МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)

(рівень вищої освіти)

Дослідження методів створення конвеєрів для потокової обробки даних

Виконав:
студент 2 курсу групи ІПЗм-21-2
Носов О.В.
(прізвище, ініціали)

Спеціальність	121 – Інженерія програмного забезпечення
Тип програми	Освітньо-наукова
Керівник	доц. Кравець Н.С.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри З.В. Дудар

2023 р

.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук (або Центр післядипломної освіти, або Навчально-науковий центр заочної форми навчання, або ЦЗФН)

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121– Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. Кафедри_____

(підпис)

«___»_____202_р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Носова Олексія Валерійовича

(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів створення конвеєрів дляпотокової обробки даних»

   затверджена наказом університету від «_03__»__04___ 2023_ р. №_83Стз__
2. Термін подання студентом роботи до екзаменаційної комісії

   «__» ____ 202__ р.

   3. Вихідні дані до роботи *технології асинхронної обробки даних, STREAMING DATA PROCESSING, APACHE KAFKA, SPARK STREAMING, ETL, модель хмарного обслуговування, пояснювальна записка.*

4. Перелік питань, що потрібно опрацювати в роботі мета роботи *аналіз стану вирішення проблемі, постановка задачі, аналіз методів дослідження, науковий аналіз та узагальнення фактичного матеріалу, отримання нових результатів, апробація отриманих результатів, узагальнення результатів дослідження з наведенням висновків і рекомендацій.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Огляд літератури	17 лютого 2023 р.	виконано
2.	Аналіз та формування вимог до програмної системи	27 лютого 2023 р.	виконано
3.	Дослідження атрибутів якості програмного забезпечення та постановка задачі	5 березня 2023 р.	виконано
4.	Дослідження інструментів створення конвеєрів обробки даних, порівняння цих інструментів	11 березня 2023 р.	виконано
5.	Тестування конвеєра	28 березня 2023 р.	виконано
6.	Опис результатів дослідження	11 квітня 2023 р.	виконано
8.	Підготовка пояснювальної записки	27 квітня 2023 р.	виконано
9.	Підготовка презентації та доповіді	30 квітня 2023 р.	виконано
11.	Нормоконтроль, рецензування	9 травня 2023 р.	виконано
12.	Занесення диплома в електронний архів	14 травня 2023 р.	виконано
13.	Допуск до захисту у зав. Кафедри	15 травня 2023 р.	виконано
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання 23 січня 2023 р.

Студент Носов О.В.
(підпис)

Керівник роботи _____ Доц. Кравець Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 66 с., 18 рис., 21 джер.

APACHE KAFKA, APACHE FLINK, PIPELINE, КОНВЕЄРИ, ОБРОБКИ ДАНИХ, ОБРОБКА ДАНИХ, ПОРІВНЯННЯ ІНСТРУМЕНТІВ СТВОРЕННЯ КОНВЕЄРІВ.

Об’єктом дослідження є методи створення конвеєрів обробки даниих.

Мета роботи – дослідити методи проектування та створення конвеєрів обробки даних в реальному часі.

Предметом дослідження є конвеєри обробки данних, їх складові, порівння та аналіз складових компонентів котрі використовються при ставоренння пайплайнів обробри данних в реальному часі. Аналізі інструментів для створення конвеєрів.

Методи дослідження. Обраними методами дослідження є огляд наукової літератури та проведення експерименту зі створення конвеєру.

Результати дослідження демонструють ефективність налаштування інструментів побудови пайплайнів, порівняння технологій та процес побудови пайплайна.

APACHE KAFKA, APACHE FLINK, PIPELINE, КОНВЕЄРИ, ОБРОБКИ ДАНИХ, ОБРОБКА ДАНИХ, ПОРІВНЯННЯ ІНСТРУМЕНТІВ СТВОРЕННЯ КОНВЕЄРІВ.

The object of study is methods of creating data processing pipelines.

The purpose of the study is to investigate methods of designing and creating real-time data processing pipelines.

The subject of the study is data processing pipelines, their components, comparison and analysis of the components used in the creation of real-time data processing pipelines. Analysis of tools for creation. Pipeline architecture. Ready-made and existing solutions.

The study compares the methods of pipeline creation, the performance of the technologies used, the configuration and effective configuration of the tools used in pipeline construction.

The results of the study demonstrate the effectiveness of configuring pipeline building tools, comparing technologies, and the process of building a pipeline.

Я,

_____Носов Олексій Валерійович_____
(прізвище, ім'я, по батькові)

студент(ка) групи _ІПЗм-21-2_ здобувач вищої освіти на другому (магістерському) рівні

кафедра_____ програмної інженерії _____,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему

__Дослідження методів створення конвеєрів для потокової обробки даних_____,
(назва роботи)

що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

ETL – Extract Transform Load.

IoT – internet of things.

GCS – Google Cloud Storage.

S3 – Amazon S3 storage.

RnD - Research and Development

ВСТУП

Конвеєр даних - це низка дій і процесів, що використовуються для передачі необроблених даних з однієї точки в іншу. Важливо зазначити що процес ETL не обов'язково є конвеєром даних, однак деякі конвеєри даних можуть підтримувати можливості потокового ETL. ETL є підмножиною конвеєра даних, який є ширшим поняттям. ETL зазвичай працює з пакетними даними, тобто дані переміщуються частинами. На відміну від нього, конвеєр даних може працювати як з пакетними, так і з потоковими даними. Дана робота зосереджена саме на досліджені методів побудови конвеєрів з потоковим типом обробки даних.

Актуальність даної теми зумовлена тим що в останні роки сучасний світ все більше та більше переходить до автоматизації життєвих процесів та завдяки впровадженню в звичайне життя комп'ютеризованих та цифрових пристроїв. Покращення рівня життя таким чином неможливо без збору та обробки величезних обсягів даних. Нейронні мережі, машинне навчання, стрімінгові сервіси, соціальні мережі, більшість онлайн заходів та подій, все це використовує конвеєри обробки даних для нормального функціонування.

Мета роботи дослідити методи проектування та створення конвеєрів обробки даних в реальному часі.

Об'єктом дослідження є методи створення конвеєрів обробки даниих.

Предметом дослідження є конвеєри обробки данних, їх складові, порівння та аналіз складових компонентів що використовються при створення пайплайнів обробки данних в реальному часі. Аналізі інструментів для створення конвеєра.

Завдання дослідження провести експеримерт з конфигураціює показників для обраних інструменів при створенні пайплайну, дослідження налаштувань інструментів що складають конвеєр.

Методи дослідження. Для досягнення цілей науково-дослідницької роботи зі створення конвеєра даних у реальному часі було обрано та використано такі методи дослідження: Огляд наукової літератури дозволяє встановити поточний стан знань у цій галузі та виявити прогалини в існуючій літературі, які має на меті заповнити це дослідження. Проведення огляду літератури для виявлення існуючих

досліджень, теорій та найкращих практик, пов'язаних зі створенням конвеєрів даних у реальному часі. Експериментальні дослідження. Проведення експериментів для оцінки продуктивності конвеєра даних у реальному часі та впливу різних конфігураційних показників на продуктивність проблемних питань.

До елементів наукової новизни можна віднести використання передових технологій. Дослідження може включати нові рішення та технології. Використання таких інструментів може значно підвищити швидкість та ефективність обробки даних, що призведе до отримання більш якісних результатів.

В результаті дослідження було проведено порівняння інструментів побудови пайпланів обробки даних, ефективність налаштування конфігураційних показників для інструментів побудови конвеєру, опис вирішених питань в процесі побудови пайплайна.

Кваліфікаційна робота була апробована на 27-му Міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXIстолітті», що проводилася у Харківському національному університеті 10 – 12 травня 2023 року."

1 АНАЛІЗ СТАНУ РОЗВ`ЯЗАННЯ ПРОБЛЕМИ ЗА МАТЕРІАЛАМИ ПУБЛІКАЦІЙ

1.1 Огляд наукової літератури

Конвеєр даних - це низка дій і процесів, що використовуються для передачі необроблених даних з однієї точки в іншу. Вони бувають різних форм і видів, але всі вони мають одну мету: переміщення даних з одного місця в інше.

Термін "конвеєр даних" повсюдно використовується у світі великих даних. Конвеєри даних складаються з трьох ключових елементів: джерела, етапу або етапів обробкита пункту призначення. Розглядаючи кожен компонент окремо, можна зрозуміти, як саме вони поєднуються. [8]

Джерело: Джерело даних - це місце, звідки будуть витягуватися дані. Це може бути база даних додатків, система Інтернету речей, дані, що надходять через API, озеро даних або сховище даних, дані з соціальних мереж і загальнодоступні набори даних. Дані можуть бути структурованими, неструктурованими або напівструктурованими.

Обробка: Обробка даних передбачає застосування перетворень до вихідних даних, щоб зробити їх готовими для аналізу. Це може бути зроблено до або після завантаження даних до місця призначення.

Місце призначення: Пункт призначення - це кінцеве місце зберігання даних, де вони будуть зберігатися. Це може бути платформа великих даних, інша база даних додатків, інше рішення для зберігання або аналітики. Звідти дані можуть бути доступні іншим системам для аналізу.[10]

1.1.1 Типи ETL

Можлива причина плутанини - схожість між конвеєром даних і процесом ETL. В процесі ETL дані витягуються з джерела, трансформуються і завантажуються в місце призначення. ETL є підмножиною конвеєра даних, який є ширшим поняттям. Процес ETL не обов'язково є конвеєром даних, однак деякі конвеєри даних можуть підтримувати можливості потокового ETL. Відмінності між ETL та конвеєром даних.

ETL зазвичай працює з пакетними даними, тобто дані оброблюються частинами. На відміну від нього, конвеєр даних може працювати як з пакетними, так і з потоковими даними. Деякі конвеєри можуть також підтримувати потокові можливості ETL.[5]

Місцем призначення в ETL є або база даних, або сховище даних, тоді як конвеєр даних може мати кілька місць призначення, навіть може ініціювати механізм звітування в режимі реального часу.

У процесі ETL необхідний етап перетворення даних, тоді як у конвеєрі даних етап перетворення можна пропустити залежно від сценарію використання.[11]

Пакетний (batch) ETL також називають традиційним ETL. Це оригінальна методика, яка використовується для вилучення даних із системи-джерела. Дані збираються пакетами через певні проміжки часу (щогодини, щодня, щотижня тощо), відповідним чином трансформуються, а потім завантажуються в систему-приймач.[5]

Потоковий (streaming) ETL витягує дані з джерела, як тільки вони вводяться, постійно поглинаючи, зберігаючи та обробляючи дані через платформу потокового передавання, перш ніж передати їх до системи призначення. Авіакомпанія застосовує потокове ETL для відображення точної інформації про те, скільки квитків на даний момент доступно на рейс. Або синоптик може покладатися на дані потокового ETL в режимі реального часу для отримання актуальної інформації про погоду.

Таблиця 1 — Порівняння типів конвеєрів2[Таблиця виконана самостійно]

	Пакетний ETL	Потоковий ETL
Переваги	a) Простота впровадження та моніторингу; b) Економічна ефективність; c) Сумісність із застарілими системами; d) Може пакетувати величезні обсяги даних протягом тривалого часу	a) Безперервна ETL з низькою затримкою; b) Обробка даних відбувається миттєво; c) Сумісність з новими технологіями; d) Аналіз даних у реальному часі;
Недоліки	a) Збій в одному наборі даних може спричинити збій у всьому пакетному процесі; b) Нові типи даних не будуть розпізнані автоматично і призведуть до неточностей;	a) Дані в реальному часі означають, що перебої та проблеми з продуктивністю є більш нагальними, ніж у випадку з пакетною обробкою; b) Миттєва обробка ускладнює гарантування якості та точності даних; c) Потребує високопродуктивних платформ для належної роботи та зменшення затримок;

У даному дослідження як предмет дослідження використовується ETL потокового типу(надалі конвеєр даних).

