ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження обробки великих данних на основі технологій Apache Spark »

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Антон Римаренко
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Антон Римаренко
група КНДМ-63

Керівник: Сергій Іщеряков
науковий ступінь, вчене звання д.т.н., професор

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Римаренко Антон Станіславович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження обробки великих данних на основі технологій Apache Spark

керівник кваліфікаційної роботи Сергій ІЩЕРЯКОВ д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: алгоритми обробки даних, емулятор сховища данних, середовище об'єктно орієнтованого проектуванняб
пояснювальна записка
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
позначка роботи, аналіз
проблемної галузі і постановка задачі,
опис проблеми обробки потоків даних,
використовувані методи та алгоритми,
опис розробленої програмної системи

5. Перелік графічного матеріалу: *презентація*

1.Мета завдання,

2. Обґрунтування доцільності розроблення

3. Постановка задачі,

4. Об'єктна модель системи

5. Базові моделі

6. Результати тестування продуктивності програмної системи

7. демонстраційні матеріали

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі та літератури	19.10-05.11.23	
2	Огляд методів аналізу данних	05.11-12.11.23	
3	Дослідження застосування хмарних технологій	13.11-19.11.23	
4	Аналіз особливостей роботи з локальними сховищами данних	20.11-25.11.23	
5	Дослідження технологій работи з базами данних	27.11-03.12.23	
6	Підготовка пояснювальної записки	04.12-10.12.23	
7	Підготоака до оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка презентаційниз матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Антон Римаренко
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Сергій Іщеряков
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 63 с., 10 рис., 2 додатки, 10 джерел.

Наукове завдання – Дослідити можливості обробки великих данних за допомогою технологій Apache Spark

Мета роботи – дослідження алгоритмів обробки великих потоків даних та будови системи Spark Streaming, порівняння патернів обробки потоку даних.

Об'єкт дослідження – є алгоритми обробки потоків даних.

Предмет дослідження – технології Apache Spark з використанням хмарних обчислень в AWS

Короткий зміст роботи: Методи розробки базуються на мовах програмування Scala, Java та Python.

Також використовуються наступні методи розробки: паралельні та розподілені обчислення, об'єктно-орієнтований та функціональний підхід розробки.

У результаті роботи були досліджені переваги та недоліки алгоритмів обробки потоків даних, структура системи Apache Spark з використанням хмарних обчислень в AWS.

КЛЮЧОВІ СЛОВА: ВЕЛИКІ ДАНІ, ОБЧИСЛЕННЯ БЛИЗЬКІ ДО РЕАЛЬНОГО ЧАСУ, ХМАРНИ ОБЧИСЛЕННЯ APACHE SPARK, BIG DATA, NEAR REAL TIME COMPUTING, HADOOP, SCALA, CLOUD COMPUTING, SPARK STREAMING, JAVA, AMAZON AWS, WEB SERVICES

ABSTRACT

Text part of the master's qualification work: 63p., 12 pic., 10 sources

The focus of this research centers on the algorithms involved in processing data streams under near real-time conditions within the framework of Big Data infrastructure, utilizing Apache Spark and leveraging cloud computing.

The primary objective is to develop an application for investigating algorithms related to data stream processing and exploring the architecture of a Big Data system by employing Spark Streaming. This involves comparing streaming processing techniques with incremental and batch engines.

The development methods primarily rely on the languages Scala, Java, and Python. Various development approaches, including parallel and distributed computing, as well as object-oriented and functional programming, are also employed.

The outcome of the research encompasses an exploration of algorithms for processing/streaming data and an examination of the structure of Apache Spark in conjunction with cloud computing through AWS.

KEYWORDS: APACHE SPARK, BIG DATA, NEAR REAL TIME COMPUTING, HADOOP, SCALA, CLOUD COMPUTING, SPARK STREAMING, JAVA, AMAZON AWS, WEB SERVICES

Зміст

ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 11
1.1 Постановка задачі, технології обчислення данних у сучасному світі 11
1.2 Дослідження методів аналізу даних у Big Data 13
1.3 Представлення даних в Big Data 17
2 ДОСЛІДЖЕННЯ МЕТОДІВ РОЗПОДІЛЕННОЇ ОБРОБКИ ДАНИХ В BD 19
2.1 Моделювання розподіленої системи оброби даних RDD 19
2.2 Обробка даних в Spark в режимі кластера 24
1.3 Spark SQL, особливості обробка даних 30
2.4 Spark Streaming, особливості обробки потоків 33
3 ПОРІВНЯННЯ АЛГОРИТТМИВ ОБРОБКИ ПОТОКІВ ДАНИХ 37
3.1 Обробка потоків даних з Spark Streaming 37
3.2 Відмовостійкість обробки потоків даних з Spark Streaming 41
3.3 Уніфікація пакетної, потокової та інтерактивної аналітики з Spark Streaming 44
3.4 Структурований потік та віконні операції 49
3.5 Порівняння API Spark та Flink 53
3.6 Інтеграція з Apache Kafka 55
ВИСНОВКИ 60
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 62

ВСТУП

Apache Spark з'явився у 2009 році в лабораторії AMPLab Каліфорнійського університету в Берклі як відповідь на обмеження фреймворку Hadoop MapReduce. Продиктований потребою у швидшій та універсальнішій обробці великих даних, Spark був розроблений з акцентом на обчисленнях в пам'яті. З роками вона перетворилася на надійну розподілену обчислювальну систему з відкритим вихідним кодом.

Основні віхи розвитку

2010 - Відкритий вихідний код: Apache Spark отримав відкритий вихідний код, заклавши основу для спільного проекту, керованого спільнотою. Цей крок ознаменував значний зсув у ландшафті великих даних.

2014 - Spark 1.0: Випуск Spark 1.0 став важливою віхою, представивши API Spark Core і підкресливши можливості Spark для розподіленої обробки даних.

2016 - Spark 2.0: У версії 2.0 з'явився API DataFrames, структуроване потокове передавання даних, а також значні покращення продуктивності та зручності використання.

Apache Spark продовжує розвиватися, з регулярними оновленнями, що впроваджують вдосконалення, оптимізацію та нові функції, що дозволяє йому залишатися на передовій технологій великих даних.

Здатність Spark кешувати дані в пам'яті, а не покладатися виключно на зберігання на диску, прискорює швидкість обробки, роблячи його до 100 разів швидшим за традиційний MapReduce.

Універсальність Spark як єдиного рушія для різноманітних робочих навантажень, включаючи пакетну обробку, інтерактивні запити, потокову аналітику та машинне навчання, спрощує розробку та зменшує потребу в декількох спеціалізованих системах.

Spark забезпечує відмовостійкість завдяки інформації про родовід, що дозволяє відновлювати втрачені дані без перезапуску всіх обчислень, підвищуючи надійність у розподілених середовищах.

Напрямки реалізації

1. Масштабована обробка даних:

Виняткова швидкість і масштабованість Spark роблять його ідеальним вибором для обробки великих масивів даних, полегшуючи роботу таких додатків, як аналіз лог-файлів, системи рекомендацій і великомасштабні ETL-процеси.

2. Потокова обробка в реальному часі:

Spark Streaming дозволяє обробляти потоки даних в режимі реального часу, знаходячи застосування в таких сценаріях, як виявлення шахрайства, інформаційні панелі в реальному часі та аналітика соціальних мереж.

3. Машинне навчання та штучний інтелект:

MLlib, бібліотека машинного навчання Spark, дозволяє розробникам створювати та розгортати масштабовані моделі машинного навчання, позиціонуючи Spark як ключового гравця в додатках на основі штучного інтелекту.

Коли ми розмірковуємо над історією Apache Spark та різноманітними сферами його застосування, стає очевидним, що його шлях є свідченням постійно мінливого ландшафту обробки великих даних. Постійні інновації та адаптивність Spark забезпечують його актуальність у формуванні майбутнього аналітики та додатків на основі даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Постановка задачі, технології обчислення данних у сучасному світі

Цифрова епоха стала свідком безпрецедентного вибуху даних, що трансформує спосіб ведення бізнесу та прийняття рішень. Зростання обсягів даних вражає: за оцінками, загальний обсяг даних, що генеруються у світі, досягне зеттабайт у найближчі роки. Цей сплеск даних зумовлений різними факторами, зокрема поширенням пристроїв, підключених до Інтернету, зростанням соціальних мереж і все більшою оцифровуванням бізнес-процесів.

Інтернет речей (IoT):

Широке впровадження пристроїв Інтернету речей призвело до напливу даних з датчиків, розумних пристроїв та інших підключених кінцевих точок. Цей постійний потік даних є різноманітним і об'ємним, що робить значний внесок у загальний інформаційний ландшафт.

Платформи соціальних мереж, а також контент, створений користувачами, роблять значний внесок у цей потік даних. Безперервне створення і поширення тексту, зображень, відео та іншого мультимедійного контенту щосекунди генерує величезні обсяги даних.

Корпоративні додатки та транзакції:

Підприємства все більше покладаються на цифрові додатки для різних операцій, що призводить до різкого зростання обсягів транзакційних даних. Від взаємодії з клієнтами до фінансових транзакцій, підприємства генерують колосальні набори даних, які вимагають складних можливостей обробки.

Можливості обробки в пам'яті Apache Spark задовольняють потребу у швидкості обробки великих наборів даних. Традиційні фреймворки пакетної обробки, такі як Hadoop MapReduce, часто страждають від вузьких місць дискового вводу/виводу, що робить їх менш придатними для ітеративних алгоритмів та обробки даних у реальному часі. Здатність Spark кешувати дані в

пам'яті значно прискорює обчислення, надаючи вирішальну перевагу в сучасному швидкоплинному бізнес-середовищі.

Існує велика кількість методів класифікацій даних і методів їх обробки.Слід зрозуміти, що відрізняє характеристики і методи BD від інших. Нерідко використовується характеристика, позначена дослідницькою компанією Gartner: «BD характеризується своїм обсягом, різноманітністю і швидкістю, з якою структуровані, слабоструктуровані і неструктуровані дані надходять по мережах передачі в процесори і сховища, поряд з процесами перетворення цих даних в цінну для бізнесу або науки інформацію».

Уніфікований аналітичний движок Spark розроблений для безперешкодної обробки різноманітних робочих навантажень. Незалежно від того, чи потрібна організаціям пакетна обробка, інтерактивні запити, потокова обробка або машинне навчання, Spark надає уніфіковану платформу. Ця універсальність усуває необхідність розгортання та управління кількома спеціалізованими системами, спрощуючи розробку та обслуговування додатків для роботи з великими даними.

В епоху, коли інформація в режимі реального часу має першорядне значення, Spark Streaming, компонент Apache Spark, дозволяє організаціям обробляти та аналізувати дані в режимі реального часу. Від моніторингу тенденцій у соціальних мережах до виявлення аномалій у фінансових операціях, Spark Streaming дозволяє компаніям швидко реагувати на мінливі сценарії.

Масштабованість Spark має вирішальне значення для обробки постійно зростаючих обсягів даних. Розподіляючи дані між кластером машин, Spark дозволяє організаціям здійснювати горизонтальне масштабування, задовольняючи зростаючі потреби в обробці великих обсягів даних. Така масштабованість гарантує, що Spark залишається надійним рішенням, оскільки обсяги даних продовжують зростати в геометричній прогресії.

MLlib, бібліотека машинного навчання Spark, відіграє ключову роль у вилученні значущої інформації з величезних наборів даних. Оскільки бізнес все більше покладається на передову аналітику та машинне навчання для отримання

конкурентної переваги, модель розподілених обчислень Spark полегшує розробку та розгортання масштабованих моделей машинного навчання.

Зростання обсягів даних у цифрову епоху змінило бізнес-ландшафт, створивши як виклики, так і можливості. Apache Spark став стрижнем у вирішенні складнощів обробки великих обсягів даних. Його швидкість, універсальність, масштабованість та підтримка обробки даних у режимі реального часу роблять його незамінним інструментом для організацій, які прагнуть використати потенціал своїх даних.

В майбутньому важливість Apache Spark в управлінні, аналізі та отриманні інформації з великих даних буде зростати. Його безперервна еволюція та адаптивність позиціонують його як ключового гравця в постійно зростаючій сфері аналізу даних, гарантуючи, що підприємства можуть орієнтуватися в складнощах ландшафту великих даних і розкрити весь потенціал своїх інформаційних активів.

