

***БАКАЛАВРСКА
РОБОТА***

БР.ІІ – 36.00.00.000 ІІЗ

Група ІІ-22-2

Дорожовець Арсен

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Дорожовець Арсен Ігорович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

БАКАЛАВРСЬКА РОБОТА

Програмне рішення для обробки поточкових даних у реальному часі

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Дорожовець А.І.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Саманів Любов Василівна, асистент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою

ІПЗ

доцент.

В.В. Бандура

“ ” 2026 р.

ЗАВДАННЯ НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Дорожовець Арсену Ігоровичу

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) "Програмне рішення для обробки поточкових даних у реальному часі"

керівник проекту (роботи) асист. Саманів Л.В.

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення

2. Аналіз вимог і постановка задачі

3. Проектування системи

4. Реалізація проекту

5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1 Загальний огляд виконання MapReduce (рис. 2.1, ст. 39)

2 Фреймворк Spark (рис. 2.2, ст. 41)

3 Архітектура Lambda (рис. 3.1, ст. 72)

4 Архітектура Карра (рис. 3.2, ст. 74)

5 Рівень обробки в режимі реального часу (рис. 3.5, ст. 83)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 2026 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.05.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	09.06.2026	виконано

Студент _____ Дорожовець А.І.
(підпис)

Керівник роботи _____ Саманів Л.В.
(підпис)

АНОТАЦІЯ

Дипломна робота містить 88 сторінок, 13 рисунків, 2 таблиці, список використаних джерел із 40 найменувань.

Метою роботи є дослідження методів та технологій обробки потокових даних у реальному часі та розробка програмного рішення.

Об'єкт дослідження: процеси обробки потокових даних у реальному часі.

Предмет дослідження: методи, моделі та інструменти розробки програмних систем для потокової обробки даних, включаючи архітектурні підходи, алгоритми обробки та механізми оптимізації продуктивності.

Результати дослідження: проведено аналіз концепції великих даних та потокової обробки, досліджено сучасні інструменти і платформи для обробки потокових даних, розглянуто архітектурні підходи до побудови систем реального часу та запропоновано власне програмне рішення.

У першому розділі розглянуто теоретичні основи великих даних, принципи потокової обробки та ключові вимоги до систем обробки даних у реальному часі.

У другому розділі досліджено методи та інструменти розробки потокових систем, включаючи сучасні платформи, методології проєктування та підходи до оптимізації використання ресурсів.

У третьому розділі розглянуто практичну реалізацію системи обробки потокових даних, проведено порівняльний аналіз існуючих рішень, досліджено архітектурні підходи та обґрунтовано запропоновану архітектуру.

Висновок: використання сучасних технологій потокової обробки дозволяє забезпечити високу продуктивність, масштабованість та надійність систем, що працюють із даними в реальному часі.

КЛЮЧОВІ СЛОВА: ПОТОКОВІ ДАНІ, BIG DATA, ОБРОБКА В РЕАЛЬНОМУ ЧАСІ, STREAM PROCESSING, АРХІТЕКТУРА СИСТЕМ, МАСШТАБОВАНІСТЬ, ПРОДУКТИВНІСТЬ, РОЗПОДІЛЕНІ СИСТЕМИ.

ANNOTATION

The thesis consists of 88 pages, 13 figures, 2 tables, and a reference list with 40 sources.

The aim of the work is to study methods and technologies for real-time stream data processing and to develop a software solution that ensures efficient, scalable, and reliable processing of streaming data.

Research object: processes of real-time stream data processing.

Research subject: methods, models, and tools for developing software systems for stream data processing, including architectural approaches, processing algorithms, and performance optimization techniques.

Research results: the concepts of Big Data and stream processing were analyzed, modern tools and platforms for stream data processing were studied, architectural approaches for building real-time systems were reviewed, and a software solution for stream data processing was proposed.

The first section describes the theoretical foundations of Big Data, principles of stream processing, and key requirements for real-time data processing systems.

The second section analyzes methods and tools for developing streaming systems, including modern platforms, design methodologies, and resource optimization approaches.

The third section covers the implementation of a stream data processing system, provides a comparative analysis of existing solutions, examines architectural approaches, and justifies the proposed system architecture.

Conclusion: the use of modern stream processing technologies ensures high performance, scalability, and reliability of real-time data systems and enables their application across various domains.

KEY WORDS: STREAM DATA, BIG DATA, REAL-TIME PROCESSING, STREAM PROCESSING, SYSTEM ARCHITECTURE, SCALABILITY, PERFORMANCE, DISTRIBUTED SYSTEMS.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОБРОБКИ ПОТОКОВИХ ДАНИХ..	9
1.1 Основи концепції великих даних (Big Data)	9
1.2 Поточкова обробка даних та її особливості	13
1.3 Вимоги, виклики та проблеми обробки поточкових даних у реальному часі.	18
1.4 Висновок по розділу	36
РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ СИСТЕМ ПОТОКОВОЇ ОБРОБКИ	37
2.1 Сучасні інструменти та платформи поточної обробки даних	37
2.2 Методологія проектування систем обробки поточкових даних	42
2.3 Методи оптимізації: розподіл ресурсів та планування обчислень у поточкових системах	46
2.4 Висновок по розділу	61
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО РІШЕННЯ	62
3.1 Порівняльний аналіз систем обробки даних у реальному часі	62
3.2 Архітектурні підходи до побудови систем реального часу	69
3.3 Розробка та обґрунтування запропонованої архітектури системи	78
3.4 Висновок по розділу	83
ВИСНОВКИ	84
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	86

					БР.ІП – 36.00.00.000 ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Програмне рішення для обробки поточкових даних у реальному часі Пояснювальна записка	Літ.	Арк.	Акрушів
Розроб.		Дорожовець А.І.						
Перевір.		Саманів Л.В.					7	
Реценз.		Вовк Р.Б.				ІФНТУНГ ІП-22-2		
Н. Контр.		Піх М.М.						
Затверд.		Бандура В. В.						

ВСТУП

У сучасному цифровому середовищі обсяг даних, що генеруються різноманітними інформаційними системами, стрімко зростає. Дані надходять у вигляді безперервних потоків із різних джерел: сенсорів, мобільних пристроїв, веб-додатків, фінансових систем та інтернету речей. У таких умовах традиційні підходи до пакетної обробки даних стають недостатньо ефективними, оскільки вони не забезпечують необхідної швидкості реагування та актуальності інформації. Це зумовлює необхідність розробки програмних рішень, здатних обробляти потокові дані у реальному часі з високою продуктивністю, масштабованістю та надійністю.

Актуальність теми полягає у зростаючій потребі організацій у швидкому аналізі даних для прийняття оперативних рішень. Сфери фінансів, телекомунікацій, електронної комерції, логістики та кібербезпеки потребують систем, які можуть обробляти великі обсяги інформації без затримок. Існуючі рішення часто мають обмеження щодо масштабованості, складності інтеграції або високих вимог до ресурсів. Тому розробка ефективного програмного рішення для обробки поточкових даних є важливим завданням сучасної програмної інженерії.

Метою роботи є дослідження методів обробки поточкових даних у реальному часі та розробка програмного рішення, яке забезпечує ефективну обробку, аналіз і передачу даних із мінімальними затримками.

Завданнями дослідження є аналіз теоретичних основ великих даних і потокової обробки, дослідження існуючих підходів і платформ для обробки даних у реальному часі, визначення вимог до таких систем, розробка архітектури програмного рішення, реалізація основних функціональних компонентів, а також оцінка ефективності запропонованого підходу.

Об'єктом дослідження є процеси обробки поточкових даних у реальному часі в інформаційних системах.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є методи, алгоритми, архітектурні підходи та програмні інструменти, що застосовуються для створення систем потокової обробки даних.

Методи дослідження включають аналіз наукових і технічних джерел, порівняльний аналіз існуючих технологій, моделювання архітектури системи, експериментальну реалізацію програмного рішення та оцінку його продуктивності.

У першому розділі роботи розглянуто теоретичні основи великих даних, принципи потокової обробки та ключові вимоги до систем реального часу. **У другому розділі** досліджено методи та інструменти розробки систем потокової обробки, включаючи сучасні платформи та підходи до оптимізації ресурсів. **У третьому розділі** представлено реалізацію програмного рішення, проведено аналіз існуючих систем і обґрунтовано запропоновану архітектуру.

Розроблене програмне рішення передбачає обробку поточкових даних у режимі реального часу, забезпечуючи високу швидкість обробки, масштабованість і надійність системи. Особлива увага приділяється ефективному розподілу ресурсів, зменшенню затримок та забезпеченню безперервності обробки даних. Очікується, що впровадження такого рішення дозволить підвищити ефективність інформаційних систем, забезпечити оперативний аналіз даних та покращити якість прийняття рішень.

Бакалаврська робота містить 88 сторінок, 13 рисунків, 2 таблиці, 3 розділи, список використаних джерел із 40 найменувань.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОБРОБКИ ПОТОКОВИХ ДАНИХ

1.1 Основи концепції великих даних (Big Data)

Цей розділ присвячено деяким основним концепціям, що використовуються у великих даних, включаючи вступ до великих даних, їх архітектуру, використання технології та концепції потоку великих даних.

Великі дані (Big data) – це нова концепція, яка була запроваджена через великий обсяг і складність даних, які стає важко обробляти за допомогою традиційних методів та інструментів баз даних. Згідно з Gartner [1], «Великі дані – це високооб’ємні, високошвидкісні та різноманітні інформаційні активи, які вимагають економічно ефективних, інноваційних форм обробки інформації для покращення розуміння та прийняття рішень». У 2010 році [2] визначили великі дані як «набори даних, які не можуть бути зібрані, керовані та оброблені звичайними комп’ютерами в прийнятному обсязі». [3] NIST зазначає, що «Великі дані означають дані, обсяг яких, швидкість збору або представлення даних обмежують можливості використання традиційних реляційних методів для проведення ефективного аналізу, або дані, які можуть бути ефективно оброблені за допомогою важливих технологій горизонтального масштабування». Характеристики великих даних узагальнено у п’яти V: Об’єм (Volume), Швидкість (Velocity), Різноманітність (Various), Достовірність (Veracity) та Цінність (Value). Об’єм представляє розмір або кількість даних від терабайта до йотабайта (This number - нуклеотидний переказ). Ми говоримо про масштабну еволюцію, адже з 2005 року обсяг даних був обмежений 0,1 ЗБ, а у 2020 році він може досягти 40 ЗБ і більше [4]. Швидкість означає, що дані необхідно обробляти та аналізувати швидко з точки зору швидкості їх збору. Різноманітність вказує на те, що дані не є одного типу, що дозволяє нам використовувати різні типи даних: структуровані, напівструктуровані та неструктуровані. Достовірність спрямована на впевненість у даних, на яких

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

базуються рішення. І останнє, але не менш важливе: цінність, що означає, що системи повинні бути розроблені не лише для ефективної обробки величезних обсягів даних, але й для фільтрації найважливіших даних з усіх зібраних даних.

Згідно з раніше наведеними визначеннями, можна сказати, що великі дані (Big Data) – це абстрактне поняття, яке дозволяє виділити такі проблеми: як зберігати, аналізувати, обробляти та витягувати потрібну інформацію з різноманітних наборів даних, швидко згенерованих та у вигляді потоку даних.

Потокова обробка – це технологія, яка дозволяє збирати, інтегрувати, аналізувати та візуалізувати дані в режимі реального часу під час їх створення[8]. Рішення для потокової обробки розроблені для обробки великих даних у режимі реального часу з високомасштабованою, високодоступною та високовідмовостійкою архітектурою. Це дозволяє аналізувати дані в русі[5]. Метою обробки в реальному часі є надання рішень, які можуть обробляти безперервний нескінченний потік даних, інтегрованих як з реальних, так і з історичних джерел, дуже швидко та інтерактивно.

прикладних сценаріїв, де своєчасні бізнес-рішення є надзвичайно важливими. Обробка безперервних потоків великих даних для аналітики в реальному часі є дуже складною під час впровадження етапу ETL для сховищ даних (DWH) або інших застосувань великих даних через природу великих даних щодо обсягу, різноманітності, швидкості, волатильності, мінливості, достовірності та цінності [6], [7].

Безперервне постачання великих даних називається потоком. Потік даних може генеруватися з одного або кількох джерел великих даних, показаних на рисунку 1.1. Широка категорія програм бере участь у безперервному генеруванні масивних даних. Аналіз цих потоків є великою проблемою, оскільки зібрані дані є неоднорідними та можуть мати будь-яку форму/природу, тобто структуровані, напів/неструктуровані, симетричні або асиметричні. Велика частина масивних даних, отриманих в результаті потоку в реальному часі, потребує обробки/аналізу в реальному часі, оскільки цінність даних розглядається в їх свіжості. Обробка потоку в реальному часі (що називається обробкою масивних

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

даних в пам'яті) зазвичай може бути потрібна у двох типах програмних областей: по-перше, де систематизуються дані.

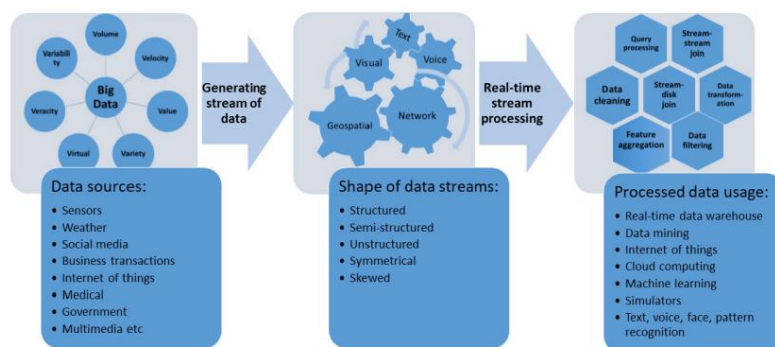


Рисунок 1.1 - Компоненти, що беруть участь в обробці потоків даних у реальному часі.

Для прийняття рішення потрібне (сховище даних у реальному часі), а також те, де потрібно генерувати певну реакцію в реальному часі, особливо при низькій затримці. Кілька застосувань другого типу також перераховані на рисунку. 1.1. Перед завантаженням у ці програми потоки необхідно обробити, щоб забезпечити якість даних. Це вимагає обробки/аналізу потоків у режимі реального часу.

Для обробки потоків потрібне кілька операцій, включаючи очищення даних, обробку запитів, об'єднання потоків, об'єднання потоків і дисків, перетворення даних тощо. На сьогоднішній день розроблено широку категорію підходів, інструментів і технологій для подолання проблем обробки потоків. Ці підходи, можливо, мають справу з різними формами та моделями зберігання даних, застосовуючи кілька операцій до цих потоків.

Для вирішення проблем обробки потоків у реальному часі наразі розроблено різні підходи: алгоритми об'єднання потоків між собою, алгоритми об'єднання потоків та дисків зі зменшеною затримкою/асиметрією даних, розподілений потоковий ETL, Mesa DWH, фреймворк потокової обробки, розподілена обробка об'єднань, сенсорні мережі, відстеження та моніторинг об'єктів, а також обробка запитів на багаторазове об'єднання у хмарному DWH.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Крім того, оброблений вихідний сигнал повинен забезпечуватися з низькою затримкою, обмеженими ресурсами, точністю та протягом кількох секунд, щоб забезпечити реакцію та прийняття рішень у реальному часі. Глибина проблем, спрямованих на обробку потоків у реальному часі, призвела до такої кількості досліджень, що необхідно провести систематичний аналіз запропонованих рішень.

Метою цього дослідження є представлення розширеного систематичного огляду літератури (СОЛ), щоб зібрати різні підходи до обробки потоків у реальному часі для всіх можливих областей застосування, зокрема, для сховищ даних у реальному часі. Ми завершили 74 дослідження з 667 414 статей для цього огляду на основі критеріїв оцінки якості. Новизна нашого СОЛ полягає в тому, що він пропонує нові критерії класифікації, дослідження обробки потоків у реальному часі, спрямовані на канали, проблеми потокового передавання сховищ даних/великих даних у реальному часі, підходи до вирішення цих проблем після емпіричної перевірки досліджень.

1.2 Потокова обробка даних та її особливості

Централізовані обчислювальні системи існують у технологічних обчисленнях вже багато років. У таких системах один центральний комп'ютер керує периферійними пристроями та виконує складні обчислення. Централізовані обчислювальні системи потребують дорогого обладнання для обробки величезних обсягів даних та одночасної підтримки кількох користувачів онлайн. За цих обставин виникли хмарні та розподілені обчислювальні системи, що використовують технологію паралельної обробки. Користувачі можуть спільно використовувати обчислювальні ресурси, які можна віртуалізувати та динамічно розподіляти в хмарі, що зазвичай характеризується масштабованими та еластичними ресурсами.

Сьогодні розподілені хмарні обчислення приваблюють численних клієнтів та постачальників послуг. Важливість систем обробки потоків зростає,

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

оскільки все більше сучасних програм накладають жорсткіші часові обмеження на певну подію. Результуючий аналіз надає інформацію, яка може забезпечити компаніям уявлення про багато аспектів їхнього бізнесу та активності клієнтів, а також дозволяє їм швидко реагувати на нові ситуації. Значна частина даних, які компанії отримують у режимі реального часу, є більш цінною в момент їх надходження. Наприклад, компанії можуть відстежувати зміни в уподобаннях громадськості щодо своїх брендів та продуктів, постійно аналізуючи потоки в соціальних мережах. Немає сенсу виявляти потенційного покупця після того, як користувач залишає сайт електронної комерції, або виявляти шахрайство з кредитною карткою після завершення транзакції [8]. Наприклад, eBay виявляє шахрайство з використанням PayPal, аналізуючи 5 мільйонів транзакцій у режимі реального часу щодня. Машинне навчання онлайн також є ще одним перспективним варіантом використання. Існує безліч варіантів використання, що роблять необхідність обробки потоків надзвичайно важливою. Коли дані надходять швидко та потребують обробки з обмеженнями в режимі реального часу, система обробки потоків зазвичай розміщується поверх хмарної інфраструктури. Обробка та аналіз потоків даних у режимі реального часу – це завдання, з якими традиційні середовища сховищ даних не можуть легко впоратися через високу затримку та вартість [9].

Система потокової обробки або система керування потоками даних (СКПД) розроблена для обробки потоків даних та керування безперервними запитами. Вона виконує безперервні запити, які не лише виконуються один раз, але й безперервно виконуються до їх явного видалення. Вона видає результати, якщо нові дані надходять у систему, і дані обробляються на льоту без необхідності їх зберігання. Дані зазвичай зберігаються після обробки. Системи потокової обробки відрізняються від систем пакетної обробки через вимогу обробки даних у реальному часі. Термін «система обробки в реальному часі» стосується системи, яка реагує в межах «реальних» часових термінів. Вона гарантує, що певний процес буде виконано протягом заданого періоду, можливо, кількох секунд, залежно від обмежень якості обслуговування. Термін «реальний

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

час» дещо зайвий, але багато систем використовують цей термін для опису себе як систем з низькою затримкою. Для цих нових вимог були запропоновані складні та гнучкі системи.

Потокові програми можна представити за допомогою орієнтованого ациклічного графа (DAG), де вершини – це оператори, а ребра – канали для потоку даних між операторами [10]. Обробка може проходити через оператори в певному порядку, де оператори можуть бути пов'язані разом, але обробка ніколи не повинна повертатися до попередньої точки в графі. У потоковій обробці використовуються дві моделі виконання [11]:

(1) Підхід потоку даних (Stream-Dataflow), де застосунок розглядається як граф потоку даних з операторами та залежностями даних між ними (іноді його називають операторним підходом). Завдання інкапсулює логіку попередньо визначеного оператора, такого як фільтр, вікно, агрегація або об'єднання, або навіть підпрограми з логікою, заданою користувачем. Потік даних між двома операторами являє собою нескінченну послідовність даних, створених завданням, яка доступна для подальшого споживання. Дані передаються та споживаються в довільному порядку між паралельними завданнями, і в результаті бракує грубої одиниці для обробки транзакцій. Все автоматично конвеєризується.

(2) Мікропакетний підхід, який пропонує рішення для обробки потоків даних у системах пакетної обробки. За допомогою мікропакетування ми можемо розглядати потокове обчислення як послідовність перетворень на обмежених множинах шляхом дискретизації розподіленого потоку даних на пакети, а потім послідовного планування цих пакетів у кластері робочих вузлів.