1.1.2 Аналіз конвеєру потокового типу

Конвеєр даних - це надійне рішення для обробки даних, але його розробка пов'язана з певними труднощами. Ці проблеми впливають на якість даних у місці призначення. Загальні виклики при розробці конвеєрів даних: [11]

Питання якості даних. Дані надходять з різних джерел, а інтеграція даних відбувається паралельно, щоб поглинути великі обсяги даних. Дані можуть бути втрачені під час завантаження через збій живлення або проблеми з сервером.

Конвеєр даних повинен бути гнучким, щоб враховувати зміни, які відбуваються в системі-джерелі, звідки надходять дані.

Автоматизація: Конвеєр даних автоматизує процес збору, обробки та зберігання великих обсягів даних.

Масштабованість: Конвеєр даних призначений для обробки великих обсягів даних, що дозволяє обробляти та аналізувати дані в режимі реального часу, навіть якщо вони зростають.

Економічна ефективність: Конвеєр даних може допомогти зменшити витрати на управління та аналіз даних за рахунок автоматизації повторюваних завдань та оптимізації використання ресурсів.

Аналітика в реальному часі: Конвеєр даних може обробляти та аналізувати дані в режимі реального часу, дозволяючи організаціям отримувати інформацію та швидко приймати обґрунтовані рішення.

Вибір інструментів і технологій для проектування конвеєрів даних є дуже важливим то варто виділяти окрему уваги вибору правильних інструментів під особоливості поставленої задачі для досягнення максимальної продуктивності конвеєра. При правильному використанні та конфігурації конвеєр здатен покривати більшість існуючих проблем. Існує багато варіантів, але вибір правильного інструменту залежить від сценарію використання.[21]

1.1.3 Аналіз інструментів створення потокового конвеєру даних

Для обробки даних необхідна потужна та універсальна концепція, яка підтримує високу продуктивність , а також гарантує відмовостійкість за допомогою надійного механізму. Оптимальна система повинна масштабуватися вертикально та горизонтально, щоб відповідати зростаючому навантаженню. Основні вимоги до такої системи включають гнучкість функціоналу для фільтрації, трансформації даних між форматами, дублювання даних на потоці та можливість підключення нових джерел. Бажано, щоб система мала нульовий або низький рівень кодування для налаштування обробки даних та забезпечувала деталізований моніторинг.[12]

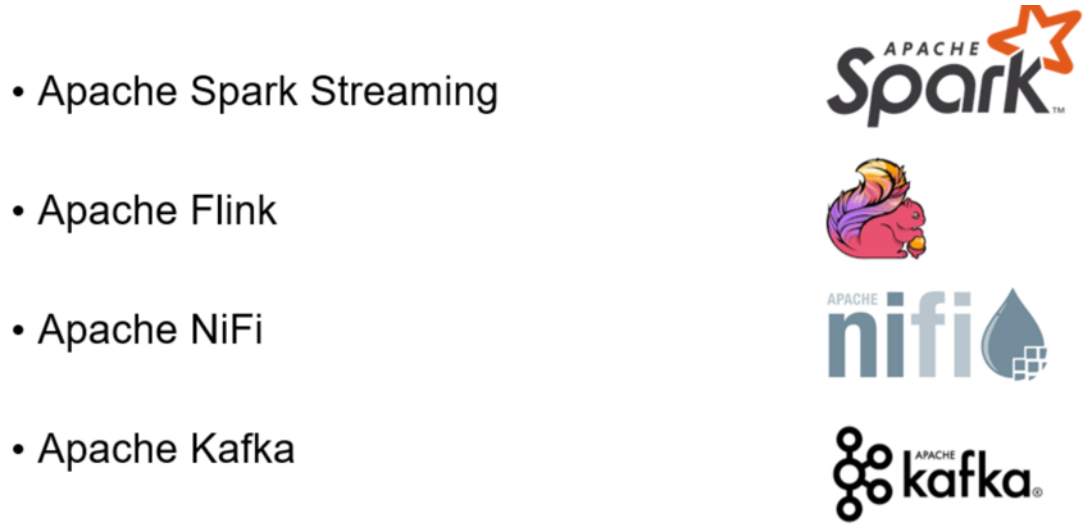

Рисунок 1.1.3 — Стек технологій[1]

Аналіз та дослідження технологій для побудови конвеєру обробки даних в реальному часі. Для аналізу було відібрано інструменти, які задовольняють вимогам до розроблюваної системи: Spark Streaming, Apache Flink, NiFi та Kafka.

Порівняння технологій Apache Nifi та Apache Spark.[2]

Таблиця 2 — Порівняння Apache Nifi та Spark[Таблиця виконана самостійно]

Елемент порівняння	Apache Nifi	Apache Spark
Що надається	Надає графічний інтерфейс користувача як формат для конфігурації системи та моніторингу потоків даних.	Масштабний фреймворк для обробки даних з приблизно нульовою затримкою забезпечується за рахунок дешевого апаратного забезпечення.
Особливості	a) Веб-інтерфейс користувача b) Висока конфігурація c) Підтвердження походження даних d) Призначений для розширення e) Безпечний f) Не для віконних обчислень g) Відсутність реплікації даних	a) Надзвичайно висока швидкість b) Багатомовний c) Розширена аналітика d) Обробка потоку в реальному часі e) Гнучкі можливості інтеграції f) Віконні обчислення g) Коефіцієнт реплікації даних 3 за замовчуванням

Продовження таблиці 2

Архітектурні компоненти	a) Веб-сервер b) Контролер потоку c) Розширення d) Репозиторій файлів потоку e) Сховище вмісту f) Репозиторій походження	a) Spark Core b) Spark Streaming c) Spark SQL d) Spark R e) Spark GraphX f) Spark MLlib
Варіанти використання	a) Управління потоком даних, а також візуальний контроль b) Довільний розмір даних c) Маршрутизація даних між різними системами	a) Потокові дані b) Машинне навчання c) Інтерактивний аналіз d) Туманні обчислення
Проблеми розгортання	Проблеми з конфігурацією та сумісністю виникають, якщо не використовувалася остання версія Java.	Для того, щоб мати кероване середовище, необхідне чітко визначене розташування кластерів, що є неправильною конфігурацією.

Кінець таблиці 2

Проблеми масштабованості та стабільності	Як правило, не повідомлялося про проблеми, пов'язані з масштабованістю та стабільністю	Досягнення стабільності є складним завданням, оскільки іскра завжди залежить від потоку.
Обмеження головним чином	Обмеження пов'язані зі швидкістю індексації походження, яка стає вузьким місцем, коли мова йде про загальну обробку великих обсягів даних.	Обмеження для Spark пов'язане зі стабільністю API, оскільки перехід від RDD до Data Frames і Data Sets часто стає складним завданням.

Порівняння технологій Apache Flink та Apache Kafka.

Таблиця 3 — Порівняння Apache Flink та Apache Kafka[Таблиця виконана самостійно]

Елемент порівняння	Apache Flink	Apache Kafka
Розгортання	це кластерний фреймворк, що означає, що фреймворк піклується про розгортання.	це бібліотека, яку може вбудувати будь-який стандартний Java-додаток, і, отже, не намагається диктувати метод розгортання;
Життєвий цикл	код обробки потоку розгортається та виконується як завдання у кластері Flink	код обробки потоку користувача виконується всередині його додатку

Кінець таблиці 3

Зазвичай належить	інфраструктурі даних або команді BI	лінія бізнес-підрозділу, яка керує відповідним додатком
Координація	флінк-майстер (JobManager),	частина потокової програми, використовує кластер Kafka для координації, балансування навантаження та відмовостійкості.
Складність і доступність	Інтерфейс Flink простий у навігації, а інтуїтивно зрозуміла документація дозволяє швидко розпочати роботу. Але Flink розгортається на кластері, що може бути складніше.	За допомогою Kafka Streams дуже легко розгортати стандартні Java та Scala додатки. Крім того, Kafka Streams працює "з коробки".понаднормовою роботою розробників.

Тож узагальнюючи Apache Nifi - це інструмент збору даних, який забезпечує просту у використанні, систему для спрощення обробки даних. На відміну від нього, Apache Spark - це надзвичайно швидка технологія кластерних обчислень, розроблена для прискорення обчислень завдяки ефективному використанню інтерактивних запитів в управлінні пам'яттю та можливостям потокової обробки.[3]

Apache Nifi надає можливості візуалізації та перетягування для кращої читабельності та загального розуміння системи. Apache Spark сам по собі не надає

можливостей для візуалізації і хороший лише в тому, що стосується програмування.

Кафка - це бібліотека для створення потокових застосунків, які перетворюють вхідні теми Кафки на вихідні теми Кафки. Kafka Streams (Kstreams) використовує внутрішні бібліотеки виробника(producer) та споживача(consumer). Вона поєднана з Kafka, і API дозволяє використовувати можливості Kafka, досягаючи паралелізму даних, відмовостійкості, низької латентності та багато іншого.

Spark stream - це розширення основного Spark API, яке забезпечує масштабовану, високопродуктивну, відмовостійку потокову обробку потоків даних у реальному часі.[2]

1.2 Постановка задачі дослідження

Задача даного дослідженя проаналізувати показники швидкості та пропускої здатності конвеєра створеного за допомогою обраних технологій. Для порівнняня інструментів треба визначити крітерії оцінювання. Відбір технологій має бути проведено за домогою аналізу по обраним критерія. Список критеріїв наведно нижче:

a) поглинання даних
b) затримка
c) масштабованість
d) відмовостійкість
e) потокова обробка
f) інтеграція
g) підтримка спільноти.

Тож для проведення відбору є наступний технологічний стек Spark Streaming, Apache Flink, NiFi та Kafka.

Наведено порівняння деяких ключових особливостей та відмінностей між Apache Kafka та Apache Flink:

Поглинання даних: Kafka в першу чергу використовується для отримання та розподілу даних, в той час як Flink є повноцінним рушієм для обробки потоків, який може виконувати складні перетворення над даними.

Затримка: Kafka забезпечує меншу затримку, ніж Flink, при отриманні та розподілі даних, оскільки не виконує складних перетворень над даними.

Масштабованість: І Kafka, і Flink мають високу масштабованість і можуть обробляти великі обсяги даних. Однак Kafka може обробляти більше розділів на тему, ніж Flink.

Відмовостійкість: І Kafka, і Flink забезпечують відмовостійкість і високу доступність. Kafka використовує модель реплікації "лідер-послідовник", в той час як Flink використовує розподілену обробку з урахуванням стану та контрольну точку.

Потокова обробка: Flink надає більш просунуті можливості обробки потоків, ніж Kafka, включаючи вікна, агрегації та алгоритми машинного навчання.

Інтеграція: Kafka добре інтегрується з іншими технологіями великих даних, включаючи Spark, Hadoop та Flink. Flink можна використовувати з Kafka для збору та розподілу даних, а також для обробки потоків.

Підтримка спільноти: Як Kafka, так і Flink мають потужну підтримку спільноти і активно розвиваються

Таким чином, Kafka найкраще підходить для збору та розповсюдження даних, в той час як Flink є повноцінним рушієм для обробки потоків, який може виконувати складні перетворення над даними. Результати порівняння наведе в таблиці 4.

