Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод обробки інформаційних потоків у туманному середовищі

(тема)

Виконав:

студент II курсу, групи СПм-22-3

Меженський О. О.
(прізвище, ініціали)

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

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

Освітня програма Системне програмування
(повна назва освітньої програми)

Керівник: проф. Кучук Г.А.
(посада, прізвище, ініціали)

Допускається до захисту

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

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Меженському Олександру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод обробки інформаційних потоків у туманному середовищі

затверджена наказом по університету від “ 01 ” квітня 2024 р. № 257 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи операційна система –*Windows*

4. Перелік питань, що потрібно опрацювати у роботі

1) Провести аналітичний існуючих методів обробки інформаційних потоків.
2) Обґрунтувати необхідність застосування концепції рефакторінга інформаційних
3) Розробити математичну модель процесу виділення мікропотоків завдань.
4) Удосконалити метод і розробити алгоритм методу обробки інформаційних потоків у туманному середовищі.
5) Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 15 слайдів.

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз літературних джерел-	02.04.24-08.04.24	1
2	Вибір та обгрунтування методики дослідження	09.04.24-16.04.24	2
3	Вибір інструментальних засобів	17.04.24-22.04.24	3
4	Розробка моделей протоколів	23.04.24-06.05.24	4
5	Проведення експериментів	07.05.24-23.05.24	5
6	Оформлення матеріалів кваліфікаційної роботи	24.05.24-03.06.24	6
7	Подання кваліфікаційної роботи керівникові та	04.06.24-07.06.24	7
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	08.06.24-12.06.24	8
	вання		

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кучук Г.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 79 с., 23 рис., 2 табл., 2 дод., 13 джерел.

ТУМАННЕ СЕРЕДОВИЩЕ, МОНОЛІТНИЙ ПОТІК, МІКРОПОТІК, РЕФАКТОРІНГ, ІНТЕРНЕТ РЕЧЕЙ

Метою кваліфікаційної роботи є зменшення часу обробки інформації, що надходить з кінцевих пристроїв Інтернету речей або Цифрових двійників шляхом розробки методу обробки інформаційних потоків у туманному середовищі, котрий, враховуючи особливості таких мереж, дозволить застосувувати потокову обробку замість пакетної.

Для досягнення мети був проведений аналіз існуючих методів обробки інформаційних потоків Інтернету речей; визначені характерні особливості туманного середовища; обґрунтована необхідність застосування концепції рефакторінга інформаційних потоків; проведено моделювання процедури виділення мікропотоків завдань із вхідного монолітного потоку; удосконалений метод обробки інформаційних потоків у туманному середовищі;проведено порівняльне дослідження запропонованого методу обробки інформаційних потоків у туманному середовищі.

Предметом дослідження методи обробки інформаційних потоків у туманному середовищі. Об’єктом дослідження є процес обробки даних при проведенні туманних обчислень.

ABSTRACT

Master’s thesis: 79 pages, 23 figures, 2 tables, 2 appendices, 13 sources.

FOG ENVIRONMENT, MONOLITHIC FLOW, MICROFLOW, REFACTORING, INTERNET OF THINGS

The goal of the qualification work is to reduce the time of processing information coming from end devices of the Internet of Things or Digital Doubles by developing a method of processing information flows in a foggy environment, which, taking into account the features of such networks, will allow the application of flow processing instead of batch processing.

To achieve the goal, an analysis of existing methods of processing information flows of the Internet of Things was carried out; characteristic features of the foggy environment are determined; justified necessity of applying the concept of information flow refactoring; simulation of the procedure for selecting microflows of tasks from the input monolithic flow was carried out; an improved method of processing information flows in a foggy environment; a comparative study of the proposed method of processing information flows in a foggy environment was conducted.

The subject of research is methods of processing information flows in a foggy environment. The object of the study is the process of data processing during fog computing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВМ – віртуальна машина

ІКТ – інформаційно-комунікаційні технології

ІР – Інтернет речей

МВ – мегабайт

ОС – операційна система

ПК – персональний комп’ютер

ТС – туманне середовище

API – Application Programming Interface

DSPE – Data Stream Processing Engines

DT – Digital Twin (Цифровий двійник)

EDA – EventDriven Architecture

IF – InfFlow

IoT – Internet of Things

MIF – MicroInfFlow

PL – processing layer

PLM – Product Lifecycle Management

SI - Smart Industry

SL – storage layer

SSL – Secure Sockets Layer

TLS – Transport Layer Security

VPN – Virtual Private Networks

ВСТУП

Актуальність теми проведеного дослідження ґрунтується на таких основних факторах:

– експоненційне зростання пристроїв і систем Інтернету речей;

– четверта індустріальна революція та розвиток технологій, пов'язаних з Цифровими двійниками;

– природні обмеження застосування моделі хмарних обчислень, пов'язані з латентністю при обробці потоків даних;

– необхідність розробки та дослідження нових підходів та моделей організації обчислювального процесу в гетерогенних розподілених обчислювальних середовищах, що забезпечують обробку потоків даних, зокрема в туманних середовищах.

Вважаючи на необхідність збору, передачі і аналізу потоків даних від Цифрових двійників та систем промислового Інтернету речей в режимах, близьких до реальному часу, для налаштування актуалізації їх віртуального стану, застосування хмарних технологій не дозволяє забезпечити потрібні характеристики наданих обчислювальних ресурсів з точки зору часу затримки і місцезнаходження сервісів обробки даних.

Можливим вирішенням цієї проблеми може бути застосування моделі туманних обчислень, яка розширює концепцію хмарних обчислень, надаючи обчислювальні ресурси ближче до джерел даних. Туманні обчислення – це багаторівнева модель, що забезпечує повсюдний доступ до загальної множини масштабованих обчислювальних ресурсів та підтримуюча розгортання розподілених застосунків та сервісів з урахуванням латентності. Хоча промислові дані часто є неструктурованими, їх можна уточнити та попередньо обробити локально на рівні туманних обчислень перед відправкою на рівень хмари для подальшої обробки. Концепція туманних

обчислень дозволяє перенести частину завдань по обробці і зберіганню даних з хмари на туманні вузли на границі мережі для зниження затримки.

Аналіз досліджень, спрямованих на декомпозицію потоків завдань на підпотоки показує, що при вирішенні цього завдання залишаються сильні зв'язки між підпотоками, а потоки завдань реалізуються в форматі пакетної обробки даних, що не дозволяє застосувати ці рішення в контексті подійно-керованої архітектури для підтримки систем Інтернету речей у туманних обчислювальних середовищах. Також, більшість методів не фокусуються на потребах туманних обчислень, які включають необхідність географічного розподілу, а також необхідність слабопов'язаної архітектури не тільки між даними і обробкою, але і в самому шарі обробки, де кожен обчислювальний об'єкт може бути реалізований як незалежний сервіс.

Отже, спираючись на вищевикладене, науково-технічне завдання, спрямоване на розробку методу обробки інформаційних потоків у туманному середовищі є актуальним та доцільним.

Предметом дослідження є методи обробки інформаційних потоків у туманному середовищі.

Об’єктом дослідження є процес обробки даних при проведенні туманних обчислень.

Метою дослідження є зменшення часу обробки інформації, що надходить з кінцевих пристроїв Інтернету речей або Цифрових двійників шляхом розробки методу обробки інформаційних потоків у туманному середовищі, котрий, враховуючи особливості таких мереж, дозволить застосувувати потокову обробку замість пакетної.

Для досягнення мети повинні бути вирішені такі часткові завдання:

1) провести аналіз існуючих методів обробки інформаційних потоків Інтернету речей;

2) визначити характерні особливості туманного середовища;

3) обґрунтувати необхідність застосування концепції рефакторінга інформаційних потоків;

4) провести моделювання процедури виділення мікропотоків завдань із вхідного монолітного потоку;

5) удосконалити метод обробки інформаційних потоків у туманному середовищі;

6) провести порівняльне дослідження запропонованого методу обробки інформаційних потоків у туманному середовищі.

Наукова новизна дослідження полягає в тому, що було удосконалено метод обробки інформаційних потоків у туманному середовищі за рахунок проведення рефакторінгу монолітного потоку завдань на мікропотоки, що дозволило зменшити час реагування на зміни показань датчиків Інтернету речей в середньому до 20%.

Практична цінність дослідження полягає у зменшенні часу реагування на зміни показань датчиків Інтернету речей.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОБРОБКИ ІНФОРМАЦІЙНИХ ПОТОКІВ

1.1 Аналіз підходів до обробки інформації Інтернету речей

Завдяки широкому поширенню Інтернету та використанню вебтехнологій в комерційних цілях, у 2000-х роках фокус проектів управління розподіленими обчислювальними системами почав поступово переходити від надання обчислювальних ресурсів для спеціальних цілей, на більш споживчу концепцію хмарних обчислень (cloud computing), в рамках якої великій кількості незалежних кінцевих користувачів забезпечується надання набору абстрактних обчислювальних ресурсів за принципом оплати за використані послуги. Хмарне середовище є великим пулом віртуалізованих ресурсів (апаратне забезпечення, платформи розробки та/або сервіси), які можуть бути динамічно переконфігуровані для адаптації до змінного навантаження, дозволяючи також оптимально використовувати ресурси.

Експонентний розвиток хмарних обчислень забезпечив нові можливості обробки даних, що надходять від інтелектуальних датчиків. Це призвело до стрімкого зростання впровадження рішень у галузі *Інтернету речей (Internet of Things, IoT*). Очікується, що світовий ринок датчиків IoT досягне 32,5 мільярда доларів в 2023 році проти 27,4 мільярда доларів в 2022 році. Фахівці корпорації Ericsson підрахували, що станом на початок 2023 року налічується понад 32 мільярдів підключених до Інтернету пристроїв IoT, у той час, як за оцінками Бюро перепису населення США чисельність населення світу на цей час наближається до 7,8 млрд осіб. Це означає, що сьогодні на кожну людину припадає майже чотири підключених пристрої.

Концепція побудови Інтернету речей нерозривно пов’язана із хмарними технологіями (рисунок 1.1). IoT можна визначити як систему взаємозв'язку сенсорних пристроїв та актуаторів, яка забезпечує можливість обміну інформацією між платформами через єдину інфраструктуру, що дозволяє

сформувати єдину операційну модель для підтримки інноваційних застосунків. Цезабезпечується завдяки постійному збору інформації, аналізу даних і поданню інформації з використанням хмари як уніфікуючої інфраструктури.

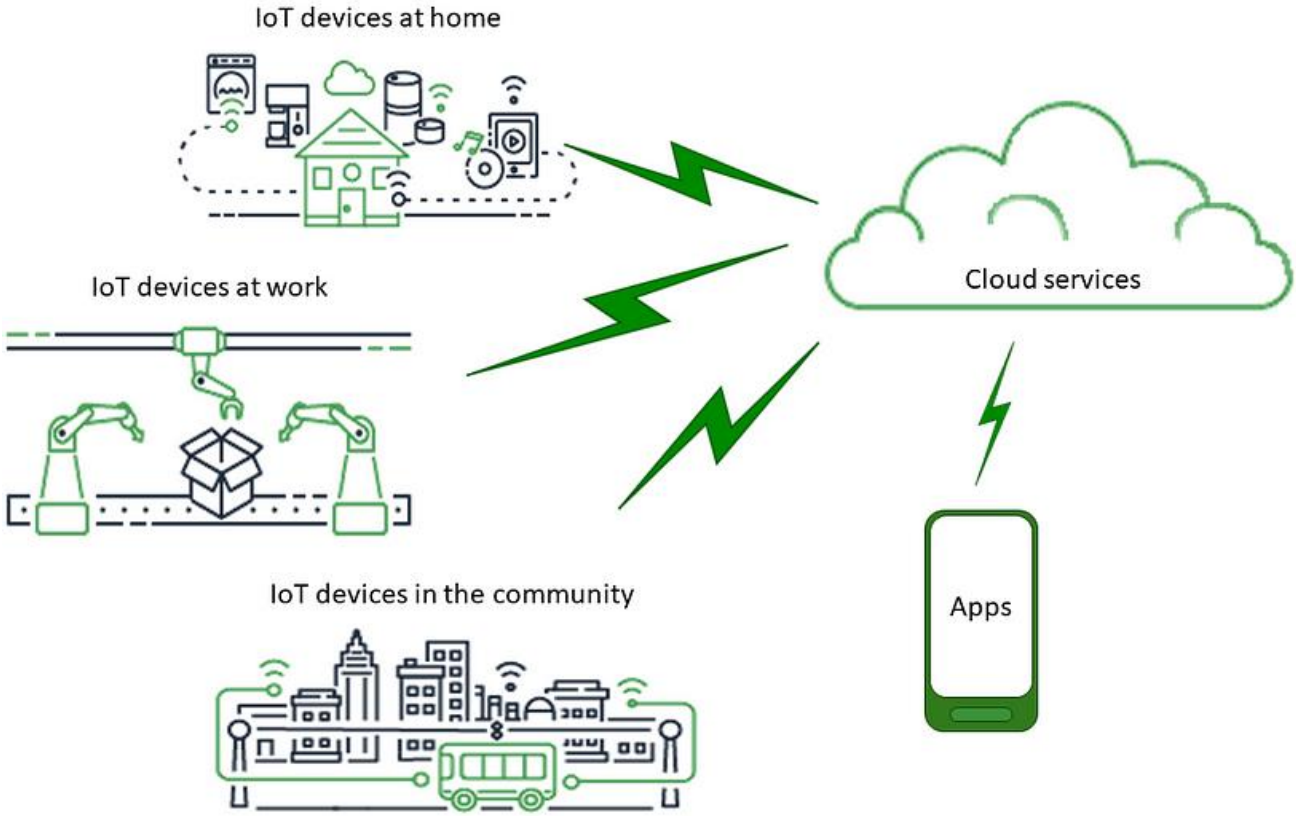


Рисунок 1.1 – Концепція Інтернету речей

Можна виділити такі ключові компоненти архітектури IoT:

– кінцеві IoT пристрої (датчики, актуатори), що забезпечують зв'язок з фізичним світом шляхом зчитування інформації простан середовища або генерацією будь-якого впливу;

– системи локальної комунікації, що включають в себе системи передачі даних і локальні обчислювальні мережі, які забезпечують передачу даних від кінцевих IoT пристроїв до найближчих пристроїв-агрегаторів;

– агрегатори, маршрутизатори і шлюзи IoT – апаратні компоненти, що забезпечують збір, агрегацію, кешування, передобробку і трансляцію

потоків даних від кінцевих IoT пристроїв в глобальну мережу, а також маршрутизацію зворотних потоків управління;

– глобальна обчислювальна мережа, що забезпечує можливість забезпечення доступу до даних і управління IoT системами з будь-якої географічної локації;

– хмарна обчислювальна інфраструктура, що надає потенційно безмежні обчислювальні ресурси для підтримки застосунків агрегації та аналізу даних IoT;

– системи обробки потоків даних, що забезпечують обробку інформації, яка надходить від пристроїв IoT в режимі реального часу;

– програми IoT, які надають кінцевим користувачам інтерфейс до можливостей та ресурсів систем IoT, зазвичай, це класичні застосунки, вебзастосунки або застосунки для мобільних платформ.

Отже можна побачити, що у даному напрямі обсяг інформаційних потоків до хмари постійно зростає, причому доволі швидко.

1.2 Специфічні особливості цифрових двійників

Через широке поширення використання технологій IoT в даний час фізичний і цифровий світи стали взаємопов'язаними, що послужило мотивом для встановлення взаємозв'язку між цими двома світами за допомогою телеметрії, підтримуваної моделюванням. Наприклад, в автоперегонах Формули-1 потік даних, зібраний з сотень датчиків, встановлених на автомобілі, і переданий на пульт технічного обслуговування, служить джерелом даних для моделювання роботи автомобіля у реальному часі. Використовуючи такі моделі, інженери можуть вносити коригування режиму роботи автомобіля віддалено, безпосередньо в режимі гонки. Використання таких підходів в індустріальній сфері називається індустріальним інтернетом речей (Industrial Internet of Things, IIoT) метою якого є створення розумної індустрії (Smart Industry) або Індустрії 4.0 (Industry 4.0) , яка інтегрує IoT з

виробничими технологіями для створення взаємопов'язаного виробничого підприємства, що аналізує інформацію для здійснення інтелектуальних дій в фізичному світі [7, 9]. Важливим додатком розумної індустрії є цифровий двійник (Digital Twin , DT, рисунок 1.2).

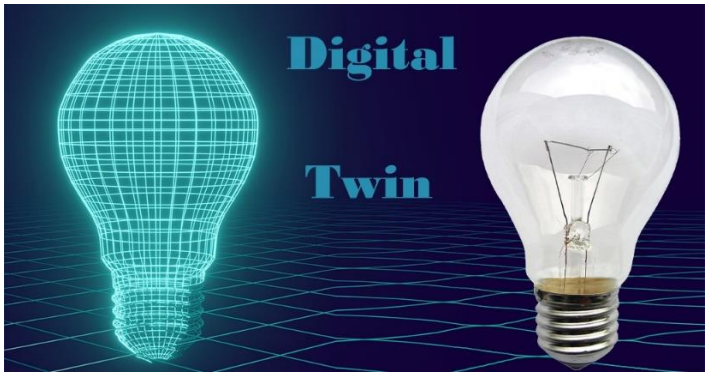


Рисунок 1.2 – Приклад цифрового двійника

Вважають, що концепція DT була вперше викладена на початку 2000-х років у контексті проектування систем управління життєвим циклом продуктів Product Lifecycle Management, PLM). DT є системою, що складається з трьох основних компонентів: фізичний об'єкт в реальному світі, його віртуальне подання в віртуальному світі, а також потоки даних та управління, які поєднують реальні та віртуальні компоненти (рисунок 1.3).