Розвиток систем обробки потоків у реальному часі в хмарі був відносно повільним, але сьогодні пропонується кілька рішень. Apache Storm [12] - це технологія розподілених обчислень у реальному часі для безперервної обробки поточкових повідомлень. Окремі логічні процесори з'єднані як конвеєр для вираження серії перетворень та надання можливостей для одночасної обробки. Heron - це механізм обробки для потокової передачі та даних у реальному часі в

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

масштабі, розроблений у Twitter як заміна Apache Storm. Spark Streaming [13] — це ще одне рішення, яке спрощує створення поточкових додатків за допомогою підходу мікропакетування. Ідея полягає в тому, щоб обробляти так само, як і пакетна обробка, але зберігаючи розміри пакетів дуже малими. Apache Samza [14] — це розподілений фреймворк для обробки потоків, який надає простий API, порівнянний з MapReduce. Існують також комерційні рішення, такі як Amazon Kinesis [15], повністю керований сервіс для обробки поточкових даних у реальному часі у великих масштабах, та IBM InfoSphere Streams [16]. Дослідження та розробки в галузі обробки потоків продовжуються та набувають великого значення в згаданому контексті Інтернету речей. У наступному розділі ми окреслимо проблеми та вимоги, яким повинні відповідати системи обробки потоків, щоб досягти успіху в програмах обробки потоків у реальному часі. Наша робота здебільшого спирається на аналіз. Також ми представимо механізми, що використовуються в епоху великих даних для вирішення вищезгаданих проблем.

Наш підхід до збору даних та критеріїв відбору

Щоб відповісти на дослідницькі питання, викладені у вступі, ми провели огляд літератури за період з січня 2010 року по березень 2018 року. Вибір періоду огляду був практичним і враховував той факт, що великі дані — це досить нове явище. Є лише один виняток. Стаття «Вимоги до обробки потоків у реальному часі» датується 2005 роком, але цінується як ключовий прототип, який досі характеризує сучасні системи обробки потоків у галузі великих даних, і ми спробували представити його, адаптувавши до цього контексту.

Бібліографічний аналіз у досліджуваній галузі включав три етапи:

- (1) Збірка пов'язаних робіт
- (2) Фільтрація релевантної роботи
- (3) Детальний огляд та аналіз сучасного стану справ

На першому кроці було виконано пошук на основі ключових слів для доповідей конференцій, статей з журналів, технічних звітів з використанням таких баз даних: IEEE Xplore, Science Direct, ACM та CiteSeer. Ці бази даних були обрані через їхнє широке охоплення відповідної літератури та розширені

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

бібліометричні функції, такі як пропонування пов'язаної літератури або цитат. Були використані ключові слова, такі як «потокова обробка», «реальний час», «великі дані», «хмарні обчислення», «розподілена обробка» у можливих комбінаціях. Після вивчення літератури з сучасного стану техніки ми перейшли до бібліографічного пошуку на основі комбінацій ключових слів, використовуючи пошукову систему Google Scholar, яка індексує повний текст та метадані наукової літератури, для додавання більш конкретної інформації. Такі терміни, як «вікна», «водяні знаки», «відновлення» та «керування станом», були додані до нашого «пулу» ключових слів, щоб дослідити механізми, що використовуються в еру великих даних для обробки вимог до потокової обробки.

Щоб вибрати основні рішення, що задовольняють вимоги потокової обробки, та провести порівняння між ними, ми провели опитування, представлених у розділі «Супутні роботи». Нам потрібно було вибрати підмножину сучасних систем, які здатні задовольнити вимоги та використовувати для цього найбільш репрезентативні механізми. Ми також хотіли зосередитися на рішеннях з відкритим кодом з активною підтримкою спільноти, що також було б корисним для другої частини нашої роботи, що стосується планування завдань.

Спочатку ми обрали двох основних представників двох доступних моделей виконання для обробки потоків, щоб зрозуміти, як вони працюють; Apache Spark Streaming [17] та Storm [18]. Spark є найактивнішим проектом з точки зору кількості спільноти та використовує техніку мікропакетів, тоді як Storm використовує модель поточкових обчислень (Heron [19] є покращеною версією Storm, але це не такий зрілий проект, і ми не змогли б знайти достатньо робіт, присвячених методам планування на ньому). Flink [20] може підтримувати ті ж можливості, що й Spark, але працює як власний поточковий рушій і створює кілька проблем для Spark в обробці потоків (наприклад, у разі відновлення та затримки). Таким чином, його було додано до нашого огляду. Flink також, здається, має більше можливостей, ніж Storm. Тим не менш, Storm має більшу спільноту, і оскільки це зрілий проект, існує велика література про методи

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

планування на ньому. Samza [21] - це розподілений фреймворк для обробки потоків, який цікавий головним чином завдяки можливостям, що виникають завдяки його розумній координації з брокером повідомлень Kafka [22]. Це забезпечує кілька переваг, таких як автоматичне керування потоком, вбудоване в систему. В результаті ми отримали 4 відомі фреймворки для обробки потоків, а саме Apache Storm, Spark, Samza та Flink.

Назви фреймворків, що розглядаються: «Spark Streaming», «Storm», «Samza» та «Flink», були додані до наших ключових слів «pool» та використані в кількох комбінаціях, щоб перевірити, як досліджувані проблеми вирішуються відомими фреймворками, надати огляд та порівняти їх на основі списку критеріїв. У розділі ключові слова «планування завдань», «планування потоків» та «розподіл ресурсів» використовувалися в поєднанні з більшістю вищезгаданих ключових слів для пошуку підходів до планування, включених до існуючих фреймворків або запропонованих з новими можливими архітектурами.

На другому етапі ми обмежили наше дослідження низкою статей найвищої якості, які вважаються найважливішими ресурсами в цій галузі. Для цього ми зосередилися на статтях з вищим впливом (тобто з більшою кількістю цитувань). Виняток було зроблено для статей, опублікованих у 2016-2018 роках, куди також приймалися статті з невеликою кількістю цитувань. Таким чином, було уникнуто сірої літератури, хоча й досить інформативної для умов індустрії великих даних. Тим не менш, офіційна документація кожної фреймворку доповнювала пов'язані статті, які шукалися кілька разів.

Ми також відфільтрували статті, що стосуються питань, що цікавлять нас (наприклад, розподілена обробка, потокова обробка, масштабованість тощо) у хмарі, які фактично не використовували великі дані та їхні притаманні характеристики (тобто обсяг, швидкість, різноманітність). Нам також довелося відфільтрувати статті, що не стосуються потокової обробки великих даних, оскільки література набагато багатша, коли йдеться про обробку вже збережених великих даних та покращення планування Hadoop MapReduce [23]. Таким чином, кількість статей, кваліфікованих відповідно до наших обмежень, була суворо

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

обмежена.

На останньому етапі ми детально вивчили відібрану літературу, щоб витягти інформацію, що стосується наших дослідницьких питань. Додаткова література, яка не була визначена на першому етапі, також була знайдена на цьому етапі, якщо на неї було посилання у розділі «пов'язані роботи». Такий підхід «снігової кулі» призвів до появи додаткових статей та веб-пунктів, з яких також було витягнуто відповідну інформацію. Витягнуту інформацію було проаналізовано та синтезовано.

1.3 Вимоги, виклики та проблеми обробки поточкових даних у реальному часі

Представлено вимоги до обробки потоків даних у реальному часі, щоб надати загальне керівництво щодо того, на що слід звертати увагу під час оцінки рішень для обробки потоків. Найбільш поширені з них можуть характеризувати доступні системи обробки потоків для великих даних способами, які представлені нижче, і можуть бути додатково вдосконалені. Система обробки потоків у реальному часі повинна:

(1) «Обробляти повідомлення в потоці без необхідності їх зберігання для виконання будь-якої операції чи послідовності операцій».

Все більше сучасних програм накладають жорсткіші часові обмеження на певну подію. Вимоги, пов'язані з продуктивністю, такі як затримка та пропускну здатність, надзвичайно важливі в поточкових програмах. Щоб відповідати цим вимогам, обробка повинна виконуватися без дорогого зберігання. MapReduce може обробляти великі набори даних, але працює з використанням постійного сховища [24]. Таким чином, він не працює, коли справа доходить до обробки даних у реальному часі, оскільки був розроблений для пакетної обробки великих обсягів даних. Обчислення в оперативній пам'яті пропонують рішення цієї проблеми та базуються на використанні розподіленої системи основної пам'яті для зберігання та обробки великих даних у реальному часі. Оперативна пам'ять

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечує вищу пропускну здатність та кращу затримку порівняно з жорстким диском. Навіть якщо фреймворк використовує пам'ять для кешування часто використовуваних даних, загальна продуктивність виконання завдання значно покращиться [25]. Незважаючи на зниження ціни на пам'ять, використання великих обсягів оперативної пам'яті для запуску всього, що знаходиться в пам'яті, може бути дорогим, тому необхідні належні механізми для ефективної обробки цього завдання.

Spark Streaming – це система, яка виникла для ефективного забезпечення обчислень в пам'яті. Вона обробляє всі дані в пам'яті, взаємодіючи лише з шаром зберігання, щоб спочатку завантажити дані в пам'ять, а потім зберегти кінцеві результати. Для реалізації обчислень в пам'яті Spark використовує модель під назвою Resilient Distributed Datasets або RDDs, її абстракцію в пам'яті для роботи з даними. Flink також пропонує обчислення в пам'яті. Її ядро побудовано на механізмі потоку даних, фундаментальною функціональністю якого є конвеєрна обробка, тобто всі завдання повинні бути одночасно онлайн, щоб дані могли проходити через завдання незалежно від вузла, на якому може знаходитися завдання [26]. Storm також працює в пам'яті для обробки великих даних зі швидкістю в пам'яті. Пам'ять є ресурсом, що має вирішальне значення для DSMS. Оскільки обчислення в пам'яті – це не лише зберігання, але й обробка великих даних у режимі реального часу, такі проблеми, як ефективне планування та відповідне переміщення коду до даних, а не даних до коду, є питаннями, з якими вона має мати справу [27].

(2) «Підтримка належної мови запитів за допомогою розширюваних потоково-орієнтованих примітивів».

Системи потокової обробки отримують вхідні потоки з одного або кількох джерел та організовують обчислення в орієнтований граф операторів явно або неявно. У першому випадку системи називаються композиційними. Вони пропонують основні структурні блоки для створення власних операторів та топологій. Користувач повинен сам реалізувати всю логіку, а оператори визначаються як реалізації класів. В останньому випадку системи називаються

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

декларативними, і розробникам надаються мови програмування високого рівня, які автоматично перетворюються системою на граф операторів [28].

Для виявлення подій, що цікавлять, або обчислення аналітики в режимі реального часу необхідні механізми запитів. Мови загального призначення, такі як Java, використовувалися як інструменти програмування в потокових додатках, але використання низькорівневих схем програмування призводить до тривалих циклів розробки та високих витрат на обслуговування.

Використання системи потокової обробки без підтримки мов запитів, подібних до SQL, вимагає ґрунтовних знань імперативного стилю програмування та розподілених систем для ефективного її використання. Підтримка мов безперервних запитів, таких як SQL, або SQL з потоковими розширеннями може допомогти в цьому напрямку. SQL залишається надійною мовою запитів з високою продуктивністю для аналітики в реальному часі [29], але має обмеження при роботі з величезними обсягами даних. Системи можуть використовувати SQL-подібні мови, які представляють обробку як запити, що багаторазово та безперервно оцінюються в міру появи нових даних. З'являються нові проекти, такі як SQLStream та Apache Calcite, для виконання запитів над великими даними, використовуючи набір потокових розширень стандартного SQL. Було докладено низку зусиль щодо використання різних мов у системах обробки подій, щоб дозволити користувачам висловлювати свої інтереси. Різні проекти обрали різні напрямки досліджень.

Існує концепція життєво важливого значення, коли йдеться про аналіз потоків даних. Вважається, що набір даних нескінченний, що створює проблеми під час обробки. Не всі оператори можна обчислювати над потоками. Механізм обробки потоків повинен знати, коли завершити операцію над потоком даних і вивести відповідь. Традиційні методи мають перевагу у знанні загального розміру набору. Вибірка – це інструмент, який вирішує цю проблему. Віконна обробка – це метод вибірки, що визначає область дії операції, і є широко використовуваним підходом у потоковій обробці. Необмежений потік даних розбивається на скінченні множини, які називаються вікнами, на основі заданих

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

критеріїв, таких як час. Вікно можна концептуалізувати як таблицю в пам'яті, де події додаються та видаляються, а обчислення обчислюються для кожного вікна подій. У [30] та [31] наведено детальний аналіз методів віконної обробки. Існують фреймворки для обробки потоків, які пропонують лише основи, такі як Storm, Samza та Spark Streaming, які підтримують лише часові вікна з розміром і слайдом, кратними розміру пакета. Навпаки, Spark Structured Streaming, новий компонент у Spark 2.0, пропонує більше можливостей роботи з вікнами. Flink також пропонує широкий спектр функцій роботи з вікнами. Перегортання вікон, ковзні вікна, вікна сеансу та глобальні вікна попередньо реалізовані, тоді як користувачі також можуть реалізувати свої власні вікна. Поняття роботи з вікнами та часом, а також семантика роботи з вікнами, що використовується відомими фреймворками для обробки потоків.

(3) «Використовуйте механізми для забезпечення стійкості до недосконалостей потоку, включаючи дані, що не впорядковані або відсутні, що зазвичай присутні в реальних потоках даних».

Неповні набори даних, пошкоджені дані, наявність викидів або упереджень у навчанні впливають на точність аналізу. Відсутність даних – це явище, неминуче в розподілених комунікаційних середовищах. Втрата передачі через розрив зв'язку між датчиками або несправність датчика призводить до відсутності значень у даних. Аварія датчика може призвести до постійної відсутності значень, тоді як тимчасове відключення або затримка в мережі призводить до тимчасової втрати, оскільки дані можуть надходити за короткий час.

Оцінка якості даних вимагає значної участі людини та експертних знань, але коли йдеться про великі обсяги неструктурованих даних, навіть напівавтоматизовані підходи непрактичні. Потокова передача даних та обробка в режимі реального часу переважають якість даних, що робить управління якістю даних більш важливим, ніж будь-коли. Наскільки нам відомо, вищезазначені проблеми досі не були ефективно вирішені, але інтерес до цього напрямку зростає. Такі методи, як виявлення викидів, перетворення даних, зменшення

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

розмірності, перехресна перевірка та бутстрепінг, є цінними інструментами в управлінні якістю даних, але донедавна дослідження якості даних в основному зосереджувалися на структурованих даних, що зберігаються в реляційних базах даних та файлових системах.

Потокові дані також зазвичай не впорядковані в часі. Дані, впорядковані за часом подій, є рідкістю в багатьох реальних розподілених джерелах вхідних даних. Наприклад, під час отримання потоку показань датчика Інтернету речей деякі пристрої можуть бути офлайн і надсилати дані через деякий час. Залишати систему з суворими часовими обмеженнями в очікуванні не є привабливим рішенням. Зберігання всіх вікон вічно також споживатиме всю доступну пам'ять. Коли справа доходить до обробки в реальному часі, має бути механізм, який дозволяє вікнам залишатися відкритими протягом певного часу. Водяні знаки є таким механізмом. Вони дозволяють поточним системам видавати своєчасні, правильні результати під час обробки даних поза порядком. Вони використовуються як евристика, припускаючи, що всі події до певного часу були дотримані. Більшість відомих систем обробки потоків (наприклад, Spark Structured Streaming, Storm, Samza, Flink тощо) реалізують методи водяних знаків.

(4) «Переконайтеся, що додатки працюють та доступні, а цілісність даних має підтримуватися постійно, незважаючи на збої. Система повинна мати можливість ефективно зберігати, отримувати доступ до інформації про стан та поєднувати її з прямими трансляціями».

Збої в мережі, брак ресурсів та помилки мережевого програмного забезпечення можуть спричинити ще більше проблем у великомасштабних розподілених обчисленнях. Коли великі набори таких компонентів працюють разом, існує висока ймовірність того, що принаймні один компонент може вийти з ладу в певний момент часу. Майже всі системи потокової обробки в індустрії великих даних забезпечують можливість автоматичного відновлення після збоїв. Було розроблено кілька методів для достатньо швидкого відновлення, щоб нормальна обробка могла продовжуватися з мінімальним впливом на загальну

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

продуктивність. Ми розрізняємо три основні категорії відновлення в потоковій обробці: точне відновлення, відновлення з відкатом та відновлення GАР. Точне відновлення не дає жодних доказів подальшого збою, але спостерігається збільшення затримки. При відновленні з відкатом вихідні дані, отримані після збою, «еквівалентні», але не обов'язково такі ж, як вихідні дані виконання без збою. Наприклад, інформація може оброблятися більше одного разу, коли трапляється збій. При відновленні GАР очікується втрата інформації на користь зменшення часу відновлення та накладних витрат часу виконання.

Storm та Samza використовують методи резервного копіювання вище за течією, які забезпечують відновлення після відкату. Загалом кажучи, система потокової обробки може використовувати метод резервного копіювання вище за течією, щоб уникнути будь-яких накладних витрат на контрольні точки, наприклад, на дискові операції вводу/виводу та структури даних (контрольні точки також доступні в обох вищезгаданих системах). Однак, коли справа доходить до стану, Storm вимагає від користувача ручного керування відновленням стану. Apache Zookeeper має використовуватися для підтримки стану кластера, тоді як Samza робить зміни стану відмовостійкими, моделюючи їх як журнал виводу (commitlog або changelog) до теми Kafka [32]. Системи, такі як Spark Streaming, які працюють з пакетами, зазвичай повторно виконують необхідні обчислення у разі збоїв, даючи той самий результат незалежно від них через RDD, який можна переобчислити детерміновано. RDD є незмінними, що означає, що жоден робочий вузол не може їх змінювати, він може лише обробити їх та вивести деякі результати. Кожен RDD може простежити своє походження через батьківські RDD і, зрештою, до даних на диску. Flink реалізує різні механізми відновлення; відкат/перезапуск для скінченних потоків та розподілене створення знімків для нескінченних потоків. Механізм контрольних точок Flink зберігає узгоджені знімки розподіленого потоку даних та стану оператора. Ці методи відновлення після збоїв призводять до різних гарантій обробки для кожної системи (і забезпечують відмовостійкість по-різному).

(5) «Мати можливість розподіляти свою обробку між кількома

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

процесорами та машинами для досягнення поступової масштабованості. В ідеалі, розподіл має бути автоматичним та прозорим».

Потоки не мають попередньо визначеного терміну служби, надходять з непередбаченою швидкістю та є об'ємними. Якщо ними не керувати ретельно, затримки обробки можуть стати неприйнятними та призвести до довгих черг на вузлі обробки, переповнення буфера та вичерпання пам'яті. Кількість обчислень потоків набагато більша, ніж кількість машин, доступних для обробки. Кілька обчислень для обробки потоків слід чергувати на одній машині, щоб зменшити кількість необхідних з'єднань та забезпечити високу пропускну здатність і підвищену продуктивність. З іншого боку, якщо завдання щільно розподілені по вузлах, відмова вузла призведе до зупинки більшої частини програми. Слід уникати перевантажених і недовикористаних машин, оскільки дисбаланс погіршує продуктивність системи.

Послідовного виконання алгоритмів недостатньо. Алгоритми повинні виконуватися паралельно з потоками даних, також розподіленими на багато розподілених обчислювальних блоків. Більше того, на відміну від обчислень без збереження стану, обчислення зі збереженням стану не можна просто реплікувати на кількох машинах з потоками даних, обробленими, наприклад, паралельно, простим циклічним алгоритмом. Горизонтальне масштабування вимагає адаптації графіка елементів обробки, експорту та збереження стану операторів для реплікації, забезпечення відмовостійкості та міграції [33]. Оскільки запити на потокову обробку часто розглядаються як тривалі, які не можна перезапустити без втрати даних, початкове призначення завдань, де елементи обробки розгортаються на доступних обчислювальних ресурсах, стає більш критичним, ніж в інших системах. Обробка повинна бути ретельно організована на наборі машин, що робить планування завдань статичним або динамічним у масштабі на наборі машин складним завданням. Це спонукало нас дослідити методи планування в еру потокової передачі великих.