Таблиця 4 — Порівняння Kafka та Flink[Таблиця виконана самостійно]

Показник	Kafka	Flink
Поглинання даних:	-	+
Масштабованість	+	-
Відмовостійкість	+	+
Потокова обробка	-	+
Інтеграція	+	-
Підтримка спільноти	+	+

Коли мова йде про швидку обробку даних у реальному часі, Apache Flink зазвичай вважається кращим вибором, ніж Apache Kafka. У той час як Kafka в першу чергу призначений для прийому та розподілу даних.

Тим не менш, Kafka все ще залишається популярним вибором для обробки даних у реальному часі в певних випадках використання. Наприклад, якщо основна увага приділяється отриманню та розподілу даних, а не складній обробці потоків, Kafka може бути більш доречним вибором. Крім того, інтеграція Kafka з іншими технологіями великих даних, такими як Spark і Hadoop, може зробити її більш привабливим варіантом у певних контекстах.

Зрештою, одна з вимог до проекту в цьому дослідження – побудова масштабованого конвеєру. Тому вибір пав на Apache Kafka з урахування можливості додаткового використання Spark для підтримки швидкості.

Порівняння деяких ключових особливостей та відмінностей між Apache NiFi та Apache Spark:

Поглинання даних: NiFi в першу чергу використовується для отримання та маршрутизації даних, в той час як Spark - це універсальний движок для обробки великих даних, який включає пакетну обробку, потокову обробку та можливості машинного навчання.

Масштабованість: І NiFi, і Spark мають високу масштабованість і можуть обробляти великі обсяги даних. Однак Spark зазвичай вважається більш масштабованим, ніж NiFi, завдяки своїй моделі розподілених обчислень.

Відмовостійкість: І NiFi, і Spark забезпечують відмовостійкість і високу доступність. NiFi використовує потокову модель програмування з автоматичним відстеженням походження даних, тоді як Spark використовує стійкі розподілені набори даних (RDD) та відмовостійкі структури даних.

Обробка потоків: NiFi надає базові можливості обробки потоків, в той час як Spark надає більш просунуті можливості обробки потоків, включаючи вікна, агрегації та алгоритми машинного навчання.

Інтеграція: NiFi добре інтегрується з іншими технологіями великих даних, такими як Hadoop, Kafka та Spark. Spark також може бути інтегрований з цими технологіями, а також з популярними базами даних і джерелами даних.

Підтримка спільноти: Як NiFi, так і Spark мають потужну підтримку спільноти та активну розробку, з великою кількістю доступних плагінів, бібліотек та інструментів.

Таким чином, NiFi найкраще підходить для збору та маршрутизації даних, тоді як Spark - це універсальний движок для обробки великих даних, який включає в себе можливості пакетної обробки, потокової обробки та машинного навчання.

Обидві платформи мають високу масштабованість, відмовостійкість і потужну підтримку спільноти. Вибір платформи буде залежати від конкретних

вимог конкретного випадку використання, таких як потреба в розширених можливостях потокової обробки, простота використання та інтеграція з іншими технологіями. Результати порівняння наведе в таблиці 5

Таблиця 5 — Порівняння Apache NiFi та Apache Spark[Таблиця виконана самостійно]

Показник	Spark	Nifi
Поглинання даних:	+	-
Масштабованість	+	+
Відмовостійкість	+	+
Потокова обробка	+	-
Інтеграція	+	+
Підтримка спільноти	+	+

АНа. Крім того, модель розподілених обчислень і обробка в пам'яті Spark може забезпечити швидшу обробку даних, ніж потокова модель програмування NiFi. Тож вибор пав на Spark.

Тож визначившись та обравши технологіх для створення ефективного, швидкого та масштабованого пайплайну обробки даних у реальному часі, було обрано Apache Kafka та Apche Spark. Після цього не меншь важливим для гарних показників продуктивності для зазначених виомг є правильна конфігурація кожного з зазначених інструментів.

Врахування критеріх конфигурації.

Нижче наведено конфігурації Kafka, які можна налаштувати, щоб збільшити продуктивність масштабування та обробки даних:

num.io.threads: Цей параметр визначає кількість потоків вводу/виводу, які використовуються для обробки запитів. Збільшення кількості потоків може допомогти підвищити продуктивність, особливо при роботі з великою кількістю запитів.

num.network.threads: Цей параметр визначає кількість мережевих потоків, які використовуються для обробки мережевого вводу/виводу. Збільшення кількості

потоків може допомогти покращити пропускну здатність мережі та зменшити затримки.

num.partitions: Цей параметр визначає кількість партішенів, які використовуються для даної теми. Збільшення кількості партішенів може допомогти покращити паралелізм та розподілити навантаження між більшою кількістю вузлів.

compression.type: Цей параметр визначає алгоритм стиснення, який використовується для стиснення даних. Увімкнення стиснення може допомогти зменшити обсяг даних, які потрібно передавати мережею, що може підвищити продуктивність.

batch.size: Цей параметр визначає максимальну кількість байт, яка може бути включена в один пакет повідомлень. Збільшення розміру пакета може допомогти зменшити кількість мережевих запитів і підвищити пропускну здатність.

linger.ms: Цей параметр визначає максимальну кількість часу, яку виробник буде чекати перед відправкою пакету повідомлень. Збільшення часу затримки може допомогти об'єднати більше повідомлень у пакет і зменшити кількість запитів, що може покращити продуктивність.

fetch.min.bytes: Цей параметр визначає мінімальну кількість байт, яку споживач буде чекати, перш ніж отримати дані від брокера. Зменшення цього значення може допомогти поліпшити затримку і скоротити час, необхідний споживачам для отримання даних.

Налаштовуючи ці параметри конфігурації, можна оптимізувати продуктивність кластера Kafka і збільшити швидкість обробки даних. Однак важливо зазначити, що оптимальні значення цих параметрів залежать від конкретних вимог сценарію використання, а їх налаштування може бути складним та ітеративним процесом.

Нижче наведено конфігурації Spark, які можна налаштувати, щоб збільшити значно збільшити швидкість обробки даних:

Пам'ять і ядра виконавця: Ці параметри контролюють обсяг пам'яті та кількість ядер процесора, виділених для кожного виконавця Spark. Збільшення цих

значень може покращити продуктивність, але потребує обережності, щоб не виділити занадто багато пам'яті або занадто багато ядер, що може призвести до помилок, пов'язаних з нестачею пам'яті або погіршенням продуктивності.

Налаштування збирання сміття (GC) виконавця: Spark використовує віртуальну машину Java (JVM), яка має збирач сміття, що може впливати на продуктивність. Ви можете оптимізувати поведінку збирача сміття JVM, налаштувавши такі параметри, як -XX:+UseG1GC або -XX:MaxGCPauseMillis.

Налаштування перетасування - це процес перерозподілу даних між виконавцями Spark. Він може бути вузьким місцем для продуктивності, тому ви можете налаштувати пов'язані з ним параметри, такі як spark.shuffle.compress, щоб покращити продуктивність.

Кешування та збереження: Spark дозволяє кешувати або зберігати дані в пам'яті або на диску, щоб уникнути повторних обчислень. Ви можете використовувати методи persist() або cache() для позначення даних для кешування. Це може значно покращити продуктивність ітеративних або ітеративно-подібних алгоритмів.

Динамічне розподілення: Динамічне розподілення дозволяє Spark регулювати кількість виконавців залежно від робочого навантаження. Ви можете увімкнути динамічний розподіл, встановивши параметр spark.dynamicAllocation.enabled у true. Це може допомогти оптимізувати використання ресурсів і зменшити витрати.

Важливо поекспериментувати з різними налаштуваннями та оцінити їхній вплив на продуктивність, щоб знайти найкращу конфігурацію для вимог задачі дослідженнг.

Правильно налаштувавши Kafka і Spark та інтегрувавши їх за допомогою Kafka-Spark Streaming, можна створити надійний і масштабований конвеєр для обробки даних у реальному часі.

2 АНАЛІЗ МЕТОДІВ ДОСЛІДЖЕННЯ

2.1 Аналіз методів дослідження.

Аналіз ефективності та результативності методів дослідження, що використовуються для створення конвеєрів даних, має вирішальне значення для забезпечення надійності отриманих в результаті дослідження висновків, які можуть бути використані для прийняття обґрунтованих рішень.

Методи дослідження. Для досягнення цілей науково-дослідної роботи використано такі методи дослідження:

a) огляд літератури

b) експериментальні дослідження.

Проведено огляду літератури для виявлення існуючих досліджень, теорій найкращих практик, пов'язаних зі створенням конвеєрів даних у реальному часі а також що найголовніше невирішені питання, узагальнення та вирішення яких і є породжує задачу цього дослідження.

Проведення експериментів для оцінки продуктивності конвеєрів даних у реальному часі та впливу різних дизайн-рішень. Експериментальні дослідження включать в себе:

a) тематичне дослідження

b) експерименти.

Ці два дослідження передбачають аналіз конкретного прикладу та надають визначення викликів, проблем та результатів. Тематичне дослідження може надати детальне розуміння практичних аспектів побудови конвеєрів даних, включаючи технічні виклики, вимоги до ресурсів та організаційні фактори. Експериментування передбачає розробку та проведення контрольованих експериментів для перевірки ефективності різних технологій, інструментів та підходів до побудови конвеєрів даних. Експерименти можуть допомогти визначити оптимальну комбінацію технологій і підходів для побудови ефективних і масштабованих конвеєрів даних.

2.2 Обгрунтування методів дослідження

Огляд літератури - це метод дослідження, який використовується для виявлення, аналізу та синтезу існуючих досліджень на певну тему. У контексті створення конвеєрів даних огляд літератури передбачає збір та аналіз існуючих досліджень з різних джерел, таких як технічно-наукова література наукові журнали, матеріали конференцій та інтернет-публікацій. Метою огляду літератури є визначення контексту дослідження та виявлення прогалин у знаннях, нових тенденцій та найкращих практик, пов'язаних зі створенням конвеєрів даних. Вивчаючи попередні дослідження на цю тему, дослідники можуть визначити, що було вивчено, які методології були використані і які ключові висновки були зроблені. Огляд літератури покриває наступні точки:

a) визначення відповідних технологій та інструментів
b) оцінка існуючих підходів
c) виявлення прогалин у знаннях
d) вибір джерел даних
e) вибір методів дослідження.

Переглядаючи літературу про створення конвеєрів даних, дослідники можуть визначити відповідні технології та інструменти, які можуть бути використані в їхніх дослідженнях. Це може допомогти у виборі найбільш підходящих інструментів для їхніх дослідницьких цілей і уникнути марнування часу і ресурсів на інструменти, які можуть виявитися непридатними. Також огляд літератури допомагає у виборі джерел даних для дослідження. Визначивши найбільш релевантні та надійні джерела даних, дослідники можуть гарантувати, що дані, які використовуються в їхньому дослідженні, є високоякісними і відповідають дослідницьким цілям. Це допоможе забезпечити достовірність і надійність результатів дослідження.

Експеримент є важливим методом дослідження при створенні ефективного конвеєра обробки даних. Експериментальне дослідження може включати в собі:

a) перевірку гіпотез

b) оцінка ефективності

c) тестування масштабованості

d) визначення вузьких місць

e) перевірка в реальних умовах

1) Перевірка гіпотез дає можливість дослідникам перевірити та підтвердити гіпотези про ефективність різних методів та інструментів створення конвеєрів даних, може допомогти підтвердити або відкинути припущення, зроблені під час огляду літератури, і надати докази на підтримку результатів дослідження.