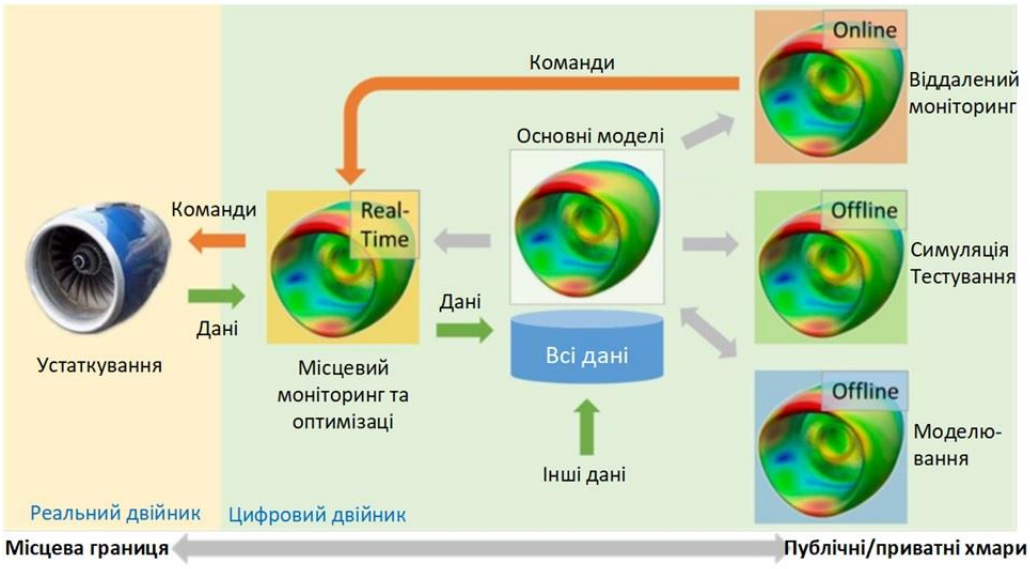


Рисунок 1.3 – Структура системи DT

DT – це інтегрована мультифізична, мультимасштабна імовірнісна симуляція складного виробу, яка використовує найбільш придатні фізичні моделі, актуальні дані сенсорів і ін. для того, щоб отримати якомога достовірніше уявлення відповідного реального об'єкта. На відміну від традиційного моделювання, віртуальне подання в DT постійно оновлюється з урахуванням стану обслуговування та продуктивності протягом усього життєвого циклу фізичного об'єкта (рисунок 1.4).

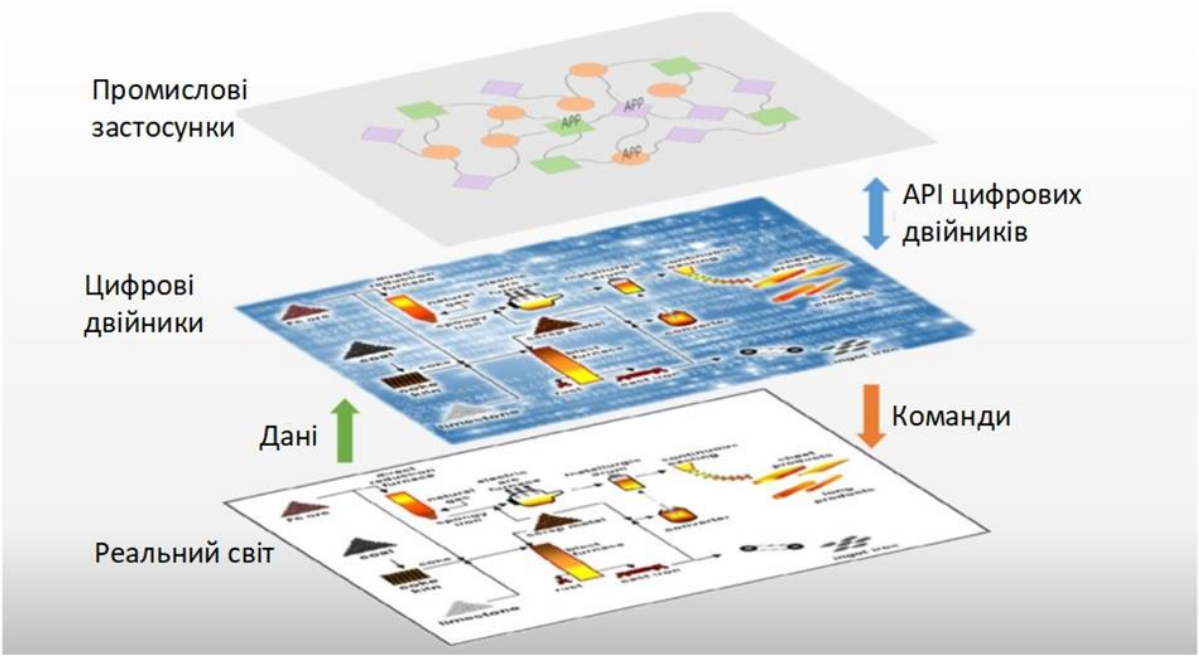


Рисунок 1.4 – Основні інформаційні потоки у системі DT

Для створення DT технологічних процесів та систем застосовуються системи математичних моделей та методів. Кожен з таких методів висуває особливі вимоги до необхідних обчислювальних ресурсів. Наприклад, методи інтелектуального аналізу даних вимагають обчислювальних ресурсів, що надають суттєві обсяги для зберігання даних, в той же час як моделі, що використовують метод кінцевих елементів, вимагають високопродуктивних обчислювальних систем (або суперкомп'ютерів).

З одного боку, надання обчислювальних ресурсів з необхідними характеристиками та можливість їх динамічного масштабування, що надається хмарними обчислювальними системами, дозволяють зробити

висновок, що концепція хмарних обчислень задовольняє таким вимогам до обчислювальної інфраструктури. З іншого боку, відмінною характеристикою DT є можливість збору, передачі і аналізу потоків даних від систем IIoT в режимах, близьких до реального часу, для налаштування і актуалізації їх віртуального стану. У цьому випадку застосування хмарних обчислень не дозволить задовольнити вимоги застосунків, чутливих до часу затримки та розташування сервісів обробки даних. Можливим вирішенням цієї проблеми може бути застосування моделі туманних обчислень, яка розширює концепцію хмарних обчислень, надаючи обчислювальні ресурси ближче до джерел даних.

1.3 Характеристика туманного середовища

Технологія хмарних обчислень забезпечує надання динамічно масштабованого, потенційно необмеженого обсягу обчислювальних ресурсів. Але висока величина затримки при передачі даних (латентність) і можливі перенавантаження мережних каналів не дозволяють хмарним рішенням забезпечити повне відповідність вимогам таких систем як, наприклад, DT, чутливим до часових затримок та розташування вузлів обробки даних. Віддалене географічне розташування хмарних центрів обробки даних призводить до неминучих проблем з латентністю при передачі та обробці даних, а також суттєвим обмеженням щодо пропускної здатності каналів зв'язку, розташованих на краю мережі (network edge) , тобто в безпосередній близькості до джерел даних та кінцевих користувачів. Розробники концепції туманних обчислень (fog computing, FC) намагаються вирішити цю проблему, переміщаючи частину завдань обробки і зберігання даних ыз хмари ближче до краю мережі, в туманні вузли (fog nodes).

Туманні обчислення - це багаторівнева модель, що забезпечує доступ до спільних масштабованих обчислювальних ресурсів та підтримує розгортання розподілених застосунків та сервісів з урахуванням латентності. Туманне

обчислювальне середовище складається з туманних вузлів (фізичних або віртуальних), розташованих між інтелектуальними кінцевими пристроями та централізованими (хмарними) сервісами. Туманні обчислення дозволяють вирішити задачу мінімізації часу від запиту до відповіді підтримуваних застосунків та забезпечують кінцевим пристроям локальні обчислювальні ресурси і, при необхідності, мережне підключення до хмарних сервісів (рисунок 1.5).

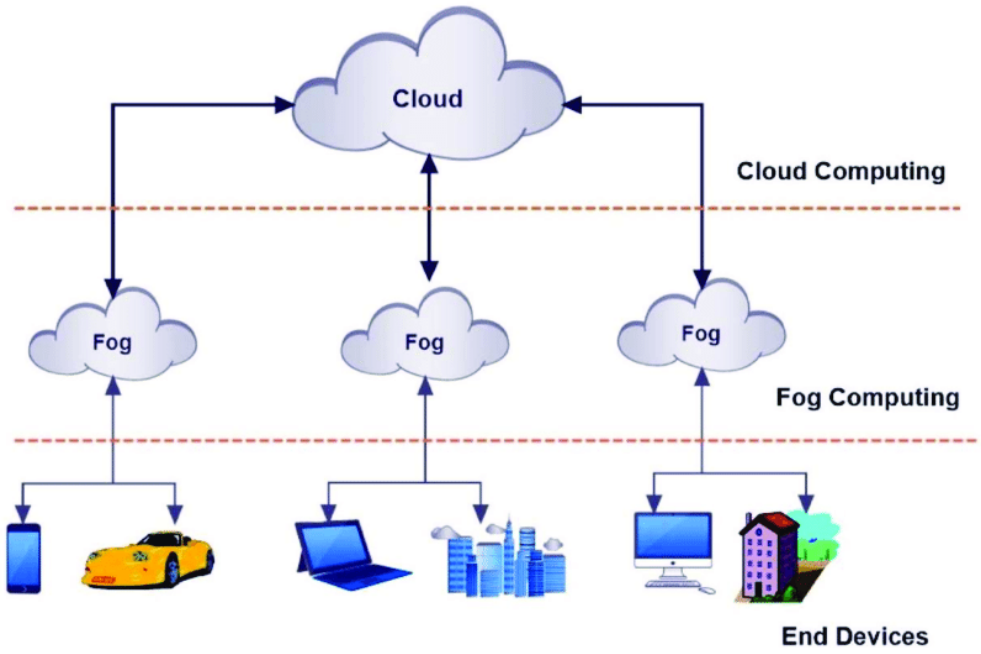


Рисунок 1.5 – Місце туманного середовища

Виділяють такі ключові характеристики туманних систем.

Контекстуальна обізнаність щодо місцезнаходження і мінімізація латентності. Туманні обчислення вирішують завдання мінімізації латентності при обробці даних завдякипоінформованості туманних вузлів щодо власного логічного місцезнаходження в контексті всієї системи і витрат на зв'язок з іншими вузлами. На відміну від централізованого підходу хмарних обчислень, послуги та застосунки, на які націлені туманні обчислення, приділяють суттєву увагу географічному і мережному розташуванню обчислювальних ресурсів при плануванні навантаження.

Гетерогенність. Туманні обчислення підтримують обробку даних різних форматів, збір і передача яких забезпечується за допомогою різних мережних комунікаційних рішень.

Функціональна сумісність (інтероперабельність) і федерація. Безперебійна підтримка таких сервісів як обробка потоків даних в реальному часу, вимагає спільної роботи кооперації великого числа компонентів. Компоненти туманного обчислювального середовища повинні бути здатні до взаємодії, а сервіси повинні бути розподілені між зонами доступності.

Масштабованість і гнучкість кластерів туманних вузлів. Туманні обчислення повинні забезпечувати адаптивність до необхідних ресурсів на рівні кластерів туманних вузлів, підтримуючи еластичне надання необхідних обчислювальних ресурсів, враховуючи зміни обчислювального навантаження і зміни стану мережі.

Взаємодія в реальному часу. Програми туманних обчислень припускають взаємодію в реальному часу, обробку даних в потоковому, а не в пакетному режимі.

Розглянуті характеристики туманних обчислювальних систем підкреслюють, що ключовий клас туманних застосунків - це географічно розподілені послуги, котрі забезпечують обробку потоків даних, підтримуючи можливість роботи в режимах, близьких до реального часу і розміщення в безпосередній близькості до джерел даних для мінімізації латентності. Проектування, розгортання і підтримка роботи таких застосунків вимагає особливого підходу від розробників програмних систем. Ключовим підходом до вирішення цього завдання може бути застосування подійно-орієнтованої архітектури, реалізованою з використанням мікросервісного підходу проектування застосунків.

1.4 Мікросервісний підхід

Туманна обчислювальна інфраструктура та програми IoT орієнтовані на обробку потоків даних з різних джерел. Подійно-керована архітектура (EventDriven Architecture – EDA) є найбільш адаптованою до цього типу програм. EDA – це системна архітектура, що складається з слабопов'язаних компонентів обробки подій, які приймають і обробляють події одночасно. У рамках даної архітектури під подією мається на увазі деяка значуща інформація про стан процесу або явища, що відбувається всередині або зовні системи, яка реалізується.

EDA за своєю природою є екстремально слабопов'язаною та високо розподіленою архітектурою програмних систем, тобто компоненти, складаючі цю систему, взаємодіють друг з одним виключно з урахуванням відкритих інтерфейсів, незалежно від окремих аспектів внутрішньої реалізації. Такий підхід забезпечує можливість внесення змін до реалізації компонента без впливу на компоненти, які його використовують, в випадку якщо поведінка і обмеження, описані у його інтерфейсі, залишаються незмінними. Протилежністю даного підходу є сильно пов'язана архітектура, в рамках якої взаємодія між компонентами реалізується з урахуванням особливості їхньої внутрішньої реалізації, та зміни в реалізації одного з них може суттєво вплинути на реалізацію множини інших компонентів, що складають систему.

В основі EDA лежать процеси генерації подій джерелами даних; комунікаційні канали і механізми поширення інформації про події між обробниками подій, а також реалізація безпосередньо обробників подій, які поєднують дані від множини джерел, що ідентифікують характерні особливості даних та формують події, містять результати обробки даних. Від програмних систем, побудованих на базі такої архітектури, очікується можливість динамічного масштабування обчислювальних ресурсів, що надаються, автоматизації розгортання та управління часом життя сервісів

обробки даних, інакше розгортання та підтримка подібних систем у масштабах, необхідних для вирішення реальних завдань, стають надзвичайно складними чи цілком неможливими.

Концепція мікросервісів вважається сьогодні найбільш перспективною для проектування слабопов'язаних систем, орієнтованих на швидку обробку подій. Мікросервіси – це підхід до проектування розподілених обчислювальних систем, де застосуток декомпозується на набір незалежних сервісів, які можуть бути розгорнуті, репліковані і зупинені незалежно один від одного. Кожен із таких сервісів орієнтований на реалізацію конкретного завдання в рамках відповідної програми. Особливості цього підходу можуть бути яскраво проілюстровані при його зіставленні з класичним підходом до проектування застосунків, який називають монолітним (рисунок 1.6, а) [9].

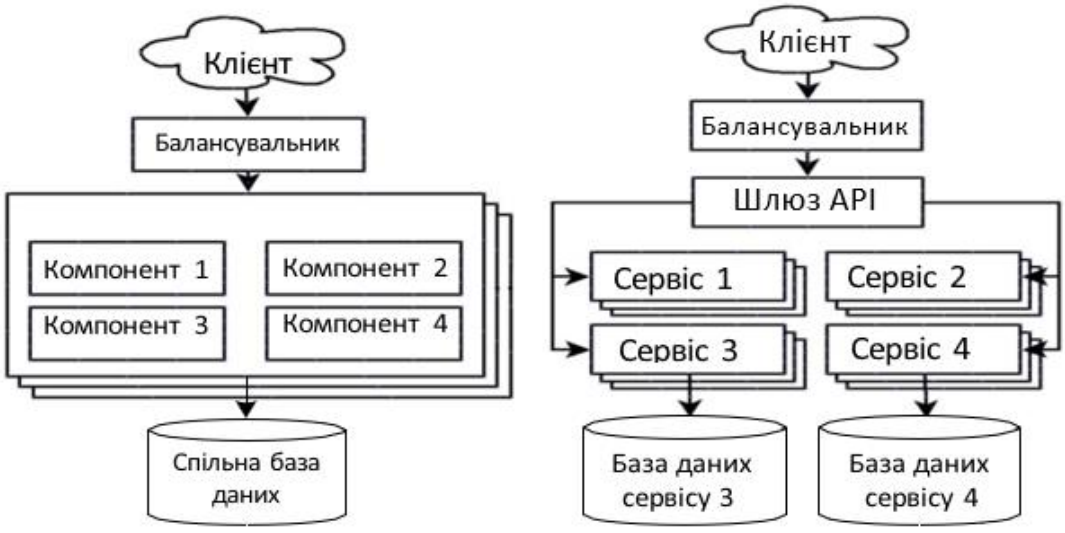


Рисунок 1.6 – Порівняння монолітного підходу і концепції мікросервісів

При проектуванні програми в рамках монолітного підходу компоненти, відповідальні за реалізацію різних галузей логіки системи, об'єднані в єдиний метакомпонент (сервер), і пов'язані між собою як безпосередніми викликами інших компонентів, так і побічно, за допомогою читання/запису даних в базу даних, загальну для всіх компонентів цього застосунку. На відміну від

монолітного підходу, виділяються такі особливості реалізації систем відповідно до концепції мікросервісів (рисунок 1.6, б):

– мікросервіси повинні використовувати відкриті, легковагі механізми для організації комунікації (частіше всього базуються на стеку протоколів TCP/IP і HTTP);

– поділ програми на мікросервіси має відбуватися на основі виділення та ізоляції частин функціонуючих процесів;

– мікросервісний підхід повинен забезпечити незалежне управління життєвим циклам екземплярів кожного компонента, включаючи можливості незалежного розгортання, масштабування і зупинення.