(6) «Мати високооптимізований механізм виконання з мінімальними накладними витратами для забезпечення реагування в режимі реального часу для

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

великогабаритних додатків»

Вищезгадані правила самі по собі не мають сенсу, якщо програма не може обробляти великі обсяги поточкових даних з дуже низькою затримкою та високою пропускною здатністю, щоб задовольнити потреби реального часу. Здатність обробляти великі обсяги даних на льоту, щойно вони стають доступними, є фундаментальною вимогою в сучасних інформаційних системах. Кількість доступних механізмів обробки потоків швидко зростає. Однак кожен механізм визначає власну модель обробки та семантику виконання, які впливають на його продуктивність. Наприклад, Apache Storm використовує модель виконання потоку даних, де потоки даних обробляються кортежем.

за допомогою кортежу на безперервних операторах. З іншого боку, потокове передавання Spark створює невеликі пакети поточкових даних та виконує їх на основі свого механізму пакетної обробки. Щоб виконати цю вимогу, вкрай важливо, щоб будь-який користувач із поточковим застосунком з великим обсягом даних ретельно вивчив надані рішення, щоб вибрати те, яке краще відповідає його випадку використання та вимогам.

Беручи до уваги вищезазначені вимоги, природа потоків створює кілька проблем обробки. Питання продуктивності та гарантії обробки повідомлень, здається, виходять на перший план при виборі фреймворку для конкретного застосування. Ми збираємося далі розглянути вищезгадані DSMS, головним чином з огляду на ці питання, щоб додатково перевірити та порівняти їхні можливості та результати їхніх вбудованих механізмів. Ефективне планування завдань сильно впливає як на обробку відмовостійкості, так і на продуктивність [34]. Більшість поточкових систем дозволяють користувачеві вказувати власне планування завдань, і інтерес до цього аспекту широко зростає, але ще багато роботи потрібно зробити. Це викликало наш інтерес до вивчення проблеми розміщення завдань та розподілу ресурсів у розподіленій системі.

Доступні фреймворки мають кілька відмінностей у своїй архітектурі та моделі обробки, яку вони використовують. Вибір системи, яка може гарантувати відмовостійку, високопродуктивну обробку потоку на основі потреб

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

користувача, є досить складним. Розглянуті механізми для обробки потоку забезпечують різні функціональні характеристики для кожної DSMS. У цьому розділі ми розділимо функціональність потокового механізму головним чином на питання в контексті продуктивності та відмовостійкості для оцінки вибраних DSMS. Для визначення переліку критеріїв, що використовуються для наших цілей, також були проведені консультації з відповідними дослідженнями.

Усі поточкові рішення, розглянуті в цьому розділі, використовують паралельну розподілену архітектуру, яка дозволяє розподіляти потоки даних та розпаралелювати їх між кластером машин. Також розглядаються такі атрибути:

Модель обробки: Вибір моделі обробки для системи варіюється від пакетної обробки та мікропакетної обробки до потокової обробки кортеж за кортежем. Системи пакетної обробки, такі як MapReduce, нас не цікавлять у цьому розділі.

Потоковий примітив: стосується основної структури даних у поточковій системі. Ці системи використовують різні терміни для позначення таких понять. Затримка: стосується часу, що минув від моменту подання завдання до отримання першої відповіді.

Керування станом: Поточкові обчислення можуть бути як без збереження стану, так і зі збереженням стану. «Програма без збереження стану розглядає кожен окрему подію та створює певний вихідний сигнал на основі цієї останньої події». Наприклад, потокова програма може отримувати дані про трафік та піднімати сповіщення у разі порушень правил дорожнього руху. «Програма зі збереженням стану створює вихідний сигнал на основі кількох подій, взятих разом». Усі види кінцевих автоматів, що використовуються для обробки складних подій (CER), є збереженими станом. Наприклад, створення сповіщення після отримання двох порушень правил дорожнього руху, які відрізняються менш ніж на 5 хвилин, є обчисленням зі збереженням стану. Розглянуті фреймворки використовують різні стратегії для зберігання станів або можуть не зберігати стан.

Пропускна здатність: стосується середньої кількості завдань, робіт або

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

операцій за одиницю часу.

Гарантія доставки: стосується «рівня правильності» результатів, отриманих після збою та успішного відновлення системи, порівняно з тим, якими були б результати без будь-яких збоїв. Системи потокової обробки характеризуються такими трьома семантиками [35]:

- *доставка «не більше одного разу»*, яка втрачає повідомлення у випадку їх неправильної обробки або у разі збою процесора. Зазвичай це найменш бажаний результат, оскільки повідомлення можуть бути втрачені.

- *доставка принаймні один раз*, яка відстежує, чи кожен вхідний параметр був успішно оброблений протягом заданого тайм-ауту. Таким чином гарантується повторна доставка та повторна обробка повідомлень після невдачі. Якщо завдання завершується невдачею, жодні повідомлення не втрачаються, але деякі повідомлення можуть бути доставлені повторно. Якщо вплив повідомлення на стан є ідемпотентним, жодних проблем не виникає, якщо одне й те саме повідомлення обробляється більше одного разу. Дублікат оновлення не змінить результат. Однак для неідемпотентних операцій, таких як підрахунок, гарантії доставки принаймні один раз можуть давати неправильні результати. Цей підхід достатньо хороший для багатьох випадків використання, але він може спричинити дублікати.

- *Семантика «точно один раз»*, яка використовує той самий механізм виявлення помилок, що й режим «принаймні один раз». Повідомлення фактично обробляються принаймні один раз, але дублікатів можна уникнути за допомогою різних методів. Такі системи гарантують, що кінцевий результат буде точно таким самим, як і у сценарії без помилок. Це найбажаніша функція, але її важко гарантувати у всіх випадках.

Модель програмування: Системи можуть бути композиційними, коли користувачам доводиться явно моделювати потокову програму як граф, або декларативними, коли користувачам надаються абстракції вищого рівня.

Мови API: стосується мов, які можна використовувати для розробки застосунку для цього фреймворку.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

Співробітники : стосується відповідної спільноти учасників на основі Github.

У таблиці 1.1 підсумовано, як чотири різні фреймворки підтримують вищезазначені функції.

Storm реалізує модель потоку даних. Топологія застосунку описується за допомогою компонентів під назвою «spouts» та «bolts», що відносяться до джерел даних та елементів, що обробляють дані у вигляді кортежів відповідно. Оскільки кортеж обробляється в міру його надходження, Storm має затримку менше секунди. Його механізм підтримує низьку пропускну здатність головним чином завдяки механізму підтвердження. Кожен запис, оброблений від оператора, надсилає підтвердження назад попередньому оператору, що вказує на те, що його було оброблено. Цей механізм також може помилково класифікувати низку записів як непідтверджені. Тому ці записи доведеться повторно обробити, що призведе до ще нижчої пропускну здатності. Він не забезпечує відновлення стану, але забезпечує гарантовану доставку та обробку даних за допомогою механізму резервного копіювання вищестоящого рівня.

У разі збою (наприклад, збою робочого процесу), якщо отримано не всі підтвердження, записи відтворюються системою споуту [36]. Таким чином, втрата даних не відбудеться, але дублікати записів можуть проходити через систему. Змінні стани можуть бути неправильно оновлені двічі. Ось чому він пропонує гарантію доставки принаймні один раз. Trident Storm пропонує мікропакетну обробку для підвищення пропускну здатності. У Storm Trident станом можна керувати автоматично, тому це гарантує узгодженість стану (доставка рівно один раз), але стан зберігається в реплікованій базі даних, що є дорогим, оскільки всі оновлення реплікуються по всій мережі. Apache Storm — це композиційна система, яка очікує, що користувач явно визначить DAG програми.

Spark Streaming групує події в короткий кадр перед обробкою отриманих даних. На низькому рівні дані представлені як RDD, а обчислення над цими RDD можуть бути представлені як перетворення або дії. Абстракція для потоків даних

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

називається Dstream і складається з послідовності RDD, що містить дані певного інтервалу потоку. DStreams дозволяють користувачам застосовувати до них перетворення.

Потокові обчислення в Spark Streaming являють собою серію пакетних обчислень з визначеним часовим інтервалом. Адаптація пакетної методології для потокової обробки включає буферизацію даних під час їх надходження в систему, що призводить до затримки в кілька секунд. Існує штраф за затримку, що дорівнює тривалості мікропакету. Очікування очищення буфера також призводить до збільшення затримки. Тим не менш, буфер дозволяє Spark Streaming обробляти великий обсяг вхідних даних, збільшуючи загальну пропускну здатність [37].

Таблиця 1.1

Класифікація рішень для потокового передавання

Платформа / Критерії	Фреймворк обробки	Примітив потоку	Затримка (Latency)	Пропускна здатність	Гарантія доставки	Модель програмування	Мови API
Storm	Потоковий (Streaming)	Кортеж (Tuple)	Субсекундна	Низька	Щонайменше один раз (Exactly once з Trident)	Композиційна	Будь-яка
Spark Streaming	Мікропакетний (Micro-Batch)	Dstream	Кілька секунд	Висока	Рівно один раз	Декларативна	Java, Scala, R, Python
Samza	Потоковий (Streaming)	Повідомлення (Message)	Субсекундна	Висока	Щонайменше один раз	Композиційна	Мови JVM
Flink	Гібридний (Hybrid)	DataStream	Субсекундна	Висока	Рівно один раз	Декларативна	Java, Scala, Python

Як бачимо, існує компроміс між низькою затримкою та високою пропускну здатністю. Системи, засновані на мікропакетуванні, можуть досягати високої пропускну здатності, але якщо обробка пакету займає більше часу в наступних операціях, ніж в операціях пакетування, мікропакет триватиме довше, ніж налаштовано. Це може призвести до того, що все більше пакетів

будуть стояти в черзі (або до зростання розміру міні-пакету) [38]. Хоча Spark Streaming може бути адекватним для багатьох проектів, він не є справжньою системою реального часу. Зазначають, однак, що хоча ця модель повільніша за справжнє потокове передавання, затримку можна достатньо мінімізувати для більшості реальних проектів завдяки використанню RDD. RDD дозволяють виконувати обчислення в пам'яті відмовостійким способом, уникаючи запису виводів на репліковані дискові системи зберігання, що призводить до трудомістких дискових операцій вводу-виводу. Spark Streaming забезпечує підтримку як обчислень зі збереженням стану, так і без збереження стану та гарантує, що обробка на пакетному рівні буде виконана рівно один раз. Це досягається шляхом відстеження ліній передачі даних у кожному DStream. Весь стан у Spark Streaming зберігається в RDD. Це декларативна система, оскільки вона вводить кілька абстракцій для представлення даних та керування різними типами обчислень. Код визначає лише ті функції, які потрібно виконати над даними, а Spark передбачає відповідний DAG з викликаних функцій.

Samza — це фреймворк для потокової обробки, заснований на моделі публікації/підписки. Він прослуховує потік даних, обробляє повідомлення, які є його поточними примітивами, по одному за раз, та виводить свій результат в інший потік. Він тісно пов'язаний із системою обміну повідомленнями Apache Kafka [39] для потокової передачі даних між завданнями та Apache YARN для розподілу завдань між вузлами в кластері. Kafka пропонує репліковане сховище даних, до яких можна отримати доступ з низькою затримкою, тому завдання Samza можуть мати затримку в невеликих мілісекундах під час виконання з ним. Samza дозволяє завданням підтримувати стан, зберігаючи його на диску (зазвичай за допомогою Kafka). Цей стан зберігається на тій самій машині, що й завдання обробки, щоб уникнути проблем з продуктивністю. Розміщуючи сховище та обробку на одній машині, Samza може досягти високої пропускну здатності [40]. Він також відстежує, чи доставлено повідомлення, і повторно доставляє його у разі збою, щоб уникнути втрати даних за допомогою системи контрольних точок, але може гарантувати доставку лише принаймні один раз.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

Якщо завдання Samza завершується невдачею та перезапускається, воно може подвійно підрахувати деякі повідомлення, які могли бути використані з моменту запису останньої контрольної точки. Топологія завдання Samza явно визначається кодом користувача [15].

Flink — це гібридне рішення. Необхідність керувати різними робочими навантаженнями в рамках узгодженої архітектури призвела до появи кількох шаблонів проектування, найпопулярнішим з яких є «Лямбда-архітектура». Складність використання різних пакетних та потокових архітектур проклала шлях до шаблону «Кеппа-архітектура», який об'єднує пакетний та потоковий шари. Flink — це матеріалізація архітектури Каппа. Незважаючи на те, що він спирається на модель потокового виконання, можливо обробляти як обмежені, так і необмежені дані, використовуючи два API, що працюють на одному розподіленому потоковому виконанні.

Базова абстракція даних для обробки потоків називається DataStream. Вона виконує довільні програми обробки потоків даних паралельно з даними та конвеєрно, що призводить до досягнення низької затримки. Модель програмування потоків даних Apache Flink забезпечує обробку подій за часом. Кортежі можуть бути зібрані в буферах з регульованим тайм-аутом, перш ніж вони будуть відправлені наступному оператору для перемикання між пропускнуою здатністю та затримкою. Вона працює у великих масштабах, працюючи на тисячах вузлів з дуже хорошими характеристиками пропускнуої здатності та затримки на основі існуючих тестів. При використанні обчислень зі збереженням стану вона забезпечує семантику рівно один раз. Apache Flink включає легкий механізм відмовостійкості, заснований на розподілених контрольних точках. Його алгоритм базується на техніці і періодично створює узгоджені знімки поточного стану розподіленої системи без пропуску інформації та без запису дублікатів. Ці знімки зберігаються у довговічному сховищі. У разі збою останній знімок відновлюється, джерело потоку перемотується до точки, коли знімок був зроблений, і відтворюється повторно [23]. Flink наразі є унікальним варіантом у світі фреймворків для обробки даних, але наразі це

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

молодий проект, і його обмеження масштабування ще не були досліджені. Це декларативна система, яка надає користувачам, таким як Spark, абстракції вищого рівня. DAG передбачає порядок перетворень, тоді як його рушій може змінювати порядок перетворень за потреби.

Наведений огляд показує, що не існує системи, здатної робити все, і немає системи, яка не робить нічого. Усі системи щось роблять, але роблять це по-різному. Порівняльний аналіз може бути гарним способом їх порівняння, особливо коли він проводився третіми сторонами. Щоб пролити світло на продуктивність вищезгаданих двигунів, було проведено кілька експериментів, які досліджують затримку, пропускну здатність та вплив їхнього механізму відмовостійкості. Деякі приклади наведено нижче:

- Порівняно Spark Streaming та Storm Trident. Він надав порівняльний аналіз обох систем для різних завдань та обробки кортежів різного розміру. Головний висновок полягав у тому, що Storm Trident був приблизно на 40% швидшим за Spark, обробляючи кортежі малого розміру. Однак, зі збільшенням розміру кортежу, Spark мав кращу продуктивність, зберігаючи час обробки.

- Розробили та впровадили реальний тест потокової передачі, зосередившись на Storm, Flink та Spark Streaming, і виявили, що Storm та Flink мають набагато меншу затримку, ніж Spark Streaming, при досить високій пропускну здатності. З іншого боку, Spark Streaming здатний обробляти вищу пропускну здатність, і його продуктивність досить чутлива до налаштувань тривалості пакетної передачі.

- Порівняли Flink зі Spark за двома бенчмарками: Intel HiBench Streaming та Yahoo Stream. Обидві системи працювали однаково за різних навантажень, але Flink продемонстрував дещо кращу продуктивність за нижчої частоти подій, головним чином завдяки методу мікропакетної обробки Spark Streaming. Використання процесора було подібним в обох системах, але коли справа доходила до використання пам'яті, Flink потребував менше пам'яті, ніж Spark.

- Запропоновано визначення бенчмарку під назвою StreamBench. Вони

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

застосували StreamBench до Spark Streaming та Storm. Вони виявили, що Spark, як правило, має більшу пропускну здатність (навіть приблизно в 5 разів, ніж у Storm) та менший вплив відмов вузлів порівняно зі Storm. Однак Storm має набагато меншу затримку (навіть у 50 разів меншу), ніж Spark, за винятком складних робочих навантажень з великим масштабом даних, для яких його затримка може бути в кілька разів більшою, ніж у Spark. Між джерелом даних та потоковою системою використовувалася система-посередник для їх розв'язання.

- Запропоновано фреймворк для бенчмарків, в якому вони відокремили тестовану систему від драйвера, щоб уникнути посередників, та провели експерименти зі Storm, Spark та Flink. Як Flink, так і Spark були стійкими до коливань швидкості надходження даних під час агрегаційних робочих навантажень, тоді як для запитів на об'єднання Flink поведився краще. Якщо затримка є пріоритетом, Flink здавався найкращим вибором і мав кращу загальну пропускну здатність під час тестування в запитах на агрегацію та об'єднання.

Щоб надати точні та персоналізовані рекомендації, необхідно розробити та впровадити більше реальних поточкових бенчмарків на вищезгаданих системах, заснованих на різних варіантах використання. Тим не менш, ми не можемо повністю покладатися на бенчмаркінг у потоковій обробці, оскільки навіть невелика зміна в конфігурації або варіанті використання може повністю змінити цифри. Більше того, вищезгадані технології, можливо, зарекомендували себе як лідери та мають сильні спільноти підтримки, але все ще розвиваються. Наприклад, більшість вищезгаданих бенчмарків не враховують випуск Spark Streaming 2.0, який долає кілька обмежень завдяки RDD, покращуючи свою продуктивність порівняно з іншими системами. Отже, нелегко скласти чіткий рейтинг з кількісно вимірюваними результатами. Однак, поєднуючи наведений вище огляд (Таблиця 1.1) з доступними бенчмарками, можна зробити деякі висновки:

- Storm працює з дуже низькою затримкою, але може доставляти дублікати та не може гарантувати впорядкування за замовчуванням. Оскільки він не забезпечує неявної підтримки керування станом, він не підходить для

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

випадків складної обробки подій. Тим не менш, він чудово підходить для нескладних випадків потокового передавання, де затримка має вирішальне значення, завдяки своїй справжній потоковій природі та зрілості.

- Samza має бути тісно інтегрована з Kafka та YARN, щоб забезпечити високу продуктивність, гнучкість та керування станом. Без них її важко використовувати в конвеєрі обробки, а розгортання такої системи вимагатиме ретельного тестування, щоб переконатися в правильності топології. Тим не менш, якщо ці технології вже інтегровані, це зріле, відмовостійке рішення, що забезпечує високу продуктивність (як щодо затримки, так і щодо пропускну здатності). Гарантія обробки принаймні один раз залишається проблемою, але вона згадується і незабаром буде вирішена в наступному випуску.

- Spark дуже популярний, зрілий та широко застосовується, його підтримує сильна спільнота. Він забезпечує високу пропускну здатність, стійкий до відмов завдяки своїй мікропакетній природі, що забезпечує лише одну гарантію, і може використовуватися, якщо сублатентність не потрібна. Тим не менш, він відстає від Flink за багатьма розширеними функціями. Його новий реліз (Structured Streaming) оснащений кількома хорошими функціями та обіцяє субсекундну затримку, але ще зарано для досягнення цієї амбіції.

- Flink забезпечує справжню потокову обробку з підтримкою пакетної обробки. Він сильно оптимізований, включає кілька інновацій, таких як легкі знімки, і, схоже, є лідером у ландшафті DSMS. Як ми бачили, більшість бажаних аспектів (низька затримка, висока пропускну здатність, гарантія точно одного разу, управління станом) забезпечено. Тим не менш, спочатку його було недостатньо, і його зрілість викликає занепокоєння. Підтримка спільноти також менша, ніж у Spark. Однак, схоже, це швидко змінюється, оскільки він почав широко впроваджуватися у світі бізнесу.

Отже, найкраща можлива відповідь для користувачів, які хочуть дізнатися, яке рішення є найкращим, полягає в тому, що це залежить від їхніх потреб. Слід також враховувати майбутні міркування. Наприклад, у простих випадках Storm може здатися гарною ідеєю. Тим не менш, якщо пізніше

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

виникнуть складні вимоги, що передбачають складну обробку подій, таку як агрегації та об'єднання, або якщо також необхідно впроваджувати пакетно-орієнтовані завдання, слід віддати перевагу передовим потоковим фреймворкам, таким як Spark Streaming або Flink. Зміни в існуючій інфраструктурі та перенавчання співробітників можуть призвести до величезних витрат часу та грошей.