2) Оцінка ефективності дозволяє оцінити ефективність різних методів та інструментів створення конвеєрів даних, допомогає виявити сильні та слабкі сторони кожного підходу і дати уявлення про те, як оптимізувати конвеєр для кращої продуктивності.

3) Тестування масштабованості дає змогу перевірити масштабованість різних методів та інструментів, також виявляє обмеження кожного підходу та надає уявлення про те, як масштабувати конвеєр для обробки більших обсягів даних.

4) Визначення вузьких місць виявляє вузькі місця (bottle neck) в конвеєрі даних та визначає області, де необхідна оптимізація. Це може допомогти підвищити ефективність і результативність конвеєра та оптимізувати використання ресурсів.

5) Перевірка в реальних умовах допомагає тестувати конвеєр в реальному середовищі, використовуючи реальні чи наближені до таких дані а також робити це в реальних або знов ж таки наближених до таких умовах. За допомогою цього можна підтвердити ефективність конвеєра в практичних додатках і дати уявлення про те, як поліпиши його організацію для кращої продуктивності та масштабованості.

2.3 Етапи проведення методів дослідження

Отже проводячи узагальнення аналізу ефективності та результативності методів дослідження можна виділити наступні етапи проведення для кожного з обраних методів а саме:

Для огляду літератури:

a) Перегляд і відбір досліджень на основі критеріїв включення.
b) Вилучення даних та інформації з відібраних досліджень.
c) Аналіз даних та синтез інформації для визначення ключових тем і висновків.
d) Критична оцінка досліджень та визначення їх якості та відповідність досліджуваному питанню.
e) Виділення невирішених питань котрі потребують окремого дослідження.
f) Розробка теоретичної бази на основі аналізу літератури та отриманих з неї результатів.

Для експерименту:

a) Сформувати гіпотезу котра потребує перевірки.
b) Планування експерименту, включаючи вибір змінних, у випадку цього дослідження набір технологій котрі підлягають перевірці та матеріалів.
c) Проведення експерименту та збір даних.
d) Інтерпретація результатів та формулювання висновків на основі аналізу.
e) Оцінити сильні та слабкі сторони експерименту та узагальнення отриманих результатів.
f) Розробка рекомендацій для майбутніх досліджень на основі отриманих результатів.

3 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

3.1 Методи створення конвеєрів

Як було описано в попередніх пунктах цієї роботи, для вирішення поставленої задачі дослідження було обрано 2 методики дослідження теоретичне дослідження котре включає огляд літератури та проведення самого дослідження а також експериментальне дослідження яке містить в собі практичну частину а саме створення конвеєру обробки даних в реальному часі.

Цей розділ тримає фокус саме на теоретичних аспектах створення конвеєрів обробки даних у реальному часі та складає нагляду картину створення конвеєру на вже існуючих прикладах, також приводиться оцінка використаних технологій, підходів, цінових політик та сервісів (для не опен-сорс технологій), загальне співвідношення та порівняння використаних технологій та систем, теоретичні відомості про типи конвеєрів, а також інформацію саме про конвеєрів з типом обробки у реальному часі.

3.2 Опис методики досліджень

Складовими теоретичної частини дослідження, є:

a) Аналіз літературних джерел, як базова складова для розуміння доменної області та специфіки середовища, загальних термінів, компонентів, позначень, проблем та невирішених питань.
b) Теоретичне дослідження різних конвеєрних систем, кращі практики їх побудови, проблеми та їх вирішення чи не вирішення при побудові та роботі конвеєра, опис та порівняння технологічного стеку використаного для їх створення, архітектура та метрики.

3.3 Розробка конвеєру

Вважаючи зазначені вимогі до системи вона має бути гнучкою до фільтрації та задавати в рамках цього конвеєру не один фільтр, а декілька. Щоб створити таку гнучку методологію, достатньо зробити невеликий DSL та реалізувати два логічні операнди. Це логічні оператори AND та OR, а також дужки для встановлення пріоритету. Вже цих операндів достатньо, щоб створювати великі вирази, які дозволять гнучко фільтрувати дані. Споживачу достатньо вказати лише цей вираз, щоб фільтрувати свої дані.

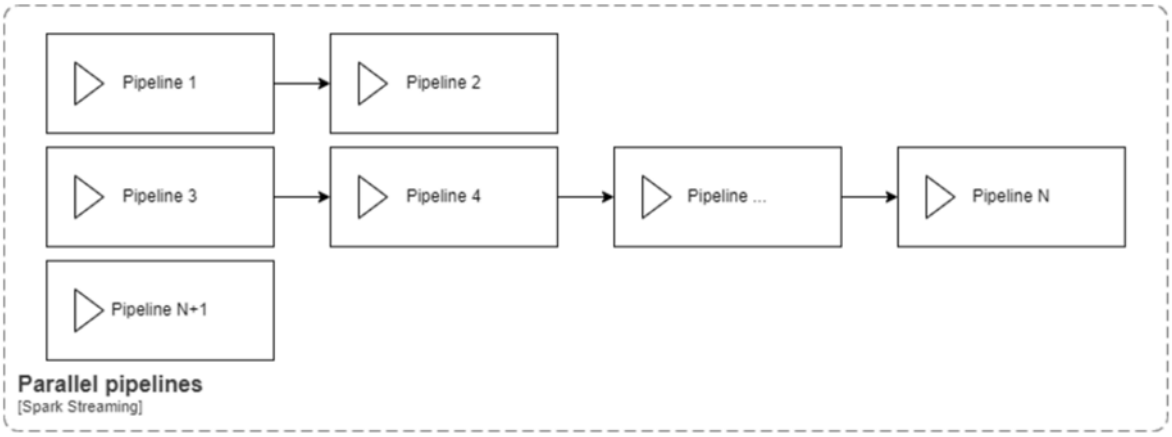

Рисунок 3.3 — Конструктор конвеєрів[3]

Однак, поточний функціонал пайплайну ще не вичерпує всіх можливостей. Бажання створити конструктор, схожий на Lego, з модульною структурою конвеєрів, що дозволяє будувати більш складні системи. Це має на меті розширити можливості реалізації різноманітних завдань. Реалізація виглядає наступним чином: звичайний клас пайплайну, який містить в основному лише два методи: start та stop. При цьому, start приймає основну, головну рушійну силу даного рішення, - StreamingContext. Він створюється один раз на рівень абстракції вище, у так званому пайплайн-менеджері, який стежить за всім життєвим циклом колекції пайплайнів. Щоб запустити або перезапустити якийсь пайплайн, достатньо оновити цю колекцію і перезапустити стрімінг-контекст.

Запуск пайплайну - в принципі важка справа. Це перезапуск стрімінг-контекста, тому перед тим, як запустити якийсь пайплайн, слід спочатку перевірити всі конфігурації. Оскільки термінальний обробник Action, краще перевірити саме його налаштування перед запуском конкретного пайплайну. І, звичайно ж, бажано реалізувати метод Stop, щоб все-таки зупиняти пайплайн, коли потрібно.

Тепер детальніше про кожен обробник в пайплайні. Source - він має приймати дані з джерела, створювати на основі цих даних стримінг і десеріалізовувати ці дані в внутрішню модель. Далі ця внутрішня модель, Message, буде подорожувати від обробника до обробника, змінюючи свій стан. Вводиться свого роду контракт.

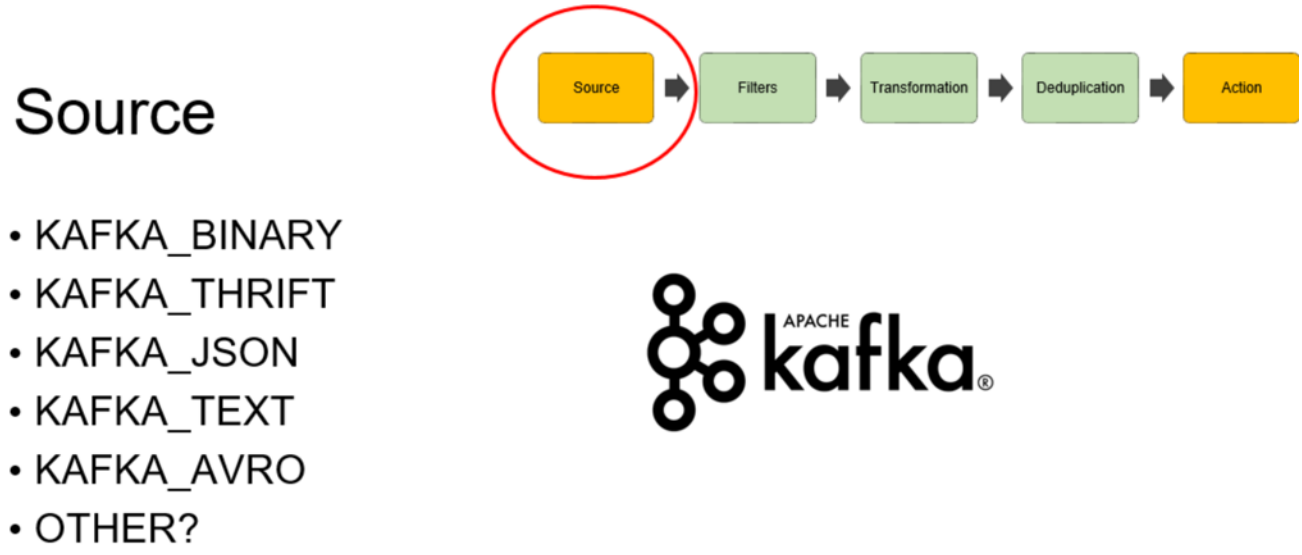

Рисунок 3.3 — Обробники Source[3]

Основне джерело даних - це Kafka, туди надсилаються в потоковому режимі всі дані. В даному дослідженні реалізовано кілька типів Source Kafka, основні - Thrift і Json. 70-80% даних зберігаються саме у форматі Thrift.

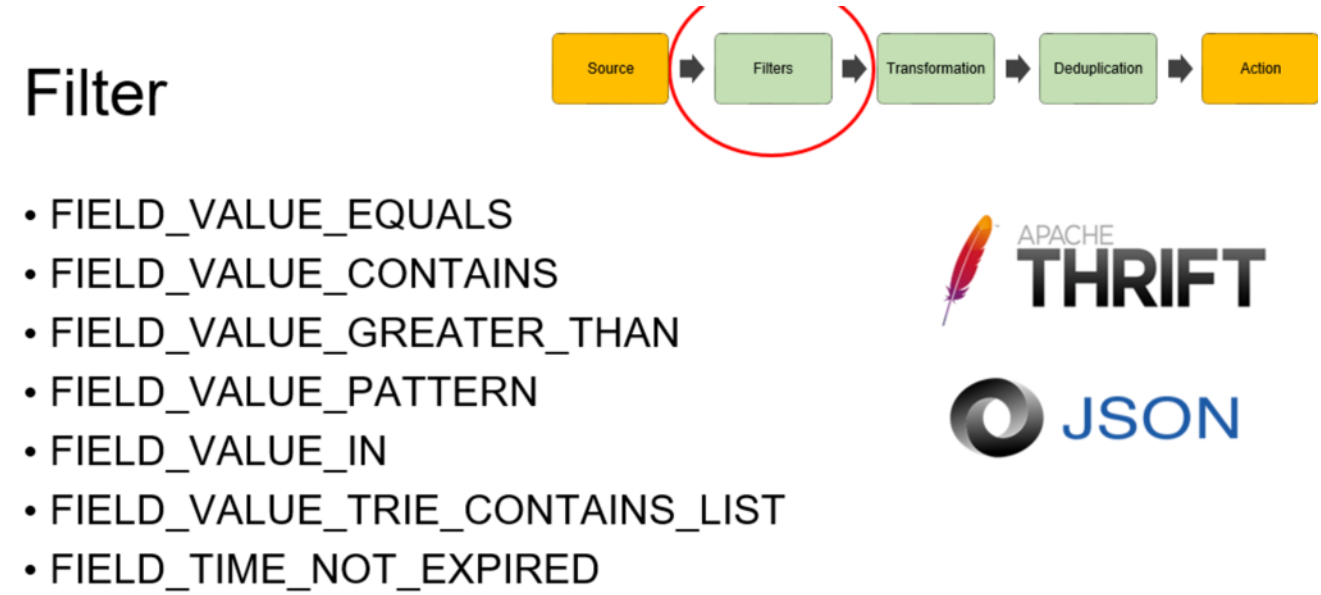

Рисунок 3.3 — Обробники Filter[3]

Можна зробити інше джерело даних, не Kafka, використовувати HDFS Avro, налаштувати цей Source на HDFS-директорію та проводити моніторинг для перевірки того як створюються нові файли та робляться зміни в уснуючих. Цю можливість надає реалізація trait Source, котра має основні методи. Основні методи - це stream, message, і onRDDEnd. Stream приймає StreamingContext, до того він «рухається» транзитивно через кожен пайплайн і кожен обробник даних. На основі StreamingContext створюється InputDStream, тобто дискретний потік даних. Після цього застосовується метод message, який залежно від типу десеріалізує дані у внутрішню модель.