Застосування концепції мікросервісів дозволяє подолати суттєві недоліки монолітного підходу, включаючи такі проблеми, як розподіл обчислювального навантаження, забезпечення безперебійного функціонування системи під час обслуговування або оновлення її компонентів. При цьому мікросервісний підхід дозволяє здійснювати незалежну розробку, розгортання, масштабування та міграцію мікросервісів з одного обчислювального ресурсу на інший.

Незважаючи на переваги даного підходу, така гнучкість не буває безкоштовною. Наприклад, комунікаційні витрати збільшуються через необхідності організації обміну даними між мікросервісами , що, в свою чергу, підвищує складність управління потоками даних.

1.5 Аналіз існуючих систем обробки інформаційних потоків

Відмінною особливістю туманних застосунків, пов'язаних з платформами Інтернету речей, є збір і обробка інформації про об'єкті фізичного світу в режимі, близькому до реального часу. Для вирішення цього завдання обчислювальні послуги повинні вміти обробляти потоки даних, що формуються сенсорами, які збирають інформацію про стан фізичних об'єктів. Продуктивність таких систем критично залежить від здатності безпечно та

ефективно збирати, передавати та аналізувати потоки даних між об'єктами реального світу та системами обробки даних.

Під потоком даних від джерела Інтернету речей будемо розуміти злічену необмежену послідовність елементів даних, які стають доступними на протязі часу, наприклад, показання датчиків IoT або котирування акцій в фінансових застосунках. Під потоковою обробкою даних будемо розуміти виконання різних завдань обробки над потоком даних, які виконуються безпосередньо в процесі надходження нових даних (на відміну від методів пакетної обробки даних, які застосовуються у разі, коли всі необхідні дані вже зібрані в окремому сховищі) (рисунок 1.7). Ці завдання можуть бути побудовою моделей, пов'язаних із прогнозуванням значень наступних елементів, виявленні частих закономірностей або аномальних значень.

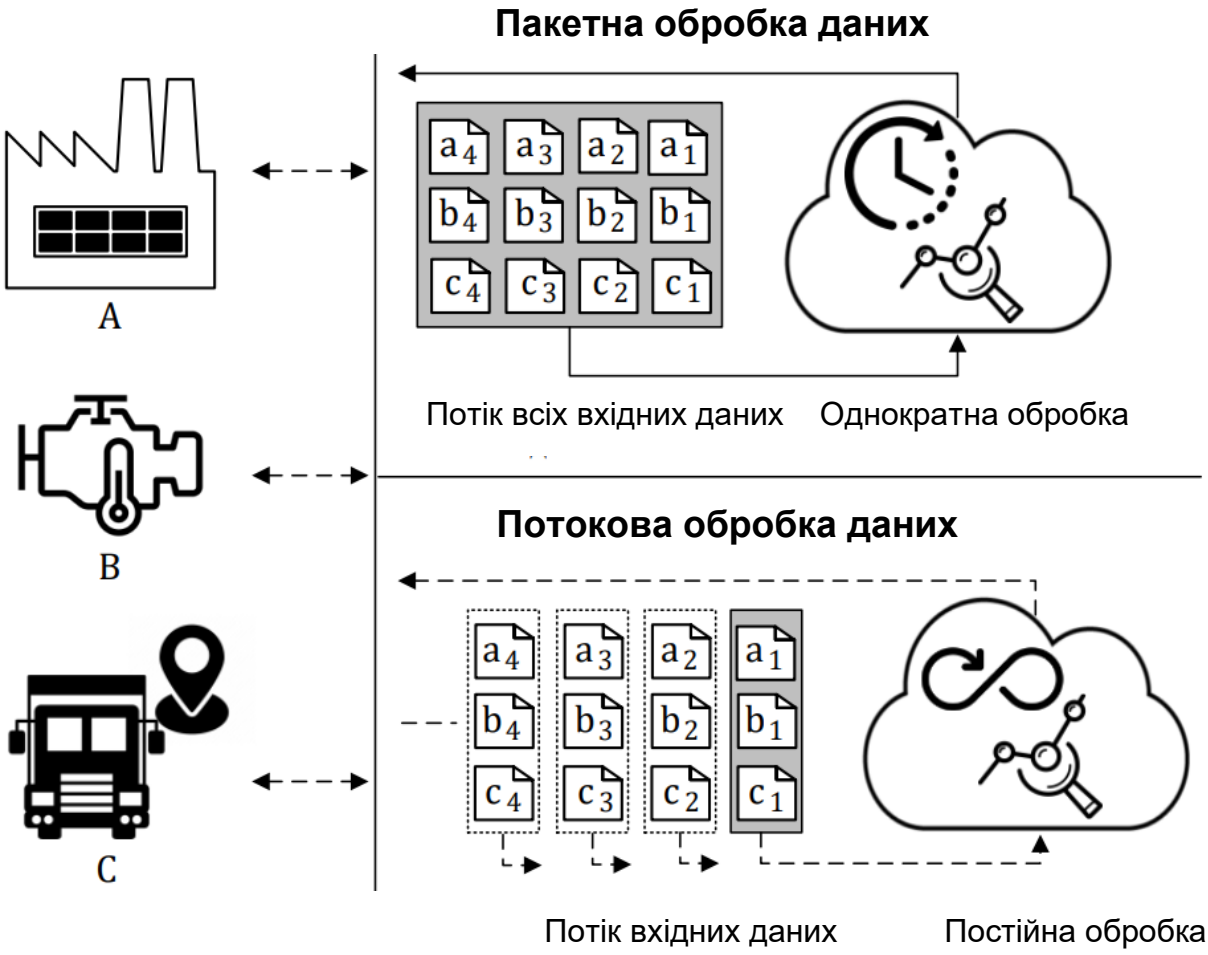


Рисунок 1.7 – Порівняння потокової та пакетної обробки даних

Алгоритми потокової обробки даних обробляють послідовно потенційно необмежені вхідні потоки, дозволяючи отримати потік результатів як на основі повного набору раніше оброблених даних, так і на основі обмеженого набору останніх даних, відібраних за допомогою ковзного методу вікна.

При реалізації пакетної обробки даних, наприклад, з використанням концепції MapReduce, всі вхідні дані повинні бути попередньо зібрані перед початком обчислювального процесу. Одним з найбільш популярних рішень для підтримки концепції MapReduce є Hadoop. Hadoop – це програмна платформа для застосунків, що забезпечують надійну і відмовостійку обробку великих даних з урахуванням кластерних обчислювальних систем (а до тисяч вузлів).

У процесі реалізації алгоритму MapReduce вхідний набір даних розбивається на велике число незалежних фрагментів, які можуть бути паралельно оброблені на фазі *map*. Вихідні результати такої обробки надалі надходять на вхід завдань *reduce*. Платформа Hadoop піклується про планування завдань, їх моніторинг і повторне виконання невдалих завдань. При цьому накладні витрати, пов'язані з підготовкою обчислювальної задачі (такі як планування, копіювання або перенесення даних та коду), не дозволяють забезпечити оперативну обробку даних, що надходять, у режимі реального часу. Варіанти використання цих концепцій також відрізняються. Наведемо приклади завдань, для рішення яких використовується пакетна обробка даних:

– система розрахунку заробітної плати співробітників, яка збирає всі відповідні дані та обробляє їх у пакетному режимі наприкінці певного періоду, наприклад, кожного місяця;

– система віртуальних краш-тестів, яка проводить моделювання на основі попередньо визначених і параметризованих моделях автомобіля та перешкоди.

Прикладами завдань, для вирішення яких застосовується потокове

оброблення даних, є такі:

– система виявлення шахрайських транзакцій в онлайн-режимі для ідентифікації і запобігання аномальних транзакцій в режимі реального часу;

– моніторинг транзакцій фондового ринку;

– моніторинг потоків даних від систем Інтернету речей для ідентифікації стану і прогнозування на протязі технологічних процесів.

Традиційно системи пакетної обробки даних застосовуються для рішення завдань високопродуктивної обробки великих даних. Рішення таких завдань може займати кілька годин і навіть днів.

Розглянемо основні компоненти систем обробки потоків даних. Виділимо такі елементи систем, які забезпечують обробку потоків даних.

Повідомлення – це основний елемент даних, оброблюваний застосунком при отриманні з вихідного потоку даних. Повідомлення складається з набору атрибутів, кожному з яких надано певне значення.

Генератор (продюсер) – це компонент системи, що забезпечує постійну генерацію повідомлень на основі оригінального джерела даних.

Потік даних – це послідовність повідомлень, які можуть бути потрібні та оброблені споживачем.

Споживач – це система, що відповідає за отримання та обробку повідомлень з потоку даних.

Оператор – базовий функціональний елемент потокового програми. Оператор отримує дані з вхідних потоків, застосовує заздалегідь певну функцію над вхідними повідомленнями і генерує вихідний потік з повідомленнями – результатами обробки.

Схема повідомлень – це специфікація, визначальна загальну структуру повідомлень у системі та їх атрибути. Кожен окремий атрибут повідомлення може бути рекурсивно визначений у вигляді схеми.

Наприклад, Apache Avro є платформою для управління схемами повідомлень, визначеними на мові JSON. Схеми Avro формуються із набору базових типів (таких як Boolean, long, int, double, float, byte, string) і структур

даних (record, array, enum, map_fixed, union).

Крім того, можна виділити ключові класи операторів над потоками даних, що функціонують у туманних середовищах.

Оператори агрегації забезпечують агрегацію підмножин вхідних повідомлень з одного або декількох потоків даних. Підмножина повідомлень, над якими виконується операція агрегації, називається вікном. У залежності від застосовуваних правил виділення підмножин, вікна агрегації можуть бути різних розмірів, такими, що не перетинаються, як, нприклад, на рисунку 1.8, а, або перекриваються, як, нприклад, на рисунку 1.8, б. Вікна, що перекриваються, інформують про наявність повідомлень, які потрапляють одночасно в кілька послідовних вікон.

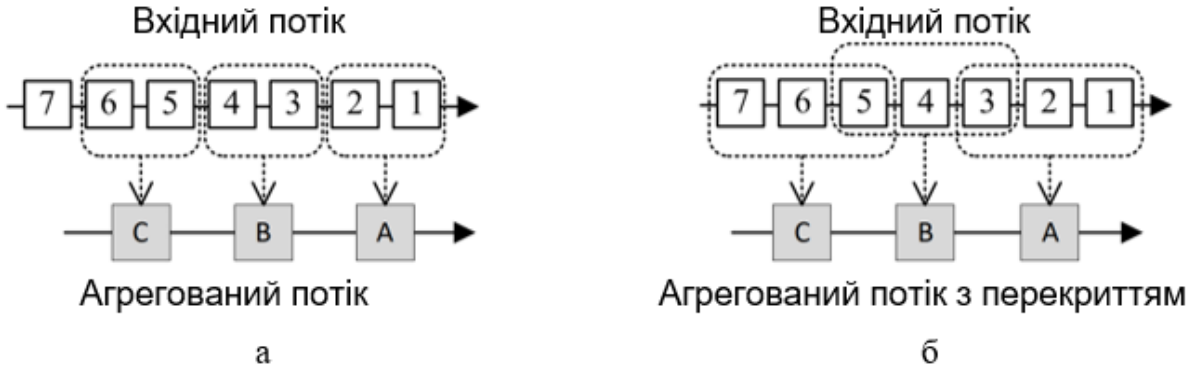


Рисунок 1.8 – Оператори агрегації

Оператори злиття забезпечують об'єднання кількох вхідних потоків даних в один загальний потік на підставі певних вимог до вирівнювання повідомлень або інших певних умов (рисунок 1.9, а). Оператори розщеплення забезпечують розподілення єдиного потоку даних на декілька потоків відповідно до заздалегідь визначеного алгоритму (рисунок 1.9, б).

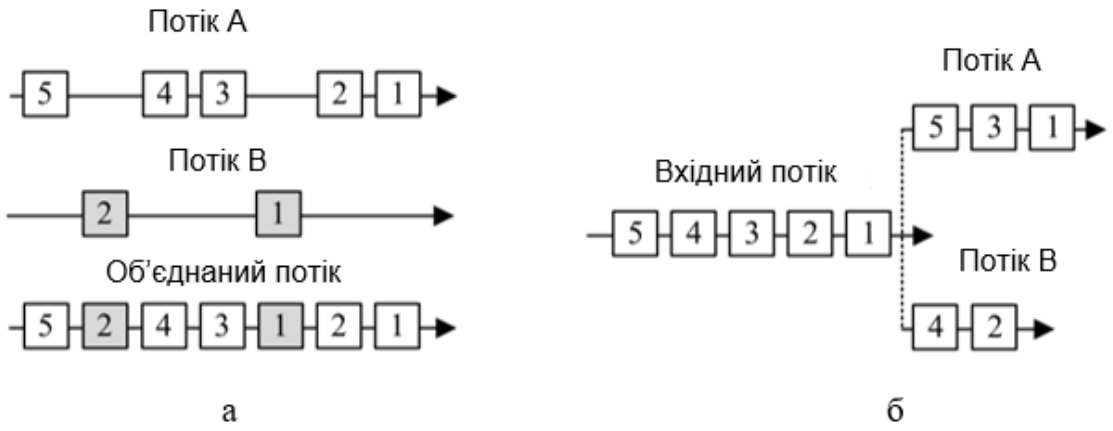


Рисунок 1.9 – Оператори злиття та розщеплення

Логічні, математичні та спеціалізовані оператори обробки даних забезпечують перетворення вхідних повідомлень відповідно із заздалегідь визначеними логічними або математичними правилами (рисунок 1.10, а). Частковим випадком операторів такого типу можуть бути оператори, які забезпечують виконання методів інтелектуального аналізу даних або машинного навчання над вхідними потоками даних (рисунок 1.10, б).

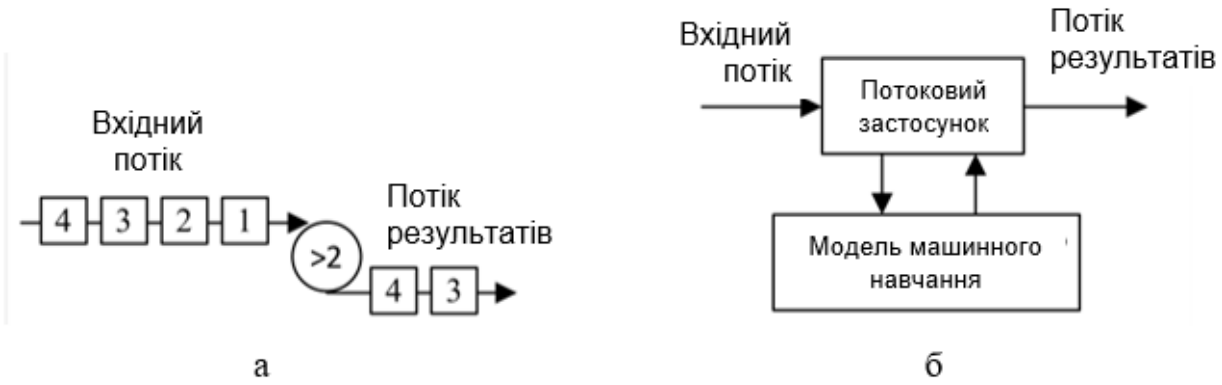


Рисунок 1.10 – Спеціалізовані оператори обробки даних

Тепер розглянемо, яку архітектуру мають системи обробки потоків даних. Можна виділити два ключові шари, необхідні організації системи обробки потоків даних: шар зберігання (storage layer, SL) та шар обробки (processing layer, PL). SL забезпечує можливість організації високої пропускної здатності длявеликих потоків даних. PL відповідає за споживання даних із SL, обробку цих даних і оповіщення SL про оновлення даних. Для

організації розподіленої обробки потоків даних на багатьох незалежних обчислювальних пристроях використовується архітектура, наведена на рисунку 1.11.