Зрештою, слід пам'ятати, що кодування в декларативних системах набагато простіше, ніж у композиційних, оскільки користувачам надаються абстракції вищого рівня, і вони імпліфікують DAG через свій код. Оптимізацію може виконувати система. Тим не менш, у композиційних системах код повністю контролюється розробником. Якщо є потреба у швидкій та легкій реалізації, перевагу слід віддавати таким системам, як Spark або Flink, але якщо потрібен повний контроль над графом додатків, тоді слід обрати Storm або Samza.

Як бачимо, найкращий вибір для кожної ситуації залежить від кількох факторів. Розуміння

Механізми та характеристики вищезгаданих архітектур полегшують вибір або принаймні фільтрацію доступних варіантів. Оцінка незавершеної роботи може допомогти користувачам зробити найкращий вибір на основі їхніх потреб. Зміни в конфігурації, налаштуванні або доступній інфраструктурі можуть змінити результати та призвести до суттєвих покращень. Планування завдань сильно впливає на продуктивність та відмовостійкість системи потокової обробки та буде розглянуто в наступному розділі.

1.4 Висновок по розділу

У першому розділі було розглянуто теоретичні основи обробки поточкових даних у реальному часі. Проаналізовано концепцію великих даних (Big Data), їх основні характеристики та значення для сучасних інформаційних систем. Досліджено особливості потокової обробки даних, зокрема безперервність надходження інформації, необхідність оперативного аналізу та

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

обмеження щодо затримок обробки. Також визначено основні виклики та проблеми, пов'язані із забезпеченням масштабованості, відмовостійкості, продуктивності та точності обробки поточкових даних. Встановлено, що ефективна обробка даних у режимі реального часу є важливою складовою сучасних програмних рішень, які працюють із великими обсягами інформації.

					БР.ІІІ – 36.00.00.000 ІІЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ СИСТЕМ ПОТОКОВОЇ ОБРОБКИ

2.1 Сучасні інструменти та платформи потокової обробки даних

Стародавні методи обробки даних, включаючи Hadoop, а саме завдання MapReduce, неадекватні для обробки в режимі реального часу. Обробка потоку даних у режимі реального часу дозволяє вам бути в курсі того, що відбувається на даний момент, незалежно від швидкості чи обсягу даних, необхідних для системи зберігання. Щоб добре зрозуміти систему, ми коротко розглянемо інші платформи, а саме Hadoop, Spark та Storm.

Apache Hadoop [10] – це програмне забезпечення з відкритим вихідним кодом, яке використовується для обробки великих даних на кластерах машин та пакетної роботи з цими наборами даних. Серце Hadoop поділено на дві основні частини, а саме MapReduce для обробки даних та HDFS для зберігання даних. Воно відоме своєю надійністю, масштабованістю та моделлю обробки.

MapReduce вперше був представлений у Google у 2004 році[11]. Це модель програмування та пов'язана з нею реалізація для обробки та генерації великих наборів даних на великих кластерах стандартних машин. Вона має високу масштабованість, може обробляти петабайти даних, що зберігаються в HDFS, на одному кластері, та є високою відмовостійкістю, що дозволяє запускати програми на кластері стандартних серверів. Ця структура базується на двох серверах: головному трекері завдань, унікальному для кластера, який отримує завдання MapReduce для запуску та організовує їх виконання на кластері. Він також відповідає за планування складових завдань на підлеглих серверах, а також за їх моніторинг та повторне виконання невдалих завдань. Інший сервер - це трекер завдань, їх кілька на кластер, він сам виконує завдання MapReduce. Кожен з трекерів завдань є одиницею обчислення кластера.

Користувачі вказують функцію map, яка обробляє пари ключ/значення для створення набору проміжних пар ключ/значення, та функцію reduce, яка

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

об'єднує всі проміжні значення, пов'язані з одним і тим самим проміжним ключем. Як показано на рисунку 2.1, спочатку бібліотека MapReduce у користувацькій програмі розділяє вхідні файли на М частин, зазвичай розміром 16-64 МБ кожна, головний обробник вибирає неактивні виконавці та призначає кожному з них завдання map або reduce. Виконавець, якому призначено завдання map, зчитує вміст відповідного вхідного розділення. Проміжні пари ключ/значення, створені функцією map, буферизуються в пам'яті. Періодично буферизовані пари записуються на локальний диск. Коли виконавець reduce зчитав усі проміжні дані для свого розділу.

Він сортує його за проміжними ключами, щоб усі входження одного й того ж ключа групувалися разом. Сортування необхідне, оскільки зазвичай багато різних ключів відповідають одному й тому ж завданню reduce. Якщо обсяг проміжних даних занадто великий, щоб поміститися в пам'яті, використовується зовнішнє сортування. Функція reduce перебирає відсортовані проміжні дані та для кожного унікального проміжного ключа, що зустрічається, передає ключ та відповідний набір проміжних значень функції reduce користувача[11].

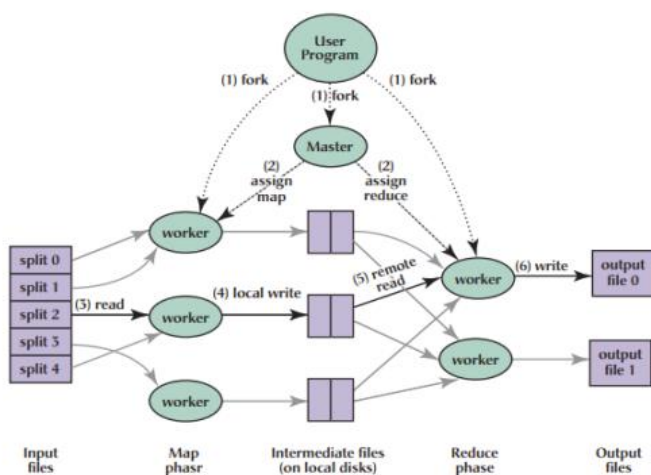


Рисунок 2.1 - Загальний огляд виконання MapReduce

MapReduce має деякі обмеження[12]:

- Розроблено для пакетної обробки
- На основі дискового підходу

- Надзвичайно жорсткий потік даних
- Пошук диска для кожного доступу
- Неefективно для ітеративних алгоритмів та інтерактивного аналізу даних

Apache Spark[13] — це фреймворк з відкритим кодом для обробки великих даних, побудований на базі Hadoop MapReduce для виконання складного аналізу та розроблений для швидкості та простоти використання. Спочатку він був розроблений Університетом Каліфорнійського університету в Берклі у 2009 році та перейшов до категорії з відкритим кодом як проект Apache у 2010 році. Spark має блискавично високу продуктивність та пришвидшує час обробки, оскільки працює в пам'яті на кластерах. Він також розроблений для роботи з пакетами, такими як Apache Hadoop, але розмір вікна пакетів дуже малий. Ядром Apache Spark є RDD (Resilient Distributed Dataset), це відмовостійкий набір елементів, розподілених по багатьох серверах, на яких ми можемо виконувати паралельні операції. [14] Елементи RDD не обов'язково повинні знаходитися у фізичному сховищі; натомість дескриптор RDD містить достатньо інформації для обчислення RDD, починаючи з даних у надійному сховищі. Це означає, що RDD завжди можна реконструювати, якщо вузли вийдуть з ладу. Поряд з основними API Spark, екосистема містить додаткові бібліотеки, які дозволяють працювати в галузі аналізу великих даних та машинного навчання. Ці бібліотеки включають потокове передавання Spark для обробки безперервного потоку даних, Spark SQL для роботи зі структурованими даними, MLlib — бібліотеку машинного навчання, та GraphX для обчислення графів, як показано на рисунку 2.2.

Потокове передавання Spark [13] – це розширення основного API Spark, яке забезпечує масштабовану, високопродуктивну та відмовостійку обробку потоків даних у реальному часі. Дані можуть надходити з багатьох джерел, таких як Kafka, Flume, Kinesis або TCP-сокети, та оброблятися за допомогою складних алгоритмів, виражених за допомогою високорівневих функцій, таких як map, reduce, join та window. [15] На рисунку 2.3 ми бачимо, що потокове передавання

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

Spark базується на режимі обробки в мікропакетному режимі. Воно отримує потоки вхідних даних у реальному часі та розділяє їх на пакети, які потім обробляються механізмом Spark протягом дуже фіксованого часу для генерації кінцевого потоку результатів пакетами. Усі вхідні потоки обробляються однаково. Той самий рекурентний таймер розподіляє пакети для всіх потоків протягом кожної пакетної тривалості.

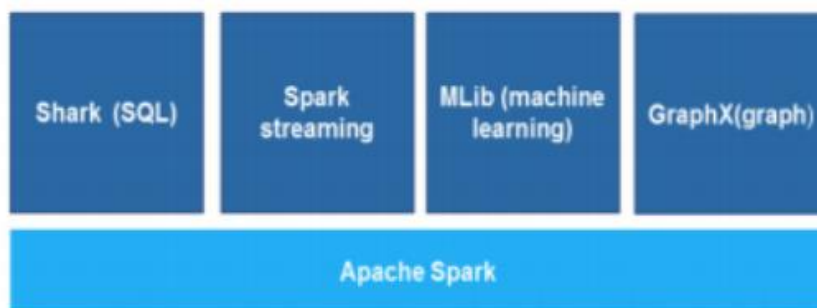


Рисунок 2.2 - Фреймворк Spark

Spark має кілька переваг над Hadoop MapReduce. По-перше, Spark пропонує комплексну та уніфіковану платформу для задоволення потреб обробки великих даних. По-друге, Spark дозволяє запускати програми на кластерах Hadoop до 100 разів швидше в пам'яті та в 10 разів швидше на диску. Крім того, його можна використовувати інтерактивно для запиту даних з вікна командної оболонки. Незважаючи на всі переваги Spark порівняно з Hadoop MapReduce, він все ще стикається з кількома обмеженнями; серед них - обробка потоку в реальному часі, яка не гарантована. Це пов'язано з тим, що Spark реалізує концепцію мікропакетів у своїй роботі та не обробляє дані в міру їх надходження, оскільки вони накопичуються протягом певного періоду часу перед обробкою. Основна різниця між Spark та Storm буде обговорена в наступних розділах.



Рисунок 2.3 - Spark Streaming

Storm, [16] технологія, реалізована Натаном Марцем для аналізу в реальному часі у грудні 2010 року, є безкоштовною розподіленою системою обчислень у реальному часі з відкритим кодом, яка дозволяє легко та надійно обробляти необмежені потоки даних. Storm виконує для обробки в реальному часі те саме, що Hadoop виконує для пакетної обробки.

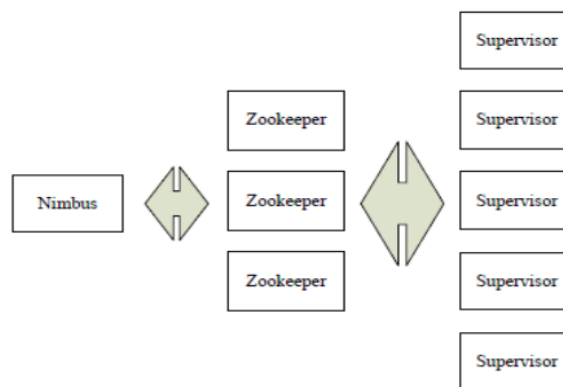
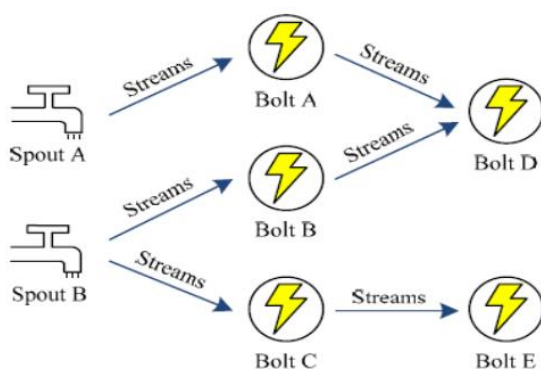


Рисунок 2.4 - Будова кластера «Шторм»

Штормовий кластер має три набори вузлів, перший з яких – це демон-процес під назвою «Nimbus», подібний до Hadoop Job Tracker. Він працює на головному вузлі для завантаження обчислень для виконання, розподілу коду по кластеру, упорядкування завдань та виявлення помилок. Другий вузол називається «Supervisor», він відповідає за запуск та зупинку робочого процесу відповідно до сигналів від Nimbus. І, нарешті, вузол «Zookeeper», який є розподіленою службою координації, що координує штормовий кластер, як показано на рисунку 2.4. Штормовий кластер зовні схожий на кластер Hadoop. У той час як на Hadoop ви запускаєте «завдання MapReduce», на Storm ви запускаєте «топології». Самі «завдання» та «топології» дуже різні – одна

ключова відмінність полягає в тому, що завдання MapReduce зрештою завершується, тоді як топологія обробляє повідомлення вічно (або доки ви її не завершите).

Топологія складається з жерлин та болтів, а зв'язки між ними показують, як проходять потоки. Ця топологія представлена у вигляді орієнтованого ациклічного графа обробки даних (DAG), який представляє всю процедуру обробки потоку. Представлення топології показано нижче на рисунку 2.5.



Рисунку 2.5 - Приклад топології

Споут (spout) – це джерело потоків, яке зчитує кортежі із зовнішнього вхідного джерела та передає їх до болтів (bolts) як потік. Болт (bolt) – це блок обробки даних топології шторму (storm), який споживає будь-яку кількість вхідних потоків, виконує певну обробку та передає нові потоки іншим болтам. Основною абстракцією в Storm є "потік". Потік – це необмежена послідовність кортежів. Кортеж – це іменований список значень, а поле в кортежі може бути об'єктом будь-якого типу. Storm надає примітиви для перетворення потоку в новий потік розподіленим та надійним способом.

2.2 Методологія проєктування систем обробки поточкових даних

Наше дослідження дотримується рекомендацій щодо систематичних оглядів, наведених у дослідженнях з програмної інженерії в [7] та [8]. Відповідно до цих рекомендацій, ми включили три основні фази в нашу

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

методологію дослідження: планування, проведення та звіт про огляд.

Фігура На рис. 2.6 показано методологію дослідження, яка демонструє - процес пошуку відповідних дослідницьких видів діяльності, визначення схеми категоризації та зіставлення статей.



Рисунок 2.6 - Методологія дослідження.

У цьому огляді було дотримано чітко структурованого процесу, який включав:

- Цілі дослідження
- Конкретизація дослідницьких питань (ДП)
- Організація пошуку в базах даних
- Вибір досліджень
- Скринінг відповідних досліджень
- Вилучення даних
- Синтез результатів
- Завершення звіту про огляд

Основні цілі нашого дослідження полягають у наступному:

а) Розробити бібліотеку статей, пов'язаних з розробкою обробки потоків у реальному часі під час ETL фази чи інші, та зробити цей набір даних доступним для інших дослідників.

б) Визначити більш значущі роботи, які надають напрямок для дослідження проблем обробки потоків у реальному часі, ETL та сховища даних

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

у реальному часі.

с) Визначити прогалини в дослідженнях ETL, обробки потоків у реальному часі та сховища даних (DWH) у нещодавніх дослідженнях.

д) Охарактеризуйте існуючі підходи та рішення для вирішення цих проблем під час впровадження потокової обробки в реальному часі для гетерогенних, структурованих/неструктурованих даних та з'ясуйте подібності та відмінності між ними.

Важливо сформулювати первинні запити на отримання інформації (RQ) для проведення цього SLR. Ці запити на отримання інформації розроблені для визначення відповідних каналів публікацій, проблем/розробок та доказів підходів до систем обробки потоків у реальному часі, як зазначено в таблиці. 2 .

Для підвищення надійності та валідності результатів нашого пошуку у включенні статей брали участь два рецензенти. Процес проведення цього огляду було сформульовано у чотири кроки, представлені нижче. На першому кроці було проведено пошук відповідних первинних досліджень у найбільш часто використовуваних цифрових бібліотеках. Відбір досліджень на основі заздалегідь визначених критеріїв включення/виключення було здійснено на другому кроці.

Ми розробили критерії оцінки якості для подальшого підвищення якості нашого огляду, описаного на третьому кроці. Потім на останньому четвертому кроці виконується зворотний метод снігової кулі для виділення важливих робіт-кандидатів.

Було проведено систематичне дослідження для фільтрації нерелевантних досліджень та вилучення відповідної інформації. Тому під час дослідження пошукових термінів були використані напівавтоматичні та ручні методи пошуку.

Окрім цього, було досліджено ще кілька цифрових бібліотек, але вони не були включені через обмеження доступності. Метою ручного пошуку є зібрати більше літератури, що стосується обробки потоків у реальному часі та сховища даних. Витягнута інформація може бути більш релевантною для обмеженого

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

обсягу пошукових термінів, тому для обмеження пошукових термінів було застосовано такі умови:

- На основі сформульованих RQ визначте основні ключові слова.
- Визначення вторинних ключових слів та синонімів для додаткових ключових слів.
- Булеві оператори «І» та «АБО» були включені до ключових слів для створення рядка пошуку.

Можливі варіанти розташування використаного рядка пошуку можна побачити на рисунку 3. Первинні ключові слова були обрані як ключові - ідентифікатори для дослідження обробки потоків у реальному часі. Первинні ключові слова були обрані разом з будь-якими вторинними або додатковими ключовими словами. Комбінація ключових слів, булевих операторів та підстановочних знаків сформувала кінцевий рядок пошуку, який має вигляд:

(реальний час АБО сховище даних реального часу*) ТА (потік АБО напівпотік АБО ETL АБО виклики АБО (вилучення*, перетворення*, завантаження*)) ТА (об'єднання АБО обробка* АБО аналіз* АБО проблема*)

На рисунку 2.7 показано остаточні рядки пошуку, що використовуються для пошуку в семи цифрових бібліотеках. Напіваавтоматичний пошук обмежувався лише назвами журналів ACM, IEEEExplore та ScienceDirect . Через обмеження в п'ять символів підстановки для пошуку в IEEEExplore, рядок пошуку для цієї бібліотеки довелося дещо змінити. Завдяки цьому налаштуванню кількість нерелевантних результатів зменшилася. Інші цифрові бібліотеки досліджувалися з налаштуванням «всі поля», оскільки вони не дозволяють більш конкретну конфігурацію пошуку. Рядок пошуку, який був занадто обмеженим, не допоміг знайти відповідні статті для цифрової бібліотеки IGI Global, тому остаточний рядок пошуку, розроблений для цієї бібліотеки, містить меншу кількість ключових слів, показаних у таблиці. 3. Остаточний рядок пошуку не вдався під час застосування до цифрової бібліотеки мовою хіндаві, тому пошук було проведено з невеликою кількістю ключових слів , що призвело до деяких релевантних результатів.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

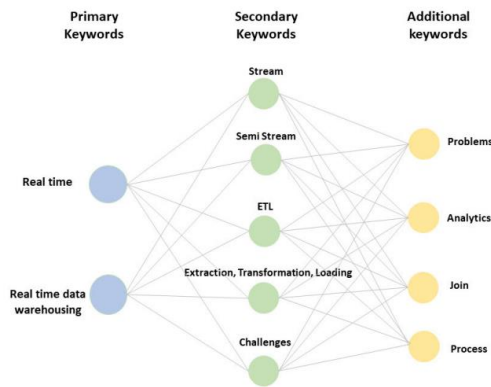


Рисунок 2.7 - Ключові слова для пошуку, що використовуються для виявлення творів, включених до нашої бази знань.

2.3 Методи оптимізації: розподіл ресурсів та планування обчислень у потокових системах

Вищезгадані фреймворки використовують спеціальні механізми обробки даних та архітектури для забезпечення необхідної продуктивності. Ефективний розподіл ресурсів та планування завдань потокових програм має велике значення в обробці потоку даних. *Розподіл ресурсів* вирішує проблему збору та призначення ресурсів різним запитам. Усередині програми ресурси необхідно ретельно планувати різним компонентам, щоб забезпечити оптимальне використання. Неправильне розміщення ресурсів може призвести як до поганого використання ресурсів, так і до нестабільності системи в цілому. Надмірне виділення ресурсів кожному оператору може призвести до їх марнування, тоді як розвантаження призводить до неправильних результатів [63]. *Планування завдань* контролює порядок виконання завдань і піклується про те, які завдання і коли розміщувати на яких раніше отриманих ресурсах [64]. Відповідні методи планування можуть покращити продуктивність системи та зменшити споживання енергії та витрати, особливо коли ми переходимо до хмари. Однак, як ми вже бачили, такі цілі, як низька затримка та висока пропускну здатність, часто конфліктують. Таким чином, планувальник повинен впроваджувати відповідний компроміс, залежно від потреб та цілей користувача.