Наступний обробник - це фільтри. Спочатку було створено загальні фільтри. Це видно з назв: є Equals, Contains, знаходження за шаблоном. Проте всі ці фільтри виявилися недостатньо ефективними. При великиї потоках даних, основне навантаження на обчислення припадає саме на фільтрацію. Центральний процесор завантажений майже на максимум саме через ці фільтри, особливо з огляду на те, що може бути велика кількість фільтрів в одному DSL.

Ще один метод – onRDDEnd, реалізовує доступність даних на джерелах. Якщо раптом у конвеєрі станеться інцидент, якийсь із обробників почне чинити збої поводитись неочікувано, генерувати винятки, то данні не втратяться. В такому випадку робиться перезавантаження пайплайну та зчитування даних з того

моменту, на якому востаннє був комміт. Відповідно з використанням такого підходу фільтри стали більш спеціалізованим та продуктивними, з урахуванням специфіки даних. Наприклад, варто звернути увагу на Trie contains list. Він містить реалізацію алгоритму Ахо-Корасіка для пошуку підрядка в рядку. В даному випадку він використовується для пошуку доменів певного рівня в потоці URL-ів. Таке використання надало змогу збільшити продуктивність порівняно зі звичайним Contains-ом на 30% і знизити навантаження на CPU. Фільтр має частково визначену функцію, яка містить за замовчуванням відразу два методи - isDefinedAt і Apply. isDefinedAt перевіряє, чи бере участь цей фільтр в обробці даних, та чи бере участь він у конвеєрі, у конкретному DSL. Якщо бере участь, то застосовують метод Apply, у якому реалізовано фільтрацію до конкретного повідомлення.

Коли дані відфільтровані - вони надходять в трансформацію. Оскільки основний формат даних - Thrift, то для споживачів важливо бачити ці дані, Thrift - це sequence byte, не юзер френдлі відображення. Відповідно, у даній реалізції використано трансформацію даних у CSV та у Json.

Рисунок 3.3 — Обробники Transformation[4]

Трансформацією прибираються зайві поля та перероблюютьсятак, як хочеться споживачеві. Наприклад, якщо на вхід прийшло сто полів - видається тільки п'ять, потрібних клієнту. І зменшується тим самим розмір самого потоку. Усе це, налаштовується споживачем на самообслуговуванні. Потрібно тільки взяти trait і реалізувати, реалізація схожа на фільтрацію. Методи аналогічні, isDefineAt і Apply з тим же призначенням.

Потім настає черга наступного обробника, дедублікації. Проблема - дедублікація даних на потоці? Коли є кінцевий набір даних – все зрозуміло: є повідомлення і записи, на основі яких можна зрозуміти, де які дані є дублем. На потоці нескінченна кількість даних, відповідно, потрібно вибрати якийсь інтервал часу, у межах якого необхідно зберігати стан об'єкта. І тільки після нього порівнювати стан конкретно інших об'єктів для пошуку дублів.

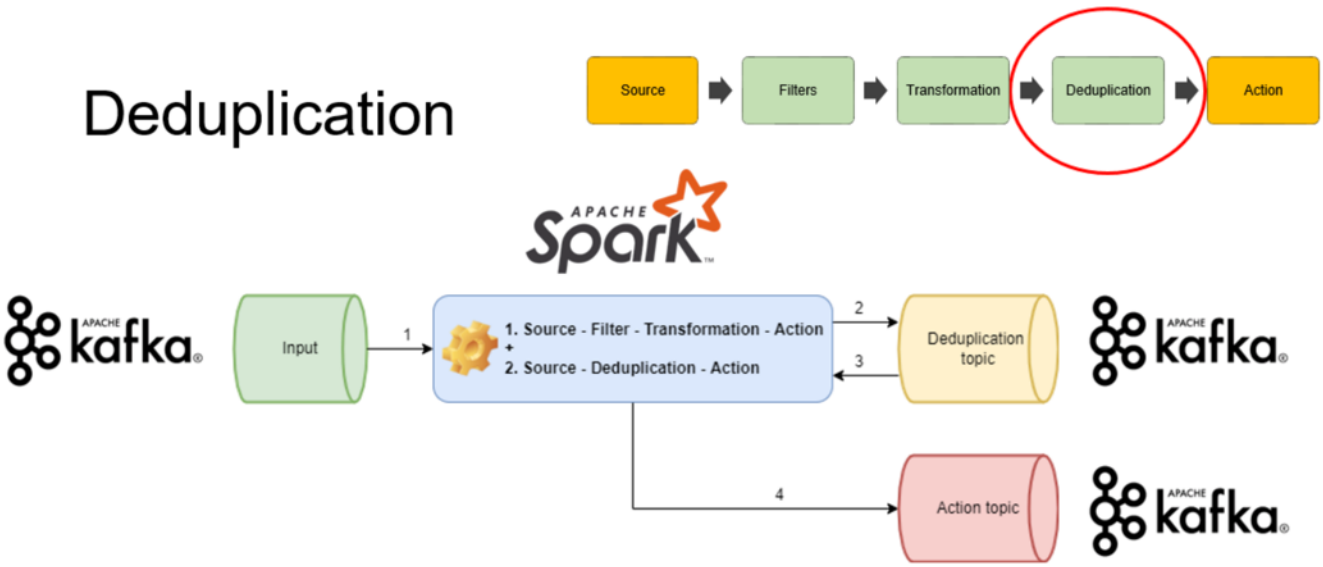

Рисунок 3.3 — Обробники Deduplication[4]

Питання полягає в тому, де зберігати ці об'єкти та як довго? Час зберігання вирішили передати на розсуд споживачів, тоді як місце зберігання варто визначити. Відступ: основна концепція полягала в тому, щоб мати самодостатній інструмент без залучення великої кількості технологій чи використання зовнішньої системи зберігання. Хорошим варіантом є Spark Streaming, який має виконавців (executors), що запускаються на різних вузлах і можуть спілкуватися між собою, а також має механізм розсилки даних (broadcast).

Можливо, Spark Streaming надасть можливість самому зберігати стан, оскільки він пропонує такий елемент як DStream - дискретний потік даних, який має метод mapWithState. Варто відмітити, що цей метод вихідних кодів Spark позначений як експериментальний. Для передачі функції дедублікації та збереження стану достатньо використати цей метод. Щодо визначення часу

зберігання, існує метод timeout, який встановлює тривалість збереження стану об'єкта, тобто визначає, як довго зберігати цей об'єкт.

Питання полягає в тому, де зберігати ці об'єкти та на який термін? Час зберігання було вирішено передати на розсуд споживачів, а місце зберігання потрібно визначити. Головна концепція полягала в створенні самодостатнього інструменту, який не вимагає великої кількості технологій чи зовнішньої системи зберігання. Оптимальним рішенням є Spark Streaming, який містить виконавців (executors), запущених на різних вузлах і здатних спілкуватися між собою, а також має механізм розсилки даних (broadcast). Spark Streaming надає можливість зберігати стан, використовуючи DStream - дискретний потік даних, що містить метод mapWithState, який в оригінальних кодах Spark позначений як експериментальний. Для використання цього методу достатньо передати функцію дедублікації, і Spark Streaming самостійно зберігатиме певний стан. Щодо визначення тривалості зберігання, існує метод timeout, який встановлює час зберігання стану об'єкта.

Це рішення є оптимальним, однак воно має свої недоліки. У випадку перезапуску додатку або інструменту може виникнути ситуація, коли відфільтровані та трансформовані дані будуть втрачені.

Вихід з ситуації виявися таким - розділити на два пайплайни конкретне рішення. Тобто взяти, віднімати дані, відфільтрувати, трансформувати, надіслати ці дані за допомогою Action у проміжний топік Kafka для дедублікації. І другим пайплайном цим же застосунком вичитати дані і просто дедублікувати, зрештою надіславши їх у конкретний топік.

Реалізація дедублікації - це метод Apply і DStream з MapWithState, туди передається функція вищого порядку, яка дедублікує дані і зберігає все за механізмом бродкастів. Таймаут вказано для timetoleave нашого конкретного стану.

Термінальний обробник. У ньому достатньо реалізувати три виходи: це Kafka, HDFS і HBASE. У Kafka можно класти дані в будь-який кластер у будь-якому напрямку, також є можливість записувати дані як файли у HDFS у різних

форматах, також можна записувати дані в HBASE за ключем, для всих цих фукнцій цього є конфігурації. При цьому в рамках пайплайну може бути кілька Action, тобто Source мінімально один, а Action може бути кілька. Це зроблено для того, щоб трансформовані і вже відфільтровані дані можна паралельно записувати в кілька топіків Kafka або в кілька директорій HDFS.