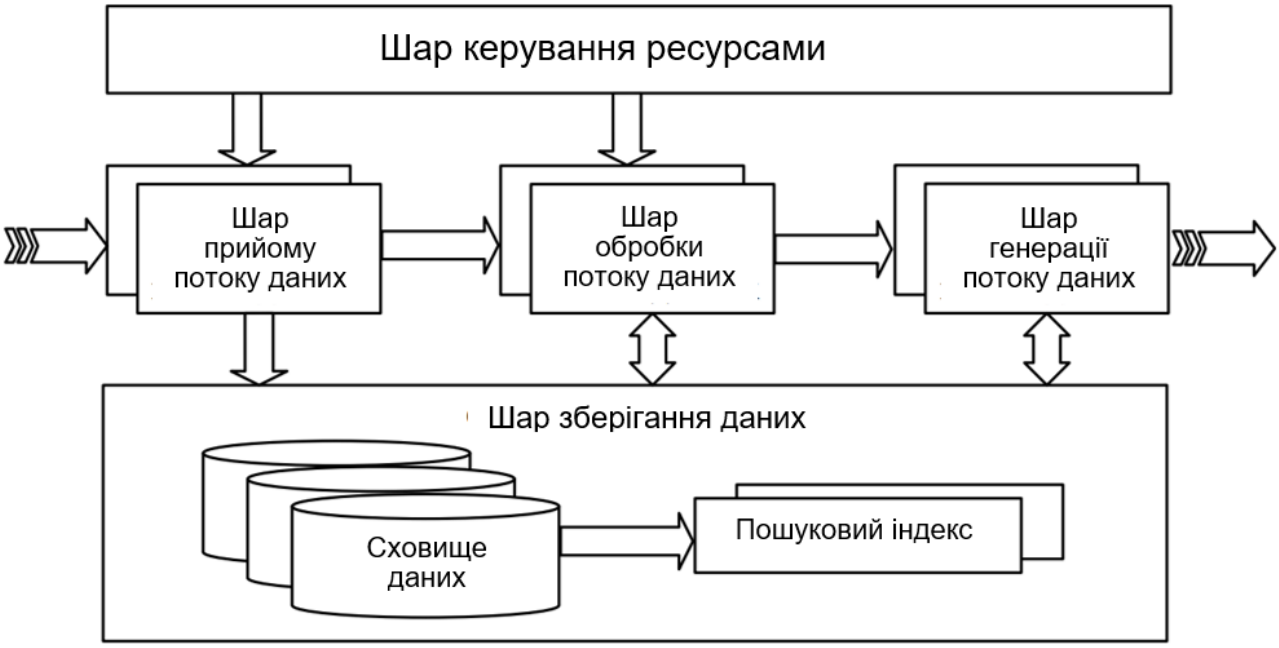


Рисунок 1.11 – Архітектура системи обробки потоків даних

В наведеній архітектурі:

1) шар прийому потоку даних включає операції по отриманню потоків даних із вхідних джерел і передачі даних в систему обробки або зберігання даних;

2) шар обробки потоку даних керує програмами для обробки потокових даних, на ньому можуть розміщуватися як окремі програми обробки даних, так і платформи обробки потоків даних (Data Stream Processing Engines, DSPE); DSPE є інструментом, що включає оператори обробки потоків даних, які можуть бути налаштовані і організовані у вигляді орієнтованого ациклічного графа для побудови конвеєра обробки потоку даних;

3) шар зберігання даних забезпечує зберігання повідомлень, проміжних та фінальних результатів обробки, виявлених патернів і

інформації з потоку даних; на цьому шарі можуть розміщуватися рішення індексації і повнотекстового пошуку в текстових даних, такі як ElasticSearch або Apache Solr;

4) шар управління ресурсами відповідає за організацію роботи і взаємодію обчислювальних вузлів, вузлів зберігання, а також управлінням часом життя даних ресурсів (включаючи виділення нових та звільнення зайнятих ресурсів) для підтримки обробки великих обсягів потоків даних;

5) шар генерації потоку даних відповідає за формування результуючого потоку даних і його відправку наступним споживачам; також, витягнута інформація і знання можуть бути переспрямовані для тимчасового або постійного зберігання в сховищі даних для подальшого аналізу.

1.6 Умови збереження стану потоків та контейнеризація

Процеси, залежні лише від поточного локального стану, тобто не від недавнього минулого чи розширеної історії всіх таких станів, називаються процесами без збереження стану (stateless). Однак розробка сервісів, орієнтованих на обробку даних IoT у туманних обчислювальних середовищах, потребує збереження стану інформаційних потоків. Такі системи прийнято називати із збереженням стану (stateful). Для виконання операцій із збереженням стану (Stateful-операцій) необхідно мати можливість ідентифікації джерела вхідних даних і визначення того, які вхідні дані були отримані з того ж джерела. Розробники Stateful-систем стикаються з низкою проблем, включаючи обмеження масштабованості і необхідність додаткових обчислювальних витрат на підтримку управління станом.

Зберігання стану всередині обчислювального сервісу обмежує можливості управління його життєвим циклом, тому стан повинен зберігатися в окремому ресурсі, наприклад, зовнішній базі даних, зовнішній файловій системі, системі кешування або проміжному програмному забезпеченні для обміну повідомленнями. Тим не менш, окремий ресурс для

роботи зі станом призводить до необхідності виділення додаткових серверних ресурсів, які можуть включати додаткові обчислювальні потужності, пам'ять та сховища даних. Крім того, такий підхід може призводити до проблем масштабування. Наприклад, це може трапитися, коли екземпляри одного і того ж сервісу оновлюють стан, а інший сервіс зчитує той же стан під час оновлення. Таке рішення не забезпечить правильність обробки стану під час операцій. Тому найчастіше стан має оброблятися послідовно. У зв'язку з цим, будь-яка операція стикається зі складністю управління станом. Складність зростає в умовах туманного обчислювального середовища, де важливе значення має можливість живої міграції завдань обробки та зберігання даних між туманними вузлами. Живою міграцією називають можливість безперебійної міграції сервісу між фізичними машинами без впливу на клієнтські процеси або програми. У цьомуу випадку обчислювальний процес припиняється тільки на час передачі загального стану, після чого обчислення поновлюється на цільовому вузлі. Реалізація такого механізму забезпечує правильність обробки, зберігання та відновлення стану та є суттєвим процесом уStateful-операціях.

Технології віртуалізації дозволяють розділити фізичний обчислювальний вузол на кілька віртуальних машин (VM), кожна з яких має власну ізольовану операційну систему та програми. Вони координуються шаром програмного забезпечення під назвою гіпервізор, який забезпечує підтримку паралельного виконання кількох VM на одній фізичній машині.

Тим не менш, накладні витрати, пов'язані з використанням VM в хмарних та туманних обчисленнях, можуть обмежити ефективність обчислювальних ресурсів. Ця проблема може бути вирішена при допомозі технології контейнеризації. Контейнеризація дозволяє запускати незалежні контейнери у вигляді окремих процесів безпосередньо на базовому ядрі операційної системи, забезпечуючи легку ізоляцію процесів усередині контейнерів. Однією з найпопулярніших платформ, що забезпечують контейнеризацію, є Docker . При створенні нового контейнера в Docker ,

новий шар (шар контейнера), доступний для запису, створюється поверх шарів, доступних тільки для читання (шар образу), а всі зміни, внесені в контейнер, записуються тільки на шар контейнера.

Всі базові образи контейнерів доступні в репозиторії образів, наприклад, Docker Hub . Щоб запустити будь-який контейнер на новому обчислювальному вузлі, необхідно завантажити тільки базовий образ з репозиторію Docker , після чого з цього образу створюється новий рівень контейнера. Складність проектування stateful-систем залежить від можливостей базової обчислювальної інфраструктури. Stateful-обчислювальною інфраструктурою буде називатися така інфраструктура віртуалізації, яка дозволяє зберігати та керувати станом усередині контейнера протягом тривалого проміжку часу. Технологія VM забезпечує можливість створення моментальних знімків та відновлення стану програм. Це дозволяє здійснювати міграцію VM між фізичними хостами з мінімальним впливом на запущені сервіси. Такі технології застосовуються в багатьох платформах керування VM, таких як VMware та vCenter. Знімки стану VM дозволяють забезпечити міграцію VM між обчислювальними вузлами без суттєвого переривання обчислювального процесу. На жаль, такий підхід не поширений у разі застосування технології контейнеризації. Реалізація такої міграції не є такою простою при використанні платформи Docker.

Існує кілька методів вирішення проблеми міграції контейнерів. Наприклад, проект CRIU та його розширення, як і раніше засновані на експериментальному режимі Docker. Також, CRIU виконує операцію "PID dance", щоб відновити процес з таким самим PID. Ця операція вимагає привілейованого доступу і має низьку продуктивність, оскільки потребує багато системних викликів і може призвести до виникнення стану гонки. З іншого боку, контейнери Linux LXD підтримують можливість створення знімків і відновлення контейнерів для резервного копіювання або міграції. Але, для цілей живий міграції, LXD все ще базується на CRIU.

Тим часом, деякі платформи оркестрації багатоконтейнерних систем, такі як Kubernetes, надають контролер StatefulSet для управління stateful-застосунками. Тим не менш, його робота, як і раніше, підлягає деяким обмеженням, таким як втрата стану при реалізації « rolling оновлень», складність чи неможливість застосування балансувальників навантаження тощо. Ключовою ж проблемою в управлінні Stateful Set є відсутність механізмів автоматичного створення Stateful, що вийшов з ладу. Тому вирішення проблеми збереження стану під час використання технології контейнеризації є активною областю досліджень. Це особливо важливо при обробці Stateful-потоків даних, де кожна точка даних може грати вирішальну роль в системі.

Щоб організувати управління станом, бажано забезпечити постійне зберігання даних про стан не тільки у обчислювальних сервісах, але і в інших частинах системи. Прикладами таких систем можуть служити зовнішня база даних, зовнішня файлова система, система кешування, проміжне програмне забезпечення обміну повідомленнями. Однак ідеальних рішень немає, тому що будь-який окремий ресурс для обробки стану вимагає додаткових серверних ресурсів: додаткової обчислювальної потужності, пам'яті і місця для зберігання даних, отже може призвести до великих затримок та погіршення якості обробки даних на крайових вузлах мережі. Проблема ускладнюється, коли обробка даних ведеться не пакетному, а в потоковому режимі.

1.7 Платформи систем обробки інформаційних потоків

Розглянемо найбільш поширені на сьогодні платформи, які забезпечують підтримку обробки потоків даних. Розподілена обчислювальна платформа Apache Flink керує станом, створюючи локальні постійні стани всередині програми. Для забезпечення відмовостійкості Flink надає механізм синхронізації миттєвих знімків стану із зовнішніми ресурсами, такими як

HDFS та/або Amazon S3. Аналітична платформа для обробки великих обсягів даних Apache Spark також забезпечує функцію stateful-обробки даних, де стан обробки потоку даних зберігається в локальній пам'яті, але для відмовостійкості Spark зберігає дані в відмовостійкій системі зберігання HDFS. Apache Kafka – це відмовостійка черга повідомлень з програмним інтерфейсом застосунків (Application Programming Interface - API). Kafka зберігає дані у «темах» (Kafka topics), які представляють собою назви категорій, в рамках яких публікуються і споживаються повідомлення. Kafka надає наступні API:

– Producer API для генерації потоків даних в теми Kafka;

– Consumer API для споживання потоків даних з тем Kafka;

– Streams API для трансформації потоку даних із вихідної теми Kafka в кінцеву тему Kafka;

– Connect API для постійного потоку даних з не-Kafka програмами (база даних, файл і т.д.) в тему Kafka або навпаки;

– Admin API для керування компонентами кластера Kafka.

Обмін даними між сервером Kafka і клієнтами здійснюється за протоколом TCP. Клієнт ініціює з'єднання, відправляє послідовність повідомлень запиту на сервер, після чого отримує повідомлення відповіді. Кожен споживач потоку даних маркується ідентифікатором групи споживачів (Group ID), і кожне повідомлення, опубліковане у темі, доставляється одному екземпляру споживача в рамках кожної Group ID. Kafka також підтримує семантику обробки повідомлень «рівно один раз» (exactlyonce processing semantics), гарантуючи, що кожен запис буде оброблено один і тільки один раз, навіть якщо в процесі обробки стався якийсь збій на клієнтах чи брокерах Kafka.

Можна виділити три основних сценарії застосунків,що взаємодіють з Kafka Streams API. По-перше, програма може служити адаптером для-не Kafka застосунків (Такі як база даних, файл і т.д.) та для забезпечення зв'язку з кластером Kafka. По-друге, застосунок може забезпечити збір даних з

датчиків. По-третє, застосунок може виступати в якості автономної одиниці обробки даних на основі Kafka Streams DSL API. Для реплікації даних Kafka у кількох географічних центрах обробки даних, на даний момент існує два офіційні інструменти: MirrorMaker та Confluent Replicator.

Кожна з платформ Kafka, Flink і Spark має свої особливості, які слід враховувати при реалізації операцій зі збереженням стану. У таблиці 1.1 наведено ключові архітектурні різниці між Kafka Stream API, Apache Flink та Apache Spark.

Таблиця 1.1 – Порівняння платформ обробки потоків даних

Критерій	Kafka Streams API	Flink та Spark
Розгортання	API, котрий може бути вбудований в застосунок і не нав'язує конкретний метод розгортання	Фреймворк, розгортаючий застосунок, або в автономних кластерах, або з використанням YARN, Mesos або контейнерів
Робота з станом, відмовостійкість	Стан зберігається локально та синхронізується з власною чергою Kafka	Стан зберігається локально, але для забезпечення відмовостій-кості може бути налаштовано зовнішнє сховище
Джерело потокових даних	Тільки з черги повідомлень Kafka, яка підтримує Connect API.	Kafka , інші черги повідомлень, файлова система і інші зовнішні системи
Результат обробки даних	Kafka, стан програми, база даних або будь-яка зовнішня система	Kafka , інші черги повідомлень, файлова система і інші зовнішні системи
Обмеженість та необмеженість	Необмежені потоки даних	Обмежені і необмежені потоки даних

Проте системи управління потоками даних зазвичай не накладають обмежень на організацію процесу обробки та перетворення, що реалізуються компонентами обробки даних. Це може привести до складнощів при проектуванні і модернізації складних конфігурацій систем обробки даних.

2 ОБРОБКА ІНФОРМАЦІЙНИХ ПОТОКІВ У ТУМАННОМУ СЕРЕДОВИЩІ

2.1 Концепція рефакторінгу інформаційних потоків

На сьогодні відсутні універсальні рішення, які дозволили б прозоро інтегрувати інформаційні потоки у туманні обчислювальні системи, що підтримують обробку потоків даних, які надходять від пристроїв IoT. Для вирішення цього завдання можна використовувати концепцію, засновану на декомпозиції IF на набори менших потоків, що носять назву мікропотоків завдань (MicroInfFlow, MIF) . Ця назва утворена з об'єднання назв двох концепцій: мікросервісів та інформаційних потоків.

Назвемо монолітним інформаційний потік, спроектований для обробки даних в пакетному режимі, такий, що складається з множини сильно пов'язаних між собою обчислювальних завдань.

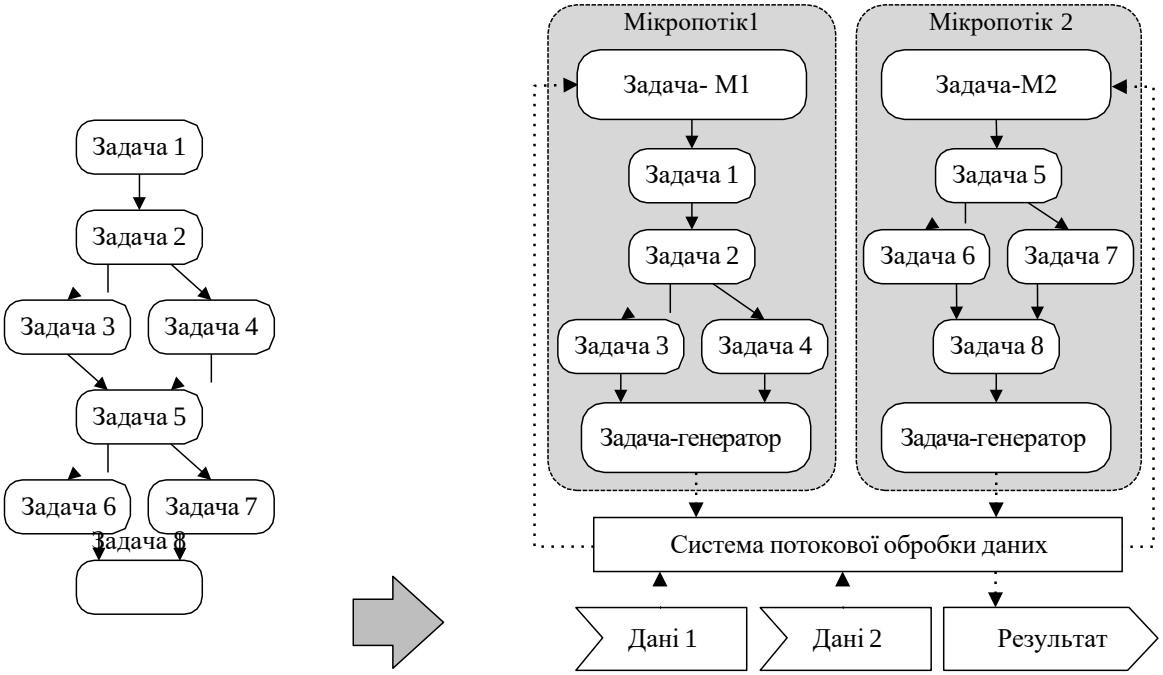
На відмінність від монолітного IF, кожен MIF представляє собою окремий обчислювальний сервіс (мікросервіс), який може бути розроблений, розгорнутий, перенесений на інший обчислювальний вузол та/або зупиненим незалежно від інших MIF. Зв'язок між MIF забезпечується за допомогою подійно-орієнтованого підходу, що підтримує перетворення виконання IF з режиму пакетної обробки даних у режим потокової обробки даних (рисунок 2.1).