1.2 Дослідження методів аналізу даних у Big Data

Безпрецедентне зростання обсягів, швидкості та різноманітності даних зумовило потребу в складних методах аналізу даних в епоху великих даних. Дослідники і практики постійно вивчають інноваційні методи і методології, щоб витягувати значущі ідеї, виявляти закономірності і приймати обґрунтовані рішення з величезних і складних наборів даних.

Масштаби даних продовжують зростати, дослідники зосереджуються на розробці масштабованих і розподілених методів обчислень. Це передбачає оптимізацію алгоритмів і фреймворків для ефективної обробки та аналізу великих наборів даних у розподілених обчислювальних середовищах. Apache Spark, Hadoop та інші платформи розподілених обчислень знаходяться в авангарді цих досліджень.

Компанія Google відіграла ключову роль у ранньому розвитку та популяризації технологій великих даних, зробивши значний внесок у розвиток

цієї галузі. Деякі ключові аспекти ролі Google у сфері великих даних на ранніх стадіях включають в себе наступні:

Впровадження Google File System (GFS) на початку 2000-х років заклало основу для розподілених систем зберігання даних. GFS була розроблена для обробки величезних обсягів даних у розподіленій інфраструктурі, забезпечуючи відмовостійкість і масштабованість.

Публікація Google про MapReduce у 2004 році представила модель програмування для обробки та генерації великих наборів даних. MapReduce спростила розробку додатків для розподіленої обробки даних, зробивши роботу з великими масивами даних більш доступною для розробників.

Bigtable від Google, представлений у статті 2006 року, сприяв розвитку баз даних NoSQL. Це була високомасштабована і розподілена система зберігання, призначена для обробки великих обсягів структурованих даних з низькою затримкою доступу.

Впливові статті Google про GFS та MapReduce надихнули на розробку Apache Hadoop, фреймворку з відкритим вихідним кодом для розподіленого зберігання та обробки великих даних. Hadoop, спочатку створений в рамках проекту Apache Nutch, віддзеркалював багато концепцій технологій Google.

Внутрішня інфраструктура, включаючи такі інструменти, як Dremel, і технології, такі як Borg (кластерний менеджер), послужили натхненням для різних проектів з відкритим вихідним кодом у сфері великих даних. Наприклад, Apache Drill черпає натхнення з Dremel від Google.

Вони активно долучилися до спільноти з відкритим вихідним кодом, ділячись ідеями та технологіями через наукові роботи та співпрацю. Це вплинуло на розвиток різних інструментів для роботи з великими даними, окрім Hadoop, таких як Apache Spark та Apache Flink.

Google Cloud Platform (GCP), сервіс хмарних обчислень Google, пропонує низку сервісів для роботи з великими даними, зокрема BigQuery, Dataflow і Dataproc. Ці сервіси дозволяють користувачам обробляти та аналізувати великі масиви даних за допомогою керованої хмарної інфраструктури.

Хоча Google не спеціалізується на обробці великих даних, випуск TensorFlow, фреймворку машинного навчання з відкритим вихідним кодом, суттєво вплинув на інтеграцію машинного навчання з аналітикою великих даних. Внесок Google у технології великих даних не лише вплинув на розвиток проектів з відкритим вихідним кодом, але й сформував підхід організацій до зберігання, обробки та аналізу даних у великих масштабах. Принципи та технології, запроваджені Google, продовжують відігравати важливу роль у ширшій екосистемі великих даних.

Дослідження в галузі машинного навчання та предиктивної аналітики спрямовані на розробку алгоритмів, здатних вивчати закономірності та робити прогнози на основі великих наборів даних. Сюди входять дослідження ансамблевих методів, глибокого навчання та навчання з підкріпленням для підвищення точності та масштабованості предиктивних моделей, що застосовуються до великих даних.

Зі зростанням уваги до прийняття рішень у режимі реального часу, дослідження спрямовані на розробку методів обробки та аналізу потокових даних у реальному часі. Це передбачає розробку алгоритмів і систем, які можуть безперервно обробляти потоки даних, забезпечуючи миттєве розуміння і швидке реагування на мінливі умови.

Багато реальних наборів даних мають графоподібну структуру, наприклад, соціальні мережі, транспортні системи та біологічні мережі. Дослідження в галузі графоаналітики зосереджені на розробці ефективних алгоритмів для таких завдань, як виявлення спільнот, аналіз центральності та виявлення аномалій у великомасштабних наборах графових даних.

Оскільки занепокоєння щодо конфіденційності та безпеки даних зростає, дослідники вивчають методи виконання змістовного аналізу чутливих даних без шкоди для особистої конфіденційності. Це передбачає розробку таких методів, як федеративне навчання, гомоморфне шифрування та диференціальна конфіденційність.

Оскільки високорозмірні набори даних стають звичним явищем, дослідження в галузі інженерії ознак та зменшення розмірності є надзвичайно важливими. Методи, які автоматично відбирають релевантні ознаки, зменшують шум і покращують інтерпретованість моделей, сприяють ефективному аналізу в умовах великих даних.

Виклики та майбутні напрямки:

Інтеграція знань з конкретних галузей у методи аналізу даних залишається складним завданням. Дослідники вивчають способи безперешкодного включення доменної експертизи в алгоритми, щоб підвищити релевантність та інтерпретованість результатів.

Інтеграція та інтероперабельність різних інструментів і платформ для аналізу даних також викликає труднощі. Поточні дослідження спрямовані на розробку стандартизованих фреймворків та інтерфейсів, що сприятимуть безперешкодній співпраці між різними інструментами та середовищами.

Зі зростанням впливу аналізу даних на суспільство зростає увага до етичних та відповідальних практик. Дослідження в цій галузі стосуються таких питань, як упередженість алгоритмів, справедливість у прийнятті рішень та прозорість процесів аналізу даних.

Інтерпретованість складних моделей має вирішальне значення для побудови довіри та взаєморозуміння в аналізі даних. Дослідження зосереджені на розробці методів для покращення пояснюваності та інтерпретованості моделей машинного навчання, особливо у сценаріях з великою кількістю складних даних.

Дослідження методів аналізу великих даних - це динамічна і багатогранна галузь, яка відіграє ключову роль у розкритті потенціалу величезних і різноманітних наборів даних. Оскільки ландшафт великих даних продовжує розвиватися, дослідники продовжуватимуть шукати інноваційні рішення для вирішення нових викликів і розширювати межі можливого в аналізі даних.

Постійна співпраця між науковими колами, промисловістю та спільнотами з відкритим вихідним кодом гарантує, що найновіші результати досліджень

перетворюються на практичні інструменти та методології, що в кінцевому підсумку формує майбутнє аналізу даних в епоху великих даних.

1.3 Представлення даних в Big Data

Дані - це необроблені факти, цифри або інформація, часто у вигляді чисел, тексту, зображень або в інших форматах. У контексті обчислень та аналітики дані є основою для отримання інсайтів, прийняття обґрунтованих рішень та розв'язання складних проблем.

Типи даних

1. Структуровані дані:

Визначення: Добре організовані дані з фіксованою схемою.

Приклад: Реляційні бази даних, електронні таблиці.

Характеристики: Легко піддаються запитам, підходять для традиційних систем баз даних.

2. Неструктуровані дані:

Визначення: Дані, що не мають наперед визначеної структури або формату.

Приклад: Текстові документи, зображення, відео.

Характеристики: Потребують розширеної обробки для аналізу, поширені у великих даних.

3. Напівструктуровані дані:

Визначення: Дані з певною структурою, але не такою жорсткою, як структуровані дані.

Приклад: JSON, XML.

Характеристики: Поєднує елементи структурованих і неструктурованих даних.

4. Тимчасові дані:

Визначення: Дані, пов'язані з часом або часовими подіями.

Приклад: Дані часових рядів, журнали подій.

Характеристики: Цінні для аналізу тенденцій та прогнозування.

5. Просторові дані:

Визначення: Дані, пов'язані з географічним розташуванням.

Приклад: Карти, GPS координати.

Характеристики: Корисні для географічного аналізу та картографування.

Організація даних у потоці великих даних:

1. Поглинання даних:

Процес: Захоплення та імпорт даних до системи.

Інструменти: Apache Kafka, Apache NiFi.

Важливість: Забезпечує безперешкодне введення даних у конвеєр великих даних.

2. Зберігання даних:

Процес: Зберігання даних для подальшого пошуку та аналізу.

Інструменти: Розподілена файлова система Hadoop (HDFS), Amazon S3.

Важливість: Забезпечує масштабоване та відмовостійке зберігання великих масивів даних.

3. Обробка даних:

Процес: Аналіз і перетворення даних для отримання інсайтів.

Інструменти: Apache Spark, Apache Flink.

Важливість: Ефективно виконує складні обчислення на великих масивах даних.

4. Аналіз та дослідження даних:

Процес: Отримання значущих висновків з даних.

Інструменти: Apache Hive, Apache Impala.

Важливість: Сприяє дослідженню та вилученню дієвої інформації.

5. Візуалізація даних:

Процес: Візуальне представлення даних для кращого розуміння.

Інструменти: Tableau, Power BI.

Важливість: Покращує передачу інсайтів за допомогою діаграм, графіків та інформаційних панелей.

6. Експорт даних:

Процес: Переміщення даних до зовнішніх систем або сховищ.

Інструменти: Sqoop, Apache NiFi.

Важливість: Полегшує обмін або архівування оброблених даних.

7. Управління даними та безпека:

Процес: Забезпечення якості, цілісності та безпеки даних.

Інструменти: Apache Ranger, Apache Atlas.

Важливість: Підтримує надійність та конфіденційність даних.

Представлення даних в контексті великих даних - це багатогранна концепція, яка передбачає розуміння природи даних, їх типів і того, як вони проходять через різні етапи конвеєра обробки даних. Оскільки організації стикаються з постійно зростаючими масивами даних, здатність ефективно організовувати, обробляти та отримувати інформацію з різних джерел даних набуває першочергового значення. Використовуючи відповідні інструменти та методи, компанії можуть використовувати силу даних для прийняття обґрунтованих рішень і отримати конкурентну перевагу в сучасному світі, що базується на даних.

2 ДОСЛІДЖЕННЯ МЕТОДІВ РОЗПОДІЛЕННОЇ ОБРОБКИ ДАНИХ В BD

2.1 Моделювання розподіленої системи оброби даних RDD

Стійкі розподілені структури даних (Resilient Distributed Datasets, RDD) - це фундаментальні структури даних в Apache Spark, що представляють собою незмінну, розподілену колекцію об'єктів.

RDD лежать в основі Apache Spark, відіграючи ключову роль у полегшенні розподіленої обробки даних. RDD - це відмовостійкі, розпаралелені колекції

даних, які можна обробляти розподіленим способом у кластері. Розуміння тонкощів RDD має вирішальне значення для використання повного потенціалу Spark у роботі з великими даними.

Програма-драйвер - це основна програма, яка виконує високорівневий потік керування завданням Spark.Вона визначає перетворення та дії, які будуть застосовані до RDD.

Spark RDD забезпечують розподілену обробку, розбиваючи дані на розділи, які обробляються незалежно на різних вузлах кластера.RDD підтримують відмовостійкість завдяки інформації про родовід, що дозволяє відновити втрачені дані у випадку збоїв у вузлах.

Перетворення створюють нові RDD з існуючих, тоді як дії обчислюють результат або вихідні дані.Перетворення RDD оцінюються ліниво, тобто вони виконуються лише тоді, коли викликається дія, що дозволяє оптимізувати плани виконання.

RDD Spark створюють лінійну діаграму, яка представляє послідовність перетворень, застосованих для створення певного RDD. У випадку відмови вузла, Spark може використовувати лінійну діаграму для відновлення втрачених даних, відстежуючи трансформації.

Програма-драйвер відповідає за визначення програми Spark, створення RDD та визначення перетворень і дій. Вона виконує основну функцію і керує SparkContext, який координує виконання завдань Spark. SparkContext є точкою входу до функціональності Spark у програмі-драйвері. Вона встановлює зв'язок з менеджером кластера Spark і координує виконання завдань між робочими вузлами.