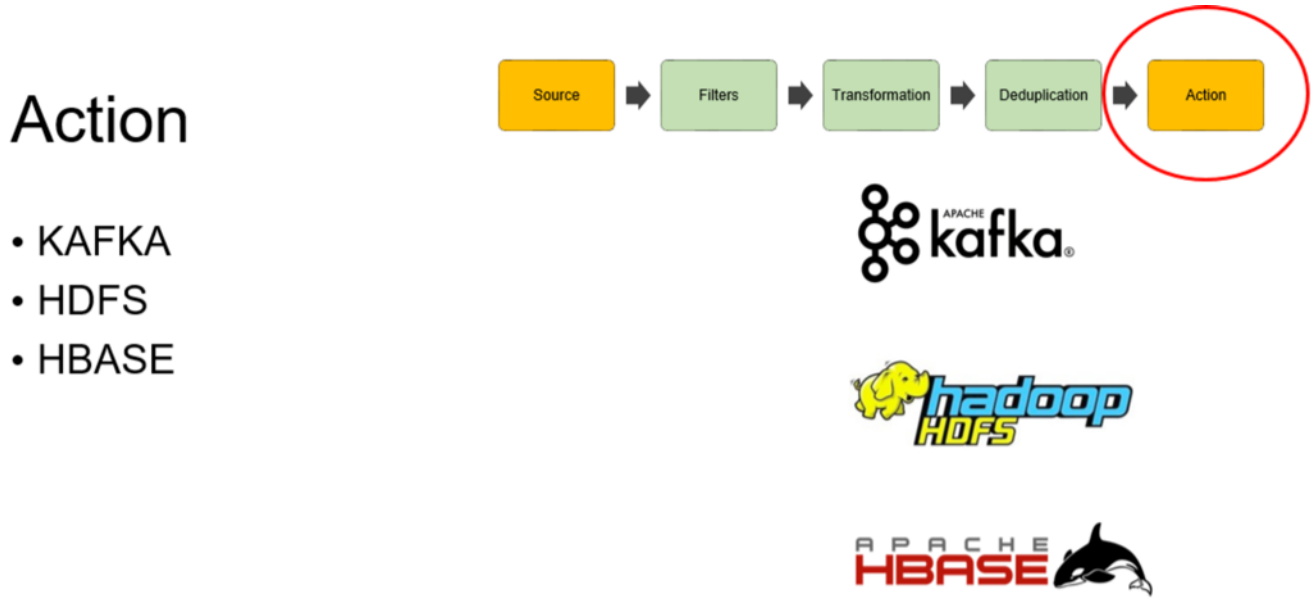

Рисунок 3.3 — Обробники Action[4]

Раніше вже зазначалося, що Action відповідає за прийняття рішення стосовно даних, коли він отримує їх у кінцевому стані. Припустимо, що існує простий DSL з кількома фільтрами, об'єднаними логічним оператором AND. Повідомлення зчитуються з джерела у потік. Існують вирази та фільтри, кожен з яких бере участь у обробці повідомлень. Отже, кожне повідомлення проходить через всі фільтри, вказані у DSL. Кожен фільтр встановлює позначку, а точніше біт у бітовій масці цього повідомлення, що відображає успішне проходження фільтрації.

Відповідно, пройшовши кожен фільтр, кожне повідомлення набуває відповідного стану. Коли повідомлення доходить до Action, він перевіряє так званий тригер. Тригер порівнює стан бітової маски конкретного повідомлення з умовою DSL-фільтрації, щоб визначити подальші дії з даними. Якщо умови бітових масок співпадають, дані передаються в Action, і Action вже знає, куди далі відправляти ці дані. Якщо ні, то дані будуть відфільтровані.

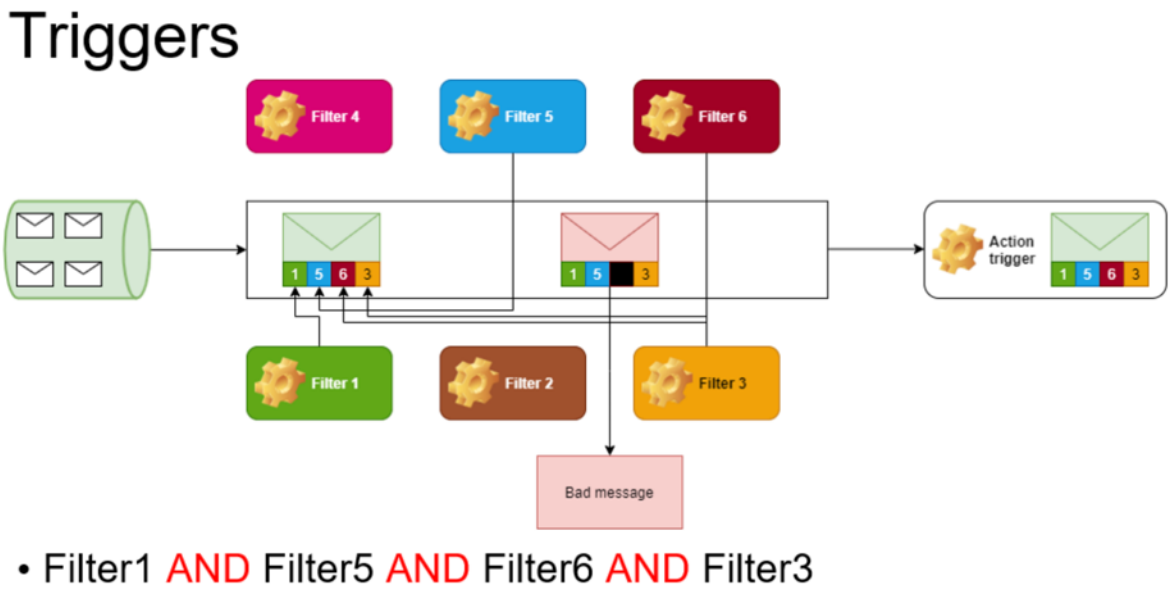

Рисунок 3.3 — Triggers[4]

Action реалізується за допомогою PartialFunction, що включає методи isDefinedAt та apply. Окрім цього, Action має додаткові методи, такі як onPartitionEnd та onRDDEnd. Метод onPartitionEnd необхідний для реалізації, наприклад, HDFS Action. У процесі запису даних у HDFS, цей метод забезпечує ротацію файлів залежно від часу та розміру. Метод onRDDEnd використовується для надсилання додаткових метрик після успішної обробки RDD, крім технічних метрик, таких як метрики даних для контролю якості даних (Data Quality).

Для зберігання конфігурацій було створення незалежного інструменту, що не вимагає від користувачів складних навичок. З цієї причини, очевидним рішенням здалося створити структурований файл для зберігання конфігурацій. Такий підхід добре працює, доки кількість пайплайнів обмежується п'ятьма. Однак, зі зростанням їх числа до 10, 20 або 30, керування та моніторинг пайплайнів стає значно складнішим.

Тому варто розглянути можливість використання Postgres для зберігання пайплайнів та обробників. Це рішення відповідає моделі, яку пропонує Postgres, дозволяючи розділити дані та етапи обробки на таблиці, пов'язані між собою. В

цьому контексті, кумулятивною таблицею виступає Action, що містить результати всіх обробників пайплайна.

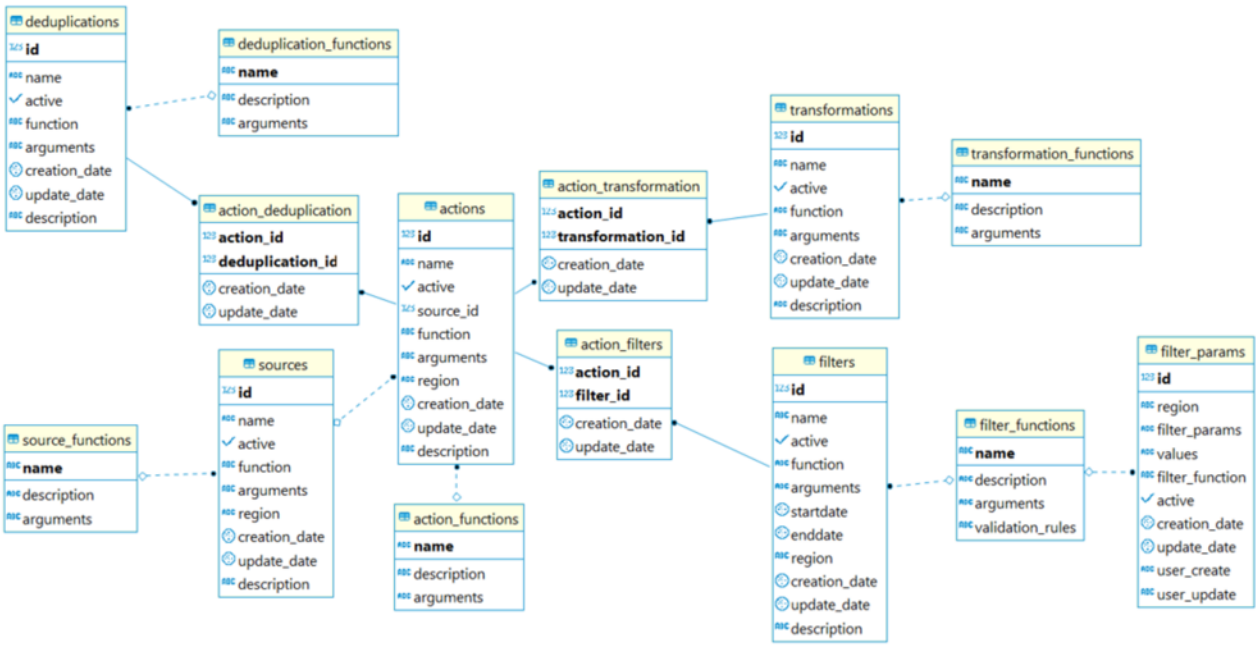

Рисунок 3.3 — Обробники Action[Рисунок виконаний самостійно]

При першому запуску розробки конфігураційні показники для Spark та Kafka не було проаналізовано та відповідно налаштовано, використовувались значення за замовчуванням. Процес оптимізації цих інструментів був довгим і складним, проте було виявлено кілька параметрів, які допомогли підвищити продуктивність та відмовостійкість системи.

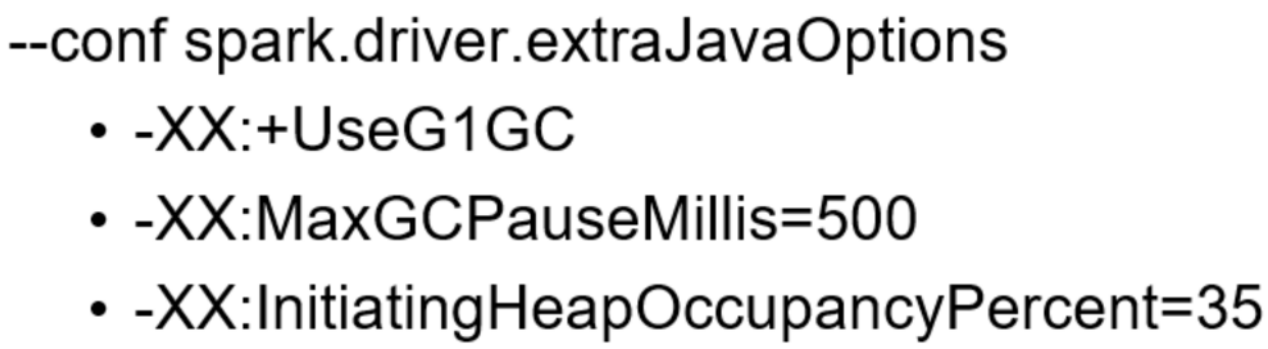

Рисунок 3.3— Spart Streaming tunning[8]

Спочатку, для зменшення навантаження на збирання сміття (GC), оптимізувати сукупність параметрів. Зокрема, важливо використовувати G1 Garbage Collector, який ефективно працює з великими об'ємами пам'яті, що підтверджено тестами. Встановлення максимальної паузи GC у 500 мілісекунд є рекомендаційним параметром, який може бути ігнорований у більшості випадків, але приблизно в 3% ситуацій допоможе зекономити ресурси.

Рекомендується налаштувати параметр заповнення купи перед паралельним збиранням сміття. Емпірично встановлено, що 35% є оптимальним значенням для ефективного використання ресурсів і зменшення пауз "stop the world".

Щодо параметру Concurrent, він дозволяє поліпшити паралельність виконання задач (job) у пайплайнів. В данному інструменті пайплайни реалізовані так, що максимум два задачі виконуються послідовно під час обробки мікробатчів. За замовчуванням, Spark Streaming налаштовано на послідовне виконання задач. Якщо налаштувати п'ять пайплайнів, то в Spark Streaming-додатку буде 10 задач, які виконуються послідовно. Однак, це не підходить в рамках цього дослідження, оскільки задачею задіяти одночасно є багато процесорів для пришвидшення обробки.