Використання цією концепції для поділу монолітного потоку робіт на набір незалежних обчислювальних сервісів мікропотоків завдань дає такі переваги:

- забезпечується можливість організації роботи інформаційного потоку в режимі потокової обробки даних;

- розв'язка сильно пов'язаного обчислювального процесу в часі та просторі, перехід до моделі асинхронної комунікації;

- забезпечується можливість перенесення обчислювального процесу IF з одного обчислювального вузла на інший за необхідності перерозподілу обчислювальної навантаження без втрати проміжних даних та без необхідності повторної обробки попередніх даних;



а б

Рисунок 2.1 – Концепція мікропотоків завдань:

a – монолітний інформаційний потік;

b – мікропотоки завдань у результаті рефакторингу монолітного потоку

- забезпечується можливість розгортання I’Fs загального потоку робіт на вузлах, що знаходяться на різних рівнях ієрархії туманної обчислювальної системи: набір IF’s забезпечувати низьку латентність, можуть бути розгорнуті в туманних вузлах поблизу пристроїв промислового IoT , в той час як інші IF’s , вимагають тривалою обробки, можуть бути перенесено в хмара;

- забезпечується можливість незалежної розробки, оновлення, розгортання та запуску IF’s , що становлять загальний потік, при умові збереження єдиного формату обміну повідомленнями;

- забезпечується можливість прозорої інтеграції потоків даних, повідомлень від пристроїв IoT в якості джерел даних для інформаційного потоку на будь-кому етапі обчислювального процесу.

Можна виділити такі ключові стадії в процесі рефакторингу монолітного інформаційного потоку завдань у множину MIF:

- поділ монолітного інформаційного потоку завдань на підпотоки завдань, кожен з яких містить частину обчислювальних завдань базового потоку завдань;

- формування спеціальних вершин споживачів і генераторів повідомлень, що забезпечують зв'язок мікропотоків завдань з платформою обробки потоків даних;

- формування підрозділів (сховищ) у платформі обробки потоків даних, відповідальних за організацію отримання та передачі повідомлень, генерованих мікропотоками завдань;

- генерація контейнерів, що забезпечують інкапсуляцію і можливість незалежного розгортання мікропотоків завдань у вигляді мікросервісів.

Надалі концепцію мікропотоків завдань та процес рефакторингу монолітного інформаційного потоку завдань у набір мікропотоків завдань розглянемо більш детально.

2.2 Виділення підпотоків завдань

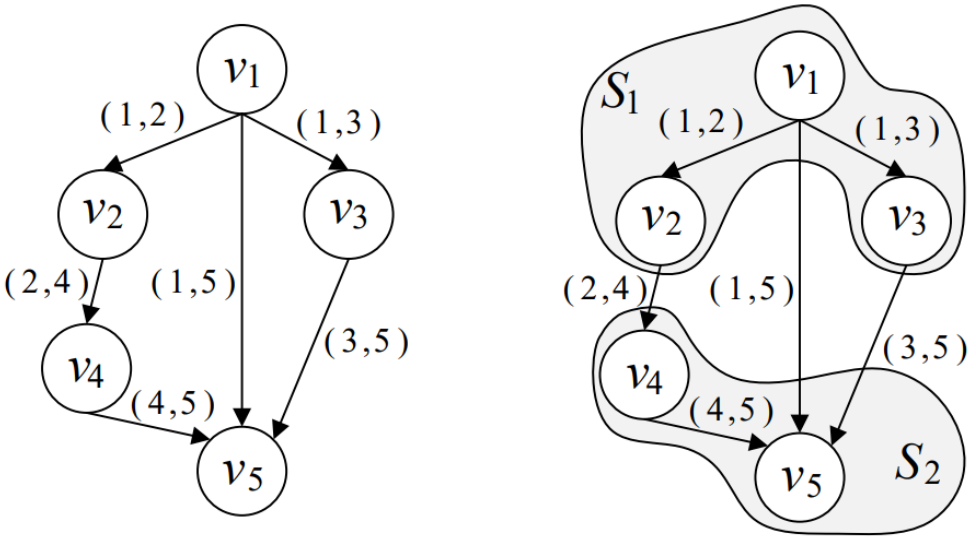
По-перше розглянемо формальний опис інформаційного монолітного потоку завдань. Такий потік завдань може бути представлений у вигляді орієнтованого ациклічного графа

$$W = (V, E), \tag{2.1}$$

де W - монолітний потік завдань; V – множина вершин, котрі репрезентують обчислювальні завдання; E – множина спрямованих ребер, що з'єднують вершини, котрі репрезентують собою залежності по даних між завданнями.

Кожна вершина v_i в V може мати вхідні та/або вихідні ребра. Вихідний ступінь $deg^+(v_i)$ вершини v_i в W визначається як кількість ребер, вихідних з v_i і спрямованих до іншим вершин. Вхідна ступінь $deg^-(v_i)$ вершини v_i в W визначається як кількість ребер, спрямованих до v_i з інших вершин. Ребро (v_i, v_j) $\in E$ є залежністю по даними від v_i до v_j.

Нехай n - це загальна кількість вершин в V. На рисунку 2.2, а наведено приклад потоку завдань W, де n приймається рівним 5.



а б

Рисунок 2.2 – Монолітний потік завдань:

а – приклад потоку завдань W, де n приймається рівним 5 ,

б – приклад потоку завдань W, розділеного на два підпотоки завдань S_1 і S_2

Отже, можна потік робіт W поділити на підпотоки, що складають множину з k підпотоків робіт (Subworkflows):

$$S = (S_1, \dots S_k),\ S = (V_i, E_i), \qquad (2.2)$$

де V_i – це множина вершин, що входять до підпотоку робіт S_i , E_i – це множина ребер між вершинами, що входять до V_i , при цьому множина вершин W розділяється підпотоками робіт на набір непересічних підмножин:

1) $\forall i \in 1..k: V_i \subset V, E_i \subset E$;

2) $\forall v \in V, \exists\, i \in 1..k : v \in V_i$;

3) $E_i = \{(v_k, v_l) \in E: v_k, v_l \in V_i\}$;

4) $\forall i, j : i \neq j \implies V_i \cap V_j = \emptyset$.

Рисунку 2.2, б показує приклад потоку завдань W, розділеного на два підпотоки завдань S_1 та S_2.

Визначимо класи ребер та вершин, пов'язаних з підпотоком завдань S_i :

1) EI_i: набір вхідних ребер з початкової вершини поза підпотоком завдань S_i і кінцевою вершиною всередині підпотоку завдань S_i:

$$EI_i = \{(v_k , v_l) \in E: v_k \notin S_i , v_l \in S_i\}; \quad (2.3)$$

2) EOi: набір вихідних ребер з початковою вершиною всередині підпотоку завдань Si і кінцевою вершиною поза підпотоком завдань Si:

$$EO_i = \{(v_k, v_l) \in E: v_k \in S_i , v_l \notin S_i; \quad (2.4)$$

3) VIi: набір вершин в Si , розташованих на головній частині ребер EIi, а також вершин, не маючих вхідних ребер:

$$VI_i = \{v_l \in S_i : (v_k , v_l) \in EI_i\} \cup \{v \in S_i : deg^-(v) = 0\}; \quad (2.5)$$

4) VOi: набір вершин в Si , розташованих на кінцях ребер EOi , а також вершин, не маючих вихідних ребер:

$$VO_i = \{ v_k \in S_i : (v_k , v_l) \in EO_i \} \cup \{ v \in S_i : deg^+(v) = 0 \}. \quad (2.6)$$

Тепер можна перейти до мікропотоків завдань.

2.3 Моделювання мікропотоків завдань

Щоб перетворити підпотік завдань S_i в мікропотік завдань MIF_i, необхідно вилучити усі вершини S_i з потоку завдань W і забезпечити комунікаційні механізми, що пов'язують MF_i з платформою потокової передачі подій через вершину споживача (*Consumer vertex, cv_i*) і вершину генератор (*producer vertex, pv_i*).

Вершина cv_i в MIF_i забезпечує споживання потоку вхідних даних від платформи потокової передачі даних і розподіляє його між вершинами VI_i.

Вершина pv_i в MIF_i діє як стік, котрий збирає вихідні дані з вершин, що становлять множину VO_i та передає їх у вигляді повідомлень на платформу потокової передачі даних. Визначимо відповідні множини ребер таким чином:

– $ECV_i = \{(cv_i, v) : v \in VI_i\}$, набір ребер, ідучих від вершини cv_i до вершин в VI_i;

– $EPV_i = \{(v, cp_i) : v \in VO_i\}$, набір ребер, ідучих від вершин в VO_i до вершини cp_i.

У цьому випадку мікропотік завдань MIF_i з підпотоку завдань S_i визначається як:

$$MWF_i = (MV_i, ME_i), \quad (2.7)$$

де $MV_i = V_i \cup \{cv_i, pv_i\}$ - множина всіх вершин, що знаходяться всередині S_i, включаючи cv_i і pv_i; $ME_i = E_i \cup ECV_i \cup EPV_i$ – множина ребер, розташованих всередині S_i, включаючи всі ребра, які йдуть з cv_i до вершин в VI_i, а також усі ребра, ідучі від вершин у VO_i до cp_i.

Рисунок 2.3 показує набір мікропотоків завдань, отриманих в результаті вилучення підпотоків завдань з монолітного потоку W, показаного на рисунку 2.2, а.

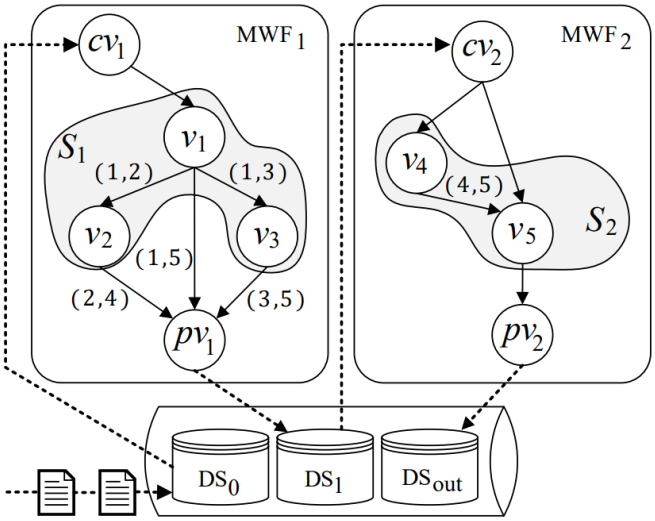


Рисунок 2.3 – Результат застосування моделі мікропотоку робіт

Модель мікропотоків завдань об'єднує моделі потоків завдань і потоковоїобробки даних.

У структурі платформи потокової обробки даних формується набір виділених каналів (сховищ потоків даних) для організації взаємодії мікропотоків завдань у вигляді обміну повідомленнями.

Кожне повідомлення є набором даних, котрий включає в себе такі інформаційні поля:

- мітку часу створення повідомлення,
- інформацію про джерело даних,
- структуровану колекцію даних, що передаються.

Необхідні наступні сховища потоків даних (рисунок 2.3):

- DS_0 відповідає за збір, зберігання і надання повідомлень, що містять набори даних, які необхідні для ініціалізації обчислювального процесу в потоці завдань;

- DS_i відповідає за отримання повідомлень, що містять проміжні дані від MWF_i, і передачу їх залежним мікропотокам завдань;

- DS_{out} відповідає за збір, зберігання і надання повідомлень, що містять результуючі дані.

Замість початкового вузла потоку завдань набори даних, необхідні для запуску обчислювального процесу, надходять в сховище потоків даних DS_0 у вигляді повідомлень. Обробка повідомлень із сховища потоку даних організована наступним чином:

1) cvi відповідного $MIFi$ витягує наступне повідомлення з $DSi{-}1$;

2) на основі аналізу отриманого повідомлення cvi ініціює передачу даних по ребрах $ECVi$ до вершин, відповідальних за виконання безпосереднього обчислювального процесу;

3) після завершення завдань обробки даних та відправлення даних по ребрах $EPVi$, pvi генерує вихідне повідомлення у DSi або $DSout$, якщо це остаточний результат.

2.4 Алгоритм методу обробки інформаційних потоків у туманному середовищі

Розглянемо алгоритм, що забезпечує рефакторинг монолітного потоку завдань в набір мікропотоків завдань. У алгоритмі рефакторингу можуть бути виділено наступні ключові кроки:

1) Ініціалізація матриці Z для представлення монолітного потоку завдань W та множини підпотоків S , на які розбивається потік завдань W.

2) Відображення в матриці Z внутрішніх і зовнішніх ребер для підпотоківзавдань $S_X \in S$.

3) Створення матриці M_X для представлення кожного мікропотоку завдань MIF_X .

Для кожного підпотоку завдань $S_X \in S$:

a) Ініціалізація матриці M_X .

b) Відображення множини внутрішніх ребер підпотоку E_X у відповідну матрицю M_X.

c) Формування множини ребер, вихідних з вершини споживача cv_X та їх відображення в M_X.

d) Формування множини ребер, вхідних в вершину генератор pv_X і їх відображення в M_X.

Для опису монолітного потоку завдань $W = (V, E)$, розділеного на набірпідпотоків завдань $S = (S_1, \dots S_k)$, визначимо матрицю $Z_{n\times n}$, де n - це загальнакількість вершин у потоці завдань:

$$Z = \begin{bmatrix} z_{1,1} & \cdots & z_{1,n} \\ \cdots & \cdots & \cdots \\ z_{n,1} & \cdots & z_{n,n} \end{bmatrix}. \quad (2.8)$$

У основі запропонованого підходу лежить модифікація механізму представлення графів як матриці суміжності. Матриця Z буде представляти

інформацію о ребрах між вершинами і про то, до якого підпотоку належить кожна вершина. Ініціалізація цієї матриці реалізується наступним чином:

$$z_{i,j} = \begin{cases} 1, & (i \neq j) \wedge \left((v_i, v_j) \in E \right); \\ 0, & (i \neq j) \wedge \left((v_i, v_j) \notin E \right); \\ x, & (i = j) \wedge (v_i \in V_x), \end{cases} \tag{2.9}$$

де $i, j \in 1..n$ – індекси елементів матриці Z; $x \in 1..k$ – індекс формованого підпотоку $S_X = (V_X, E_X)$.

Значення $z_{i,i}$ на діагоналі матриці у цьому випадку відповідає індексу x підпотоку завдань S_X, якому належить вершина з індексом i, в той час, коли інші значення елементів матриці $z_{i,j}$ позначають наявність ребра між вершинами v_i та v_j, де 0 означає відсутність ребра між v_i і v_j , а «1» означає наявність відповідного ребра. На рисунку 2.4 показаний приклад ініціалізації матриці Z для прикладу потоку завдань W з двох підпотоків.

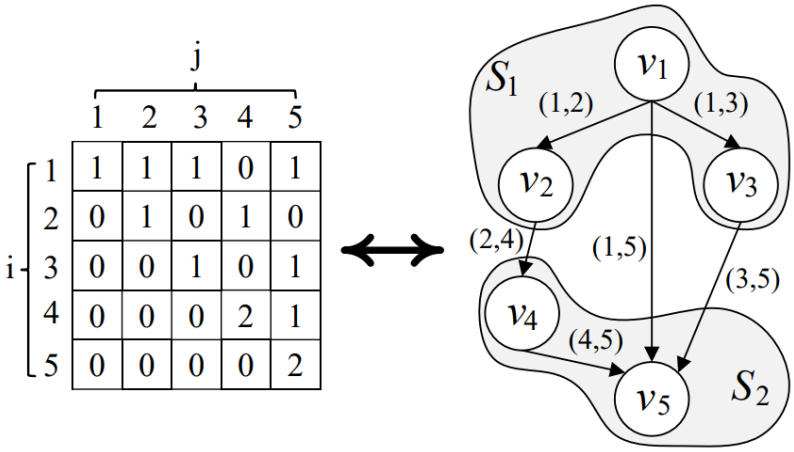


Рисунок 2.4 – Приклад ініціалізації матриці Z, отриманий за формулою (2.4) для прикладу потоку завдань W із двох підпотоків

На наступному кроку алгоритму відбувається визначення внутрішніх і зовнішніх ребер для кожного підпотоку в Z:

$$z_{i,j} = \begin{cases} z_{i,i}, & (i \neq j) \wedge (z_{i,j} = 1) \wedge (z_{i,i} = z_{j,j}); \\ -1, & (i \neq j) \wedge (z_{i,j} = 1) \wedge (z_{i,i} \neq z_{j,j}); \\ z_{i,j}, & \textit{у інших випадках.} \end{cases} \quad (2.10)$$

Так як на кроці ініціалізації (2.9) у значенні елементів матриці Z, розташованих на діагоналі ($z_{i,i}$) було збережено індекс підпотоку, в якому знаходиться вершина v_i , ці значення були використані у формулі (2.10) для розпізнавання внутрішнього та зовнішнього ребер для кожного підпотоку в Z. У першому випадку визначається множина внутрішніх ребер, які пов'язували дві вершини v_i та v_j перебувають у тому самому підпотоці . Це реалізується шляхом зіставлення відповідних діагональних значень $z_{i,i}$ і $z_{j,j}$ для ребра (v_i, v_j), представленого в $z_{i,j}$ = 1.

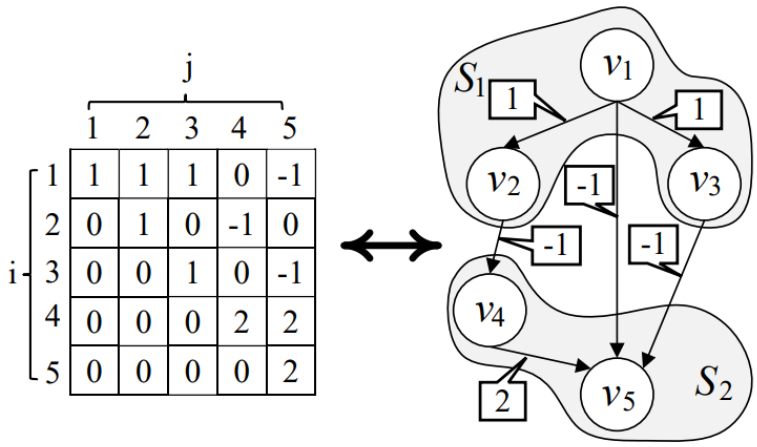
Якщо значення $z_{i,i}$ і $z_{j,j}$ для ребра, представленого $z_{i,j}$ однакові, це означає, що вершини v_i і v_j знаходяться в одному і тому ж підпотоці, і значення $z_{i,j}$ встановлюється рівним значенням індексу відповідного підпотоку, збереженого в відповідних діагональних елементах матриці Z.