Програма-драйвер створює завдання, які надсилаються на робочі вузли для виконання.Виконавці, запущені на робочих вузлах, виконують ці завдання паралельно на розділах RDD. Програма-драйвер спілкується з виконавцями для розподілу завдань і збору результатів.Вона керує загальним планом виконання, організовуючи паралельну обробку RDD на всьому кластері.

Операції на RDD

Операції перетворення:

Перетворення створюють новий RDD з існуючого, застосовуючи функцію до кожного елемента. Приклади включають map, filter і flatMap.

Всі перетворення в Spark є лінивими, в тому, що вони не обчислюють свої результати одразу. Натомість, вони просто пам'ятають перетворення, застосовані до деякого базового набору даних (наприклад, файл). Перетворення обчислюються лише тоді, коли дія вимагає повернення результату до програми драйвера. Ця конструкція дозволяє Spark працювати більш ефективно. Наприклад, ми можемо зрозуміти, що набір даних, створений через map, буде використовуватися у зменшенні і повертати тільки результат зменшення на драйвер, а не на більший нанесений набір даних.

Дії виконують обчислення над RDD і повертають результат програмі-драйверу або записують дані на зовнішню систему зберігання. Приклади включають count, collect і saveAsTextFile.

Парні RDD представляють пари ключ-значення і підтримують операції, характерні для цієї структури.Приклади включають reduceByKey, groupByKey і join.

Операції перемішування:

Shuffling -це процес обміну даними між розділами. В результаті, рядки даних можуть переміщатися між робочими вузлами, якщо їх вихідний і цільовий розділи знаходяться на різних машинах. Такі операції, як groupByKey, reduceByKey та join можуть спричинити shuffling. Shuffle - це дорога операція, оскільки вона включає в себе I/O диска, серіалізацію даних і мережевий I/O. Щоб

організувати дані для shuffle, Spark генерує набір тасків для організації даних, а також набір тасків зменшення, щоб агрегувати його. Ця номенклатура походить від MapReduce і безпосередньо не стосується Spark.

Використання groupByKey може мати значний вплив на продуктивність, коли дані вже розбито на розділи і відсортовано. Можна вважати гарною практикою надавати перевагу groupBy(...) (RelationalGroupedDataset) над groupByKey(...) (KeyValueGroupedDataset[K, V]).Якщо вам дійсно потрібно використовувати KeyValueGroupedDataset[K, V], використовуйте groupBy(...).as[K, V] замість groupByKey(...). Це дозволяє оптимізувати запит у Spark.

Shuffle також генерує велику кількість проміжних файлів на диску. Від Spark 1.3 ці файли зберігаються доти, доки відповідні RDD більше не будуть використовуватися. Це робиться для того, щоб файли перетасовування не потребували повторного створення, якщо повторні обчислення будуть необхідні.Збір сміття може відбутися лише після тривалого періоду часу, якщо програма зберігає посилання на ці RDD або якщо GC не починає часто виникати. Це означає, що довгострокові задачі Spark можуть споживати великий обсяг дискового простору. Каталог тимчасового зберігання задається параметром конфігурації spark.local.dir під час налаштування контексту Spark. У більшості випадках операція groupByKey може бути замінена операцією reduce by key

Стійкість і кешування:

Однією з найважливіших можливостей Spark є збереження (або кешування) набору даних у пам'яті через операції. Під час зберігання RDD, кожен вузол зберігає будь-які його розділи, які він обчислює в пам'яті, і повторно використовує їх в інших діях на цьому наборі даних (або наборів даних, отриманих з неї). Це дає змогу майбутнім діям працювати набагато швидше (часто більше 10x). Кешування є ключовим інструментом для ітераційних алгоритмів і швидкого інтерактивного використання.

Кешування дозволяє зберігати RDD у пам'яті або на диску, зменшуючи потребу у повторних обчисленнях.Такі операції, як persist і cache полегшують кешування.
Spark не переміщує дані між вузлами випадковим чином. Переміщення є трудомісткою операцією, тому вона виконується лише тоді, коли немає іншого виходу.

В Apache Spark створення RDD та їх перетворення призводить до формування спрямованого ациклічного графа (Directed Acyclic Graph, DAG). DAG поділяється на етапи, і кожен етап складається з послідовності перетворень, які можуть виконуватися паралельно. Етапи створюються на основі операцій, які викликають перестановку даних або вимагають переміщення даних між розділами.

Процес побудови графа взаємозв'язків RDD можна розділити на три основних етапу:

– виділення необхідної інформації для побудови графа взаємозв'язків;
– написання методів вилучення даної інформації;
– візуалізація графа на основі зібраних даних.

За замовчуванням кожне перетворене RDD може бути перераховано кожен раз при виконанні дії на ньому. Тим не менш, можливо також зберігати RDD в пам'яті, використовуючи метод persist (або кеш), в цьому випадку Spark зберігатиме елементи навколо кластера для набагато швидшого доступу наступного разу, коли йде запит. Існує також підтримка постійних RDD на диску або реплікації на декількох вузлах.

Spark використовує DAG для оптимізації плану виконання, визначаючи можливості для конвеєризації вузьких перетворень і мінімізації перетасування даних.DAG допомагає Spark приймати обґрунтовані рішення щодо найефективнішого способу обробки даних у розподіленому кластері].
Spark UI - це веб-інтерфейс користувача, що надається Apache Spark для моніторингу та налагодження додатків Spark.Він дає уявлення про продуктивність, використання ресурсів та деталі виконання завдань Spark.Spark UI відображає

інформацію про кожне завдання Spark, включаючи візуалізацію DAG, етапи та деталі їх виконання зображені на рис 2.1.

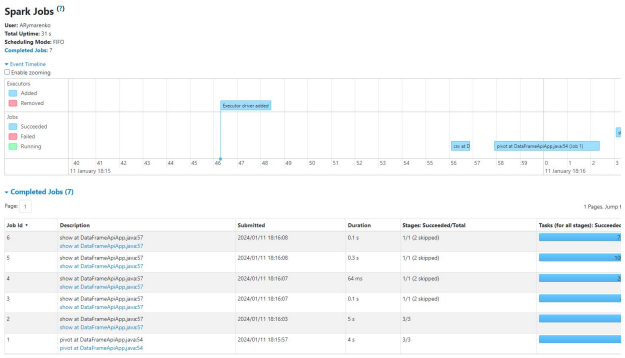


Рисунок 2.1 – приклад Spark UI

Користувачі можуть дослідити структуру DAG, зрозуміти етапи та відстежувати прогрес виконання завдань.Інтерфейс користувача Spark надає метрики, пов'язані з виконанням завдань, включаючи тривалість завдання, розміри вхідних/вихідних даних та використання ресурсів.

Користувачі можуть виявляти вузькі місця в продуктивності та аналізувати статистику на рівні завдань для оптимізації.Інтерфейс показує інформацію про середовище Spark, налаштування конфігурації, а також список активних виконавців та використання ними ресурсів.Користувачі можуть перевіряти конфігурації Spark і вирішувати проблеми, пов'язані з розподілом ресурсів.

2.2 Обробка даних в Spark в режимі кластера

Apache Spark, призначений для розподіленої обробки даних, працює в різних кластерних режимах для ефективного використання ресурсів на декількох машинах. Вибір режиму кластера залежить від таких факторів, як масштабованість, відмовостійкість та базовий менеджер кластерів.

Spark - це простий вбудований кластерний менеджер, який дозволяє запускати додатки Spark на кластері. Кластер Spark складається з головного вузла та декількох робочих вузлів. Головний вузол керує плануванням і розподілом

завдань між робочими вузлами. Додатки Spark надсилають завдання на головний вузол, який потім координує їх виконання між робочими вузлами.

Об'єкт SparkConf займає центральне місце в конфігуруванні додатків Spark, дозволяючи користувачам встановлювати різні властивості, щоб налаштувати їх виконання. При локальному запуску Spark додаток виконується на одній машині, використовуючи всі доступні ядра. На відміну від цього, на Amazon EMR (Elastic MapReduce), додатки Spark масштабуються за допомогою кластерів у хмарі.

SparkConf використовується для конфігурації додатків Spark, надаючи простий інтерфейс для налаштування таких властивостей, як ім'я додатку, основна URL-адреса та ресурси виконання. Об'єкт SparkContext, ініціалізований за допомогою SparkConf, слугує точкою входу для функціональності Spark, забезпечуючи зв'язок з менеджером кластеру та координацію розподілених завдань. Об'єкт SparkConf ініціалізується конфігураційними властивостями, такими як ім'я програми, головна URL-адреса (наприклад, "local[*]" для використання всіх доступних ядер) та інші налаштування. Ця конфігурація керує поведінкою програми Spark під час виконання.

Об'єкт SparkContext, створений з SparkConf, діє як координатор між програмою-драйвером і середовищем виконання Spark. Він керує виконанням завдань, координує дії з менеджером кластера (в даному випадку, локальною машиною) і займається розподілом завдань між робочими вузлами (локальними потоками або процесами).

Завдання Spark виконуються на локальній машині, використовуючи багатопотоковість для розпаралелювання операцій на доступних ядрах. Весь додаток Spark працює в межах однієї віртуальної машини Java (JVM). Автономний режим.Поширені практики локального запуску Spark включають використання скрипту spark-submit або запуск додатків Spark безпосередньо з інтегрованого середовища розробки (IDE), наприклад, ноутбуків PyCharm або Jupyter. Локальний запуск Spark забезпечує спрощене середовище для тестування і розробки без накладних витрат розподіленого кластера.

Запуск Spark локально означає спрощене налаштування для розробки та тестування. Однак на AWS EMR Spark легко інтегрується з хмарним середовищем, динамічно масштабуючи ресурси за потреби. JavaSparkContext, компонент SparkContext, взаємодіє з JVM, полегшуючи зв'язок між Spark та базовим середовищем виконання Java. Ця взаємодія дозволяє Spark ефективно виконувати завдання, використовуючи переваги управління пам'яттю JVM та незалежність від платформи.

Amazon EMR (Elastic MapReduce) - це хмарна платформа для роботи з великими даними, яку пропонує Amazon Web Services (AWS). EMR спрощує процес забезпечення, управління та масштабування розподілених обчислювальних кластерів для обробки великих наборів даних. Вона використовує популярні фреймворки з відкритим кодом, такі як Apache Spark, Apache Hadoop, Apache Hive та Apache HBase, що робить її універсальною для різних завдань обробки великих даних.

EMR підтримує Spark, швидку і універсальну платформу розподілених обчислень, що дозволяє користувачам виконувати обробку даних, машинне навчання та аналітику в масштабах. Додатки Spark на EMR виграють від безперешкодної інтеграції з іншими сервісами AWS, включаючи Amazon S3 для зберігання даних та Amazon DynamoDB для роботи з базами даних NoSQL, усі сервіси з якими може інтегрувати AWS EMR можно знайти на рис. 2.2 .

Рисунок 2.2 Інтеграція EMR кластеру з іншими сервісами

Коли користувачі ініціюють кластер EMR, вони можуть вказати бажані програмні додатки та конфігурації для кластера. EMR піклується про базову інфраструктуру, автоматично створюючи та налаштовуючи екземпляри Amazon EC2 як робочі вузли. Користувачі можуть вибирати різні типи екземплярів, щоб пристосувати кластер до своїх конкретних вимог.

Життєвий цикл кластера EMR:

Користувачі ініціюють кластер EMR через консоль управління AWS, AWS CLI або SDK. Параметри конфігурації включають програмні додатки, типи екземплярів і кількість екземплярів у кластері. EMR динамічно надає екземпляри Amazon EC2 на основі заданої конфігурації.Екземпляри слугують головним і робочим вузлами, причому головний вузол координує загальне виконання.

Додатки Spark, завдання Hadoop або інші завдання розподілених обчислень передаються на виконання кластеру. Головний вузол розподіляє завдання між робочими вузлами, використовуючи обчислювальну потужність всього кластера.
EMR підтримує розподілену обробку даних за допомогою таких фреймворків, як Spark, що дозволяє користувачам аналізувати, трансформувати та отримувати інформацію з великих наборів даних.Він легко інтегрується з іншими сервісами AWS, полегшуючи зберігання, пошук і взаємодію з різними компонентами екосистеми AWS. Кластери EMR можна динамічно масштабувати, додаючи або видаляючи робочі вузли відповідно до вимог робочого навантаження. Динамічне масштабування дозволяє користувачам адаптувати ресурси до мінливих вимог до обробки.