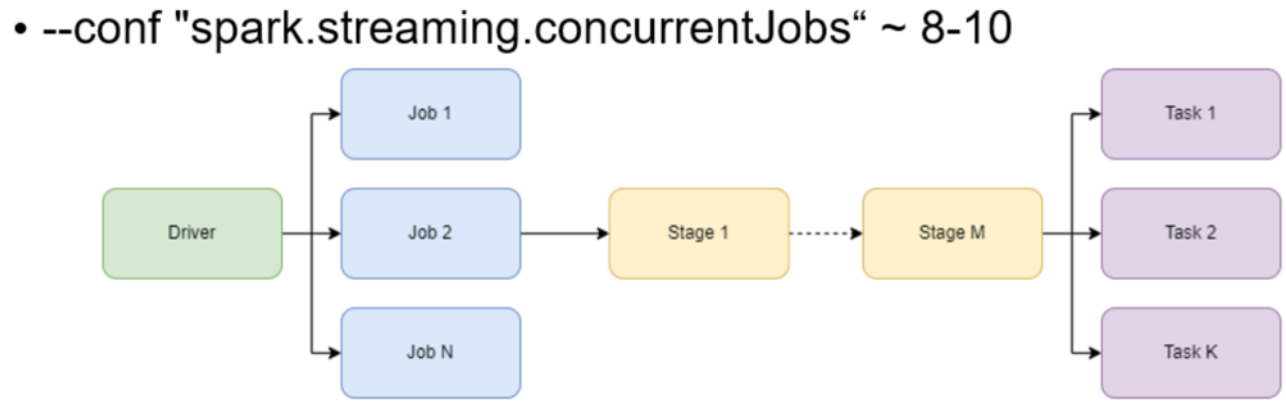

Рисунок 3.3 – Spart Streaming tunning[8]

Отже, при налаштуванні п'яти пайплайнів встановлюно 10 одночасних задач (concurrent job), що дозволяє усім пайплайнам працювати паралельно. Іноді задача може "зависнути" через проблемну таску, що займає один процесор, у той час як інші процесори просто очікують. Встановленням правильного значення concurrent

job можна запустити наступний мікробатч без очікування завершення попереднього. Параметр Speculation є важливим, оскільки він дозволяє автоматично перезапускати "завислі" таски у Spark Streaming. Spark Streaming визначає "завислу" таску на основі параметрів multiplier і quantile.

Припустимо, що multiplier=3 і quantile=0.9. Це означає, що якщо час виконання таски втричі перевищує медіанний час виконання 90% інших тасок, вона вважається "завислою". Наприклад, якщо з 10 задач, дев'ять виконалися за одну секунду, а одна працює вже чотири секунди, необхідно її перезапустити.

- --conf "spark. Speculation=true"
- --conf "spark.speculation.multiplier=3"
- --conf "spark.speculation.quantile=0.90“

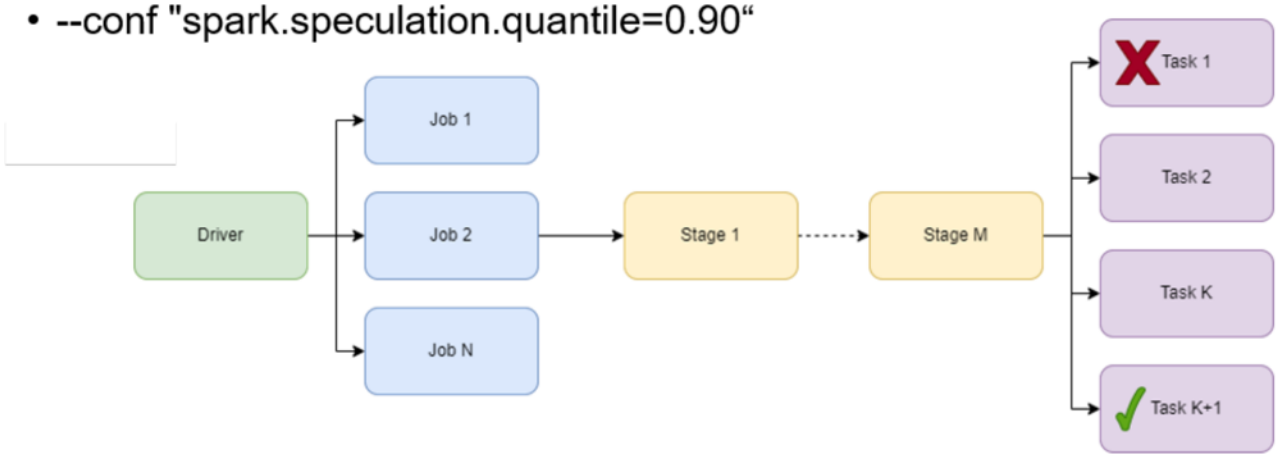

Рисунок 3.3 — Spart Streaming tunning[8]

У 70% випадків Spark успішно перезапускав "завислі" задачі, які після перезапуску продовжували працювати ефективно. Це може відбуватися через різні інтеграції з джерелами даних, де вказані часові обмеження (таймаути). Наприклад, якщо інтервал між батчами встановлений на 10 секунд, але потрібно обробляти дані кожні 10 секунд, але таймаут встановлений на 4 хвилини. Внаслідок цього, таска може "зависати" протягом чотирьох хвилин, перш ніж таймаут спрацює, і задача перезапуститься. В таких випадках, параметр Speculation виявився надзвичайно корисним для оптимізації роботи Spark Streaming.

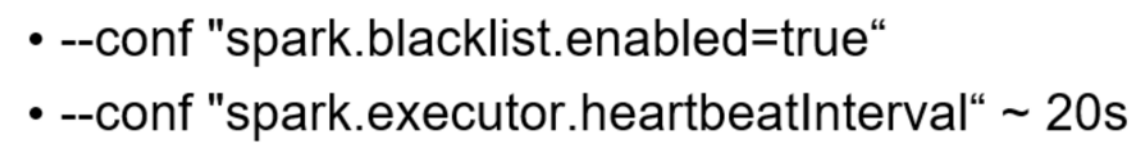

Рисунок 3.3 ─ Оптимізація, налаштування чорного списку[8]

Ще одним важливим параметром є Blacklist. У ситуації, коли Spark Streaming запускає екзек'ютор на певній ноді, яка постійно викликає збої, цей параметр додає зазначену ноду до чорного списку. Протягом певного часу екзек'ютори на цій ноді не запускаються, що дозволяє зберегти продуктивність системи, особливо під час пікового навантаження. У такому випадку не потрібно вручну відключати ноду від кластера для аналізу причин проблеми, оскільки вона продовжує перебувати в кластері. Цей параметр допомагає автоматизувати процес ідентифікації та вирішення проблем із нодами, що працюють ненадійно.

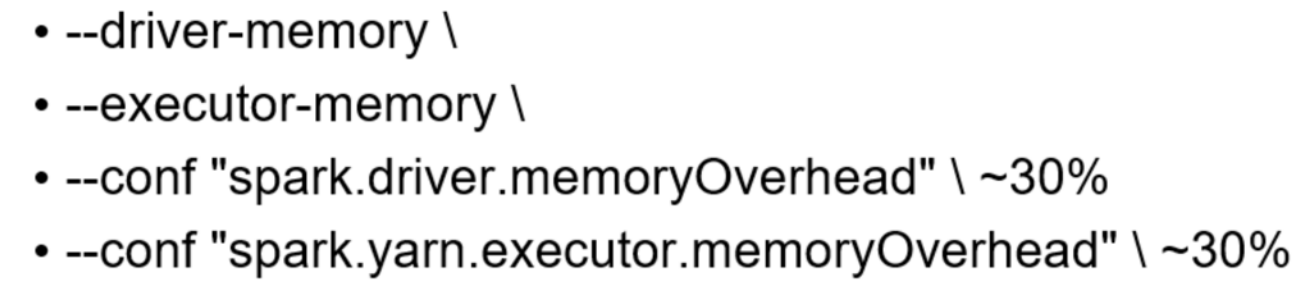

Рисунок 3.3 ─ Оптимізація, налаштування пам’яті[8]

Ще одним важливим параметром є Memory. Більшість користувачів знайомі з параметрами driver memory та executor memory, однак часто забувають про memory overhead. В Spark memory model цей параметр необхідний для задоволення різноманітних потреб JVM, наприклад, для інтернування рядків. При обробці великих потоків даних, дефолтне значення у 10% від основної пам'яті може бути недостатнім. Тому, використовуючи емпіричний підхід, ми встановили, що

приблизно 30% від цієї пам'яті достатньо для наших потреб, і екзек'ютори при цьому не викликають помилки через нестачу пам'яті (out of memory).

Рисунок 3.3 ─ Оптимізація, налаштування Kafka[8]

Варто звернути увагу на конкретні параметри Kafka, особливо на maxRatePerPartition, який має корелювати з інтервалом мікробатча. MaxRatePerPartition говорить про те, яку максимальну кількість записів буде прочитано за одну секунду з однієї партиції одного топіка.

Це, звісно, потрібно обчислювати в потоці емпіричним шляхом. Дуже допомогла ось така табличка, знайдена на просторах інтернету.

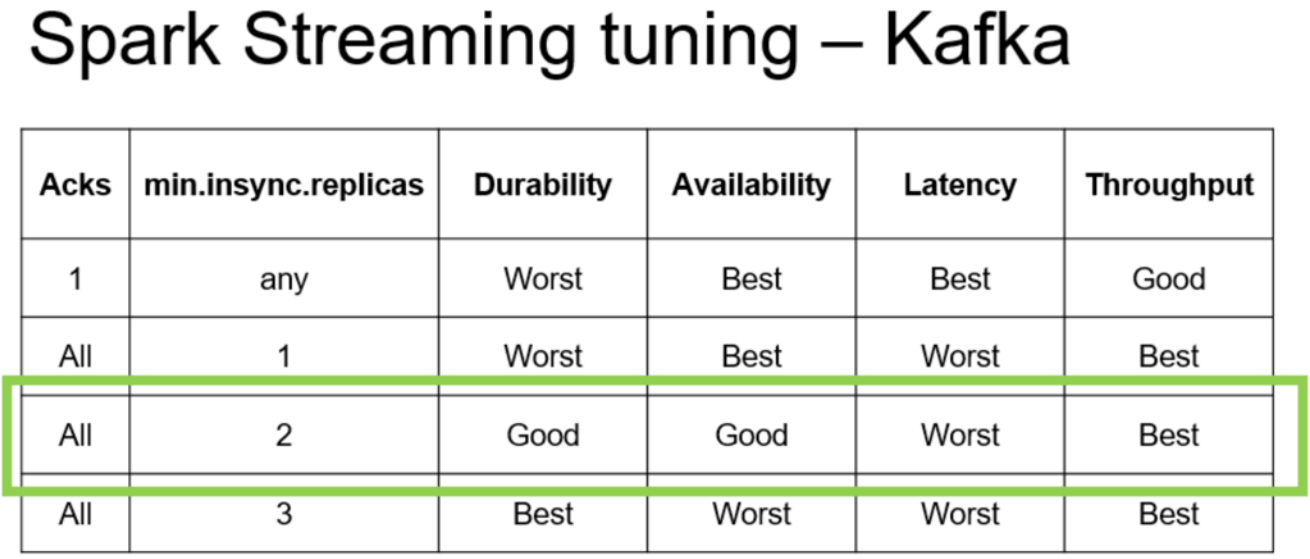

Рисунок 3.3 ─ Вибір оптимального значення[8]

Дослідження розглядає параметри та їх оптимальне використання, зокрема Acks та min.insync.replicas. Важливим аспектом цього дослідження було досягнення високої пропускної здатності, тоді як Latency не набувала значного

пріоритету, оскільки весь Latency в Kafka споживався на інтервали Spark Streaming. З огляду на високу доступність, було визначено, що Acks повинен бути встановлений на All, а min.insync.replica - на двійку.