У другому випадку, якщо значення відповідних діагональних елементів $z_{i,i}$ і $z_{j,j}$, що характеризують вершини v_i і v_j по-різному, то вершини знаходяться в різних підпотоках, і (v_i, v_j) – це зовнішнє ребро. У цьому випадку значення ребра $z_{i,j}$ встановлюється рівним "–1".

На рисунку 2.5, а показано приклад застосування кроку, описаного у формулі (2.10). Результатом цього кроку є сформована матриця Z.

На наступному кроці алгоритму відбувається ініціалізація матриць M_X представлення мікропотоків завдань MIF_X на основі підпотоків S_X. Визначимо M_X як матрицю, використовувану для визначення мікропотоку завдань MIF_X на основі підпотоку S_X і вершин cv_X і pv_X. Щоб уникнути плутанини, для індексації рядків і стовпців елементів в матрицях представлення мікропотоків M_X будемо використовувати відповідно змінні a

та b (рисунок 2.5).

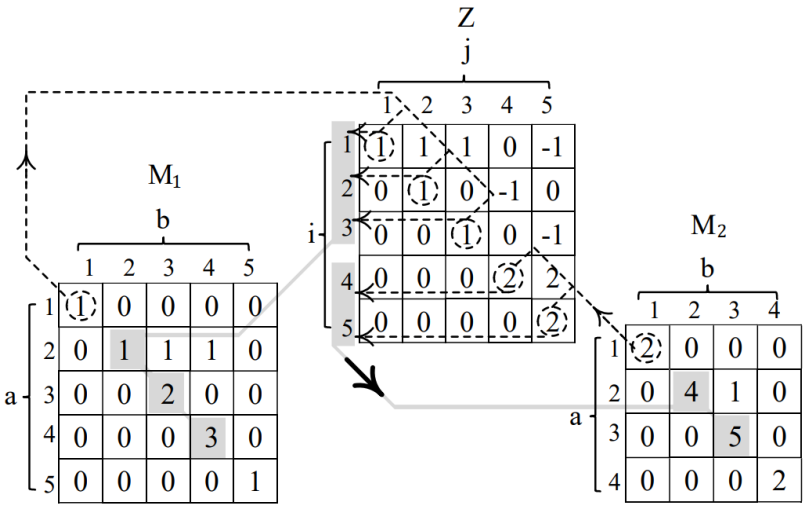


а б

Рисунок 2.5 – Внутрішні і зовнішні ребра підпотоків:

а – матриця Z після виконання кроку (2.10),

b – візуалізація W після виконання кроку (2.10)



а б

Рисунок 2.6 – Матриця Z (а) та ініціалізація матриць M_1 і M_2

Розмірність r_x матриці M_x визначається кількістю рядків у матриці Z, у яких значення на діагоналі рівні x :

$$r_x = |\{ a \in 1..n: z_{a,a} = x\}|. \quad (2.11)$$

Для представлення вершини споживача (cv_x) та вершини генератора (pv_x) необхідно в матрицю M_x додати ще 2 рядки і 2 стовпці таким чином:

$$M_x = \begin{bmatrix} m_{x_{1,1}} & \cdots & m_{x_{1,r_x+2}} \\ \cdots & \cdots & \cdots \\ m_{x_{m_{x_{1,1}},1}} & \cdots & m_{x_{r_x+2,r_x+2}} \end{bmatrix}. \quad (2.12)$$

Представлення вузла cv_x буде розміщено на позиції $m_{x1,1}$, а pv_x - на позиції $m_{xr+2,r+2}$

Діагональні значення в матриці M_x обох цих вузлів встановимо рівними x. В решті діагональних значень збережемо індекси i вершин потоку завдань W, які входять у підпотік S_x , тобто мають значення, що дорівнює x.

Збережемо значення індексів i всіх вершин з S_x в D_x де x – це індекс підпотоку S_x:

$$D_x = \{\forall i \in 1..n: z_{i,i} = x\}. \quad (2.13)$$

Значення $d \in D_x$ будуть використані в (2.14) для заповнення діагональних значень для матриці M_x:

$$m_{x_{a.a}} = \begin{cases} d_{a-1}, & 1 < a < r_x + 2; \\ x, & (a = 1) \vee (a = r_x + 2), \end{cases} \quad (2.14)$$

де $x \in 1..k$ –індекс підпотоку S_x; $a \in 1..r_x + 2$ – індекс рядки в матриці M_x.

На рисунку 2.6 бачимо приклад застосування виразів (2.11) – (2.14) для ініціалізації матриць мікропотоків M_1 і M з Z.

Після ініціалізації матриць M_x для кожного підпотоку S_x , реалізується розміщення в них уявлення всіх внутрішніх ребер підтпооків E_x . Для того, щоб перенести внутрішні ребра E_x з Z в M_x застосовується така формула:

$$m_{x_{a,b}} = \begin{cases} 1, & z_{m_{x_{a,a}} m_{x_{b,b}}}; \\ 0, & \text{в інших випадках}; \end{cases}$$
$$\forall m_{x_{a,b}} \in M_x : (a,b \in 2..r_x + 1) \wedge (a \neq b). \quad (2.15)$$

У цьому випадку застосовується інформація про суміжні вершини,

збережена в діагональних позиціях M_X як посилання, де a – індекс рядка в M_X, а b - індекс стовпця в M_X .

На рисунку 2.7 показаний приклад застосування (2.15) для перенесення внутрішнього ребра (v_1, v_2) з E_1 в M_1 , а також результати представлення внутрішніх ребер підпотоків завдань з множин E_1 і E_2 в M_1 і M_2 відповідно.

Після представлення всіх внутрішніх ребер з E_X в M_X відбувається визначення вихідних ребер для вершини споживача cv_X в матрицях описи підпотоків M_X . Існує 2 типи вершин в M_X , які повинні отримувати вхідні ребра від cv_X :

- початкова вершина W, яка не має вхідних ребер;
- вершини, які отримують вхідні ребра від вершин, що знаходяться в інших підпотоках.

Формула (2.16) визначає значення позицій у першому рядку матриць M_X, що визначають вихідні ребра з вершини cv_X :

$$m_{x_{1,b}} = \begin{cases} 1, & \left(\left(\forall i \in 1..n\right) \wedge \left(i \neq m_{x_{b,b}}\right) : z_{i,m_{x_{b,b}}} = 0\right) \\ & \wedge \left(\left(\exists i \in 1..n\right) : z_{i,m_{x_{b,b}}} = -1\right); \\ 0, & \text{в інших випадках;} \end{cases}$$
$$\forall m_{x_{1,b}} \in M_x : b \in 2..r_x + 1. \qquad (2.16)$$

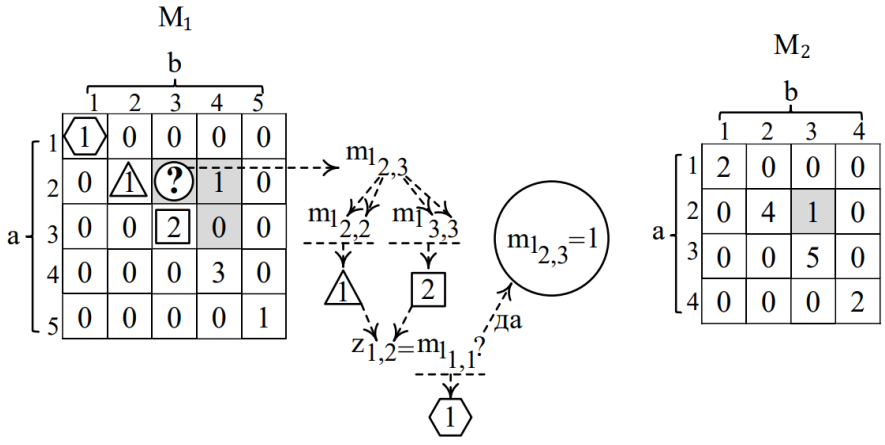


Рисунок 2.7 – Результати застосування формули (2.15)
для подання внутрішніх ребер множин E_1 і E_2 у M_1 і M_2 відповідно

Перший випадок для значення «1» у формулі (2.16) визначає ребра з cv_x до тих вершин, які від самого початку не мають вхідних ребер в W. Для отримання індексу стовпця, за яким здійснюється пошук у матриці Z, використовується значення $m_{xb,b}$, де $b \in 2..r_x + 1$. Якщо значення всіх елементів матриці (крім діагонального) Z в стовпці $m_{x\ b,b}$ рівні 0, це значить, що від самого початку у вершини з індексом $m_{xb,b}$ не було вхідних ребер в Z , таким чином вона має бути з'єднана з cv_x в M_x .

На рисунку 2.8 показаний приклад застосування цього випадку формули (2.16) для визначення наявності ребра, що має бути представлено в позиції $m1_{1,2}$ в M_1.

Другий випадок для значення «1» в формулі (2.16) визначає ребра з cv_x в ті вершини, які мають вхідні ребра з інших підпотоків. Аналогічним чином, значення $m_{xb,b}$ використовується для отримання індексу стовпця, по якому проводиться пошук у матриці Z.

Наявність хоча б одного елемента зі значенням «1» у відповідному стовпці означає, що дана вершина отримує дані з зовнішнього підпотоку.

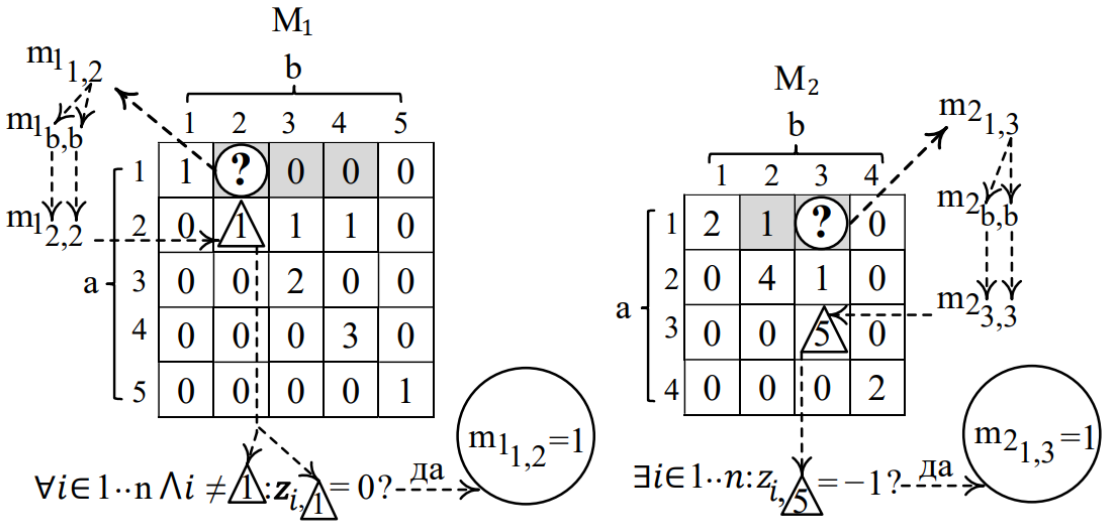


Рисунок 2.8 – Застосування формули (2.16) до M_1 і M_2 для з'єднання cv

Дана вершина повинна отримати вхідне ребро з cv_x в M_x. На рисунку 2.8 показаний приклад застосування цього випадку формули (2.16) для

визначення наявності ребра, яке повинно бути представлено в позиції $m2_{1,3}$ в M_2.

Фінальним кроком алгоритму рефакторингу є визначення ребер, які мають бути завершені у вершині генератора pv_x для кожної матриці підпотоку M_x.

Є 2 типи вершин в S_x , які повинні бути пов'язані ребрами з pv_x:

- вершина, що не має вихідних ребер в E_x ;
- вершини, вихідні ребра яких йдуть до вершин, що знаходяться в інших підпотоках.

Формула (2.17) визначає значення позицій в останньому стовпці (r_x + 2) матриць M_x , визначальних вхідні ребра в вершину pv_x :

$$m_{x_{a,r_x+2}} = \begin{cases} 1, & \left(\left(\forall b \in 2..r_x+1\right) \wedge \left(a \neq b\right) : m_{x_{a,b}} = 0\right) \\ & \wedge \left(\left(\exists j \in 1..n\right) : z_{m_{x_{a,a}},j} = -1\right); \\ 0, & \text{в інших випадках;} \end{cases}$$
$$\forall m_{x_{a,r_x+2}} \in M_x : a \in 2..r_x+1. \qquad (2.17)$$

Перший випадок для значення «1» у формулі (2.17) визначає ті вершини, які від самого початку не мають вихідних ребер. Такі вершини з'єднуються ребрами з pv_x.

На рисунку 2.9 показаний приклад застосування цього випадку формули (2.17) для визначення наявності ребра, яке повинно бути представлено в позиції $m2_{2,4}$ в M_2 .

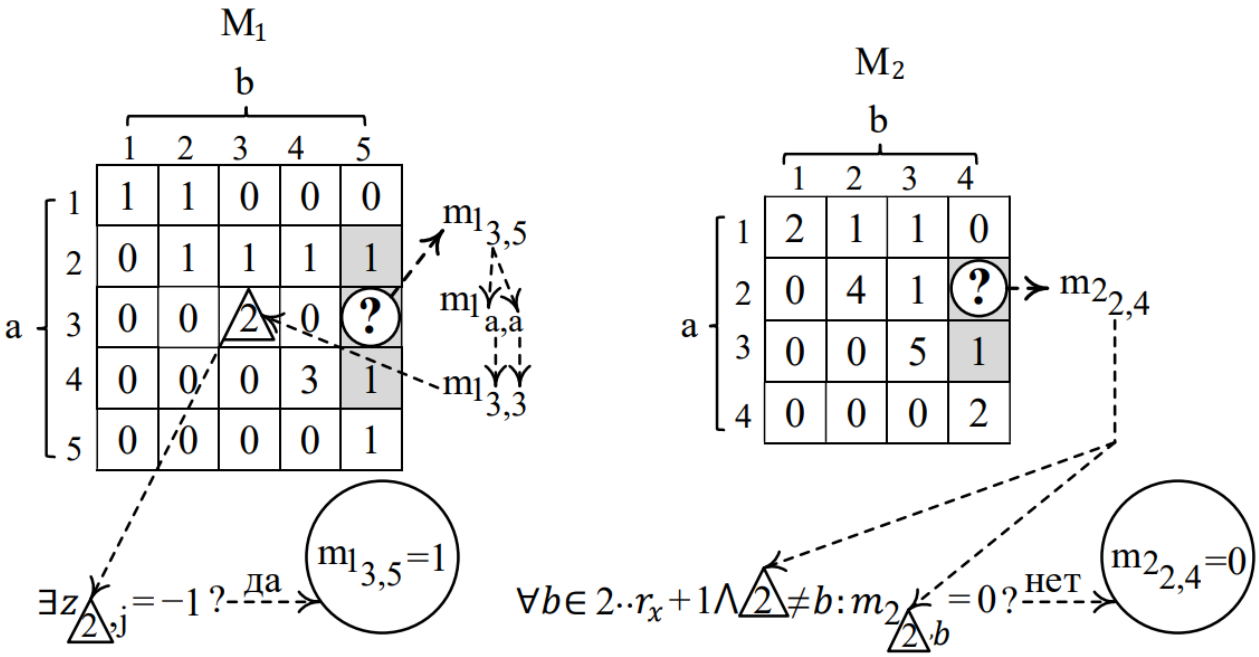


Рисунок 2.9 – Застосування формули (2.17) до M_1 і M_2 для з'єднання pv

Другий випадок для значення «1» в формулі (23) , визначає ребра в pv_x з тих вершин, які спочатку мали вихідні ребра у вершини, які є у інших підпотоках. Значення $m_{x_{a,a}}$ використовується для отримання індексу рядка, за яким проводиться пошук елементів у матриці. Наявність хоча б одного елемента зі значенням «1» у рядку з індексом $m_{x_{a,a}}$ у матриці Z означає, що ця вершина передає дані в зовнішній підпотік. У цьому випадку дана вершина повинна бути з'єднана вихідним ребром з pv_x в M_x .

На рисунку 2.9 показаний приклад застосування цього випадку формули (2.17) для визначення наявності ребра, яке має бути представлено в позиції $m1_{3,5}$ в $M1$.

На рисунку 2.10 показані фінальні стани матриць M_1 і M_2, що представляють мікропотоки завдань MIF_1 та MIF_2 , сформовані на основі розбиття монолітного потоку завдань W, наведеного як приклад на рисунку 2.4.