EMR надає можливості для оптимізації витрат, дозволяючи користувачам обирати між екземплярами на вимогу та точковими екземплярами, виходячи з бюджетних міркувань. Після завершення завдань обробки користувачі можуть завершити роботу кластера EMR, щоб припинити стягувати плату за ресурси. EMR спрощує складнощі управління розподіленими обчислювальними

кластерами, роблячи його доступним для користувачів, щоб використовувати потужність обробки великих даних у хмарі без необхідності в складному управлінні інфраструктурою.

В Apache Spark менеджер кластера відповідає за розподіл ресурсів і планування завдань на кластері машин. Кластерний менеджер відіграє вирішальну роль в управлінні виконанням додатків Spark і забезпеченні ефективного використання ресурсів. Як працює кластер менеджер можна подивитись на рис. 2.3.

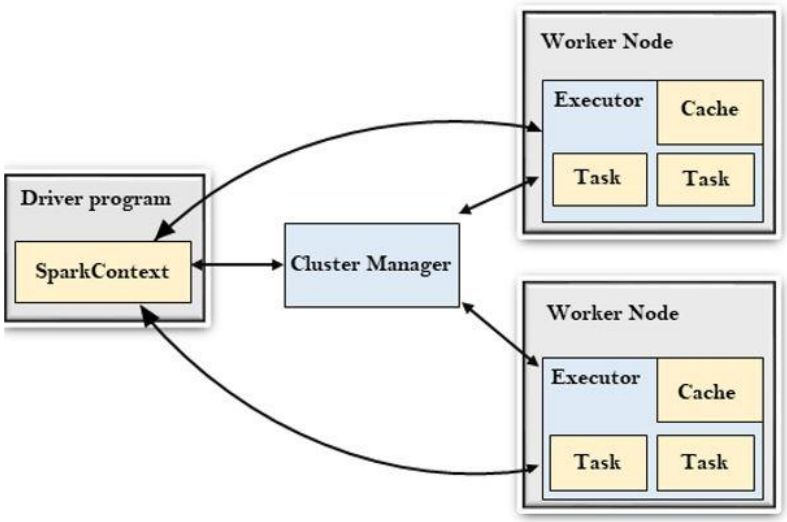


Рисунок 2.3 Кластер менеджер Spark

Існує кілька кластерних менеджерів, які підтримуються Spark:

1. Автономний менеджер кластерів:

Spark постачається з автономним кластерним менеджером, який є простою і вбудованою опцією для запуску додатків Spark на кластері. його легко налаштувати і він не вимагає зовнішнього кластерного менеджера. надає базові функції для розподілу ресурсів і планування. підходить для малих і середніх кластерів.

2. Apache Mesos:

Apache Mesos - це кластерний менеджер загального призначення, який можна використовувати для запуску додатків Spark разом з іншими розподіленими системами. Mesos забезпечує тонкий розподіл ресурсів та ізоляцію між додатками. Завдання Spark виконуються в контейнерах Mesos. Підходить для змішаних робочих навантажень з різними фреймворками.

В основі Mesos лежить архітектура "ведучий-ведений". Майстер Mesos керує ресурсами кластера, в той час як агенти (або підлеглі) Mesos працюють на кожному вузлі кластера, пропонуючи доступні ресурси майстру. Така абстракція дозволяє Mesos підтримувати декілька фреймворків одночасно, що робить його універсальним вибором для великомасштабних, багатокористувацьких середовищ.

3. Apache Hadoop YARN:

Apache Hadoop YARN (Yet Another Resource Negotiator) - це менеджер ресурсів в екосистемі Hadoop. YARN управляє ресурсами і планує завдання для додатків Spark в кластері Hadoop. Підтримує багатокористувацьку оренду і розподіл ресурсів між різними додатками.

4. Kubernetes:

Spark підтримує роботу на Kubernetes, системі оркестрування контейнерів. Kubernetes забезпечує динамічний розподіл ресурсів і автоматичне масштабування. додатки Spark працюють в контейнерах, керованих Kubernetes.

Кожен кластерний менеджер має свої сильні сторони та варіанти використання, і вибір часто залежить від таких факторів, як існуюча інфраструктура, вимоги до розгортання та інтеграція з іншими розподіленими системами. Програми Spark написані таким чином, щоб абстрагуватися від специфіки базового кластерного менеджера, що дозволяє користувачам переключатися між різними кластерними менеджерами з мінімальними змінами в коді.

Під час подання Spark-програми для визначення кластерного менеджера використовується прапорець --master. Наприклад:

spark-submit --master mesos://<mesos-master>:<port> your_application.jar

Розуміння характеристик і можливостей різних кластерних менеджерів має важливе значення для оптимізації розгортання додатків Spark на основі конкретних вимог вашого сценарію використання та інфраструктури.

2.3Spark SQL, особливості обробка даних

Spark SQL - це модуль в Apache Spark, призначений для обробки структурованих даних. Він надає програмний інтерфейс для запитів до структурованих і напівструктурованих даних за допомогою SQL-запитів. Spark SQL дозволяє розробникам легко інтегрувати SQL-запити з програмами Spark, пропонуючи уніфікований і гнучкий підхід до обробки даних. Бфльш детально ощнайомитисть з архітекруторою можна на рис. 2.3.

Spark SQL надає уніфікований програмний інтерфейс для обробки структурованих даних, що дозволяє розробникам використовувати SQL запити разом з програмами Spark. Він сумісний з мовою запитів Hive, що дозволяє користувачам запускати існуючі запити Hive у Spark. Spark SQL використовує оптимізатор Catalyst для оптимізації планів запитів, покращуючи продуктивність SQL запитів. Spark SQL легко інтегрується з DataFrame, дозволяючи користувачам переключатися між SQL запитами та операціями API DataFrame.

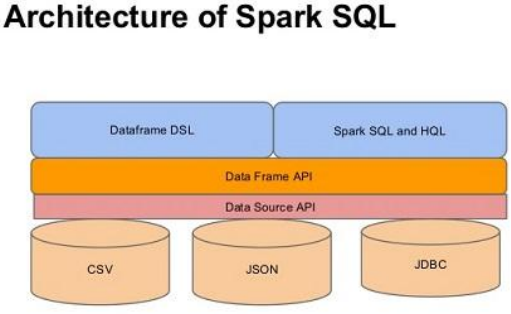


Рисунок 2.3 Архітектура Spark SQL

Щоб почати використовувати Spark SQL, створюється SparkSession за допомогою методу SparkSession.builder(). Розробники можуть встановити різні параметри конфігурації під час створення, такі як ім'я програми та додаткові властивості Spark. Метод config використовується для встановлення конкретних параметрів конфігурації для SparkSession, таких як URL-адреса головного сервера Spark, параметри пам'яті та інші.

Параметри конфігурації впливають на поведінку та продуктивність SQL-операцій Spark. Розробники можуть вказати додаткові конфігурації, такі як кількість розділів Після налаштування SparkSession можна виконувати SQL запити за допомогою методу spark.sql(). Користувачі можуть виконувати такі операції, як SELECT, JOIN і FILTER безпосередньо над структурованими даними. Spark SQL підтримує концепцію наборів даних, які представляють розподілені колекції даних. Набір даних може бути створений з існуючого RDD (Resilient Distributed Dataset - стійкий розподілений набір даних) або шляхом зчитування даних із зовнішніх джерел, таких як файли Parquet або JSON.

Набори даних можуть бути перетворені в DataFrame, що забезпечує безшовну інтеграцію між реляційною та функціональною парадигмами програмування. Розробники можуть налаштовувати формат серіалізації наборів даних, вибираючи між вбудованими кодувальниками або користувацькими кодувальниками для конкретних типів даних.

DataFrames і Datasets є основою при роботі зі Spark SQL та являє собою абстракції, побудовані на основі фреймворку, які забезпечують структуровану і розподілену колекцію даних. Вони пропонують більш високий рівень і більш зручний API у порівнянні з RDD (Resilient Distributed Datasets - стійкі розподілені набори даних).

DataFrame - це незмінний розподілений набір даних, організований в іменовані стовпці, який являє собою таблицю даних з рядками і стовпцями.

DataFrame може бути створений шляхом читання структурованих даних з різних джерел, таких як Parquet, JSON, CSV, або шляхом перетворення існуючого RDD.

Dataset - це розподілена колекція даних зі строгою типізацією, яка є розширенням DataFrames, пропонуючи переваги як RDD, так і DataFrames. Набори даних дозволяють розробникам працювати зі строго типізованими об'єктами, використовуючи прецедентні класи (Scala) або JavaBeans (Java). Це забезпечує безпеку типів під час компіляції та кращу інтеграцію з мовою програмування. Набори даних можуть бути створені шляхом перетворення існуючих DataFrame або шляхом читання даних із зовнішніх джерел. Набори даних забезпечують кращу продуктивність порівняно з DataFrame, особливо коли мова йде про операції, які включають складні перетворення або функції, визначені користувачем.

DataFrames і Datasets забезпечують абстракції вищого рівня для обробки структурованих даних у Spark. DataFrames підходять для SQL-подібних операцій, в той час як набори даних пропонують сильну типізацію для підвищення безпеки під час компіляції та оптимізації продуктивності. Обидві абстракції співіснують і можуть використовуватися взаємозамінно, залежно від конкретних вимог задач обробки даних.

Також Oracle дозволяє роботу із зведеними таблицями, ікі були вперше представлені для роботи із програмами Microsoft. Зведені таблиці у Spark - це потужний інструмент для перетворення та узагальнення даних. Вони дозволяють користувачам перетворювати рядки на стовпці, забезпечуючи чітке і стисле представлення агрегованих даних. Операція зведення у Spark зазвичай застосовується до фреймів даних і передбачає групування даних за одним або кількома стовпчиками, а потім обертання результату для створення структури, схожої на матрицю.

Обертання передбачає вибір стовпця, який стане новим стовпцем у результуючому фреймі даних, значень з іншого стовпця для заповнення клітинок і функції агрегування для визначення того, як заповнити клітинки, де

перетинаються кілька значень. Цей процес особливо корисний для створення зведених таблиць, перехресних таблиць або виконання спеціальних агрегацій даних.

Зведена операція є універсальною і може застосовуватися в широкому спектрі випадків використання, включаючи фінансову звітність, бізнес-аналітику та попередній аналіз даних. Вона дозволяє аналітикам і дослідникам даних ефективно узагальнювати і візуалізувати складні набори даних, отримуючи уявлення про закономірності і тенденції.У Spark операція зведення часто поєднується з іншими перетвореннями DataFrame і діями для досягнення конкретних цілей аналізу. Крім того, Spark надає такі функції, як pivot і groupBy, щоб полегшити виконання зведеного аналізу, а користувачі можуть налаштувати логіку агрегації відповідно до своїх конкретних вимог.

Spark SQL надає потужний та уніфікований інтерфейс для обробки структурованих даних, легко інтегруючи SQL запити з додатками Spark. Ініціалізація включає в себе створення SparkSession, налаштування параметрів і виконання SQL запитів. Крім того, Datasets пропонує гнучкий та ефективний спосіб представлення розподілених колекцій даних в рамках Spark SQL.

2.4 Spark Streaming, особливості обробки потоків

Spark Streaming - це масштабований і відмовостійкий движок обробки потоків, побудований на основі Apache Spark. Він розширює можливості ядра Spark для обробки та аналізу потокових даних у реальному часі. Spark Streaming дозволяє розробникам писати потокові додатки, використовуючи високорівневі API на Java, Scala або Python. Він працює, розділяючи вхідний потік даних на невеликі партії і обробляючи кожну партію за допомогою того ж самого механізму Spark, який забезпечує пакетну обробку. Такий підхід, відомий як мікроблок, забезпечує переваги відмовостійкості та простоти використання, одночасно дозволяючи обробляти дані в режимі, близькому до реального часу.

Spark Streaming може безпосередньо споживати дані з Apache Kafka, популярної розподіленої потокової платформи. Flume, HDFS, Amazon Kinesis та користувацьких приймачів.

Spark Streaming вводить поняття дискретизованих потоків (DStreams), які представляють собою послідовність даних RDD (Resilient Distributed Datasets - стійкі розподілені набори даних) у часі. D-потоки підтримують високорівневі операції, такі як відображення, зменшення, віконні операції та операції з урахуванням стану. Основною одиницею абстракції є RDD, а DStream - це, по суті, серія RDD, кожна з яких відповідає невеликому часовому інтервалу. Ці інтервали визначаються інтервалом пакетів, параметром, що налаштовується користувачем і вказує, як часто потік даних ділиться на пакети.

