуті такого товариства.

На сьогоднішній день установчими документами ПрАТ «Київстар» встановлено відсутність обов'язковості застосування печатки в господарській діяльності товариства.

Враховуючи зазначене та чинний правовий порядок в Україні, ПрАТ «Київстар» повідомляє про незастосування з 07.09.2020 року у своїй господарській діяльності печатки підприємства під час укладання договорів, підписання первинних бухгалтерських та будь-яких інших документів, надання звітності, що обгрунтовується чинним на момент надсилання цього листа законодавством України.

З повагою,
Начальник відділу кадрів **Мельниченко О.П.**

Додаток Г

"ЗАТВЕРДЖУЮ"
Проректор з науково-педагогічної
роботи ДУІКТ,
Доктор технічних наук, професор
_____ Беркман Л.Н.
«14» вересня 2023 р.

АКТ

про реалізацію результатів дисертаційної роботи
Хоменчука Владислава Олеговича на тему: «Методика побудови адаптивних кластерів комп'ютерних систем для збору та обробки інформації з пристроїв IoT» у навчальному процесі Державного університету інформаційно-комунікаційних технологій

Комісія у складі: голови комісії - директора Навчально-наукового інституту інформаційних технологій, доктора технічних наук, професора Бондарчука А.П., та членів комісії - завідувача кафедри Інженерії програмного забезпечення автоматизованих систем, доктора технічних наук, професора Сторчак К.Н.; завідувача кафедри Штучного інтелекту, доктора технічних наук, доцента Зінченко О.В.; завідувача кафедри Інженерії програмного забезпечення доктора технічних наук, доцента Замрій І.В. розглянула дисертаційну роботу Хоменчука Владислава Олеговича на тему: «Методика побудови адаптивних кластерів комп'ютерних систем для збору та обробки інформації з пристроїв IoT» та публікації автора за матеріалами дисертаційної роботи. Результати впроваджено в начальний процес Навчально-наукового інституту інформаційних технологій, а саме:

1. Розроблену методику навчання нейронної мережі для розподілу навантаження.
2. Розроблену модель нейронної мережі для аналізу даних з довгою короткочасною пам'яттю.

На основі аналізу представлених матеріалів комісія установила, що отримані наукові результати використовуються в навчальному процесі Державного університету інформаційно-комунікаційних технологій при викладанні навчальних дисциплін: «Моделювання IoT advance» та «Штучний інтелект».

Голова комісії

Директор ІНІТ
доктор технічних наук, професор — А.П. Бондарчук

Члени комісії

Завідувач кафедри Інженерії програмного забезпечення автоматизованих систем, доктор технічних наук, професор — Сторчак К.Н.

Завідувач кафедри Штучного інтелекту, доктор технічних наук, доцент — Зінченко О.В.

Завідувач кафедри Інженерії програмного забезпечення, доктор технічних наук, доцент — Замрій І.В.

Документ підписано у сервісі Вчасно (початок)
_____ALL (4).pdf

Документ підписано у сервісі Вчасно (продовження)
_____ALL (4).pdf

Документ відправлено: 12:53 15.12.2023

Власник документу

Електронний підпис
12:53 15.12.2023
Ідентифікаційний код: 3485714550
ХОМЕНЧУК ВЛАДИСЛАВ ОЛЕГОВИЧ
Власник ключа: ХОМЕНЧУК ВЛАДИСЛАВ ОЛЕГОВИЧ
Час перевірки КЕП/ЕЦП: 12:53 15.12.2023
Статус перевірки сертифікату: Сертифікат діє
Серійний номер: 248197DDFAB977E504000000962309011BED2E04
Тип підпису: кваліфікований