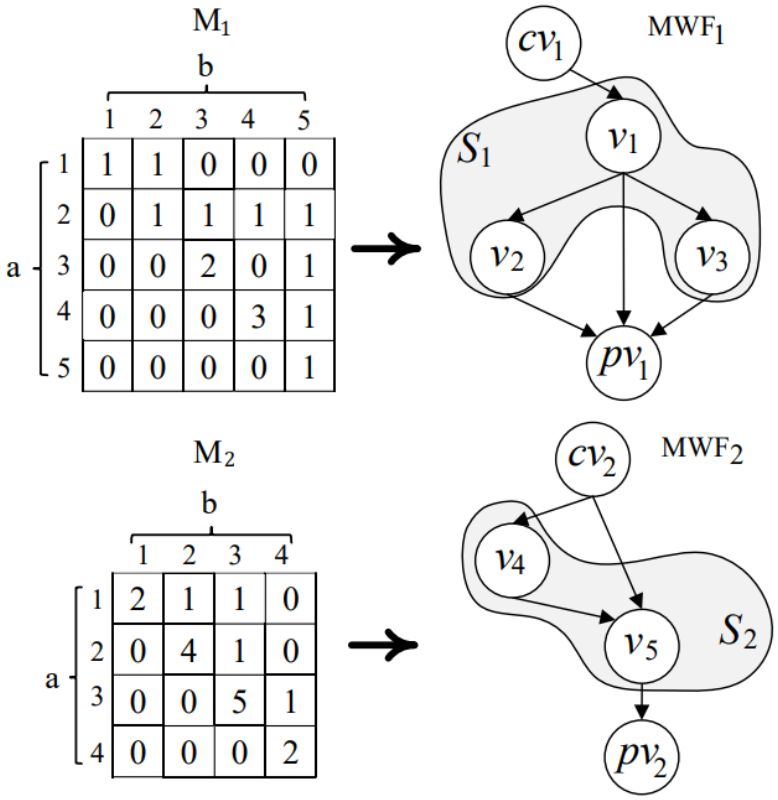


Рисунок 2.10 – Фінальний стан матриць M_1 і M_2, що репрезентують MIF_1 і MIF_2

У другому розділі представлено метод обробки інформаційних потоків у туманному середовищі. Розглянуті концепція рефакторінга інформаційних потоків та процес виділення підпотоків завдань. Виділені ключові стадії в процесі рефакторингу монолітного інформаційного потоку завдань у множину мікропотоків. Запропонована математична модель мікропотоків завдань для обробки потоків даних у розподілених обчислювальних середовищах, таких як туманні обчислення. Модель включає алгоритм рефакторингу залежностей в монолітному потоці в набір автономних мікропотоків завдань. Такий поділ підтримує незалежність реалізації, виконання, розробки, супроводу та кросплатформного розгортання. мікропотоків завдань на незалежних обчислювальних вузлах. Дана модель використана при розробці алгоритму методу обробки інформаційних потоків у туманному середовищі.

3 ДОСЛІДЖЕННЯ МЕТОДУ ОБРОБКИ ІНФОРМАЦІЙНИХ ПОТОКІВ У ТУМАННОМУ СЕРЕДОВИЩІ

3.1 Імітаційне моделювання елементів потокової обробки даних у туманному середовищі

Для оцінки методу обробки інформаційних потоків у туманному середовищі, що базується у вигляді мікропотоків робіт, був розроблений набір модулів, розширюючих можливості системи забезпечення потокової обробки даних з використанням платформи Apache Kafka. У процесі запуску та виконання, реалізовані модулі можуть зчитувати інформацію як із змінних середовища, значення яких встановлені на вузлі, на якому розгорнуть потік робот, так і зі свого графічного інтерфейсу, якщо змінні середовища не встановлені. Таким чином, забезпечується можливість виконання даних модулів як в інтерактивному режимі з підтримкою користувача інтерфейсу, так і в автономному режимі.

3.1.1 Модуль реалізації вершини споживача мікропотоку робіт

Розроблений модуль KafkaConsumer є реалізацію вершини споживача мікропотоку робіт.

Модуль KafkaConsumer підписується на тему Kafka, підключаючись до сервера Apache Kafka за протоколом TCP.

Після підключення, цей модуль отримує потік даних (у послідовності повідомлень) з цієї теми. Кожне повідомлення містить набір полів даних, які формуються за допомогою системи серіалізації даних Apache Avro. Модуль забезпечує десеріалізацію кожного отриманого повідомлення у набір токенів, які потім відправляються на вихідний порт цього модуля.

Для підтримки десеріалізації даних модулю необхідна схема Apache

Avro Schema для вхідних повідомлень. Модуль отримує цю схему з сервера реєстру схем.

Лістинг 3.1 – Модуль KafkaConsumer

```
procedure KafkaConsumer
D Стадія ініціалізації
if environmentVariables = null
PARAMS ← GUI
else
PARAMS ← environmentVariables
con ← ConnectToKafkaConsume ( PARAMS.
SOURCE_KAFKA_SERVER,
PARAMS. SOURCE_GROUP_ID , PARAMS. SOURCE_ TOPIC ,
PARAMS. SOURCE_CONSUME_TIMEOUT )
schema ← ReadSchema ( PARAMS. SOURCE_SCHEMA_REG_SERVER)
D Цикл обробки повідомлень
message ← ConsumeMessage ( con, Schema )
rcvMsg ← GetRecieveTimestamp ( Message )
record ← InitializeRecord ( )
if PARAMS. SOURCE_PARMS_TO_READ <> null
for each field in message :
if field. name ∈ PARAMS. SOURCE_PARMS_TO_READ :
record [ PARAMS. SOURCE_TOPIC ]. Insert ( field )
else
for each field in message :
record [ PARAMS. SOURCE_TOPIC ]. Insert ( field )
record . Insert ( rcvMsg )
SendRecordToOutputPort ( record)
goto 8
end procedure
```

Лістинг 3.1 є кодом процесу роботи модуля KafkaConsumer.

3.1.2 Модуль реалізації вершини генератора мікропотоку робіт

Модуль KafkaProducer є реалізацією вершини генератора мікропотоку робіт. Цей модуль отримує токени в якості вхідних даних на вхідний порт, серіалізує кожну отриману запис токенів в вигляді повідомлення в форматі Apache Avro , а потім відправляє це повідомлення відповідної темі Apache Kafka на сервері Kafka по протоколу TCP. Для підтримки серіалізації даних,

модулю необхідна схема Apache Avro Schema для вихідних повідомлень. Модуль отримує дану схему з сервера реєстру схем.

Лістинг 3.2 – Модуль KafkaProducer

```
1: procedure KafkaProducer
   D Стадія ініціалізації
2: if environmentVariables = null
3: PARAMS ← GUI
4: else
5: PARAMS ← environmentVariables
6: con ← ConnectToKafkaProduce ( PARAMS. OUT_KAFKA_SERVER,
   PARAMS. OUT_TOPIC )
7: schema ← PARAMS. OUT_AVRO_SCHEMA
8: SendSchema ( PARAMS. OUT_SCHEMA_REG_ SERVER , schema)
   D Цикл обробки вхідних записів токенів
9: record ← RecieveRecordFromInputPort ( )
10: message ← InitializeMessage ( schema )
11: if PARAMS. OUT_PARMS_TO_READ <> null
12: for each field in record :
13: if field. name ∈ PARAMS. OUT_PARMS_TO_READ :
14: message . Insert ( field )
15: else
16: for each field in record :
17: message . Insert ( field )
18: ProduceMessage ( con, message)
19: goto 9
20: end procedure
```

Лістинг 3.2 є кодом роботи модуля KafkaProducer.

3.1.3 Модуль моніторингу змін у серіях даних

Модуль DetectStateChange розроблений для постійного моніторингу змін у серіях даних, одержуваних від окремого джерела даних або від набору джерел даних. У рамках даного модуля окреме джерело даних позначається

як «тема» (topic). У кожну одиницю часу модуль отримує на вхід запис токенів. Ідентифікатор кожної теми є ідентифікатором одного з джерел серії даних. Якщо потрібно виявити зміну стану в певному наборі джерел, він може вказати набір ідентифікаторів джерел, що його цікавлять, параметром inTopic , вказавши їх ідентифікатори через пробіл (наприклад, як «topic1 topic2 topic3»).

Лістинг 3.3 – Модуль DetectStateChange

```
1: procedure DetectStateChange
   D Стадія ініціалізації
2: if environmentVariables = null
3: PARAMS ← GUI
4: else
5: PARAMS ← environmentVariables
6: recordStore ← InitializeStore ( )
   D Цикл обробки вхідних записів токенів
7: recordIn ← RecieveRecordFromInputPort ( )
8: recordOut ← InitializeRecord ( )
9: if recordStore . IsEmpty ()
10: recordStore ← recordIn
11: goto 7
12: for each inTopicName, inTopicValue in recordIn :
13: if inTopicName ∈ PARAMS. inTopic :
14: for i = 1 to inTopicValue. p. Length ():
15: if inTopicValue. p[i ]. Name () ∈ PARAMS. inPP and
    recordStore[inTopicName]. p[i] <> inTopicValue. p[i] :
16: outTopic ← recordOut[inTopicName]
17: if outTopic. IsEmpty ():
18: outTopic. index ← inTopicValue. index
19: outTopic. rcvTime ← inTopicValue. rcvTime
20: outTopic. edge[i] ← inTopicValue. p[i]
21: outTopic. outTs[i] ← inTopicValue. ts[i]
22: recordOut[inTopicName] ← outTopic
23: if not recordOut. IsEmpty ()
24: recordStore ← recordIn
    SendRecordToOutputPort ( recordOut)
25: goto 7
26: end procedure
```

Лістинг 3.3 є кодом роботи модуля DetectStateChange.

3.1.4 Модуль аналізу кореляції зв’язків між змінами

Модуль CorrelateStateChange вирішує завдання виявлення зв'язків між змінами, які відбулися в однієї серії даних зі змінами, які відбулися в іншій серії даних шляхом аналізу кореляції між ними. Суть розв'язуваної задачі зводиться до розрахунку часової різниці між підвищеним (1) та зниженим (0) станами кожної із вхідних пар сенсорів.

Лістинг 3.4 є кодом роботи модуля CorrelateStateChange.

Лістинг 3.4 – Модуль CorrelateStateChange

```
procedure CorrelateStateChange

D Стадія ініціалізації

if environmentVariables = null
PARAMS ← GUI
else
PARAMS ← environmentVariables
rStore ← InitializeCorrelateStore ( )

D Цикл обробки вхідних записів токенів

rIn ← RecieveRecordFromInputPort ( )
rOut ← InitializeRecord ( )
for each topic in rIn :
if topic . Name () ∈ PARAMS. inTopic :
for each edge, ts in topic :
edgeIndex ← PARAMS. xxEdges. Index ( edge . Name ())
if edgeIndex <> null :
yyEdgeName ← PARAMS. yyEdges [ edgeIndex ]
if rStore [ topic . Name ()][ yyEdgeName ][ edge ] = null :
continue
dtName ← PARAMS. outDT [ edgeIndex ]
tsName ← PARAMS. outTS [ edgeIndex ]
cRec ← rStore [ topic . Name ()][ yyEdgeName ][ edge ]
rOut [ topic . Name ()][ dtName ] ← xxTscRec. ts
rOut [ topic . Name ()][ tsName ] ← cRec. ts
SendRecordToOutputPort ( rOut)
for each edge, ts in topic :
edgeIndex ← PARAMS. yyEdges. Index ( edge . Name ())
if edgeIndex <> null :
yyEdgeName ← PARAMS. yyEdges [ edgeIndex ]
rStore [ topic . Name ()][ yyEdgeName ][ edge ] . ts ← ts
goto 7
end procedure
```

3.2 Дослідження монолітного інформаційного потоку завдань у туманному середовищі

Для тестування алгоритму рефакторингу монолітного потоку завдань на набір мікропотоків завдань, представленого у розділі 2, у якості монолітного потоку завдань, на підставі якого проводилося тестування запропонованого алгоритму, із спеціалізованого датасету був взятий один із типових потоків завдань « Montage 25», представлений на рисунку 3.1.

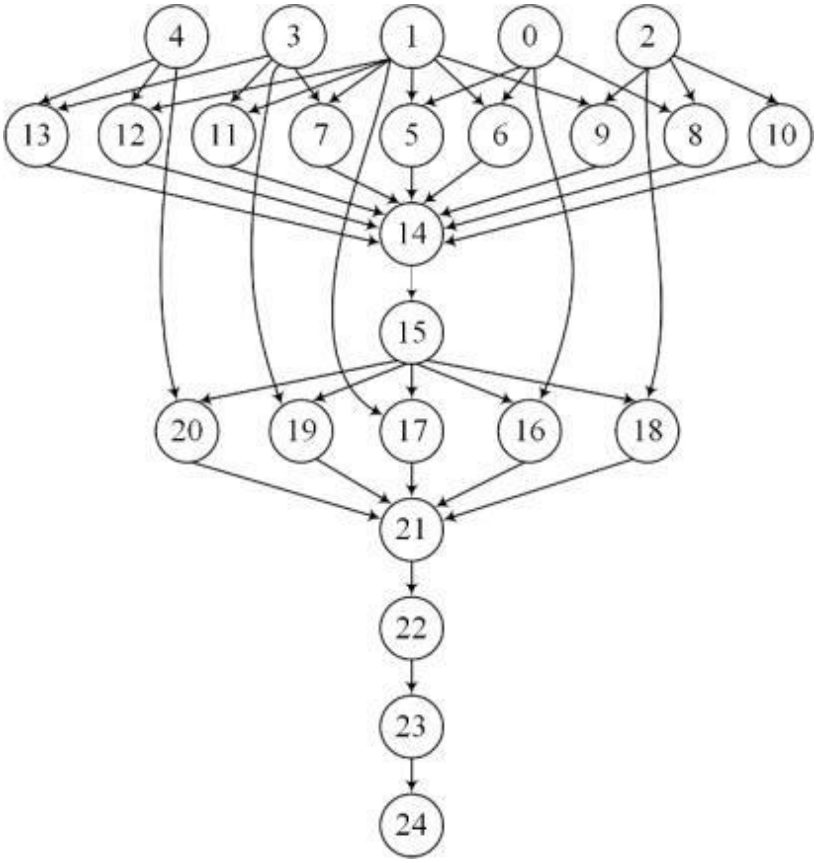


Рисунок 3.1 – Типовий потік завдань « Montage 25»

Процес рефакторингу потоку завдань Montage 25» на набір мікропотоків завдань починається з ручного поділу вхідного потоку завдань на підпотоки шляхом вказівки вершин потоку завдань, що входять в кожен із підпотоків. На підставі множини вершин у кожному з підпотоків, а також наборів ребер з початкового монолітного потоку завдань здійснюється його

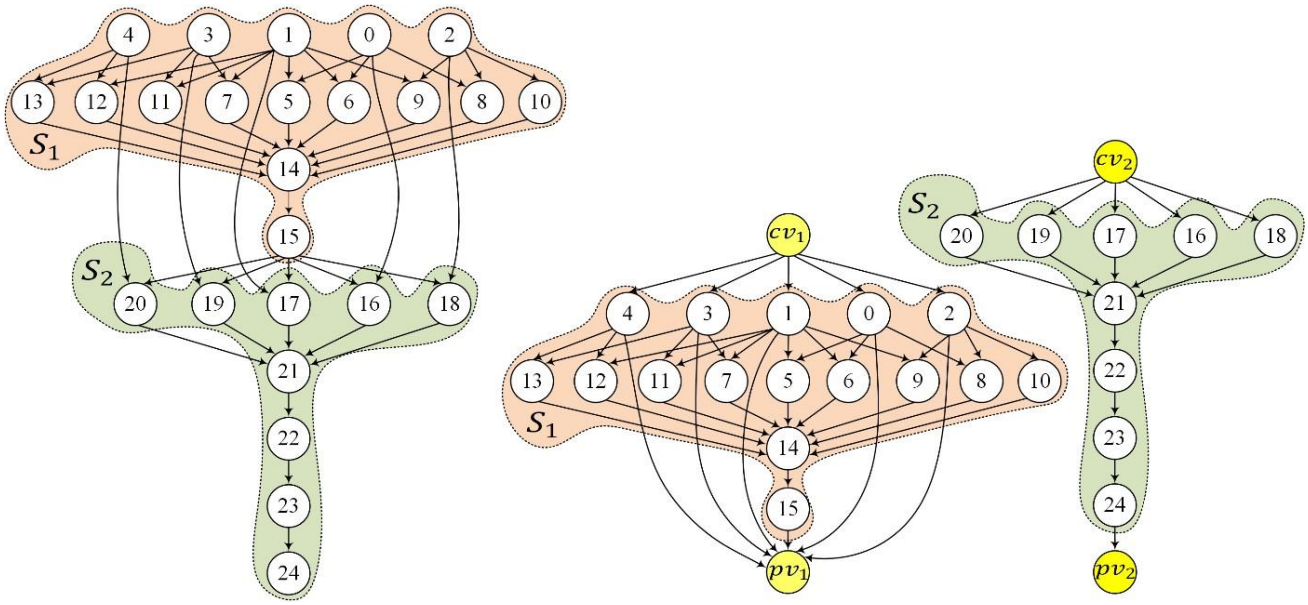
рефакторинг в набір мікропотоків завдань.

Експеримент проводився на персональному комп'ютері, оснащеному процесором Intel (R) Core (TM) i54210U та 16 ГБ оперативної пам'яті.

У рамках першого випробування потік завдань «Montage 25» був розділений на два мікропотоки завдань.

На рисунку 3.2, а представлена візуалізація розподілу «Montage 25» на два підпотоки: $S1$ та $S2$.

На рисунку 3.2 б і 3.2 в показано візуалізацію результатів виконання алгоритму рефакторингу над потоком завдань «Montage 25» для виділення двох мікропотоків завдань.