Операції DStream виконуються над кожною порцією даних незалежно, забезпечуючи паралельну та масштабовану обробку. Кожна партія обробляється рушієм Spark, який застосовує визначені користувачем операції та перетворення для отримання бажаного результату. Крім того, Spark Streaming забезпечує відмовостійкість, зберігаючи метадані та інформацію про походження оброблених партій, що дозволяє проводити переобчислення у разі збоїв у вузлах. Ця модель обробки мікропакетів у поєднанні з лінійною інформацією для забезпечення відмовостійкості формує основу роботи DStreams всередині Spark Streaming, забезпечуючи гнучкий і стійкий фреймворк для обробки даних у реальному часі.

До DStreams можна застосовувати стандартні перетворення Spark, такі як `map`, `filter` та `reduceByKey`. Ці операції виконуються над кожною партією даних, забезпечуючи масштабований і розпаралелений підхід до обробки потоків.

Дискретизовані потоки (DStreams) у Spark Streaming - це безперервний потік даних, розділений на невеликі, визначені користувачем часові інтервали.

DStreams - це, по суті, послідовність стійких розподілених наборів даних (Resilient Distributed Datasets, RDD), де кожен RDD містить дані з певного часового інтервалу. Стандартні операції Spark Streaming, такі як фільтрація та

зменшення за ключем, застосовуються до DStreams для виконання перетворень над базовими RDD. Нижче представлення того, як ці операції працюють:

Операція фільтр(filter):

Операція фільтрації у Spark Streaming застосовується до кожного RDD у DStream незалежно. Для цього використовується визначена користувачем функція, яка оцінює кожен елемент RDD і повертає булеве значення.

Наприклад, якщо у вас є DStream з пар ключ-значення, що представляють дані датчиків, ви можете використовувати фільтр для вибіркового збереження елементів на основі певної умови, наприклад, відфільтрувати значення нижче певного порогу.

```
val filteredStream = originalStream.filter(sensorData => sensorData._2 > threshold)
```

Операція фільтрації у Spark Streaming застосовується до кожного RDD у DStream незалежно. Для цього використовується визначена користувачем функція, яка оцінює кожен елемент RDD і повертає булеве значення.

Наприклад, якщо у вас є DStream з пар ключ-значення, що представляють дані датчиків, ви можете використовувати фільтр для вибіркового збереження елементів на основі певної умови, наприклад, відфільтрувати значення нижче певного порогу.

Операція ReduceByKey:

Операція reduceByKey - це перетворення, яке часто використовується у Spark Streaming для виконання агрегації пар ключ-значення в межах кожного RDD. Вона оперує парами елементів з однаковим ключем і застосовує визначену

користувачем асоціативну і комутативну функцію для зменшення значень, пов'язаних з кожним ключем.

Наприклад, якщо у вас є DStream з пар (слово, кількість), ви можете використати reduceByKey для підсумовування кількості для кожного слова у пакетах.

```
val wordCounts = pairsDStream.reduceByKey((count1, count2) => count1 + count2)
```

Віконні операції:

Spark Streaming підтримує віконні операції, які дозволяють застосовувати перетворення у ковзних часових вікнах. Ці операції, такі як reduceByKeyAndWindow або window, дають змогу аналізувати дані протягом певних часових інтервалів.

Наприклад, ви можете використовувати reduceByKeyAndWindow для обчислення агрегованих результатів за ковзним вікном останніх N партій.

```
val windowedWordCounts = pairsDStream.reduceByKeyAndWindow(
  (count1, count2) => count1 + count2,  // Reduce function
  (count1, count2) => count1 - count2,  // Inverse reduce function
  Seconds(windowSize),  // Window duration
  Seconds(batchInterval)  // Slide duration
)
```

Стандартні операції та перетворення дозволяють розробникам виражати складну логіку обробки потоків, використовуючи знайомі конструкції Spark. Незалежно від того, чи це фільтрація нерелевантних даних, агрегування значень за ключем або аналіз даних у часових вікнах, операції DStreams і Spark Streaming забезпечують потужну і гнучку основу для обробки даних у реальному часі.

Оброблені дані зі Spark Streaming можуть бути легко інтегровані з рушієм Spark Core, що дозволяє поєднувати обробку в реальному часі та пакетну обробку в одному додатку. Spark Streaming зазвичай використовується для додатків, що передбачають обробку даних у реальному часі, таких як системи моніторингу та

оповіщення, виявлення аномалій та аналітика в реальному часі. Інтеграція з Apache Kafka робить його особливо придатним для обробки високопродуктивних, відмовостійких потоків даних.

Загалом, Spark Streaming розширює можливості Apache Spark до області обробки даних в режимі реального часу, надаючи уніфіковану та універсальну платформу для побудови наскрізних рішень для обробки даних та аналітики.

3 ПОРІВНЯННЯ АЛГОРИТТМИВ ОБРОБКИ ПОТОКІВ ДАНИХ

3.1 Обробка потоків даних з Spark Streaming

Spark Streaming - це потужне розширення Apache Spark, яке дозволяє обробляти та аналізувати дані в режимі реального часу. Spark Streaming - це окрема бібліотека в Spark для обробки безперервних потокових даних. Вона надає нам DStream API, який працює на основі Spark RDD. DStreams надає нам дані, розділені на фрагменти у вигляді RDD, отримані від джерела потокового передавання, які потрібно обробити, і після обробки надсилає їх до місця призначення

Spark Streaming підтримує різні джерела вхідних даних, включаючи Kafka, Flume, HDFS та користувацькі приймачі. Дані надходять невеликими порціями через певні часові інтервали. Віконні операції дозволяють обробляти дані протягом ковзних часових інтервалів. Найпоширеніші віконні операції включають віконні підрахунки, суми та середні значення, які дають уявлення про тенденції в часі. Єдиний механізм виконання Spark та уніфікована модель програмування для пакетного та потокового доступу призводять до деяких унікальних переваг перед іншими традиційними потоковими системами.

На високому рівні сучасні розподілені конвеєри обробки потоків працюють наступним чином:

Отримують потокові дані з джерел даних (наприклад, журнали в реальному часі, дані системної телеметрії, дані пристроїв Інтернету речей тощо) в деяку систему збору даних, таку як Apache Kafka, Amazon Kinesis тощо.

Обробляйте дані паралельно на кластері. Це те, для чого призначені механізми потокової обробки, про що ми детально поговоримо далі.

Виводити результати в наступні системи, такі як HBase, Cassandra, Kafka тощо.

Традиційна потокова обробка передбачає обробку даних у режимі реального часу в міру їх надходження, часто з акцентом на низьку затримку і майже миттєвий аналіз подій. У контексті традиційних систем потокової обробки дані, як правило, надходять безперервно, а алгоритми працюють з ними поетапно, забезпечуючи миттєві результати. Ці системи часто покладаються на архітектуру, керовану подіями, і призначені для додатків, де своєчасна реакція на події має вирішальне значення, таких як фінансові транзакції, системи моніторингу або додатки Інтернету речей.Як працює традиційна система обробки потоків відображено на рис. 3.1.

Для обробки даних більшість традиційних систем потокової обробки розроблені з використанням моделі безперервного оператора, яка працює наступним чином:

Існує набір робочих вузлів, на кожному з яких працює один або декілька безперервних операторів.

Кожен безперервний оператор обробляє потокові дані по одному запису за раз і пересилає записи іншим операторам у конвеєрі.

Існують оператори-джерела, які отримують дані від систем поглинання, та оператори-поглиначі, які виводять дані до систем, що знаходяться нижче за течією.

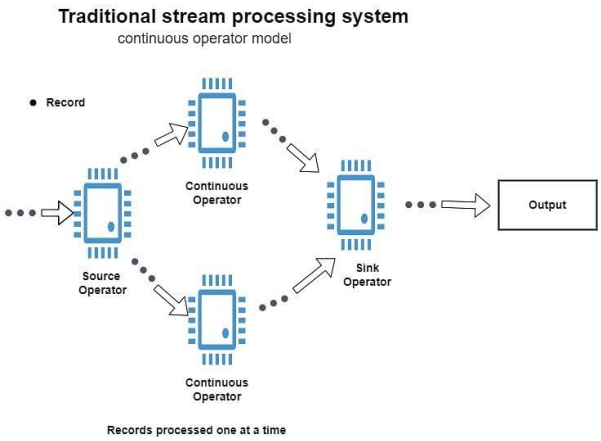


Рисунок 3.1 Традиційна система потокової обробка

Безперервні оператори - це проста і природна модель. Однак, зважаючи на сучасну тенденцію до більш масштабної та складної аналітики в реальному часі, ця традиційна архітектура також зіткнулася з деякими проблемами. Ми розробили Spark Streaming, щоб задовольнити наступні іимоги:

Швидке відновлення після збоїв і відставання - зі збільшенням масштабу зростає ймовірність того, що вузол кластера вийде з ладу або непередбачувано сповільниться (тобто відставатиме). Система повинна мати можливість автоматично відновлюватися після збоїв і відставання, щоб надавати результати в реальному часі. На жаль, статичний розподіл безперервних операторів між робочими вузлами ускладнює для традиційних систем швидке відновлення після збоїв і відстаючих.

Балансування навантаження - Нерівномірний розподіл навантаження між працівниками може спричинити вузькі місця в системі з безперервним оператором. Це частіше трапляється у великих кластерах і при робочому навантаженні, що динамічно змінюється. Система повинна мати можливість динамічно адаптувати розподіл ресурсів залежно від робочого навантаження

Об'єднання потокових, пакетних та інтерактивних робочих навантажень У багатьох випадках використання привабливим є інтерактивний запит до потокових даних (зрештою, потокова система зберігає їх у пам'яті) або поєднання їх зі

статичними наборами даних (наприклад, попередньо розрахованими моделями). Це важко зробити в системах з безперервними операторами, оскільки вони не призначені для динамічного введення нових операторів для спеціальних запитів. Для цього потрібен єдиний рушій, який може поєднувати пакетні, потокові та інтерактивні запити.

Розширена аналітика, така як машинне навчання та SQL-запити - більш складні робочі навантаження вимагають постійного навчання та оновлення моделей даних або навіть запиту "найсвіжішого" представлення потокових даних за допомогою SQL-запитів. Знову ж таки, наявність спільної абстракції для цих аналітичних завдань значно полегшує роботу розробника.

Замість того, щоб обробляти потокові дані по одному запису за раз, Spark Streaming дискретизує потокові дані на крихітні, субсекундні мікропакети. Іншими словами, приймачі Spark Streaming приймають дані паралельно і буферизують їх у пам'яті робочих вузлів Spark. Потім оптимізований до затримок движок Spark виконує короткі завдання (десятки мілісекунд) для обробки пакетів і виводить результати в інші системи. Зауважте, що на відміну від традиційної моделі безперервного оператора, де обчислення статично розподіляються між вузлами, у Spark завдання призначаються працівникам динамічно на основі локалізації даних і доступних ресурсів. Це забезпечує краще балансування навантаження та швидше відновлення після збоїв, як ми проілюструємо далі.

Крім того, кожен пакет даних є стійким розподіленим набором даних (Resilient Distributed Dataset, RDD), який є базовою абстракцією відмовостійкого набору даних у Spark. Це дозволяє обробляти потокові дані за допомогою будь-якого коду або бібліотеки Spark.

Розділення даних на малі мікро-партії дозволяє спростити API. Наприклад, розглянемо просте навантаження, в якому потік вхідних даних необхідно розділити за допомогою ключа і обробити. У традиційному підході під час запису, якщо один з розділів є більш обчислювально інтенсивним, ніж інші, вузол статично призначений для того, щоб цей розділ став вузьким місцем і сповільнив

би конвеєр. У Spark Streaming навантаження є збалансованим між екзекюторами деякі єкзекютори оброблятимуть декілька більш довгих завдань ніж інші та будуть обробляти більш короткі завдання.Балансування навантаження це нерівномірний розподіл навантаження на обробку між робочими може призвести до вузьких місць у безперервній системі оператора.