Стосовно моніторингу, використовуються стандартні підходи. В Spark Streaming є багато технічних метрик. Якщо деяких метрик не вистачає, можна написати експортери на Python. Всі дані збираються в Pushgateway, звідки вони передаються в Prometheus та візуалізуються у Grafana. У разі проблем на продакшн-середовищі, сповіщення про це надходять через alert-manager на електронну пошту. Логи збираються стандартним способом, використовуючи ELK-стек: обробка відбувається в Logstash, зберігається в Elasticsearch та аналізується і візуалізується у Kibana.

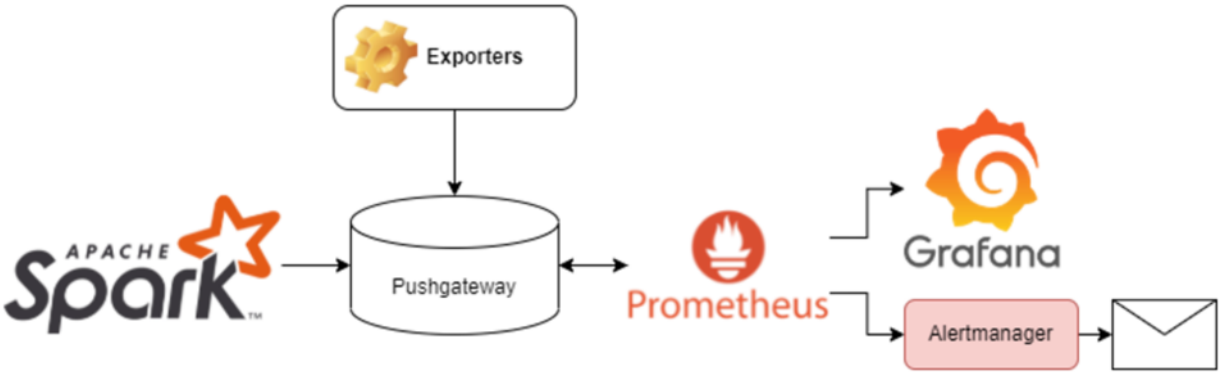

Рисунок 3.3 — Моніторинг[8]

У межах даного дослідження було використано кластер, що включає 18 DataNode, 3 вузли для керування кластером та додатковими компонентами. Характеристики кластеру передбачають наявність 36 ядер та 384 гігабайтів оперативної пам'яті. Варто зазначити використання технології Optane, яка додає 624 гігабайти до загального обсягу RAM. У якості менеджера ресурсів використовується YARN, який розпізнає терабайт пам'яті на одному вузлі. Оперативна пам'ять Optane має меншу швидкість, ніж звичайна RAM, але вона допомагає вирішувати завдання дедублікації. Крім того, кластер має 8 терабайтів вільного простору на жорстких дисках.

Configuration	Core	RAM, Gb	Optane, Gb	HDD, Gb
2xIntel Xeon Gold 6140(2.3GHz/18-core/24.75MB/140W) Processor (with heatsink)	36	384	624	8525

18 DataNodes + EdgeNode + Manage Node + NameNode

Рисунок 4.3.1 — Опис апаратної частини[Рисунок виконаний самостійно]

Також є окремий кластер під Kafka, під проміжні топіки. Це 5 KafkaNode, з практично такими ж характеристиками, тільки без Optane і з меншою кількістю HDD.

Configuration	Core	RAM, Gb	HDD, Gb
2xIntel Xeon Gold 6140(2.3GHz/18-core/24.75MB/140W) Processor (with heatsink)	36	384	6400

5 KafkaNode

Рисунок 4.3.2 — Опис апаратної частини[Рисунок виконаний самостійно]

Рекомендується розділити інстанси інструменту на два інстанси Spark Streaming, розділивши фільтрацію та дедублікацію, а також з'єднати їх проміжним Kafka топіком. Це пов'язано з тим, що фільтрація вимагає більше CPU та пам'яті для інтернування рядків, а для дедублікації необхідно більше пам'яті та деякої кількості CPU для ефективної роботи збірника сміття.

4.4. Аналіз результатів дослідження.

В результаті дослідження було проведено аналіз інструментів для створення конвеєрів обробки даних у реальному часі. З наведених інструментів Apache NiFi, Apache Spark, Apache Flink та Apache Kafka згідно до вимог дослідження а саме масштабованості та швидої обробки великих ланих було відібрано два інструменти з найращіми показниками а саме Apache Spark та Apache Flink.

Другим етапом дослідження було налаштування оптимальних показникив конфігураціх для кожного з інструмента. Попердня версіяя пайплайну використовувала показники за замочуванням що заважало нормальній роботі конвеєра та приводило до помилок та неочікуваної поведінки.

Третім етапом дослідження була побудова конвеєра обробки з використанням обраних інстурменів та підбором конігураційних показникік. Для тестування та оцінки метрик було використано тестування навантаження, а саме використано тестову вибірку подібну до реальної, заміри тестування показали гарні результати.

Підсумовуючи, очікувалося отримання 5-10 млн подій на секунду. Згідно з метриками Grafana, розподіленими за доменами, наразі досягнуто середньої кількості 6 млн подій на секунду та 7 млн під час пікових навантажень. Таким чином, ще збережено запас у 3 млн подій.

Рисунок 4.4 — Метрики[[Рисунок виконаний самостійно]]

Протягом року забезпечено 99,9% відмовостійкість, завдяки комбінації технологій та механізмам відмовостійкості Spark і Kafka. Масштабованість, забезпечена Spark і Kafka, включає як вертикальну, так і горизонтальну

масштабованість. Функціонал фільтрації, трансформації та дедублікації реалізований у пайплайні

Щодо налаштування обробки даних з мінімальним кодуванням (low code), для налаштування пайплайну достатньо виконати кілька інсертів.

ВИСНОВКИ

Об’єктом та метою дослідження данної магістерської роботи є методи створення конвеєрів для обробки даних в режимі реального часу. Аналіз інструментів для створення. Архітектура конвеєрів.

У результаті виконня роботи було проведено огляд та аналіз науково-технічної літератури що складає частину теоретичного дослідження роботи. Також було виконано аналіз практичну частину дослідження з описом та аналізом отриманого результату. Підготовка для дослідження методів та розроблена необхідна документація, визначено типи конвеєрів, необхідні вимоги для створення та підтримики конвеєра даних. Проведено аналіз предметнлї галузі під час котрого було визначено ключові концепції аналізу, проектування та створення конвеєрів даних, приклад кілької ключових міркувань, які слід враховувати при створення конвеєрів обробки даних. Технологічний стек, який було обговорено (Apache Kafka, Apache Spark), забезпечує надійну та масштабовану платформу для побудови конвеєрів обробки даних у реальному часі. Побудова ефективних і масштабованих конвеєрів обробки даних вимагає ретельного аналізу джерел даних, завдань обробки, архітектури, інструментів і технологій. Добре спроектований конвеєр може обробляти великі обсяги даних, забезпечувати низьку затримку, а також високу доступність і відмовостійкість.

Була також проведена оцінка продуктивності, вона має вирішальне значення для оптимізації конфігурації конвеєра і гарантування того, що конвеєр зможе впоратися з очікуваним обсягом даних і навантаженням на обробку. Вимірюючи такі показники продуктивності, як пропускна здатність, затримка, використання ресурсів, масштабованість і відмовостійкість, було виявлено найбільш оптимальну конфігурацію під вимоги бізнесу та цілі дослідження. Також було виявлено потенційні вузькі місця і оптимізовано конвеєр для ефективної обробки даних.

Однак конвеєри обробки даних у реальному часі також можуть стикатися з потенційними проблемами, такими як затримка в мережі, збої в роботі обладнання, проблеми з узгодженістю даних, проблеми з масштабуванням і загрози безпеці. Ці проблеми необхідно контролювати, щоб забезпечити надійність та ефективність

конвеєра. Ретельно розробляючи та оптимізуючи конфігурацію конвеєра, організації можуть створити надійні, масштабовані та високопродуктивні конвеєри, здатні впоратися з очікуваним обсягом даних і навантаженням на обробку.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. Marz, N., & Warren, J. (2015). Big Data: Principles and best practices of scalable real-time data systems. Manning Publications.
2. Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). Learning Spark:
3. Lightning-fast big data analysis. O'Reilly Media, Inc.
4. Dunning, T., & Friedman, E. (2018). Streaming Systems: The what, where, when, and how of large-scale data processing. O'Reilly Media, Inc.
5. Networked Systems Design and Implementation. USENIX Association.
6. Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging systemfor log processing. In Proceedings of the NetDB.
7. "Building Scalable and Fault-Tolerant Real-Time Data Processing Systems" by N. Marz.
8. MLOps Blog. “Comparing Tools For Data Processing Pipelines”, https://neptune.ai/blog/comparing-tools-for-data-processing-pipelines
9. "Real-time Data Processing with Apache Spark Streaming" by Matei Zaharia.
10. "Real-time Big Data Analytics: Emerging Architecture" by Yifeng Jiang.
11. "Real-time Data Processing with Apache Storm" by Taylor Goetz and James Roper.
12. Real-Time Data Warehousing: Data Management in Modern Time by Richard Hackathorn.
13. Real-Time Analytics with Storm and Cassandra" by Shilpi Saxena.
14. Real-Time Stream Processing and Analytics" by Srinath Perera and Anjana Wijekoon.
15. The Lambda Architecture: Principles for Architecting Real-Time Big Data Systems" by Nathan Marz.
16. Real-Time Analytics and Machine Learning: Principles, Techniques and Applications" by Parag Kulkarni.
17. "Apache NiFi: The Complete Guide - Part 1" by Raghavendar TS.
18. Real-time Data Analytics: Designing, Implementing, and Deploying Apache Flink by Shashank Tiwari.

19. Designing Real-Time Data Processing Systems by Wilson Lee.
20. Shliakhov, V., Chetverykov, G., Bozhko, I., Shliakhova, N. Processing and analysis of rhinomanometric signals by F-transform approximation Data Stream Mining & Processing (DSMP), IEEE First International Conference on, 2016, pp 314–317.
21. A. Yerokhin, A. Nechyporenko, O. Turuta, A. Babii. Application of Big Data methods in E-learning systems // 5th International Conference on Computational Linguistics and In-telligent Systems (COLINS-2021), Kharkiv, Ukraine, April 22-23, 2021. – CEUR Workshop Proceedings, 2021, 2870, Volume I, pp. 1302-1311

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

20. Shliakhov, V., Chetverykov, G., Bozhko, I., Shliakhova, N. Processing and analysis of rhinomanometric signals by F-transform approximation Data Stream Mining & Processing (DSMP), IEEE First International Conference on, 2016, pp 314–317.
21. A. Yerokhin, A. Nechyporenko, O. Turuta, A. Babii. Application of Big Data methods in E-learning systems // 5th International Conference on Computational Linguistics and In-telligent Systems (COLINS-2021), Kharkiv, Ukraine, April 22-23, 2021. – CEUR Workshop Proceedings, 2021, 2870, Volume I, pp. 1302-1311