Що стосується планування завдань, запити передаються до даних, і

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

Hadoop має можливість переміщувати обчислення ближче до місця фактичного розміщення даних, замість переміщення великих даних на обчислення. Це мінімізує перевантаження мережі та збільшує загальну пропускну здатність системи. Локальність даних відіграє вирішальну роль. Тим не менш, MapReduce неефективний для обробки величезних обсягів даних у режимі реального часу. Це фреймворк, який неефективний для виконання інтерактивних завдань та запитів у режимі реального часу.

Підходи до планування та розподілу ресурсів для потокової обробки даних у реальному часі принципово відрізняються від традиційної пакетної обробки. Це набагато складніше через динамічну природу потоків вхідних даних. На відміну від систем пакетної обробки, системи потокової обробки призначені для обробки незв'язаних потоків даних. Завдання в першому випадку врешті-решт завершиться, тоді як завдання в другому продовжується вічно, якщо користувач не завершить його. Метою потокової обробки є створення надійної, високопродуктивної та низькозатримуючої обробки подій, яка може безперервно аналізувати та реагувати на поточкові дані в реальному часі. У системах пакетної обробки обчислення призначаються вузлам, де зберігаються необхідні дані, тоді як у поточковій передачі більшість взаємодіючих завдань розміщуються разом на одному вузлі або стійці. Більше того, обчислення в пам'яті, що використовуються в системах потокової обробки, в сотні разів швидші, ніж обчислення, що виконуються на диску, як це традиційно для таких систем, як Hadoop.

Системи потокової обробки, що дотримуються мікропакетного підходу, мають кілька переваг порівняно з системами потокової обробки, які обробляють дані по одному запису за раз, такі як швидке відновлення після збоїв, краще балансування навантаження та масштабованість. Мікропакетні системи оптимізовані для пропускну здатності, але мають збільшений час відповіді на запити, оскільки кожен вхідний запит має чекати, поки буде сформовано пакет. Надзвичайно малі пакети можуть мінімізувати цю додаткову затримку, але це призведе до додаткових накладних витрат. З іншого боку, потокова обробка на основі оператора працює краще, коли нам доводиться мати справу зі суворими

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

обмеженнями реального часу, але схильна до збоїв. Щоб забезпечити необхідну продуктивність системи при високому навантаженні вхідних даних, необхідні додаткові механізми обробки та ефективне планування потокових програм.

Ми поділяємо рішення щодо планування на офлайн (з використанням статичних алгоритмів) та онлайн (з використанням динамічних алгоритмів). Алгоритми можуть спиратися на заздалегідь визначені характеристики потокових топологій (офлайн) або можуть збирати інформацію за допомогою систем моніторингу (онлайн). Офлайн-рішення спираються на знання, які планувальник має до розміщення та запуску будь-якого завдання. Онлайн-рішення стосуються інформації, зібраної під час виконання користувацького застосунку, після початкового розміщення його завдань по вузлах кластера. Ми збираємося далі розглянути та класифікувати роботи, що пропонують евристичні алгоритми (або іноді навіть цілі архітектури), які намагаються вибрати оптимальне розміщення ресурсів та виконавців завдань для максимізації їхньої продуктивності.

Spark Streaming пакує дані через невеликі інтервали часу (мікропакети) та пропускає їх через детерміновані паралельні операції. Завдання застосовуються до DStream відповідно до дій, що виконуються користувачами в застосунку. Кожне завдання розділяється на завдання (найменший блок виконання в Spark), а планувальник Spark відповідає за їх призначення доступним ресурсам. Планування мікропакетних завдань у Spark Streaming для максимізації продуктивності та ефективності використання ресурсів є досить громіздким, оскільки пакетне виконання часто включає кілька завдань зі складними та динамічними залежностями в робочому навантаженні.

За замовчуванням планувальник Spark запускає завдання за методом FIFO, і кожна програма намагається використовувати всі доступні вузли. Хоча залежності завдань можна фіксувати за допомогою FIFO, такий підхід може призвести до збільшення затримки, коли довготривале завдання затримує завдання, що відстають від нього. Користувачі можуть обмежити кількість вузлів, які може використовувати програма, і вручну встановити рівень

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

паралелізму, але він залишатиметься незмінним протягом усього виконання програми. Низький паралелізм може призвести до неефективного використання ресурсів, тоді як високий паралелізм може планувати більше взаємозалежних завдань і, зрештою, призвести до низької продуктивності. Такий тип планування є статичним, оскільки загальні рішення щодо планування визначаються до запуску системи. Порівняли Spark Streaming зі Storm, повідомляючи про їхню пропускну здатність. На Storm негативно вплинули менші розміри записів і він виконав 115 тис. записів/с/вузол для 100-байтових записів порівняно з 670 тис. у Spark. Порівняння між Spark Streaming та іншими DSMS можна знайти в підрозділі. Однак хороший планувальник повинен враховувати динамічний попит на ресурси між завданнями та інші фактори планування, такі як розмір пакета або рівень паралелізму, щоб мати змогу гарантувати обмеження в реальному часі в Spark Streaming. Послідовне планування завдань за замовчуванням у Spark є неефективним.

Розподіл ресурсів та планування завдань у *Storm* передбачає прийняття рішень у режимі реального часу щодо того, як реплікувати болти та розподіляти їх по вузлах кластера. Storm має набагато меншу затримку, ніж Spark Streaming, завдяки моделі мікропакетного виконання Spark (навіть у 50 разів менша. Користувач вказує кількість одночасних завдань для кожного компонента. Кожне з цих завдань, що виконуються паралельно, містить однакову логіку обробки, але працює з різними даними та може виконуватися в різних фізичних місцях. Таким чином, завдання з одного болта або витоку, ймовірно, розміщуються на різних фізичних машинах. Планувальник за замовчуванням використовує стратегію циклічного розподілу для рівномірного розподілу завдань по слотах вузла. Така стратегія не є оптимальною.

Запропонували два планувальники для Storm: автономний та онлайн. Перший аналізує граф топології та визначає можливі набори болтів для планування на одному вузлі, розглядаючи спосіб їхнього з'єднання. Досліджуються шаблони зв'язку між виконавцями, щоб розмістити виконавці, які найбільше спілкуються, в одному слоті. Нарешті, слоти призначаються

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

робочим вузлам циклічним способом. Вони протестували продуктивність свого підходу як на загальній топології, так і в більш реалістичних умовах. Затримка обробки події була покращена автономним планувальником порівняно зі стандартним, приблизно на 20%, а міжвузловий трафік - на 13%.

Алгоритми планування Storm за замовчуванням не враховують вимоги до ресурсів та їх доступність і зазвичай стають неефективними. Він надає планувальник (IsolationScheduler), який дозволяє користувачеві вказувати топології для запуску на виділеному наборі машин у кластері, але налаштування ізоляції є статичним [30]. Реалізували *R-Storm*, підхід до планування з урахуванням топології та ресурсів у Storm. Враховуючи топологію Storm, використовується метод обходу в ширину (BFS) для створення часткового впорядкування компонентів, щоб розмістити суміжні компоненти в тісній послідовності один до одного та запланувати їхні завдання якомога ближче. Маючи впорядкований список завдань, які потрібно запланувати, вузол-кандидат визначається шляхом знаходження найближчої евклідової відстані в 3-вимірному просторі між двома точками: вимогами завдання та доступними потужностями конкретного вузла. Планування, обчислене R-Storm, забезпечило на 30%-50% вищу пропускну здатність, ніж та, що обчислюється планувальником Storm за замовчуванням. Спільне розташування завдань, які взаємодіють один з одним, на одній машині або одній серверній стійці мінімізує затримку мережевого зв'язку. Завантаження процесора R-Storm було на 70-350% вищим порівняно зі Storm, оскільки R-Storm використовував менше машин для досягнення того ж рівня продуктивності.

Хоча R-Storm забезпечує кращу продуктивність, ніж циклічний алгоритм за замовчуванням, він не може контролювати продуктивність, коли відбувається спільне використання процесора. Запропонували стратегію, що враховує продуктивність, яка не залежить від стратегій використання ресурсів, що застосовуються R-Storm, і базується на *генетичному алгоритмі (ГА)*. Основною частиною генетичного алгоритму є його функція придатності, метою якої є побудова розкладу та оцінка його продуктивності. Моделі продуктивності

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

виконавців на заданих вузлах необхідні для оцінки їхньої пропускну здатності та будуються або на статистичних даних, зібраних під час виконання, або на історичних даних. Продуктивність кожного виконавця залежить не лише від вузла, до якого він призначений, але й від його батьківського виконавця. У своїх експериментах вони дозволили спільне використання процесора між завданнями та довели, що максимальної продуктивності завдань можна досягти за допомогою мінімального спільного використання процесора, споживаючи при цьому максимальну кількість ядер. Результати їхніх бенчмарків показали до 40% кращу продуктивність топології з точки зору пропускну здатності ГА порівняно з R-Storm у гетерогенних кластерах та на 16% в гомогенних. Планувальник ГА зміг впоратися з високим навантаженням у кількох топологіях, тоді як продуктивність R-Storm та Storm залишалася майже однаковою та низькою.

Представили ієрархічну адаптивну схему планування під назвою *P-Scheduler*. P-Scheduler також є стратегією, що враховує топологію, яка використовує ваги на ребрах графа, що зберігають швидкість передачі даних. Ці значення швидкості отримуються шляхом початкової передачі відомої топології до кластера та її запуску на планувальнику Storm за замовчуванням. Кількість вузлів, необхідних для запуску топології, обчислюється на основі навантаження на топологію. Ці вузли спільно використовують всю топологію, і їхні зв'язки визначаються. Алгоритм призначає найвищі пари зв'язку топології одному й тому ж виконавцю (JVM), а інші виконавці завантажуються підрозділом усієї топології. Топологія розділяється за допомогою METIS, спеціалізованого програмного забезпечення для розділення графів. Завдяки такій схемі зменшується міжпроцесний та міжвиконавчий трафік. P-Scheduler також є стратегією, що враховує робоче навантаження, в тому сенсі, що більшість топологій Storm мають майже однакове робоче навантаження, і загалом не існує проблем із балансуванням навантаження. Автори реалізували P-Scheduler в однорідному кластері з 8 робочими вузлами та протестували його на трьох топологіях. Результати показали, що P-Scheduler скорочує середній час обробки кортежів майже на 50% в результаті зменшення міжвузлового та міжробочого

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

зв'язку.

Запропонували алгоритм планування для гетерогенних кластерів, який використовує знання часу проектування та методи бенчмаркінгу. Щоб розгорнути процесори у доступні слоти, зберігаючи при цьому робоче навантаження в гетерогенному кластері, система повинна:

- а. отримати знання про характеристики продуктивності вузлів для різних типів обчислень та
- б. знайти спосіб аналізу обчислювальних характеристик вхідних завдань та вхідних даних.

Детальна специфікація окремих вузлів топології надається користувачем під час проектування кластера. Що стосується характеристик завдань, пропонується провести бенчмаркінг кожної програми на певному кластері перед її запуском у робочому середовищі для оцінки профілів споживання завданнями. Для оцінки їхнього підходу їхній прототип був реалізований як підключаємий планувальник для Apache Storm. Приріст продуктивності їхнього планувальника з точки зору всіх кортежів, оброблених усім додатком, склав 4,1% порівняно зі стандартним планувальником Storm та 17,5% порівняно з найгіршим можливим планувальником. Програми, що використовують графічні процесори та FPGA для деяких своїх компонентів, можуть отримати найбільшу користь від їхнього підходу.

Як ми вже згадували, планування завдань за замовчуванням у Spark Streaming недостатньо. Існує опція, доступна під час використання Spark з , яка пропонує динамічний спільний доступ до ядер процесора. Таким чином, програма може повертати ресурси кластеру, якщо вони їй не потрібні, та запитувати їх знову пізніше, коли є потреба. Ця функція корисна, коли кілька програм спільно використовують ресурси в кластері, але програмі може знадобитися деякий час, щоб повернути ядра, коли у неї є робота, що призводить до менш передбачуваної затримки.

Flink відстежує розподілені завдання, вирішує, коли планувати наступне завдання (або набір завдань), та реагує на завершені завдання або збої виконання.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

JVM-процеси Flink (TaskManagers) забезпечують певну кількість потоків виконання в кластері, породжених для запуску одного конвеєра паралельних завдань. Конвеєр складається з кількох послідовних завдань. Паралелізм програм Flink визначається ступенем паралелізму потоків та окремих операторів. Flink використовує одноразове розподілення завдань на тривалий час. Він використовує алгоритм негайного планування та планування в черзі. Перший повертає слот негайно, коли є запит, тоді як другий ставить запит у чергу та повертає слот, коли він доступний. Таким чином, у певному сенсі планувальник Flink працює довільним чином. Тим не менш, система є гнучкою для переналаштування конвеєрів на більшу або меншу кількість працівників та перерозподілу стану програми на вимогу. Такий підхід мінімізує накладні витрати на управління та дозволяє адаптуватися до змін апаратного або програмного забезпечення або часткових збоїв, які потенційно можуть виникнути.

Запропоновано *A-планувальник*, адаптивний підхід до планування, який динамічно планує кілька завдань одночасно, використовуючи різні політики на основі їхніх залежностей. Залежність даних між завданнями визначається шляхом профілювання DAG програми під час прийняття завдання (з урахуванням топології). Розподільник ресурсів застосовує справедливе планування для незалежних завдань та FIFO для залежних. Він також збирає статистику продуктивності, таку як наскрізна затримка для кожного завдання та пропускна здатність системи, щоб автоматично налаштувати параметри паралелізму завдань та політики спільного використання ресурсів (що виконується адаптивним оптимізатором налаштування). Проблему налаштування було сформульовано як процес навчання з підкріпленням, який використовує підхід, що враховує продуктивність. Їхні експериментальні результати показують, що *A-планувальник* може зменшити наскрізну затримку на 42%, покращити пропускну здатність робочого навантаження на 21% та енергоефективність на 13% порівняно зі Spark Streaming.

Drizzle також є стратегією, що враховує топологію та використовує

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

комунікаційну структуру DAG. Головною метою цієї стратегії є відокремлення інтервалів обробки від інтервалів координації, що використовуються для відмовостійкості. Для досягнення цього стратегія використовує центральний планувальник, який реалізує планування мікропакетних груп, усіх груп одночасно. Це дозволяє уникнути вузьких місць у централізованому плануванні, а обробка даних повністю відокремлена від рішень щодо планування. Використовується адаптивний алгоритм налаштування розміру групи, натхненний контролем перевантаження TSP. Під час виконання групи лічильники використовуються для відстеження часу, проведеного в різних частинах системи, а політика, аналогічна AIMD, визначає частоту координації для завдання. КORTEЖІ обробляються протягом мілісекунд, тоді як функції координації виконуються протягом секунд. Додаткові стратегії використовуються для підвищення продуктивності. Зокрема, попереднє планування використовується, щоб робочі машини могли відстежувати залежності даних і запускати завдання, коли залежності виконуються. Методи оптимізації запитів використовуються для досягнення кращої пропускну здатності. Стратегію порівнювали зі Spark та Flink з точки зору наскрізної затримки обробки та часу відновлення після збоїв, і повідомлялося, що вона має в 3-4 рази меншу затримку, ніж Spark. Ще одним показником порівняння було відновлення після збоїв, яке виконується в 4 рази швидше порівняно з Flink.

Ченом та ін. також запропоновано фреймворк попереднього планування під назвою *Lever*. Автори зосереджуються на проблемі відстаючих завдань; перепланування відсталих завдань протягом періоду виконання завдання збільшує час обробки мікропакетів та призводить до дорогого переміщення даних, оскільки дані вже відправлені. *Lever* відстежує та періодично збирає та аналізує історичні профілі завдань, що повторюються, та прогнозує відстаючі завдання. Потім він оцінює пропускну здатність вузла та вибирає відповідних помічників. Він також приймає рішення щодо перепланування до того, як модуль пакетної обробки відправить дані, щоб уникнути збільшення часу обробки мікропакету. Своєчасні рішення щодо планування мінімізують затримку

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

обробки. Чен та ін. впровадили Lever у Spark Streaming, і коли його оцінили, він скоротив час виконання завдання на 30-40% та значно перевершив традиційні методи.

Коли йдеться про мікропакетні системи, питання визначення розміру пакетів є фактором вирішального значення, оскільки воно впливає на затримку та може полегшити планування та перепланування завдань і даних. Онлайн-алгоритм автоматично адаптує розмір пакета, щоб вплинути на продуктивність системи. Динамічно регулюючи пакетний інтервал, можна впоратися з коливаннями в робочих умовах та контролювати загальну затримку кожної події. Автори використовували модуль керування для збору інформації зі статистики минулих завдань (наприклад, час обробки, затримка в черзі тощо), щоб вивчити характеристики робочого навантаження та адаптувати розмір пакета онлайн, коли відбуваються зміни швидкості передачі даних, робочого навантаження та ресурсів. Їхній алгоритм керування базується на методі оптимізації з фіксованою точкою ітерацій, і їхні експерименти показують, що стабільність системи може бути забезпечена для широкого діапазону робочих навантажень (фільтрація, скорочення, об'єднання, вікно), незважаючи на великі коливання швидкості передачі даних та робочих умов. Під час оцінювання їхній підхід зміг забезпечити затримку, порівнянну з мінімальною затримкою, досягнутою за допомогою будь-якого статично визначеного методу пакетного інтервалу. Тим не менш, він вразливий до великих затримок циклу. Раптові значні зміни робочого навантаження можуть призвести до великих затримок.

Досліджували вплив розміру пакета на рішення щодо планування та продуктивність потокових навантажень у Spark Streaming. Їхня ідея базується на динамічному регулюванні інтервалів пакетів відповідно до кількості подій, що надходять у поточному інтервалі. Зменшення інтервалу поблизу піків даних виявилось хорошим вибором для згладжування загального часу затримки. Виходячи з існуючої структури планування Spark Streaming, таймер використовується на початковому етапі для розподілу пакетів. Вони використовували типові потокові тести, щоб продемонструвати, що їхня

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

політика ефективна в обробці нестабільних поточкових вхідних даних, що призводить до обмеження загальної затримки подій.

Алгоритм планування за замовчуванням Storm є статичним і залишається далеким від оптимального. Універсальні рішення планування для надання ресурсів кільком додаткам, що конкурують за хмарні ресурси, такі як Mesos та YARN, припускають, що додаток вже знає обсяг необхідних йому ресурсів і як розподілити ці ресурси внутрішньо. Розробили та впровадили *DRS*, динамічний планувальник ресурсів, який застосовується до систем обробки потоків на основі операторів. Їхній алгоритм враховує кількість операторів у додатку та максимальну кількість доступних процесорів, які можуть бути їм виділені, і намагається знайти оптимальне призначення процесорів, яке призводить до мінімального очікуваного загального часу перебування. Вони оцінили загальний час перебування вхідного сигналу, моделюючи систему як відкриту мережу черг (OQN). Модель продуктивності побудована на основі комбінації однієї з моделей Ерланга та мережі Джексона. Система контролює фактичний загальний час перебування та перевіряє, чи падає продуктивність, чи може система виконати обмеження з меншою кількістю ресурсів, і за необхідності переплановує. Він багаторазово додає один процесор до оператора з максимальною граничною вигодою, доки розрахунковий загальний час перебування не перевищує параметр обмеження реального часу. DRS використовує логіку потокової обробки Storm та демонструє надійну продуктивність, пропонуючи найкращу конфігурацію розподілу ресурсів, навіть коли основні умови теорії черг, яку вона використовує, не повністю виконуються. Загалом, накладні витрати DRS у більшості протестованих випадків становлять менше мілісекунд, що призводить до незначного впливу на затримку системи.