Spark Streaming підтримує операції з урахуванням стану, що дозволяє програмам підтримувати та оновлювати стан для всіх пакетів даних. Це корисно для сценаріїв, де обчислення повинні враховувати історичні дані.

3.2 Відмовостійкість обробки потоків даних з Spark Streaming

Швидке відновлення після збоїв і відставання є важливими аспектами відмовостійкості в Apache Spark, гарантуючи, що додатки Spark можуть ефективно обробляти збої вузлів і пом'якшувати вплив повільно працюючих або відстаючих завдань. Ці механізми сприяють загальній відмовостійкості та надійності додатків Spark у розподілених обчислювальних середовищах.

У разі відмови вузла традиційні системи змушені перезапускати неперервний оператор на іншому вузлі і відтворювати частину потоку даних, щоб переобчислити втрачену інформацію. Зауважте, що переобчислення виконує лише один вузол, і конвеєр не може продовжувати роботу, доки новий вузол не наздожене його після повторного відтворення. У Spark обчислення вже дискретизовано на невеликі, детерміновані завдання, які можуть виконуватися будь-де без шкоди для коректності. Таким чином, завдання, що вийшли з ладу, можуть бути перезапущені паралельно на всіх інших вузлах кластера, таким чином рівномірно розподіляючи всі переобчислення між багатьма вузлами, і відновлюючись після збою швидше, ніж при традиційному підході, що відображено на рис. 3.2.

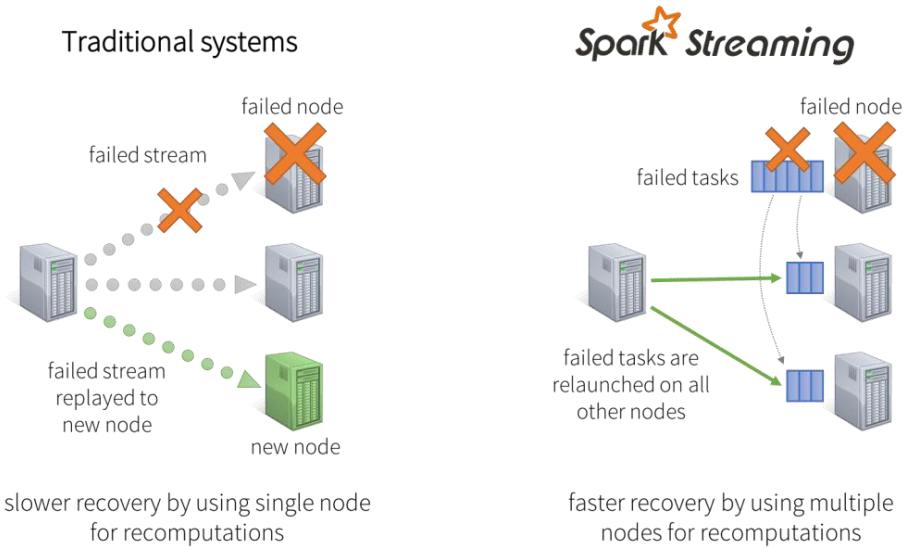


Рисунок 3.2 Швидше відновлення після збоїв завдяки перерозподілу обчислень

Швидка відмова у Spark означає швидке виявлення та реагування на відмову вузлів у кластері Spark. Коли робочий вузол виходить з ладу через апаратні проблеми або з інших причин, Spark швидко ідентифікує збій за допомогою механізмів серцебиття. Після ідентифікації Spark перерозподіляє втрачені завдання на інші доступні вузли в кластері, гарантуючи, що додаток продовжить обробку даних без значних перебоїв. Швидка відмова має важливе значення для підтримки високої доступності та безперервної роботи додатків Spark, особливо у великомасштабних розподілених середовищах, де збої вузлів неминучі.

Відновлення відстаючих вирішує проблему повільного виконання або відставання завдань у завданні Spark. У розподіленому обчислювальному середовищі окремі завдання можуть виконувати довше, ніж інші, що потенційно затримує загальний хід виконання завдання. Щоб пом'якшити цю проблему, Spark використовує спекулятивне виконання, механізм, який визначає повільні завдання і повторно виконує додаткові спекулятивні копії цих завдань на інших вузлах. Система відстежує хід виконання цих спекулятивних завдань і вважає результатом те, яке завершиться першим. Такий підхід допомагає подолати вплив відстаючих, гарантуючи, що загальна робота буде виконана вчасно.

Контрольні точки та механізми відновлення в Apache Spark є невід'ємними компонентами, які сприяють відмовостійкості та надійності розподілених додатків обробки даних. Ці функції гарантують, що додатки Spark можуть відновлюватися після збоїв, підтримувати узгоджений стан і продовжувати обробку даних безперебійно. Стан програми Spark періодично зберігається у надійній розподіленій файловій системі, такій як HDFS (Hadoop Distributed File System - розподілена файлова система Hadoop). Ці дані з контрольними точками включають інформацію про родовід та метадані, необхідні для реконструкції графіка родоводу RDD (Resilient Distributed Dataset - стійкий розподілений набір даних).

Основна мета контрольних точок - скоротити графік родоводу і створити більш стійку основу. Це зменшує обчислювальну складність і підвищує відмовостійкість, оскільки переобчислення з контрольної точки вимагає менших обчислень, ніж реконструкція всього графа.

Хоча контрольні точки підвищують відмовостійкість, вони збільшують витрати на зберігання, оскільки дані контрольних точок потрібно зберігати в розподіленій файловій системі. Користувачі повинні ретельно зважити вимоги до зберігання і вибрати відповідні місця для зберігання контрольних точок.

Швидка реакція на збої та відновлення відстаючих завдань разом підвищують відмовостійкість та продуктивність додатків Spark. Швидко реагуючи на збої вузлів і стратегічно повторно виконуючи завдання, що затримуються, Spark мінімізує вплив непередбачуваних подій на загальне виконання завдання. Однак важливо дотримуватися балансу, оскільки надмірне спекулятивне виконання може призвести до зайвого споживання ресурсів. Правильне налаштування параметрів, пов'язаних зі спекулятивним виконанням, таких як кількість спекулятивних копій та пороги завершення завдання, є важливим для оптимізації продуктивності та використання ресурсів додатків Spark.

Apache Spark слугує важливим механізмом підвищення відмовостійкості та відновлення у розподілених додатках обробки даних. Періодично зберігаючи стан

програми, Spark забезпечує стійкість до збоїв вузлів і полегшує ефективне відновлення, сприяючи надійності Spark-додатків у великомасштабних розподілених середовищах.

3.3 Уніфікація пакетної, потокової та інтерактивної аналітики з Spark Streaming

Об'єднання пакетної, потокової та інтерактивної аналітики в Apache Spark Streaming являє собою потужну парадигму, яка дозволяє розробникам легко інтегрувати та перемикатися між різними режимами обробки даних в рамках одного середовища. Spark Streaming розширює ядро Spark, щоб забезпечити обробку потоків у реальному часі, зберігаючи при цьому сумісність з можливостями пакетної обробки та інтерактивної аналітики Spark.

Однією з ключових особливостей Spark Streaming є його уніфікований API, який дозволяє розробникам використовувати однакові високорівневі конструкції як для пакетної, так і для потокової обробки. Така уніфікація спрощує процес розробки, оскільки розробники можуть використовувати наявні знання про API Spark для пакетної обробки для потокових додатків. Наприклад, перетворення map та reduceByKey, що використовуються у пакетній обробці, можна легко застосувати до DStreams (дискретизованих потоків) у Spark Streaming.

Приклад пакетної обробки в Spark Streaming:

```
SparkConf sparkConf = new SparkConf().setAppName("BatchProcessingExample");
JavaStreamingContext streamingContext = new JavaStreamingContext(sparkConf, Durations.seconds(1));

JavaDStream<String> inputDStream = streamingContext.textFileStream("/path/to/batch/data");
```

```
JavaPairDStream<String, Integer> wordCounts = inputDStream
    .flatMap(line -> Arrays.asList(line.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((count1, count2) -> count1 + count2);

wordCounts.print();
streamingContext.start();
streamingContext.awaitTermination();
```

Spark Streaming відмінно справляється з обробкою потоків даних у реальному часі. DStreams представляють собою серію RDD (Resilient Distributed Datasets - стійкі розподілені набори даних) через певні проміжки часу. Розробники можуть застосовувати перетворення і дії до DStreams, що полегшує перехід від пакетної до потокової аналітики. Ті ж самі операції Spark, що використовуються в пакетній обробці, можуть бути використані для обробки даних в реальному часі з невеликими змінами.

Приклад звичайного підрахунку слів у Spark Streaming:

```
JavaDStream<String> inputDStream = streamingContext.socketTextStream("localhost",
9999);
JavaPairDStream<String, Integer> wordCounts = inputDStream
    .flatMap(line -> Arrays.asList(line.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((count1, count2) -> count1 + count2);
```

Інтероперабельність Spark поширюється на багаті бібліотеки, такі як MLlib (машинне навчання), SQL, DataFrames та GraphX. Легко інтегрується з Spark SQL,

дозволяючи розробникам виконувати інтерактивну аналітику як пакетних, так і потокових даних. Розробники можуть використовувати DataFrame API для вираження SQL-подібних запитів до структурованих даних, забезпечуючи уніфікований підхід до аналізу даних.

RDD, згенеровані DStreams, можуть бути перетворені у DataFrames (програмний інтерфейс до Spark SQL) і запитані за допомогою SQL. Наприклад, використовуючи JDBC сервер Spark SQL, ви можете показати стан потоку будь-якому зовнішньому додатку, який розмовляє на мові SQL.

На практиці здатність передавати пакетні дані і використовувати механізм Spark призводить до порівнянної або навіть вищої пропускної здатності порівняно з іншими потоковими системами. З точки зору затримок, Spark Streaming може досягати затримок у кілька сотень мілісекунд. Розробники іноді запитують, чи не збільшує затримки мікропакетування за своєю суттю затримки. На практиці, затримка при пакетній обробці є лише невеликою складовою наскрізної затримки конвеєра.

Наприклад, багато програм обчислюють результати у ковзаючому вікні, і навіть у системах з безперервним оператором це вікно оновлюється лише періодично (наприклад, 20-секундне вікно, яке ковзає кожні 2 секунди). Багато конвеєрів збирають записи з декількох джерел і чекають короткий проміжок часу, щоб обробити запізнілі або неправильні дані. Нарешті, будь-який алгоритм автоматичного запуску має тенденцію чекати деякий час, щоб запустити тригер. Тому, порівняно з наскрізною затримкою, пакетна обробка рідко додає значних накладних витрат. Насправді, збільшення пропускної здатності за рахунок DStreams часто означає, що вам потрібно менше машин для обробки того ж робочого навантаження.

Структурований потік - це масштабований і відмовостійкий механізм обробки потоків, побудований на основі Spark SQL. Ви можете виразити свої потокові обчислення так само, як ви б виразили пакетні обчислення над статичними даними. Механізм Spark SQL подбає про його інкрементне і

безперервне виконання та оновлення кінцевого результату по мірі надходження потокових даних. Ви можете використовувати Dataset/DataFrame API в Scala, Java, Python або R для вираження потокових агрегацій, вікон часу подій, об'єднання потоку в пакет тощо.

Обчислення виконуються на тому ж оптимізованому механізмі Spark SQL. Нарешті, система забезпечує наскрізну гарантію відмовостійкості з точністю до одного разу завдяки контрольним точкам і журналам Write-Ahead Logs. Structured Streaming забезпечує швидку, масштабовану, відмовостійку, наскрізну одномоментну обробку потоків без необхідності для користувача міркувати про потокову передачу даних. Об'єднання потокових, пакетних та інтерактивних робочих навантажень у багатьох випадках використання привабливим є інтерактивний запит до потокових даних (зрештою, потокова система зберігає їх у пам'яті) або поєднання їх зі статичними наборами даних (наприклад, попередньо розрахованими моделями). Це важко зробити в системах з безперервними операторами, оскільки вони не призначені для динамічного введення нових операторів для спеціальних запитів. Для цього потрібен єдиний рушій, який може поєднувати

Apache Spark 2.0 додає першу версію нового високорівневого API, Structured Streaming, для створення безперервних додатків. Основна мета - полегшити створення наскрізних потокових додатків, які інтегруються зі сховищами, обслуговуючими системами і пакетними завданнями узгоджено і відмовостійко. У цій статті ми пояснюємо, чому це важко зробити за допомогою сучасних розподілених потокових механізмів, і представляємо Структурований потоковий зв'язок. пакетні, потокові та інтерактивні запити.