а б в

Рисунок 3.2 – Рефакторинг « Montage 25» на два мікропотоки завдань

В рамках другого експерименту той же потік завдань «Montage 25» був розділений на три мікропотоки завдань.

На рисунку 3.3 а представлена візуалізація розподілу цього потоку завдань на три підпотоки $S1$, $S2$ і $S3$.

На рисунку 3.3, б, в, г представлена візуалізація результатів рефакторингу потоку завдань « Монтаж 25» на три мікропотоки завдань.

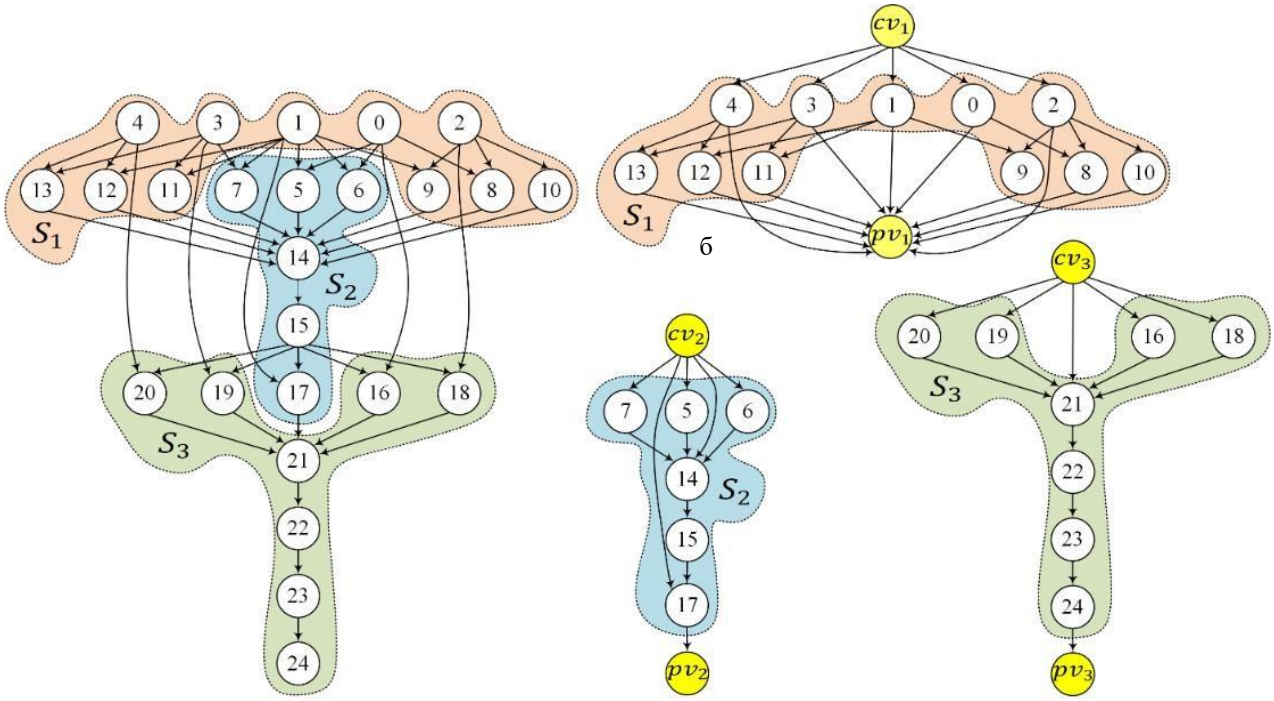


Рисунок 3.3 – Рефакторинг « Montage 25» на три мікропотоки завдань

3.3 Порівняння запропонованого методу обробки інформаційних потоків у туманному середовищі з існуючим підходом

Класичні потоки завдань реалізують концепцію пакетної обробки даних, що означає необхідність попередньої підготовки масиву даних для обробки. На відміну від монолітного підходу, запропонований метод з використанням мікропотоків завдань підтримує можливість потокової обробки даних. Критерієм оцінки в рамках даного експерименту є середній час реакції на подію, інформація про яку може бути отримана з потоку даних в реальному часу.

В рамках даного експерименту моделюється типова поведінка монолітних та мікропотоків завдань. Обчислювальний процес монолітного потоку завдань організовано у вигляді пакетної обробки даних. Для запуску обробки даних у пакетному режимі, необхідно провести їх попередній збір та розміщення на сервері зберігання, з якого вони будуть взяті початковим

вузлом монолітного потоку завдань. Після того, як здійснено планування обчислювальних завдань, що становлять монолітний потік завдань, і вони розподілені по обчислювальним вузлам обчислювальної системи, запускається зчитування даних та їх пакетна обробка. Потік завдань завершує своє виконання після завершення виконання останнього завдання та звільнення зайнятих системних ресурсів. Щоб запустити монолітний потік завдань для нового пакета даних, усі перераховані вище завдання мають бути виконані заново. Оскільки пакет даних має бути заздалегідь підготовлений для обробки монолітним потоком завдань, необхідно використання спеціальних стратегій розподілу даних між обчислювальними ресурсами. Цей етап перерозподілу даних також надає значний вплив на загальний час виконання потоку завдань. З іншого боку, при обробці кожної нової порції потоку даних необхідно докладати додаткові обчислювальні зусилля перетворення інформації з потоку даних (набір незалежних повідомлень) в пакет даних (що містить повний набір даних за певний період). У даному експерименті збиральник потоку даних реконфігурується для збору всіх повідомлень, що містять інформацію за заданий період часу. Складальник даних забезпечує серіалізацію зібраних даних у вигляді пакета даних (вхідного файлу), що містить усі дані за необхідний період, а потім запускає монолітний потік даних для обробки цього пакета. Результати обробки пакета даних генеруються після закінчення останнього завдання в монолітному потоці завдань.

У рамках мікропотоку завдань організовано процес потокової обробки даних в режимі постійного виконання, без заздалегідь заданого певного часу завершення обробки даних. Такий підхід дозволяє уникнути затримок, характерних для пакетної обробки даних, пов'язаних з необхідністю попереднього накопичення пакета даних, планування виконання потоку даних на обчислювальних вузлах, накладних витратах на розгортання та запуск процесів обробки даних тощо. Також, обробка потоку даних дозволяє забезпечити на порядки нижчий час реакції системи на події, інформація о

яких може бути ідентифікована в потоці даних.

Експеримент реалізований на одному вузлі, оснащеному процесором Intel (R) Core (TM) i54210U та 16 ГБ оперативної пам'яті. Виконується одночасний запуск симулятора датчиків Інтернету речей, мікропотоку завдань та монолітного потоку завдань, що забезпечують обробку потоку даних, одержуваних з цих датчиків. Після завершення генерації даних проводиться очікування отримання інформації про події, ідентифіковані мікропотоком завдань і монолітним потоком завдань. В рамках експерименту було зроблено два випробування. У рамках випробування 1 реалізується обробка потоку даних протягом 30 хвилин. У рамках випробування 2 реалізується обробка потоку даних на протязі 1 години.

У кожному випробуванні симулятор датчиків формує потік даних, де кожне повідомлення потоку даних відображає інформацію з 8 датчиків в момент генерації цього повідомлення. Завдання обробки даних складається в тому, щоб забезпечити відстеження в зміні значень датчиків, і зафіксувати час цієї зміни. Зміна стану датчиків може статися будь-якої миті часу протягом тестового періоду. Затримка між генерованими повідомленнями складає близько 9,5 мілісекунд.

В рамках випробування 1 протягом 30 хвилин була проведена обробка 18000 повідомлень потоку даних. При цьому як монолітний, так і мікропотік завдань ідентифікували 150 подій, пов'язаних із зміною стану датчиків. В рамках випробування 2 на протязі 60 хвилин була зроблена обробка 378948 повідомлень потоку даних. При цьому як монолітний, так і мікропотік завдань ідентифікували 383 події, пов'язані із зміною стану датчиків. Результати, отримані в ході випробувань, наведено в таблицях 3.1 та 3.2.

Як метрики для оцінки результатів випробувань були використані наступні показники:

- час відповіді: середній час відповіді на подію в потоці даних, який розраховується як середнє значення для різниці між моментом генерації події джерелом даних і моментом виявлення цієї події при її обробці;

- об`єм вхідних даних: середній об`єм вхідних даних, необхідних на початку кожного обчислення;

- обсяг результуючих даних: середній обсяг результуючих даних, одержуваних в результаті обробки кожної порції даних.

Таблиця 3.1 – Результати випробування 1 (обробка 30 хвилин потоку даних)

Метод	Час відповіді	Об'єм вхідних даних	Обсяг результуючих даних
Стандартний	73,2 мс .	1,4 Кбайт	1,6 Кбайт
Запропонований	1.4 мс	1 Кбайт	1 Кбайт

Таблиця 3.2 – Результати випробування 2 (обробка 60 хвилин потоку даних)

Метод	Час відповіді	Об'єм вхідних даних	Обсяг результуючих даних
Стандартний	164,2 мс	2,8 Кбайт	2,4 Кбайт
Запропонований	1,4 мс	1 Кбайт	1 Кбайт

Результати, отримані в цьому експерименті, дозволяють зробити наступні висновки в плані обробки потокових даних при необхідності реакції на події в режимі, близькому до реального часу.

При організації обробки потоку даних за допомогою мікропотоку завдань середній час відповіді виявилося значно меншим, ніж час відповіді, що забезпечується монолітним потоком завдань. Це досягається за рахунок того, що мікропотік завдань отримує дані та обробляє дані в потоковому режимі, забезпечуючи надання результатів, як тільки вони стають доступними, в той же час як монолітний потік чекає, коли завершиться етап підготовки даних для запуску їх обробки.

Середній час відповіді на події при обробці у форматі мікропотоку завдань не залежить від загального періоду виконання, тоді як при реалізації обробки даних у пакетному режимі монолітного потоку завдань, середній час

надання відповідей зростає при збільшення періоду виконання у зв'язку із збільшенням періоду підготовки даних і збільшенням розміру вхідних даних.

Усереднений об`єм даних, необхідних для ініціалізації обчислювального процесу в форматі мікропотоку завдань, менший по порівнянні з монолітним потоком завдань. Аналогічно, середній розмір повідомлень з результатами обробки даних мікропотоку завдань також менше порівняно з монолітним потоком завдань.

Отже, запропонований метод більше підходить для розгортання в умовах обмежених мережних та обчислювальних ресурсів, зокрема, на вузлах туманних обчислювальних систем.

ВИСНОВКИ

Сукупність отриманих у кваліфікаційній роботі результатів дозволило вирішити актуальне науково-технічне завдання дослідження, спрямоване на розробку методу обробки інформаційних потоків у туманному середовищі.

В результаті проведених досліджень отримані такі наукові та практичні результати:

Проведений аналіз існуючих методів обробки інформаційних потоків Інтернету речей. На прикладі таких ключових програм IoT як цифрові двійники можна бачити, що хмарні обчислення не дозволяють забезпечити необхідний рівень обслуговування в частині мінімізації латентності та обробки даних у безпосередній фізичній близькості від джерел даних.

Визначені характерні особливості туманного середовища. Істотними особливостями організації таких систем є застосування подійно-орієнтованої архітектури та слабозв'язаної мікросервісної моделі організації обчислювальних сервісів для обробки потоків даних від систем IoT . Однак виникають складнощі при проектуванні та модернізації складних конфігурацій систем обробки даних. Фрагментація потоків робіт в пакетному режимі не дозволяє забезпечити динамічне масштабування частин потоку робіт, а також не дозволяє організувати прозору інтеграцію потоків даних від систем IoT .

Обґрунтована необхідність застосування концепції рефакторінга інформаційних потоків. Розглянуті концепція рефакторінга інформаційних потоків та процес виділення підпотоків завдань. Виділені ключові стадії в процесі рефакторингу монолітного інформаційного потоку завдань у множину мікропотоків.

Проведено моделювання процедури виділення мікропотоків завдань із вхідного монолітного потоку. Запропонована математична модель мікропотоків завдань для обробки потоків даних у розподілених обчислювальних середовищах, таких як туманні обчислення. Модель

включає алгоритм рефакторингу залежностей в монолітному потоці в набір автономних мікропотоків завдань. Такий поділ підтримує незалежність реалізації, виконання, розробки, супроводу та кросплатформного розгортання. мікропотоків завдань на незалежних обчислювальних вузлах. Дана модель використана при розробці алгоритму методу обробки інформаційних потоків у туманному середовищі.

Удосконалений метод обробки інформаційних потоків у туманному середовищі за рахунок проведення рефакторінгу монолітного потоку завдань на мікропотоки, що дозволило зменшити час реагування на зміни показань датчиків Інтернету речей в середньому до 20%. Алгоритм реалізації запропонованого методц має такі ключові кроки: формування матриць монолітного потоку завдань та множини підпотоків; поступове виділення у базовій матриці внутрішніх і зовнішніх ребер для підпотоків завдань та створення матриць для кожного мікропотоку завдань. На відміну від типових рішень обробки потоків даних, таких як Spark, Storm і Kafka, застосування запропонованого методу дозволяє інтегрувати у туманні обчислювальні середовища існуючі обчислювальні процеси, реалізовані на основі потоків завдань.

Проведено порівняльне дослідження запропонованого методу обробки інформаційних потоків у туманному середовищі. Для оцінки методу обробки інформаційних потоків у туманному середовищі, що базується у вигляді мікропотоків завдань, був розроблений набір модулів, розширюючих можливості системи забезпечення потокової обробки даних з використанням платформи Apache Kafka. У рамках експерименту з рефакторингу потоку завдань було проведено випробування та перевірка програмної утиліти, що реалізує алгоритм рефакторингу, який забезпечує поділ монолітного потоку завдань на набір незалежних мікропотоків завдань. У рамках експерименту по порівнянню монолітного потоку завдань та мікропотоку завдань для обробки потоків даних у реальному часі було показано перевагу концепції мікропотоків завдань у плані зменшення часу відгуку на події, які можуть

бути ідентифіковано в потоці даних Інтернету речей. Результати експериментів показують, що запропонований метод обробки інформаційних потоків у туманному середовищі розширює можливості існуючих підходів завдяки поєднанню гнучкості контейнерних технологій і стійкості підходу потокової передачі даних. Він підтримує можливість обробки подій від різних джерел (таких як датчики Інтернету речей) всередині обчислювальних потоків завдань. Впровадження та тестування даної архітектури на базі системи управління потоками завдань Kepler і платформи потокової обробки даних Apache Kafka із застосуванням розроблених модулів показує, що пропонований підхід успішно вирішує завдання обробки даних Інтернету речей, забезпечуючи достатньо невеликі розміри мережної затримки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Z. Kolesnyk, O.Mezhenskyi, O. Davykoza, H.Kuchuk Fog computing technology in distributed systems. Системи управління, навігації та зв'язку. Полтава : Національний університет «Полтавська політехніка імені Юрія Кондратюка», 2024. Вип. 1(75). С. 94–100. doi: 10.26906/SUNZ.2024.1.094.

2. . Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030. Statista. Telecommunications. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide.

3. Hou, X.; Li, Y.; Chen, M.; Wu, D.; Jin, D.; Chen, S. Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures. IEEE Trans. Veh. Technol. 2016, 65, 3860–3873.

4. ITU Internet Reports. The Internet of Things. ITU, November, 2005. URL: https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf.

5. Chiang, M.; Zhang, T. Fog and IoT: An overview of research opportunities. IEEE Internet Things J. 2016, 3, 854–864.

6. Kowalczyk A. European IoT Spending to Reach Nearly $227 Billion in 2023, Despite Ongoing Market Uncertainty, Says IDC. June 2023. URL: https://www.idc.com/getdoc.jsp?containerId=prEUR250941023&utm_medium=embedd&utm_campaign=idc_embedd&utm_source=referral.

7. Bonomi F., Milito R., Zhu J., et al. Fog Computing and Its Role in the Internet of Things Proceedings of the 1st ACM Mobile Cloud Computing Workshop, MCC'12 (Helsinki, Finland, August, 17, 2012). ACM Press, 2012. P. 13–15. DOI: https://doi.org/10.1145/2342509.2342513.

8. Burton A. Houdini Procedural City Generator. October 2020. URL: https://80.lv/articles/houdini-procedural-city-generator-09.

9. Peralta, G.; Iglesias-Urkia, M.; Barcelo, M.; Gomez, R.; Moran, A.;

Bilbao, J. Fog computing based efficient IoT scheme for the Industry 4.0. In Proceedings of the 2017 IEEE Int. Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics, San Sebastian, Spain, 24–26 May 2017; pp. 1–6.

10. Dastjerdi A.V., Gupta H., Calheiros R.N., Ghosh S.K., Buyya R. Fog computing: Principles, architectures, and applications. In: Buyya R., Dastjerdi A. (ed.) Internet of Things: Principle & Paradigms. Morgan Kaufmann, Burlington, Massachusetts, USA; 2016.

11. Lavassani, M.; Forsström, S.; Jennehag, U.; Zhang, T. Combining fog computing with sensor mote machine learning for industrial IoT. Sensors. 2018, 18, 1532.

12. Arkian H.R., Diyanat A., Pourkhalili A. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. Journal of Network and Computer Applications. 2017; 82:152-165. (In Eng.) DOI: https://doi.org/10.1016/j.jnca.2017.01.012.

13. Wen, Z.; Yang, R.; Garraghan, P.; Lin, T.; Xu, J.; Rovatsos, M. Fog orchestration for internet of things services. IEEE Internet Comput. 2017, 21, 16–24.