Окрім використання всіх доступних робочих вузлів, планувальник Storm не враховує зв'язки між завданнями, які будуть розміщені на цих вузлах. Міжвузловий та міжпроцесний трафік – це фактори, що суттєво впливають на продуктивність системи. Запропонували динамічний підхід до планування завдань, який враховує зв'язки між завданнями та зменшує трафік між вузлами,

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

призначаючи завдання, що взаємодіють один з одним, одному вузлу або сусіднім вузлам. Топологія отримується шляхом апріорного запису робочого навантаження вузлів та комунікаційного трафіку через комутатори. Вони використовували матричну модель для опису проблеми планування завдань у реальному часі. Їхня процедура обробки намагається зменшити трафік між вузлами через комутатори, знизити навантаження на пропускну здатність та збалансувати робоче навантаження вузлів, вибираючи відповідний хост-вузол, коли виникає тригер (керований вузлом або завданням). Вони оцінили свій алгоритм, розгорнувши власну платформу обробки потоків, та порівняли своє рішення з алгоритмами, вбудованими в Storm та S4, використовуючи балансування навантаження та комунікаційний трафік через комутатори як індикатори. Зі зростанням кількості завдань, що виконуються на цих платформах, балансування навантаження в SpeedStream покращувалося. Більше того, було виявлено менше поточкових даних, що проходять через комутатори, і цей трафік було зменшено, що зменшило навантаження на пропускну здатність кластера.

Також є планувальником, який враховує трафік, працює динамічно та розширює підхід, що враховує топологію, що використовувався в першому. Для цього він контролює ефективність розкладу під час виконання за допомогою журналу продуктивності та адаптує розподіл виконавців до зміни навантаження та відповідно до шаблонів зв'язку програми, щоб зменшити міжвузловий та міжслотовий трафік у кластері. Тим не менш, планувальник не перевіряє весь шаблон зв'язку між виконавцями та розглядає кожну пару окремо. Таким чином, деякі пари, що взаємодіють, можуть опинитися на різних вузлах. Користувацький планувальник може періодично перевіряти, чи можна розгорнути новий, більш ефективний розклад з точки зору міжвузлового трафіку. Затримка обробки події була нижче 20-30%, а міжвузловий трафік нижче 20% відносно планувальника Storm за замовчуванням в обох протестованих топологіях (загальній та тій, що базується на реалістичному наборі).

T-Storm – це ще одна спроба мінімізувати міжвузловий та міжпроцесний

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

трафік. Інформація про робоче навантаження та трафік збирається під час виконання моніторами навантаження для оцінки майбутнього навантаження за допомогою методу прогнозування машинного навчання. Генератор розкладів - періодично зчитує вищезазначену інформацію з бази даних, сортує виконавців у порядку спадання їхнього трафікового навантаження та призначає виконавців слотам. Виконавці з однієї топології призначаються в один слот для зменшення міжпроцесного трафіку. Загальне робоче навантаження виконавців не повинно перевищувати пропускну здатність працівників, а кількість виконавців на слот розраховується за допомогою параметра керування. T-Storm об'єднує працівників та робочі вузли для досягнення кращої продуктивності з ще меншою кількістю робочих вузлів, дозволяє гарячу заміну алгоритмів планування та коригує параметри планування на льоту.

Перевантаження системи також є предметом інтересу. Запропоновано алгоритм динамічного планування розподілу (DAS) для обробки великих потоків даних у мобільних інтернет-сервісах. Автори генерують структуру, яка називається графом потокових запитів (SQG), на основі операторів та зв'язків між відповідним входом та виходом. SQG - це прямий ациклічний граф, а ребро між двома вузлами представляє чергу завдань. Вага ребра - це кількість завдань у черзі. Ребро з мінімальною вагою вибирається для надсилання кортежів, а список буферів встановлюється для зберігання деяких кортежів перед наступним плануванням. Стратегія планування DAS постійно оновлюється кожною логічною машиною окремо. Розділивши загальну проблему планування на загальну підпроблему кожного оператора, накладні витрати зменшуються, а точність підвищується. Автори порівняли DAS із SAMR, алгоритмом динамічного планування, заснованим на MapReduce. Було виявлено, що DAS має коротший час відгуку та займає менше системних ресурсів, головним чином тому, що SAMR потребує постійного розкладання завдань та зчитування інформації про локальну історію, тому задіяно більше дискових операцій.

Еластичність має вирішальне значення в онлайн-середовищі для визначення того, як масштабувати потік даних, що змінюється з часом, та

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

планувати ресурси відповідно до поточної швидкості надходження потоку. Запропонували еластичну систему онлайн-планування (*E-stream*), яка працює з кількома групами дистанційного керування (DAG), використовуючи доступну потужність обчислювальних вузлів та вхідну швидкість потоку даних. *E-stream* кількісно визначає вартість обчислень та зв'язку, зв'язки між вхідним та вихідним потоком вершини та регулює ступінь паралельності вершин у графі, який необхідно масштабувати. Підграфи додатково будуються для зменшення вартості зв'язку між пов'язаними вершинами та часу відгуку DAG. Кожен підграф замінюється логічно еквівалентною вершиною та розглядається як «вершина» на етапі планування. Коли справа доходить до планування, *E-stream* контролює, чи потребують DAG більше ресурсів, та переплановує їх, використовуючи стратегію найпершого часу завершення (EFT) на основі пріоритетів, гарантуючи ступінь справедливості системи. Вихідними даними є послідовність планування для кількох DAG на основі максимальної та мінімальної справедливості з гарантованим часом виконання (*makespan*, загальний час, необхідний для виконання графа). *E-stream* був розроблений на базі Storm. У порівнянні зі Storm, він забезпечив прийнятний час відгуку навіть при використанні 50 DAG, що було коротшим, ніж у планувальника Storm за замовчуванням (приблизно в 3 та 7 разів швидше для різних застосувань). Більше того, ступінь справедливості *E-stream* був кращим, ніж у планувальника Storm.

Зосереджуються на цілях рівня обслуговування (SLO), які необхідно підтримувати у разі непередбачуваних змін навантаження та зниження продуктивності апаратного чи програмного забезпечення. Вони представили Dhalion, систему з можливостями саморегулювання, та впровадили її поверх Heron. Dhalion періодично викликає політику, яка перевіряє стан потокової програми, збираючи метрики з менеджерів метрик Heron, виявляє можливі проблеми (наприклад, повільну обробку, брак ресурсів тощо), діагностує їх та виконує відповідні дії. Було розроблено дві політики. Перша динамічно надає ресурси для максимізації пропускної здатності, враховуючи навантаження вхідних даних (варіації робочого навантаження). Друга приймає як вхід SLO

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

пропускної здатності, відстежує спостережувану пропускну здатність під час роботи топології та автоматично налаштовує програму Heron (налаштовуючи паралелізм споутів або болтів) або забезпечує необхідне виділення ресурсів, якщо це необхідно. У своїх оціночних експериментах вони протестували обидві політики. Перша політика коригувала ресурси топології на льоту, коли виникали піки робочого навантаження та спостерігався зворотний тиск. Друга політика успішно автоматично налаштувала топологію, коли вона досягла стаціонарного стану, коли спостережувана пропускну здатність дорівнювала або перевищувала значення SLO пропускної здатності.

На відміну від більшості підходів, використовували не кластерне середовище, а розподілену систему, розкидану між кількома невеликими центрами обробки даних. Вони реалізували розподілений алгоритм, який фактично використовує мережевий алгоритм планування для Storm. Атрибути QoS (затримка, використання та доступність) вважаються відомими, і вони зосереджені на моделюванні обробки даних між кожною парою вузлів (вартісний простір) та на розміщенні операторів (на основі операторів) у цьому вартісному просторі. Усвідомлення QoS забезпечується QoSMonitor, а алгоритм Вівальді надає точні оцінки між парами вузлів. Нарешті, AdaptiveScheduler отримує цю інформацію, щоб визначити, які виконавці можна ефективно перемістити, щоб остаточно розмістити їх у відповідному вузлі (для підтримки цієї процедури використовується еталонна модель MAPE [83]). Завдяки перегрупуванню операторів на вузли з вищою доступністю, затримка зменшується приблизно на 72% порівняно зі Storm, і майже на 17% менше кортежів довелося надсилати повторно. Більше того, запропонований планувальник забезпечує ефективніше балансування навантаження.

Представлено цілу архітектуру потокового механізму реального часу в багатопроцесорному середовищі, звертаючи увагу на значення швидкості. Запити та деякі їхні характеристики призначаються кластерам, але кожен запит приймається до обробки, якщо його термін виконання може бути дотриманий. Час відповіді системи на запит обчислюється за допомогою функції. Цей

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

алгоритм, що враховує продуктивність, порівнює час виконання кожного запиту з терміном виконання запиту. Алгоритм вибирає запит з найвищим пріоритетом і розподіляє його між відповідним кластером процесорів, використовуючи алгоритм First-Fit на основі коефіцієнта використання (йому потрібно обчислити час виконання кожного запиту). Кожен кластер вибирає запит зі своєї черги очікування, використовуючи алгоритм EDF (Earliest Deadline First - найперший термін). Потім вибраний запит обробляється паралельно за допомогою запропонованого методу диспетчеризації з урахуванням термінів; кортеж обробляється процесором, а потім пересилається найкращому наступному процесору для продовження його обробки на основі терміну виконання завдання. Для оцінки та порівняння цього прототипу зі Storm було перевірено кілька параметрів. Результати показали, що цей підхід перевершує Storm за показниками пропорційного коефіцієнта пропуску дедлайнів (метрика для оцінки систем обробки потоків у реальному часі, ~50% від Storm), затримки кортежів (~66% від Storm) та пропускної здатності (~1,4% від Storm), водночас він має деякі недоліки щодо використання пам'яті (~1,2 від Storm) та втрати кортежів (~1,9 від Storm).

2. 4 Висновок по розділу

У другому розділі досліджено методи та інструменти розробки систем потокової обробки даних. Проведено аналіз сучасних платформ і технологій, які забезпечують збір, передачу та обробку поточкових даних у реальному часі. Розглянуто методологію проектування таких систем із урахуванням вимог до продуктивності, надійності та масштабованості. Особливу увагу приділено методам оптимізації, зокрема ефективному розподілу ресурсів та плануванню обчислень у розподілених середовищах. Отримані результати підтверджують, що правильний вибір архітектурних рішень та інструментів суттєво впливає на ефективність функціонування систем потокової обробки даних.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО РІШЕННЯ

3.1 Порівняльний аналіз систем обробки даних у реальному часі

У цьому розділі ми порівнюємо різні інструменти, що використовуються для обробки потоків у реальному часі, і відповідно до цього порівняння ми оберемо найбільш підходящий інструмент.

Наведене вище порівняння показує, що Storm є найкращим інструментом для обробки потоків у реальному часі, Hadoop виконує пакетну обробку, а Spark здатний виконувати мікропакетування. Storm використовує Spouts та Bolts для поетапної обробки, щоб уникнути невід'ємних накладних витрат затримки, що виникають через пакетування та мікропакетування.

Як ми бачили, в останні роки багато дослідників представили різні підходи та алгоритми планування завдань. Різноманітність кластерів, де відбувалася оцінка вищезгаданих систем, не дозволяє нам проводити безпечні порівняння між ними. Більше того, відмінності в властивих додаткам характеристиках та використовуваних потоках даних (наприклад, їх швидкість передачі, їх реалістичний характер тощо) також є основними перешкодами для безпечних порівнянь. Наша таксономія показує, що існує більше доступних удосконалень, які використовують операторну модель в потоковій обробці, ніж мікропакетну. Однак існує достатньо підходів до планування для систем пакетної обробки, таких як Spark, які також можуть підійти для систем, що використовують мікропакетну модель, але вони не включені до нашого огляду, оскільки вони не були протестовані в потоковій обробці. Також здається, що більшість ефективних рішень щодо планування в потоковій обробці приймаються онлайн або принаймні базуються на попередніх онлайн-знаннях. Спостережувані параметри, які впливають на рішення представлених методів

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

планування, - це головним чином топологія, доступні ресурси та мережа, робоче навантаження та продуктивність системи.

Таблиця 3.1:

Порівняння інструментів обробки великих даних

Критерій	Hadoop	Spark	Storm
Модель вихідного коду	Відкритий (Open source)	Відкритий (Open source)	Відкритий (Open source)
Архітектура	Master/slaves	Master/slaves	Однорангова (Peer)
Модель обробки	Пакетна (Batch)	Мікро-пакетна	У реальному часі (поелементно)
Обробка Big Data	Пакетна	Пакетна та потокова	Потокова
Досяжна затримка	Висока	Кілька секунд (< 1с)	Менше секунди (< 100мс)
Програмування API	Java, Python та Scala	Java, Python та Scala	Будь-яка мова
Гарантована обробка	Exactly-once (рівно один раз)	Exactly-once (рівно один раз)	At least once (щонайменше один раз)
Зберігання даних	Так	Так	Ні
В оперативній пам'яті	Ні	Так	Так
Відмовостійкість	Так	Так	Так
Гарантії порядку	Так	Так	Ні
Інструмент координації	Zookeeper	Zookeeper	Zookeeper

Топологічні проблеми: Виходячи з нашого огляду, коли йдеться про статичні підходи, більшість рішень щодо планування в досліджуваних системах спираються на топологію, яку необхідно виконати. Топологія також відіграє другу за важливістю роль у прийнятті рішень у динамічних підходах (вона посідає друге місце після характеристик робочого навантаження). Коли немає великого навантаження на обробку, пропускна здатність системи значною мірою залежить від затримок мережевого зв'язку. Враховуючи топологію, можна знайти шаблони зв'язку, зменшити міжвузловий та міжпроцесний трафік, і тоді очікується, що пропускна здатність різко зросте. Наприклад, лінійна та зірчаста топології (перевірені в R-Storm) обов'язково передбачають більшу кількість комунікацій, і не дивно, що покращення пропускної здатності є більш значимими порівняно зі Storm, коли ці топології тестувалися, оскільки Storm не враховує міжвузловий та міжпроцесний трафік. Призначення найбільш комунікаційних завдань разом на одному вузлі або стійці є необхідністю, яка відрізняє потокову

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

обробку від пакетної обробки, яка натомість звертає увагу на локальність даних (призначаючи обчислення вузлам, де зберігаються необхідні дані).

Проблеми з ресурсами: обробка великих даних вимагає великої кількості циклів процесора, пам'яті, пропускної здатності мережі та обсягу дискового вводу/виводу. Особливо в потоковій обробці пам'ять набуває вирішального значення. Важливо ефективно планувати завдання, спосіб, який мінімізує час виконання завдань та збільшує використання ресурсів. Усвідомлення ресурсів є загальною потребою як у статичних, так і в динамічних підходах, що досліджуються. Такі стратегії намагаються скористатися перевагами використання вузлів. У GA Storm автори показали, що їхня стратегія працює краще, коли завдання розподіляються між максимальною кількістю ядер, таким чином, ми маємо меншу можливу кількість завдань на ядро. Максимальної продуктивності завдань можна досягти за допомогою мінімального спільного використання процесора, споживаючи при цьому максимальну кількість ядер. Це очевидне припущення, проте воно ставить питання: як стратегія, що враховує ресурси, повністю використовуватиме процесор? У відповідь на це питання більшість стратегій, представлених вище (крім GA Storm), повинні повністю виділити один процесор на компонент топології. Це означає, що в гетерогенному середовищі процесори з більшою обчислювальною потужністю слід планувати для обробки завдань, які потребують більшої обробки, тоді як в гомогенному середовищі планування базується на припущенні, що доступні процесори мають достатню потужність.

Неоднорідність кластерів впливає на рішення щодо статичного планування. У вищезгаданих системах автори зазвичай вказують, чи спрямовані запропоновані схеми на однорідне чи на неоднорідне середовище. У R-Storm вузли, на яких плануються завдання, визначаються функцією відстані, яка базується на доступності ресурсів. Однак стратегія планування була реалізована для однорідних кластерів, де передбачається, що всі процесори мають подібну обчислювальну потужність. У випадках, коли існують різні архітектури процесорів, планування з урахуванням ресурсів не може бути легко застосоване.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

Наприклад, у R-Storm процесор, вибраний з групи доступних процесорів на основі критерію «найближчих» доступних ресурсів, не гарантовано працюватиме належним чином (повна обробка), якщо заздалегідь не відомо, що всі процесори ідентичні. Смірнов та ін. експериментально довели, що пропускну здатність значною мірою визначається типом процесора. Аніелло та ін. мали справу з неоднорідними вузлами та враховували швидкість процесора у своїх прогнозах. Їхня стратегія базувалася на переміщенні кортежів по ланцюжку переходів після того, як вони були відправлені джерелом, доки їх обробка не завершиться сигналом підтвердження. Як приклад переміщення кортежів, якщо виконавець використовує 10% використання процесора на процесорі 1 ГГц і мігрує на вузол з процесором 2 ГГц, використання процесора становитиме 5%. З подібними процесорами таке планування вимагатиме балансування навантаження для збільшення використання. Однак балансування навантаження обов'язково передбачає ретельний вибір кортежів, призначених кожному процесору, та деякі апріорні знання щодо розміру кортежів. Більше того, Рихлі та ін. показали, що їхній підхід до планування з урахуванням ресурсів, найгіршого, стандартного та найкращого варіантів забезпечує однакові результати на однорідній платформі, тоді як значне покращення найкращого планування порівняно з найгіршим та середнім плануванням показано зі збільшенням неоднорідності, оскільки найкращий розклад повністю використовує відмінності в апаратному забезпеченні. Іншою стратегією, що враховує топологію та може працювати на гетерогенній платформі, є P-Scheduler. У P-Scheduler автори використовують однорідну платформу. Однак у цьому випадку їхня головна мета — мінімізувати затримки мережі, призначаючи завдання з високою частотою зв'язку тим самим вузлам. У такій схемі архітектура процесора не має основного значення, проте автори припускають певну неоднорідність швидкостей передачі доступних потоків, щоб вони могли використовувати швидші потоки для передачі кортежів з вищою швидкістю, коли це необхідно. Хоча вони намагаються зменшити час передачі, вони стверджують, що це також зменшить час обробки кортежів.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

Проблеми робочого навантаження: Коли йдеться про динамічні підходи, робоче навантаження, здається, впливає на більшість рішень щодо планування. Характеристики робочого навантаження набувають вирішального значення, коли йдеться про мікропакетні системи обробки, оскільки вони допомагають визначити відповідний розмір пакета для рішень щодо планування. Загалом, ідея вирішення цієї проблеми полягає в тому, щоб спробувати відповідно налаштувати розміри потоків. Коли використовуються менші потоки, відбувається швидша адаптація до змін системи. В системах на основі оператора підходи, що враховують робоче навантаження, можуть допомогти зменшити перевантаження в робочому вузлі. Коли обчислювальних ресурсів машини недостатньо для обробки потреб, її потужність може бути встановлена на частку від її фактичної потужності, щоб запобігти перевантаженню (як це робить T-Storm), але цього недостатньо. Великі піки навантаження призводять до вузьких місць, можливого зворотного тиску та зниження загальної пропускної здатності системи. Крім того, часті методи балансування навантаження та міграції станів можуть збільшити накладні витрати та, як наслідок, затримку. Тим не менш, знижена швидкість планування може призвести до неточності. Коливання робочого навантаження вимагають ретельної обробки.

Еластичність стосується здатності хмари дозволяти сервісу виділяти додаткові ресурси або вивільняти ресурси на вимогу, щоб відповідати робочому навантаженню програми. Тим не менш, без налаштування паралелізму компонентів, пропускна здатність топології досягає межі, вище якої додавання більшої кількості машин не покращить продуктивність. Планування топології серед непотрібної кількості машин може призвести до збільшення затримки зв'язку. Зазвичай користувачам потрібно багато зусиль та кілька налаштувань, щоб визначити ступінь паралелізму на кожному етапі потокового конвеєра, оскільки користувачі мають обмежені знання про поведінку системи під час виконання. Такі системи, як E-Stream, відстежують швидкість потоку, і в разі її збільшення створюють нові екземпляри для певної кількості вершин. Саморегульовані потокові системи, такі як Dhalion, також допомагають у цьому

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

напрямку, оскільки вони враховують специфікації потокових програм, такі як швидкість вхідних даних, а також політики, що визначають мету користувача для динамічного вирішення проблеми. Інший цікавий підхід полягає в розділенні загальної проблеми планування на підзадачі кожного оператора (як це робить DAS), щоб зменшити накладні витрати та підвищити точність.