За замовчуванням, внутрішні запити структурованих потоків обробляються за допомогою механізму мікроклітинної обробки, який обробляє потоки даних як серію невеликих пакетних завдань, таким чином досягаючи наскрізних затримок до 100 мілісекунд і гарантії відмовостійкості принаймні з першого разу. Однак, починаючи з Spark 2.3, був запроваджений новий режим обробки з низькою

затримкою, який називається Безперервна обробка, що дозволяє досягти наскрізних затримок до 1 мілісекунди з гарантією відмов принаймні з першого разу. Не змінюючи операції Dataset/DataFrame у ваших запитах, ви зможете вибрати режим відповідно до вимог вашої програми.

У Structured Streaming ми вирішуємо проблему семантики роблячи сильну гарантію щодо системи: в будь-який момент часу вихід програми еквівалентний виконанню пакетного завдання над префіксом даних. Наприклад, у нашому додатку для моніторингу таблиця результатів у MySQL завжди буде еквівалентна взяттю префікса потоку оновлень кожного телефону (незалежно від того, які дані потрапили до системи до цього моменту) і запуску SQL-запиту, який ми показали вище. Ніколи не буде "відкритих" подій, які підраховуються швидше, ніж "закриті", дублікатів оновлень при збоях і т.д. Структурований потік автоматично забезпечує узгодженість і надійність як всередині движка, так і при взаємодії з зовнішніми системами (наприклад, при транзакційному оновленні MySQL).

Ця гарантія цілісності префікса дозволяє робити висновки про три проблеми:

1.Вихідні таблиці завжди узгоджуються з усіма записами у префіксі даних. Наприклад, поки кожен телефон завантажує свої дані у вигляді послідовного потоку (наприклад, в один і той же розділ в Apache Kafka), ми завжди будемо обробляти і рахувати його події по порядку.

2.Відмовостійкість обробляється цілісно за допомогою структурованих потоків, в тому числі при взаємодії з вихідними поглиначами. Це було основною метою при підтримці безперервних додатків.

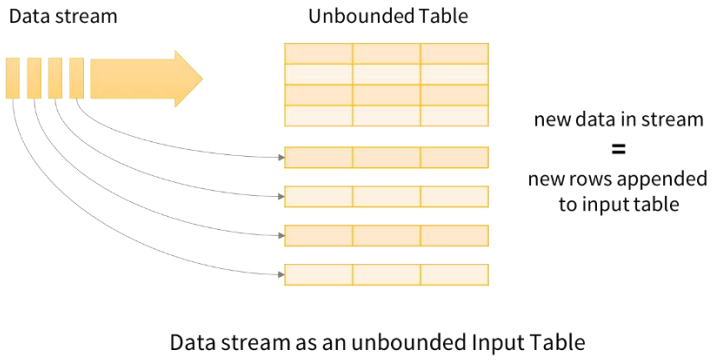
3. Вплив невпорядкованих даних очевидний. Ми знаємо, що дані про результати роботи згруповані за діями і часом для префікса потоку. Якщо пізніше ми отримаємо більше даних, ми можемо побачити поле часу за годину в минулому, і ми просто оновимо відповідний рядок в MySQL. Структурований потік також підтримує API для фільтрації надто старих даних, якщо користувач цього хоче. Але, по суті, невпорядковані дані не є "особливим випадком": в запиті сказано

згрупувати за полем часу, і побачити старий час нічим не відрізняється від побачити повторювану дію.

Перевага структурованого потокового передавання полягає в тому, що API дуже простий у використанні: це просто API DataFrame і Dataset API Spark. Користувачі просто описують запит, який вони хочуть виконати, місця вводу і виводу, і, за бажанням, ще кілька деталей. Потім система виконує запит інкрементно, підтримуючи достатній стан для відновлення після збою, збереження результатів у зовнішньому сховищі тощо.

3.4 Структурований потік та віконні операції

Структурований потік розглядає всі дані, що надходять, як необмежену вхідну таблицю. Кожен новий елемент у потоці - це як рядок, що додається до вхідної таблиці. Насправді ми не будемо зберігати всі вхідні дані, але наші результати будуть еквівалентні тому, якби ми мали всі дані і виконували пакетне завдання, що відобрадено на рис. 3.4.



Рисинок 3.4 Потік данних та його обробка

Потім розробник визначає запит до цієї вхідної таблиці, як до статичної таблиці, щоб обчислити кінцеву таблицю результатів, яка буде записана до вихідного сховища. Spark автоматично перетворює цей пакетний запит на

потоковий план виконання. Це називається інкременталізацією: Spark з'ясовує, який стан потрібно підтримувати, щоб оновлювати результат кожного разу, коли надходить новий запис. Нарешті, розробники визначають тригери, щоб контролювати, коли оновлювати результати. Кожного разу, коли спрацьовує тригер, Spark перевіряє наявність нових даних (новий рядок у вхідній таблиці) та інкрементно оновлює результат.

Остання частина моделі - це режими виводу. Кожного разу, коли таблиця результатів оновлюється, розробник хоче записати зміни до зовнішньої системи, наприклад, S3, HDFS або бази даних. Зазвичай ми хочемо писати вивід інкрементно.

Для цього Структурований Потік надає три режими виводу:

Додавання : До зовнішнього сховища будуть записані лише нові рядки, додані до таблиці результатів з моменту останнього тригера. Застосовується лише для запитів, де існуючі рядки в таблиці результатів не можуть змінюватися (наприклад, карта на вхідному потоці).

Завершення: Вся оновлена таблиця результатів буде записана до зовнішнього сховища.

Оновлення: У зовнішньому сховищі будуть змінені лише ті рядки, які були оновлені в таблиці результатів з моменту останнього тригера. Цей режим працює для поглиначів виводу, які можна оновлювати на місці, наприклад, таблиця MySQL.

Наш пакетний запит має на меті підрахувати кількість дій, згрупованих за (дія, година). Щоб виконати цей запит інкрементно, Spark підтримуватиме певний стан з підрахунками для кожної пари, і оновлюватиме його, коли з'являтимуться

нові записи. Для кожного зміненого запису буде виведено дані відповідно до режиму виведення. На рис. 3.5. показано виконання цього запиту у режимі оновлення.

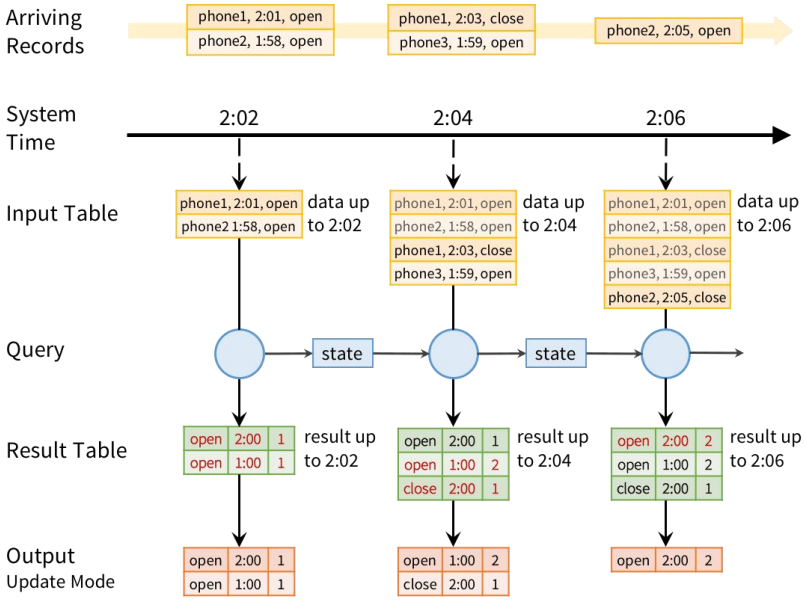


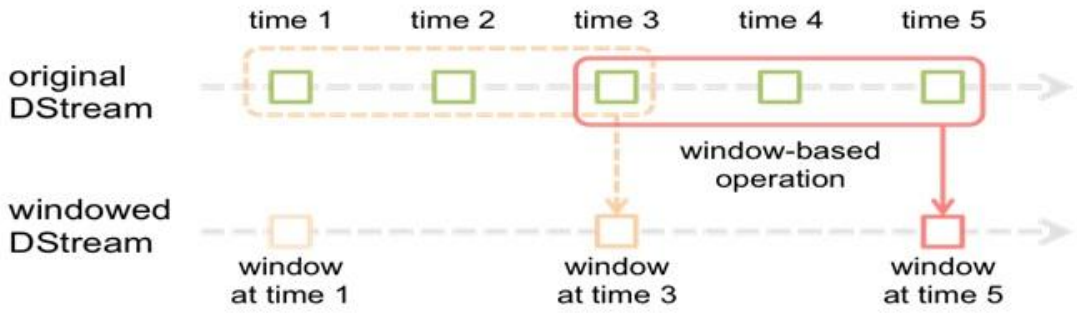
Рисунок 3.5 Режим оновлення данних

У кожній точці тригера ми беремо попередні згруповані підрахунки і оновлюємо їх новими даними, які надійшли з моменту останнього тригера, щоб отримати нову таблицю результатів. Потім ми відправляємо до поглинача лише ті зміни, які вимагаються нашим режимом виводу - тут ми оновлюємо записи для пар (дія, година), які змінилися під час цього тригера в MySQL (показано червоним кольором).

Зверніть увагу, що система також автоматично обробляє запізнілі дані. На рисунку вище подія "відкрити" для phone3, яка сталася о 1:58 на телефоні, потрапляє до системи лише о 2:02. Тим не менш, навіть якщо вже минуло 2:00, ми оновлюємо запис на 1:00 в MySQL. Однак, гарантія цілісності префіксів у структурованому потоці гарантує, що ми обробляємо записи з кожного джерела в порядку їх надходження. Наприклад, оскільки подія "закриття" phone1 надходить після події "відкриття", ми завжди будемо оновлювати лічильник "відкриттів" до того, як оновимо лічильник "закриття".

Потоки в структурованому потоковому передаванні представляються як DataFrame або Dataset з властивістю isStreaming, встановленою в true. Ви можете створювати їх за допомогою спеціальних методів читання з різних джерел. Ви можете використовувати звичайні операції DataFrame/Dataset для перетворення даних. Це дозволяє розробникам тестувати свою бізнес-логіку на статичних наборах даних і без проблем застосовувати її до потокових даних без зміни логіки.

Віконні операції в Spark Streaming дозволяють розробникам виконувати обчислення над ковзаючими вікнами даних, що уможливлює агрегацію на основі часу. Spark Streaming підтримує операції з урахуванням стану, що дозволяє додаткам підтримувати та оновлювати стан для пакетів даних.



Рис

инок 3.6 Обробка віконных пакети в Structure Streaming

Коли вікно ковзає над вихідним DStream, вихідні RDD, які потрапляють у вікно, об'єднуються. Також виконується операція, яка виробляє іскрові RDD з DStream, що знаходиться у вікні. Отже, у цьому конкретному випадку операція застосовується над останніми 3 часовими одиницями даних, а також ковзає на 2 часові одиниці.

В основному, будь-яка віконна операція Spark вимагає вказівки двох параметрів.

Довжина вікна - визначає тривалість вікна (3 на рисунку).

Інтервал ковзання - визначає інтервал, з яким виконується операція з вікном (2 на рисунку).

3.5 Порівняння API Spark та Flink

Flink розроблений з архітектурою, орієнтованою на потік, і пропонує справжню обробку потоків, керовану подіями. Він природно підтримує безперервну обробку потоків даних з низькою затримкою. Основною абстракцією Flink є DataStream API, який дозволяє розробникам працювати безпосередньо з потоковими даними, що робить його добре придатним для додатків у режимі реального часу. Незважаючи на те, що Spark Streaming надає можливості потокової обробки, його основна модель мікропакетної обробки вносить невелику затримку в обробку. Spark насамперед відомий своїми можливостями пакетної обробки, а потоковий компонент є розширенням моделі пакетної обробки.