Можна запропонувати ще два потоково-орієнтовані підходи: (1) Використання пріоритетів: Коли існує певний механізм для призначення пріоритетів певним потокам, ми продовжуємо обробляти ці потоки (таким чином їхня обробка не відкладається) і переміщуємо менш важливі потоки до нових вхідних вузлів для подальшої обробки, та (2) Відкидання менш важливих потоків: У такому рішенні менш важливі кортежі можна відкинути, щоб зменшити затримки. Завдання тут полягає в тому, щоб вирішити, скільки кортежів потрібно відкинути, щоб зберегти точність. Цю проблему можна звести до задачі оптимізації оптимальної кількості кортежів, які потрібно відкинути, щоб зберегти певний рівень точності.

Еластичність стає ще складнішою в середовищах потокової обробки, де обчислення, як правило, є станомісткими. Гарантування відмовостійкості та динамічного балансування навантаження для операторів зі станом вимагає передачі станів, що є досить громіздким. Як ми бачили, Flink використовує спеціальні механізми для обробки станів.

Проблеми з продуктивністю: Продуктивність системи можна контролювати під час виконання або отримувати зі статистики старих виконань чи бенчмарків. Моніторинг характеристик продуктивності, таких як загальний час перебування (DRS), час виконання запитів (підхід Сафаєї) або пропускна здатність (A-Scheduler), може допомогти системам або відповідно налаштувати свою поведінку в режимі онлайн, або побудувати моделі для ефективного прийняття рішень. Планувальник GA Storm також є прикладом стратегії, що враховує продуктивність, але його планування є статичним. Спільне використання процесора використовується для максимізації продуктивності, але для стратегій, що враховують продуктивність, існує висока залежність від

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

однорідності середовища. В однорідних середовищах стратегія GA Storm не покращує продуктивність, оскільки розрахункова пропускна здатність усіх виконавців здається подібною, тоді як виконавці, що розвиваються, генеруються з їхніх батьків.

Моніторинг продуктивності системи набуває критичного значення за наявності збоїв. Різні методи *відмовостійкості* важливі для ефективного використання наданих ресурсів. Потоки повинні мати можливість відновлюватися після збоїв, щоб потокові програми працювали без проблем. У системах потокової обробки відмовостійкість є широкою галуззю досліджень, і вона привертає все більше уваги. У цьому абзаці ми лише зробимо деякі корисні зауваження щодо відновлення після збоїв досліджуваних DSMS, використовуючи відповідні методи планування. Особливо, коли йдеться про динамічні стратегії, де рішення приймаються під час виконання, стратегії відновлення після збоїв мають першочергове значення. Як показано в літературі, на вибір стратегії значною мірою впливають ресурси та час, необхідний для відновлення. Як правило, описані вище стратегії планування обробляють відновлення після збоїв одним із наступних способів:

(a) Відновлення на основі контрольних точок. У цій стратегії вершина періодично перевіряє свій потік. Коли вершина виходить з ладу, вона завантажує останню контрольну точку та відновлює виконання. Встановлення контрольних точок створює накладні витрати під час нормального виконання та не є ідеальним рішенням для вузлів з високим навантаженням. Це пов'язано з тим, що існування багатьох контрольних точок обов'язково вимагає великих структур даних, що збільшує їх загальну вартість. Один зі способів зменшення витрат - розміщувати контрольні точки лише в кінці потоків. Однак, коли потік великий, у разі збою потрібно відтворити занадто багато записів. Інший варіант - обробляти збої в попередньому плануванні, але немає гарантії, що таке рішення є оптимальним у системі реального часу.

(b) Відновлення на основі реплікації. Інша стратегія полягає в тому, щоб кілька екземплярів одного й того ж вузла запущено одночасно: вони можуть бути

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

підключені до тих самих вхідних і вихідних потоків. Коли один екземпляр виходить з ладу, відновлення досягається шляхом отримання того самого знімка з іншого екземпляра. Очевидно, що в цьому випадку немає потреби використовувати великі бази даних, але є три недоліки: а) існування кількох екземплярів, б) вибір відповідних вузлів, які запускають ці екземпляри, що іноді вимагає ретельного відбору залежно від таких факторів, як відстань, та в) складність прогнозування потоків, які потрібно буде реплікувати заздалегідь, тому планування реплік є ще одним питанням, яке необхідно враховувати. У таких випадках слід розробити ефективні підходи для розрахунку мінімальної реплікації, щоб виконати вимогу надійності.

Як бачимо, продуктивність DSMS залежить від багатьох факторів. Планування потоків, здається, є більш ефективним, коли воно працює онлайн, включаючи безперервні, поступові рішення щодо планування, які враховують зміни в швидкості надходження потоків та використанні ресурсів кластера. Ємність та можливості базового кластерного середовища, в якому відбувається обробка, завжди обмежуватимуть його продуктивність. Цілком очевидно, що планування завдань є критичним фактором для продуктивності систем. Досягнення низької затримки та високої пропускну здатності обробки потоків, коли ресурси можуть бути масштабовані, вимагає ефективного планування завдань, яке зменшує кількість міграцій завдань, розподіляє кількість залежних та незалежних завдань майже оптимальним чином для зменшення загального часу обчислення завдання та покращує використання ресурсів кластера.

3.2 Архітектурні підходи до побудови систем реального часу

У цій роботі ми представляємо короткий огляд деяких архітектур обробки даних у реальному часі, а саме Лямбда та Каппа.

Лямбда-архітектуру запропонував Натан Марц. Ця архітектура поєднує переваги моделей обробки, пакетної обробки та обробки в реальному часі, щоб забезпечити кращі результати з низькою затримкою.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

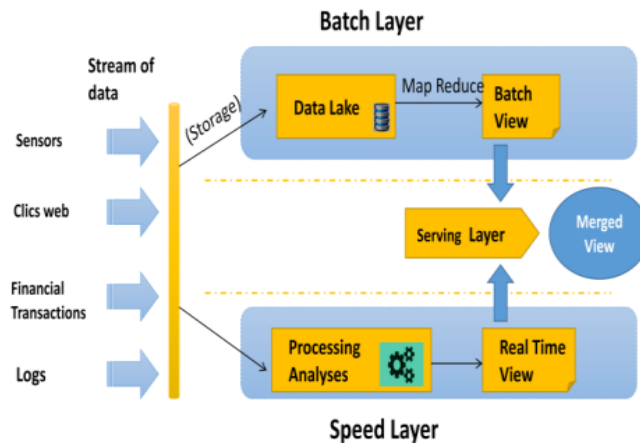


Рисунок 3.1 - Архітектура Lambda

На рисунку 3.1 показано базову архітектуру лямбда-доменів [18]. Вона поділена на три рівні:

- Пакетний шар – керує історичними даними та перераховує результати.
- Швидкісний шар – отримує дані, що надходять, та виконує поступові оновлення результатів пакетного шару.
- Сервісний рівень – дозволяє здійснювати різні запити результатів, надісланих з пакетного та швидкісного рівнів.

Усі нові дані надсилаються як на пакетний рівень, так і на рівень швидкості. Пакетний рівень відповідає за зберігання основного набору даних і послідовно обчислює представлення цих даних за допомогою алгоритму MapReduce. Результати пакетного рівня називаються «пакетними представленнями».

Обслуговуючий рівень індексує попередньо обчислені представлення, створені пакетним рівнем. Це масштабована база даних, яка замінює нові пакетні представлення, коли вони стають доступними. Через затримку пакетного рівня результати, доступні з обслуговуючого рівня, завжди застарівають на кілька годин. Обслуговуючий рівень може бути реалізований за допомогою NoSQL-технологій, таких як HBase, Apache Druid тощо.

Рівень швидкості компенсує високу затримку оновлень обслуговуючого

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

рівня. Роль цього рівня полягає в обчисленні в режимі реального часу даних, які не були враховані в останньому пакеті пакетного рівня. Він створює представлення в режимі реального часу, які завжди актуальні, та зберігає їх у швидкому сховищі. Рівень швидкості може бути реалізований за допомогою технологій потокової передачі даних, таких як Apache Storm або Spark Streaming.

Однак, лямбда-архітектура має деякі обмеження; по-перше, це бізнес-логіка, яка реалізується двічі на рівні реального часу та пакетної обробки. Розробникам потрібно писати один і той самий код на обох рівнях. Друге зауваження полягає в необхідності опанування більшої кількості фреймворків. І, нарешті, існують простіші рішення, коли потреба менш складна.

Архітектура Карра [19], як її описали у LinkedIn у 2014 році, є шаблоном архітектури програмного забезпечення. Карра — це спрощення архітектури лямбда, що означає, що вона схожа на систему архітектури лямбда з видаленою системою пакетної обробки. [19] Канонічне сховище даних у системі архітектури Карра — це незмінний журнал, який можна лише додавати. З журналу дані передаються через обчислювальну систему та надходять до допоміжних сховищ для обслуговування.

Фактично, і навіть більше, ніж лямбда-архітектура, архітектура Карра не дозволяє постійного зберігання даних. Вона більше орієнтована на їх обробку. Хоча й більш обмежена, архітектура Карра залишає певну свободу у виборі реалізованих компонентів.

На відміну від лямбда-архітектури, яка використовувала два різні шляхи коду для пакетного та швидкісного шарів, Карра використовує лише один шлях коду для двох шарів, що зменшує складність системи[20]. Перевагою архітектури Карра є можливість для користувачів розробляти, тестувати, налагоджувати та експлуатувати свої системи на основі єдиної платформи обробки. На малюнку нижче представлено архітектуру Карра:

Наведена нижче діаграма представляє коротке порівняння двох архітектур, як обговорювалося раніше, а саме Lambda та Карра, за певними критеріями.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

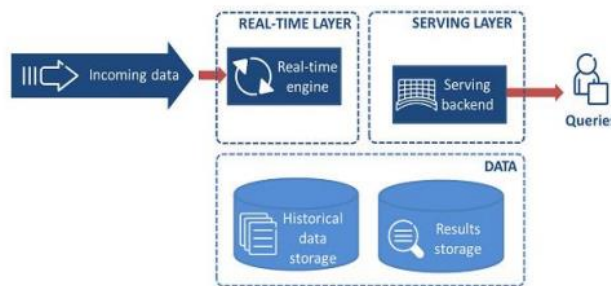


Рисунок 3.2 - Архітектура Карра

У сучасному середовищі великі дані здебільшого описуються своїми основними характеристиками, такими як великий обсяг, різноманітність та неоднорідність даних, у яких усі процеси вимагають нових технологій та методологій для вилучення з них цінної інформації. На цьому етапі основні проблеми, що обговорювалися в цьому дослідженні, були згруповані за трьома основними категоріями: характерні проблеми, проблеми обробки та проблеми управління, як показано на рис. 3.2. У решті роботи проблеми обговорюються більш детально та уточнюються під вісьмома окремими заголовками.

Таблиця 3.2:

Порівняння архітектур обробки даних у реальному часі

Критерій	Lambda-архітектура	Карра-архітектура
Архітектура	Незмінна (immutable)	Незмінна (immutable)
Відмовостійкість	Так	Так
Масштабованість	Так	Так
Постійне зберігання	Так	Ні
Рівні (Layers)	Пакетний, реального часу та сервісний	Реального часу та сервісний
Дані для обробки	Пакетні та потокові	Потокові
Гарантії обробки	"Так" у пакетному; приблизні у потоковому	Exactly-once (рівно один раз)
Парадигма переобробки	У кожному пакетному циклі	Тільки при зміні коду
Реальний час	Неточний	Точний

А. Обсяг, різноманітність та неоднорідність

Неструктуровані дані можуть містити всі типи даних у великому обсязі та різних форматах. Ці дані можуть включати різні платформи обміну інформацією, такі як соціальні мережі, форуми, електронна пошта, чати, онлайн-спільноти, онлайн-магазини тощо. Крім того, дані можуть містити комбінацію різних форматів файлів, таких як зображення, відео, аудіо та текст. Аналітичні процеси обробки даних є складнішими та дорожчими через цю складну структуру даних. Ця неоднорідна та складна структура великих даних є серйозною проблемою, яку необхідно подолати в процесі аналізу великих даних.

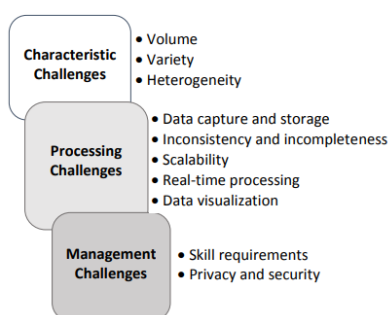


Рисунок 3.3 - Огляд проблем, пов'язаних із великими даними

Наявні традиційні методи обробки даних застосовуються лише до однорідних та структурованих даних. Тому, враховуючи складну структуру великих даних, потрібна розробка інноваційних методів обробки даних. Крім того, гетерогенна природа великих даних повинна бути ретельно структурована та перетворена на однорідні дані для обробки традиційними методами аналізу. Системи обробки даних та джерела великих даних можуть бути об'єднані на спільній платформі, і таким чином дані можуть бути надані в більш відповідній структурі для цих процесів аналізу.

В. Збір та зберігання даних

У сучасному цифровому середовищі щодня генерується 2,5 квінтільйона байтів даних, і цей обсяг постійно зростає день у день. Збір та зберігання даних у таких великих обсягах може виконуватися за допомогою різних технічних процесів, які потребують значних витрат. Кілька записів журналів періодично видаляються в різних джерелах даних через вартість необхідних систем

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

зберігання. Поява моделей великих даних змінює традиційні методології збору та зберігання даних, системи доступу до даних та реалізації на основі даних. Оскільки існуючі технології зберігання не можуть забезпечити необхідну продуктивність з точки зору збору, обробки та зберігання даних у режимі реального часу, обробка та аналітика великих даних вимагає високошвидкісних шаблонів доступу до вводу/виводу (I/O), оскільки ці процеси неможливо досягти за допомогою сучасних жорстких дисків (HDD) на основі випадкового доступу до вводу/виводу. У зв'язку з цим технології твердотільних накопичувачів (SSD) та фазозмінної пам'яті (PCM), що використовують послідовні шаблони вводу/виводу, є переважно кращими для зберігання великих даних, ніж традиційні HDD. Щоб подолати цю проблему, для досягнення передових процесів зберігання почали використовувати інноваційні технології зберігання, такі як пряме підключення сховища (DAS), мережеве підключення сховища (NAS) та мережа зберігання даних (SAN). Крім того, розподілені, паралельні та хмарні центри обробки даних забезпечують новий підхід до збору та зберігання даних у схвалений спосіб.

с. Непослідовність та неповнота

Великі дані – це комбінація даних, отриманих з різних джерел у різних форматах. У цьому випадку під час обробки даних, отриманих з різних джерел, може виникнути невідповідність та неповнота даних, тому обробка даних може бути не досягнута на бажаному рівні. Враховуючи гетерогенну природу великих даних, спочатку слід підвищити надійність даних, зібраних з різних джерел, шляхом перевірки їх як у часі, так і в просторі. Після цього слід визначити загальну однорідну структуру даних, а дані, отримані з різних джерел, слід об'єднати на монотипній основі. Також процес передачі даних необхідно регулярно перевіряти, щоб переконатися, що процес завершений. Таким чином, помилки невідповідності та неповноти можна контролювати під час обробки великих даних.

д. Масштабованість

Останнім часом оптимізація швидкозростаючих даних стає складним

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

процесом для забезпечення масштабованості, що є надзвичайно важливим завданням, яке необхідно вирішити в сучасних системах баз даних. Вважається, що масштабована система є масштабованою, незалежно від того, чи зростає продуктивність та потужність системи після встановлення обладнання чи програми. Масштабованість великих даних базується на основах масштабування баз даних від одного вузла до великих кластерів [13]. Одним із процесів, що використовуються більшістю основних програм систем керування базами даних (СКБД), є розділення великих таблиць на основі масивів значень у ключовому полі. Таким чином, бази даних можна масштабувати на кластері окремих серверів великих даних. Крім того, з появою 64-бітних багатоядерних процесорів постачальники СКБД зосередилися на багатопотокових реалізаціях, які суттєво збільшують потужність обробки транзакцій. Мережі зберігання даних (SAN), мережеві сховища (NAS) та сховища з прямим підключенням (DAS) разом зі швидкими локальними мережами розширюють можливості структур великих баз даних та розподілених обчислювальних процесів. У масштабуванні даних використання хмарних обчислювальних процесів з кожним днем зростає. Як наслідок, забезпечення синхронізації між хмарними обчислювальними процесами є одним із основних завдань обробки великих даних.

Е. Обробка в режимі реального часу

Зі збільшенням обсягу та різноманітності даних своєчасність обробки великих даних стає все складнішим завданням. Своєчасність можна визначити як часову затримку між генеруванням та отриманням даних. Дані повинні бути отримані в цей період для досягнення ефективної аналітики. Наприклад, у системі моніторингу дорожнього руху, яка миттєво спостерігає за мільйонами транспортних засобів, пошук альтернативних шляхів та розрахунок часу прибуття вимагають обробки даних у режимі реального часу. У цьому відношенні будь-яка затримка або неправильний розрахунок може призвести до неправильного напрямку руху швидкої допомоги, і в цьому аспекті своєчасність має життєво важливе значення для людського життя. Таким чином, своєчасність є одним з найважливіших показників якості даних через зростаючий попит на

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

системи підтримки прийняття рішень у режимі реального часу. При обробці в режимі реального часу дуже важливо забезпечити необхідний час для виконання необхідних операцій з потоками даних великого обсягу. Крім того, своєчасність великих даних може полегшити обробку потоків подій, щоб забезпечити прийняття рішень у режимі реального часу. Різноманітні набори даних, згенеровані з різних джерел даних, слід об'єднати в аналітичну платформу, щоб мінімізувати потенційні часові затримки для обробки в режимі реального часу.

Ф. Візуалізація даних

Візуалізація даних використовується для чіткого та ефективного представлення інформації користувачам за допомогою різних методів представлення, таких як графіки, таблиці, діаграми та анімація. Користувачам може бути корисно розуміти та інтерпретувати великі обсяги оброблених даних простішим способом. Візуалізація великих даних не така проста, як традиційні невеликі набори даних. У складних та великомасштабних наборах даних процес візуалізації вимагає багатьох інноваційних підходів та компонентів. Визначення найбільш підходящої моделі представлення даних має вирішальне значення для досягнення зрозумілої візуалізації. Розвиток традиційних підходів до візуалізації є відносно значним, але не на бажаному рівні. У візуалізації великомасштабних даних для візуалізації та представлення даних пропонується багато генеративних методів, таких як вилучення ознак та геометричне моделювання. Таким чином, підходи до візуалізації даних слід переробити з урахуванням тенденцій та вимог великих даних. По-перше, візуалізація великих даних повинна забезпечувати ефективний огляд, а потім гнучке масштабування та фільтрацію. Процес візуалізації слід виконувати так, щоб зробити великомасштабні дані більш керованими, зрозумілими та функціональними.

Г. Вимоги до навичок

Індустрія великих даних – це ринок праці, що розвивається та інноваційний, на якому трудові ресурси ефективно використовуються. У цьому конкурентному середовищі затребувані навички, компетенції та вимоги постійно змінюються та розвиваються. Від фахівців з великих даних очікується широкий

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

спектр знань та навичок, що відповідають вимогам ринку праці у сфері великих даних. З цієї причини фахівці з великих даних повинні завжди вдосконалювати свої знання та навички, оновлюючи їх до сучасного рівня. З розвитком індустрії великих даних та зростанням потреб бізнесу, щодня відкриваються нові кар'єрні можливості для фахівців з великих даних, і тому дефіцит кваліфікованої робочої сили постійно зростає в наші дні. Дефіцит фахівців з великих даних стане значним викликом для задоволення потреб ринку, що розвивається, у цій галузі. Згідно з останньою статистикою, до 2018 року лише у Сполучених Штатах може виникнути дефіцит від 140 000 до 190 000 співробітників з глибокими аналітичними знаннями та навичками, а також 1,5 мільйона менеджерів, аналітиків та спеціалістів, які використовують результати аналітики великих даних для прийняття ефективних рішень [14]. У цій галузі, що розвивається, підготовка спеціалістів з великих даних із сильним технічним досвідом є надзвичайно складним завданням, яке необхідно вирішити.

н. Конфіденційність та безпека

Деяка конкретна інформація, така як інформація про здоров'я людини, членство в онлайн-покупках, номери банківських рахунків та профілі в соціальних мережах, потребує високого рівня захисту з точки зору конфіденційності та безпеки. У зв'язку з цим найважливішою загрозою для безпеки персональної інформації є те, що персональні дані нерегулярно накопичуються численними платформами соціальних мереж. Безсумнівно, конкретні ресурси великих даних повинні бути захищені законами та нормативними актами з точки зору конфіденційності та безпеки. Міжнародна корпорація даних (IDC) рекомендувала п'ять рівнів підвищення безпеки: конфіденційність, дотримання вимог, зберігання, конфіденційність та блокування. Приватна персональна інформація особи в поєднанні з її спільними даними надає важливі факти та деталі про особу. Виявлення конкретної персональної інформації особливо сприяє кримінальним розслідуванням. З іншого боку, люди можуть не бажати, щоб інші поширювали або використовували їхню персональну інформацію. З цієї причини, у сфері

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

конфіденційності та безпеки даних, що використовуються в аналітиці великих даних, необхідні додаткові дослідження та нормативні акти, які виявляють необхідні рівні безпеки.

3.3 Розробка та обґрунтування запропонованої архітектури системи

Відповідно до архітектур та платформ, представлених у попередніх абзацах, ми представили різні переваги та недоліки кожної з цих архітектур, і відповідно до цієї інформації ми розробили нову архітектуру з відкритим вихідним кодом, яка враховує кілька критеріїв, серед яких обробка великих даних у режимі реального часу з високою швидкістю. Вона також дозволяє необмеженій кількості користувачів створювати багато нових та інноваційних функцій і вносити кілька покращень.

Ця архітектура повинна приймати-фільтрувати-аналізувати та обробляти вхідні потоки даних з низькою затримкою, тому система повинна реагувати досить швидко, і це залежить від використовуваної архітектури обробки (іскра, шторм тощо) або розміру даних та складності виконуваних обчислень. З іншого боку, необхідно враховувати, як вибрати найефективніший інструмент; він має бути простим у використанні, щоб не створювати для користувачів, будь то аналітики чи розробники, проблем з інфраструктурою.

Ідеально, нам потрібна архітектура, яка дозволяє досить легко переходити до масштабування та візуально змінювати розподіл ресурсів. Крім того, щойно налаштовані ресурси повинні безперешкодно приєднуватися до кластера та обробляти зміни навантаження або трафіку, не перериваючи обробку поточкових даних у всьому світі.

І нарешті, архітектура реального часу повинна забезпечувати візуалізацію даних у реальному часі. Вона повинна дозволяти динамічне створення інформаційних панелей, користувацької графіки та розширень інтерфейсу користувача.

На рисунку 3.4 представлено як традиційну архітектуру великих даних,

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

так і запропоновану архітектуру. Традиційна архітектура містить три рівні, а саме: зберігання, обробку та аналіз, тоді як запропонована нами архітектура представлена наступним чином.

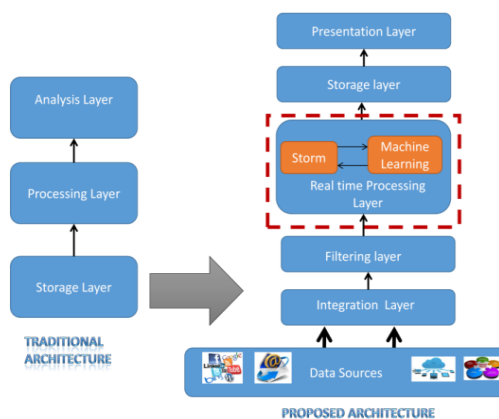


Рисунок 3.4 - Пропонована архітектура

Дані надходять з різних пристроїв та обладнання, таких як датчики, мережі, кіберінфраструктура, веб, електронна пошта, соціальні мережі та багато іншого. Ці дані, які надходять у вигляді потоку з різних джерел з високою швидкістю, отримуються рівнем інтеграції за допомогою набору інструментів та функцій (наприклад, Apache Kafka). Після отримання дані фільтруються за допомогою операцій вилучення-перетворення-завантаження (ELT) (наприклад, PIG). Іншими словами, дані очищаються, а їхня якість аналізується тощо. Цей рівень фільтрації служить для підготовки даних до рівня обробки в реальному часі, який має на меті обробку даних у реальному часі з дуже низькою затримкою. Як показано на рисунку 3.4, на цьому рівні використовуються дві технології, а саме Storm, яка є інструментом для обробки в реальному часі, та машинне навчання. Використання машинного навчання на цьому рівні дозволяє архівувати дані. Його метою є візуалізація попередніх тенденцій за допомогою методу запиту/відповіді на подібні вхідні дані. ML постійно навчається на нових даних, що надходять, що полегшує обробку. Storm, з іншого боку, також використовується на цьому рівні для обробки даних у режимі реального часу. Він використовує поняття топології, яка являє собою мережу Spout та Bolt. Як

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

зазначалося раніше, потоки надходять від Spout, який транслює дані, що надходять із зовнішніх джерел у топології Storm. Щодо Bolts, ми можемо реалізувати деякі функції, такі як функції, фільтри, об'єднання, агрегації тощо. Отже, функцію Map можна реалізувати в Bolt, щоб позначати слова в потоці. Отриманий потік, що надходить від Bolt 'Map', перетікає в наступний Bolt, який реалізує функцію 'Reduce' для агрегації слів у числа.

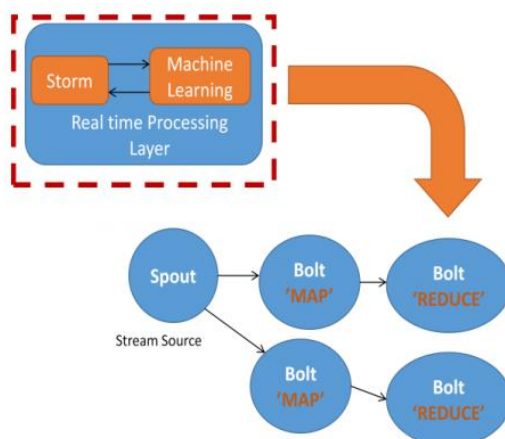


Рисунок 3.5 - Рівень обробки в режимі реального часу

Після завершення обробки дані зберігатимуться або в NoSQL-базі, або безпосередньо в розподіленій файловій системі, такій як HDFS, і, нарешті, рівень візуалізації представить користувачеві результат кінцевих даних у потоковому режимі.

Реалізували сучасний стан різних концепцій, що призвело до проведення ретельного порівняння інструментів обробки потоків даних. Головною метою цього порівняння є показати, що архітектура великих даних базується на пакетній обробці, яка не може обробляти дані в режимі реального часу. Завдяки цьому ретельному порівнянню, Storm було обрано як інструмент для обробки даних, оскільки це програмне забезпечення з відкритим вихідним кодом, яке дозволяє обробку в режимі реального часу з дуже низькою затримкою.

Дані, отримані в режимі реального часу, можуть стати надто цінними в момент їх надходження та підтримки прийняття цінних рішень. З'явилося кілька хмарних систем, які дозволяють розподілену обробку потоків великих даних. У

цій роботі представлені механізми, що використовуються провідними системами управління потоками (DSMS) для вирішення проблем, що виникають при обробці потоків у контексті великих даних. Ми також описали таксономію, яка може полегшити порівняння різних функцій, що пропонуються фреймворками для обробки потоків. На основі цієї таксономії ми надали огляд чотирьох фреймворків для обробки потоків з відкритим кодом. Наше дослідження дає уявлення про фактори, які необхідно враховувати під час вибору платформи, враховуючи конкретний випадок використання. Наприклад, Flink є хорошим вибором, якщо потрібна складна обробка потоків. Однак Spark Streaming є більш зрілим проектом і має більшу спільноту. Storm також є зрілим проектом і може забезпечити кращу затримку з меншою кількістю обмежень, але не може гарантувати узгодженість станів. Він також пропонує лише основні будівельні блоки для побудови топології для користувачів, тоді як Spark і Flink надають API вищого рівня як декларативні системи. Системи потокової передачі майже в реальному часі конкурують одна з одною за кількома факторами, але поки що немає явного переможця. Кожна зі згаданих тут платформ має свої переваги та недоліки. Розуміння сильних та обмежених сторін вищезгаданих фреймворків разом з потребами користувачів полегшує вибір відповідного рішення. Складність поточкових обчислень та різноманітність робочих навантажень створюють труднощі для порівняльного аналізу цих систем, але невеликі зміни в параметрах конфігурації або в доступній інфраструктурі можуть змінити результати. Отже, необхідні подальші дослідження для тестування та оцінки розподілених платформ обробки потоків у різних хмарних експериментах, використовуючи не лише критерії, що стосуються продуктивності, але й безпеки або інтеграції з іншими інструментами. Великі та активні спільноти, які підтримують ці проекти обробки, продовжують впроваджувати інновації, але можуть отримати користь від подальших пропозицій та вдосконалень цих інструментів.

Продуктивність системи обробки потоків залежить від багатьох факторів. Планування завдань перетворилося на критичний фактор, який може суттєво

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

впливати на продуктивність хмарних фреймворків. Механізми, що використовуються вищезгаданими системами, не є дуже складними та не враховують базову доступну інфраструктуру, яка в деяких випадках може бути неоднорідною. Таким чином, планування завдань стало предметом інтересу для багатьох дослідників. Однак, наскільки нам відомо, немає великого дослідження підходів до планування завдань для фреймворків обробки потоків великих даних, —яке б класифікувало та обговорювало представлені підходи. Ми проаналізували 22 різні підходи та визначили найважливіші фактори, що впливають на рішення запропонованих планувальників, а саме топологію, ресурси, продуктивність системи та робоче навантаження. Ми показали, як рішення, засновані на цих факторах, впливають на розглянуті системи. Наше опитування дозволило нам класифікувати проблеми планування за різними категоріями. Більшість статичних підходів спрямовані на забезпечення початкової стратегії розміщення, щоб забезпечити мінімальну роботу в кластері. Однак зміни робочого навантаження та системних умов вимагають використання методів планування під час виконання, які допомагають зменшити дисбаланс у використанні ресурсів та покращують продуктивність системи обробки потоків. Наше опитування виявляє деякі обмеження, і ми можемо окреслити деякі можливі напрямки розвитку на майбутнє. Поточна таксономія не включає комерційно доступні платформи обробки потоків і спирається лише на платформи обробки потоків з відкритим кодом, документація та вихідний код яких є у вільному доступі. Існують також більш перспективні фреймворки, такі як Apache Apex, нативна платформа Hadoop YARN, яка об'єднує потокову та пакетну обробку. Останнім часом існує ініціатива щодо об'єднання різних пакетних та потокових обробок-двигуни обробки даних на спільній основі. Наприклад, Apache Beam можна використовувати для написання API незалежно від фреймворку, а потім запускати той самий код на будь-якому двигуні. Такі ініціативи та їхні механізми можна було б розглянути для майбутніх досліджень. Більше того, більшість представлених підходів оцінюються на основі таких показників, як затримка, пропускна здатність та відмовостійкість, але що стосується відмовостійкості, то

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

все ще проводяться широкі дослідження. Існує лише кілька робіт, які розраховують лише кількість втрачених кортежів. Аналіз ризиків та витрат платформ обробки потоків з використанням різних методів планування також може бути цікавим у майбутньому, оскільки вартість відіграє важливу роль в угодах про рівень обслуговування (SLA) між постачальниками послуг та клієнтами.

3.4 Висновки по розділу

У третьому розділі виконано порівняльний аналіз систем обробки даних у реальному часі та досліджено архітектурні підходи до побудови високопродуктивних програмних рішень. На основі проведеного аналізу було розроблено та обґрунтовано архітектуру програмної системи для обробки поточкових даних. Запропоноване рішення враховує вимоги до масштабованості, відмовостійкості та мінімізації затримок під час обробки інформації. Результати дослідження показали, що використання сучасних технологій потокової обробки дозволяє забезпечити ефективне функціонування систем у режимі реального часу та створює основу для подальшого розвитку високонавантажених інформаційних платформ.

					БР.ІІІ – 36.00.00.000 ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено та обґрунтовано підходи до розробки програмного рішення для обробки поточкових даних у реальному часі, що є комплексним процесом, спрямованим на створення високопродуктивної, масштабованої та відмовостійкої системи. У роботі проаналізовано сучасні концепції обробки великих даних і поточкових обчислень, визначено ключові вимоги до систем реального часу, а також досліджено існуючі інструменти й архітектурні підходи. Отримані результати дозволили сформуванню ефективної моделі обробки потоків даних, що відповідає сучасним вимогам до швидкості, надійності та гнучкості.

На початковому етапі було проведено аналіз предметної області, що включав визначення характеристик великих даних та особливостей потокової обробки. Було встановлено, що ключовими викликами є забезпечення низької затримки обробки, масштабованість системи, стійкість до збоїв та здатність працювати з безперервними потоками даних. Визначено функціональні та нефункціональні вимоги, які стали основою для подальшого проєктування системи.

У другому розділі досліджено методи та інструменти розробки систем потокової обробки даних. Особливу увагу приділено сучасним фреймворкам та платформам, що підтримують обробку даних у реальному часі, а також підходам до оптимізації продуктивності, розподілу ресурсів і планування обчислювальних задач. Розглянуто методологічні аспекти побудови таких систем, що дозволило сформуванню узагальненого підходу до їх розробки.

Практична частина роботи була спрямована на аналіз існуючих архітектур обробки поточкових даних та розробку власної архітектури системи. Було проведено порівняння популярних рішень, визначено їхні переваги та недоліки, а також обґрунтовано вибір оптимальної архітектури. Запропонована система забезпечує ефективну обробку поточкових даних у режимі реального часу, підтримує горизонтальне масштабування та має механізми відновлення після збоїв.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

У процесі дослідження було також приділено увагу питанням оцінки ефективності системи, включаючи аналіз продуктивності, затримок обробки та використання ресурсів. Отримані результати підтвердили доцільність використання сучасних підходів до потокової обробки даних і ефективність запропонованої архітектури.

Загалом, розроблене програмне рішення відповідає поставленим цілям та вимогам: воно забезпечує обробку великих обсягів даних у реальному часі, має високу продуктивність і масштабованість, а також є надійним у роботі. Перевагами підходу є гнучкість архітектури, можливість інтеграції з різними джерелами даних і підтримка сучасних технологій обробки потоків. Водночас існують певні обмеження, зокрема складність налаштування системи та необхідність оптимізації при високих навантаженнях.

У перспективі можливим є розширення функціональності системи шляхом впровадження інтелектуального аналізу даних, використання методів машинного навчання для прогнозування подій у потоках даних, а також інтеграції з хмарними платформами для підвищення доступності та масштабованості рішення.

					БР.ІІ – 36.00.00.000 ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Apache Kafka Documentation. (2025). kafka.apache.org.
<https://kafka.apache.org/documentation/>
2. Apache Flink Documentation. (2025). flink.apache.org.
<https://flink.apache.org/docs/>
3. Apache Spark Streaming Guide. (2025). spark.apache.org.
<https://spark.apache.org/docs/latest/streaming-programming-guide.html>
4. Azure Stream Analytics Documentation. (2025). learn.microsoft.com.
<https://learn.microsoft.com/en-us/azure/stream-analytics/>
5. Google Cloud Dataflow Documentation. (2025). cloud.google.com.
<https://cloud.google.com/dataflow/docs>
6. Amazon Kinesis Data Streams Developer Guide. (2025).
aws.amazon.com. <https://docs.aws.amazon.com/streams/latest/dev/>
7. Kleppmann, M. (2017). Designing Data-Intensive Applications.
oreilly.com. <https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/>
8. Stonebraker, M., et al. (2015). The 8 Requirements of Real-Time Stream Processing. ACM SIGMOD Record, 44(1), 42–47.
<https://doi.org/10.1145/2723372.2742797>
9. Akidau, T., et al. (2015). The Dataflow Model. Proceedings of VLDB, 8(12), 1792–1803. <https://doi.org/10.14778/2824032.2824076>
10. Karau, H., Warren, R. (2017). High Performance Spark. oreilly.com.
<https://www.oreilly.com/library/view/high-performance-spark/9781491943199/>
11. Kreps, J. (2014). Questioning the Lambda Architecture. Confluent Blog.
<https://www.confluent.io/blog/questioning-the-lambda-architecture/>
12. Marz, N., Warren, J. (2015). Big Data: Principles and Best Practices of Scalable Systems. manning.com. <https://www.manning.com/books/big-data>

					БР.ІІІ – 36.00.00.000 ІІЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

13. Akidau, T. (2018). Streaming Systems. oreilly.com.
<https://www.oreilly.com/library/view/streaming-systems/9781491983867/>
14. Microsoft Azure Architecture Center. (2025). Streaming architectures.
<https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/stream-processing>
15. Google Cloud Architecture Framework. (2025). cloud.google.com.
<https://cloud.google.com/architecture/framework>
16. AWS Big Data Blog. (2025). Streaming Data Solutions. aws.amazon.com.
<https://aws.amazon.com/blogs/big-data/>
17. Confluent Platform Documentation. (2025). docs.confluent.io.
<https://docs.confluent.io/>
18. IBM Streaming Analytics Documentation. (2025). ibm.com.
<https://www.ibm.com/docs/en/streaming-analytics>
19. DataBricks. (2025). Structured Streaming Guide. databricks.com.
<https://docs.databricks.com/structured-streaming/>
20. NIST. (2024). Big Data Interoperability Framework. nist.gov.
<https://www.nist.gov/big-data>
21. Dean, J., Ghemawat, S. (2008). MapReduce. Communications of the ACM, 51(1), 107–113. <https://doi.org/10.1145/1327452.1327492>
22. Zaharia, M., et al. (2013). Discretized Streams. SOSP.
<https://doi.org/10.1145/2517349.2522737>
23. Carbone, P., et al. (2015). Apache Flink: Stream Processing. IEEE Data Engineering Bulletin, 38(4), 28–38.
24. Kreps, J., Narkhede, N., Rao, J. (2011). Kafka: A Distributed Messaging System. NetDB.
25. OWASP. (2024). Top Ten Security Risks. <https://owasp.org/www-project-top-ten/>
26. Microsoft .NET Documentation. (2025). learn.microsoft.com.
<https://learn.microsoft.com/en-us/dotnet/>

					БР.ІІІ – 36.00.00.000 ІІЗ	Арк.
						87
Змн.	Арк.	№ докум.	Підпис	Дата		

27. Docker Documentation. (2025). docs.docker.com.
<https://docs.docker.com/>
28. Kubernetes Documentation. (2025). kubernetes.io.
<https://kubernetes.io/docs/>
29. Prometheus Monitoring Documentation. (2025). prometheus.io.
<https://prometheus.io/docs/>
30. Grafana Documentation. (2025). grafana.com. <https://grafana.com/docs/>
31. Newman, S. (2021). Building Microservices. oreilly.com.
<https://www.oreilly.com/library/view/building-microservices/9781492034018/>
32. Burns, B., et al. (2019). Kubernetes: Up and Running. oreilly.com.
33. Hohpe, G., Woolf, B. (2003). Enterprise Integration Patterns. addison-wesley.com.
34. Red Hat. (2024). Event-Driven Architecture Guide. redhat.com.
<https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture>
35. Fowler, M. (2003). Patterns of Enterprise Application Architecture. martinowler.com.
36. Gartner. (2025). Trends in Real-Time Data Processing. gartner.com.
<https://www.gartner.com/en/information-technology>
37. Chen, M., Mao, S., Liu, Y. (2014). Big Data: A Survey. Mobile Networks and Applications, 19(2), 171–209.
38. Kamburugamuve, S., et al. (2018). Survey on Stream Processing Systems. IEEE Communications Surveys.
39. Li, S., et al. (2023). Real-Time Data Processing Systems. Journal of Big Data, 10(45). <https://doi.org/10.1186/s40537-023-00745-2>
40. Singh, A., Patel, D. (2024). Scalable Stream Processing Architectures. IEEE Access, 12, 55678–55692. <https://doi.org/10.1109/ACCESS.2024.3378901>

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Програмне рішення для обробки
потоків даних у реальному часі "

Обсяг пояснювальної записки: 88 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____