API Flink робить сильний акцент на обробці часу подій. Він надає вбудовану підтримку для обробки подій з порушенням порядку і запізненням, що робить його придатним для додатків, які вимагають точної послідовності подій. Хоча Spark Streaming підтримує обробку часу подій, він може не так ефективно обробляти події з порушенням порядку та запізненнями, як Flink.

У роботі Flink робить сильний акцент на обробці часу подій. Він надає вбудовану підтримку для обробки подій з порушенням порядку і запізненням, що робить його придатним для додатків, які вимагають точної послідовності подій. Він пропонує розширені можливості управління станами, що дозволяє гнучко обробляти обчислення з урахуванням станів. Він підтримує такі функції, як вікна часу подій, вікна сеансів і зіставлення шаблонів, що робить його придатним для складних потокових сценаріїв. Flink надає стислий і виразний API для обробки потоків.

Програми у Flink за своєю суттю є паралельними та розподіленими. Під час виконання потік має один або декілька розділів потоку, а кожен оператор має одну

або декілька підзадач оператора. Підзадачі оператора не залежать одна від одної і виконуються в різних потоках і, можливо, на різних машинах або контейнерах.

Кількість підзадач оператора - це паралелізм цього конкретного оператора. Різні оператори однієї програми можуть мати різний рівень паралелізму, що відображено на рис. 3.8

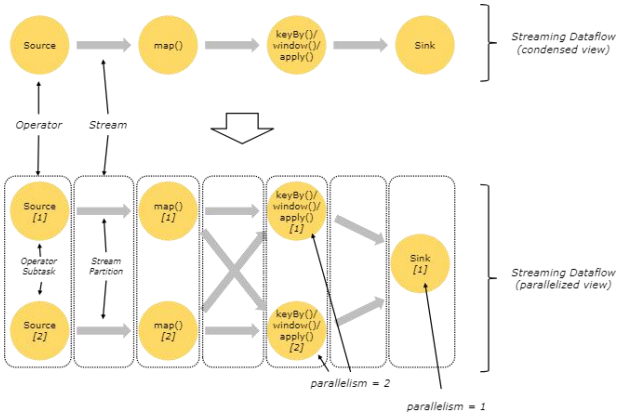


Рисунок 3.7 Виконання програми Flink

DataStream API дозволяє розробникам легко визначати складні потокові робочі процеси. Хоча він може мати крутішу криву навчання, він пропонує розширені можливості та гнучкість для складних завдань обробки потоків. Справжня модель обробки потоків DataStreaming підтримує меншу затримку порівняно зі Spark Streaming, що робить його придатним для додатків із суворими вимогами до низької затримки. Мікропакетування Spark Streaming вносить певну затримку через фіксовані інтервали обробки.

Flink API включає в себе надійний механізм відмовостійкості, пропонуючи семантику обробки "точно за один раз" і "принаймні за один раз". Контрольні точки та реплікація станів сприяють стійкості Flink до збоїв. Flink розроблений як автономний фреймворк для обробки потоків з роз'ємами для різних джерел і стоків даних. У той час Spark має добре розвинену екосистему, що включає бібліотеки для машинного навчання (MLlib), обробки графіків (GraphX) та аналітики на основі SQL (Spark SQL). Інтеграція Spark Streaming з ширшою екосистемою Spark забезпечує комплексне рішення.

Flink ідеально підходить для додатків з низькою затримкою, керованих подіями, де точна обробка часу подій має вирішальне значення. Його зазвичай обирають для аналітики в режимі реального часу, виявлення шахрайства та систем моніторингу, де важлива своєчасна і точна обробка даних. Spark Streaming добре підходить для випадків використання, які можуть терпіти дещо більшу затримку обробки, таких як аналіз журналів, ETL (Extract, Transform, Load) та сценарії пакетної потокової передачі даних. Ширша екосистема Spark робить його універсальним для різних завдань обробки даних.

3.6 Інтеграція з Apache Kafka

Apache Kafka швидко стає однією з найпопулярніших платформ для прийому потокового мовлення з відкритим кодом. Ми бачимо таку ж тенденцію і серед користувачів Spark Streaming. Тому в Apache Spark 1.3 ми зосередилися на значному покращенні інтеграції Kafka зі Spark Streaming. Це призвело до наступних доповнень:

Новий прямий API для Kafka - це дозволяє обробляти кожен запис Kafka рівно один раз, незважаючи на збої, без використання Write Ahead Logs. Це робить конвеєри Spark Streaming + Kafka більш ефективними, забезпечуючи при цьому сильніші гарантії відмовостійкості.

Python API для Kafka - щоб ви могли почати обробку даних Kafka виключно з Python.

Kafka - це потенційна платформа для обміну повідомленнями та інтеграції для Spark Streaming. Вона діє як центральний вузол для потоків даних у реальному часі, які обробляються в Spark Streaming за допомогою складних алгоритмів. Після обробки даних Spark Streaming або публікує результати в іншій темі Kafka, або зберігає їх у HDFS, базах даних чи дашбордах.

Kafka є найсучасніша система обміну повідомленнями. Вона також добре масштабується, тому користується популярністю в індустрії. Вона працює за

моделлю "видавець-передплатник" або "pub-sub". У цій моделі видавець може робити свій внесок у тему, а кілька підписників можуть отримати доступ до цієї теми. Тема може розглядатися як еквівалент таблиці в системі баз даних. Це добре інтегрується зі Spark Streaming для подальшої обробки потоків даних.

Spark Streaming підтримує Kafka з моменту його створення, і Spark Streaming використовується разом з Kafka у виробництві в багатьох місцях . Однак спільнота Spark з часом вимагала кращих гарантій відмовостійкості та сильнішої семантики надійності. Щоб задовольнити цю вимогу, у Spark 1.2 було запроваджено запис журналів наперед (Write Ahead Logs, WAL). Він гарантує, що жодні дані, отримані з будь-яких надійних джерел даних (наприклад, транзакційних джерел, таких як Flume, Kafka та Kinesis), не будуть втрачені через збої (тобто, принаймні, семантику). Навіть для ненадійних (тобто нетранзакційних) джерел, таких як звичайні старі сокети, це мінімізує втрату даних.

Для джерел, які дозволяють відтворювати потоки даних з довільних позицій у потоці (наприклад, Kafka), ми можемо досягти ще сильнішої семантики відмовостійкості, оскільки ці джерела дозволяють Spark Streaming мати більше контролю над споживанням потоку даних. У Spark 1.3 введено поняття прямого API, за допомогою якого можна досягти семантики "точно один раз" навіть без використання Write Ahead Logs. Давайте розглянемо деталі прямого API Spark для Apache Kafka.

Хід виконання інтеграції виглядає наступним чином:

1. Налаштуйте Кафку
2. Створіть тему
3. Опублікуйте в темі Кафки
4. Налаштуйте завдання Spark для читання з теми Кафки

Схема інтеграції Kafka представлена на рис. 3.8.

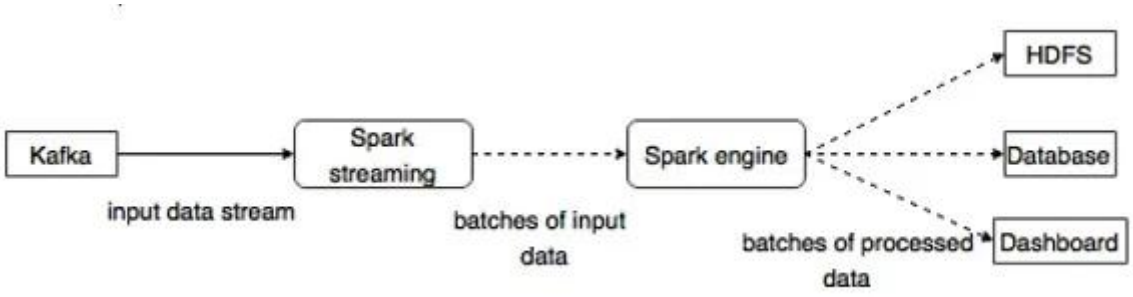


Рисунок 3.9 Процес інтеграції Kafka

На високому рівні попередня інтеграція Kafka працювала з журналами попереднього запису (Write Ahead Logs, WAL) наступним чином:

1. Дані з Kafka безперервно надходили до приймачів Kafka, запущених у працівниках/виконавцях Spark. Для цього використовувався високорівневий споживчий API Kafka.

2. Отримані дані зберігаються в пам'яті робочого/виконавця Spark, а також у сховищі WAL (реплікованому на HDFS). Отримувач Kafka оновлює зміщення Kafka до Zookeeper тільки після того, як дані були збережені до журналу.

3. Інформація про отримані дані та їхнє розташування у WAL також надійно зберігається. У разі збою ця інформація використовується для повторного зчитування даних і продовження обробки.

Хоча такий підхід гарантує, що жодні дані з Kafka не будуть втрачені, все ще існує невелика ймовірність того, що деякі записи можуть бути оброблені більше одного разу через збої (тобто принаймні один раз семантика). Це може статися, коли деякі отримані дані надійно зберігаються у WAL, але система дає збій перед оновленням відповідних зсувів у Kafka в Zookeeper. Це призводить до неузгодженості - Spark Streaming вважає, що дані було отримано, але Kafka вважає, що дані не було успішно надіслано, оскільки зміщення у Zookeeper не було оновлено. Отже, Kafka надішле дані знову після того, як система відновиться після збою.

Ця неузгодженість виникає через те, що обидві системи не можуть атомарно оновлювати інформацію, яка описує те, що вже було надіслано. Щоб уникнути цього, лише одна система повинна підтримувати узгоджене уявлення про те, що було надіслано або отримано. Крім того, ця система повинна мати повний контроль над відтворенням потоку даних під час відновлення після збоїв. Тому ми вирішили зберігати всю спожиту інформацію про зсуви лише у Spark Streaming, який може використовувати Simple Consumer API Kafka для відтворення даних з довільних зсувів, якщо це буде потрібно через збої.

Для його створення (основним розробником був Cody) новий API Direct Kafka використовує зовсім інший підхід, ніж Receivers і WAL. Замість того, щоб безперервно отримувати дані за допомогою Receivers і зберігати їх у WAL, ми просто вирішуємо на початку кожного пакетного інтервалу, який діапазон зсувів використовувати. Пізніше, коли завдання кожного пакету виконано, дані, що відповідають діапазонам зсувів, зчитуються з Kafka для обробки (подібно до того, як зчитуються HDFS-файли). Ці зміщення також надійно зберігаються (з контрольними точками) і використовуються для перерахунку даних для відновлення після збоїв.

Треба зауважити , що Spark Streaming може перечитувати і переробляти сегменти потоку з Кафки, щоб відновитися після збоїв. Однак, завдяки однократному характеру перетворень RDD, остаточні переобчислені результати є точно такими ж, як і без збоїв.

Таким чином, цей прямий API усуває необхідність як у WAL, так і в приймачах для Kafka, гарантуючи, що кожен запис Kafka буде ефективно отриманий Spark Streaming рівно один раз. Це дозволяє побудувати конвеєр Spark Streaming + Kafka з наскрізною однократною семантикою (якщо ваші оновлення до наступних систем є ідемпотентними або транзакційними). Загалом, це робить такі конвеєри потокової обробки більш відмовостійкими, ефективними та простими у використанні.

ВИСНОВКИ

У ході виконання роботи будо досліджено методи обробки данних за допомогою API Apache Spark, проблему обробки у режимі реального часу та паралельної обробки данних.

У процесі дозлідження в першому розділи було виявлено що в епоху великих даних вирішення глобальних проблем вимагає потужних інструментів, і Apache Spark стає ключовим гравцем у цьому середовищі. Завдяки своєму універсальному API, Spark полегшує обробку величезних наборів даних, забезпечуючи масштабоване вирішення складних проблем.

У другому розділі ми більш детально розглянули методи обробки данних, ознайомилися з архітектурою та основними компонентами. Дослідити роботу низькорівневих та високорівневих примітиви платформи. Проаналізувати як масштабованість та можливості розподілених обчислень роблять Apache Spark робить його надійним рішенням для роботи з великими обсягами даних. Ознайомитися з методами інтеграції латфоми до хмарних середовищ.

У результаті роботи вдалося досягти поставлених задач та порівняти алгоритми обробки потоків близьких до режиму реального часу з урахуванням ефективного використання низькорівневих примітивів обробки великих даних.

У третьому розділі проводився аналіз та порівняння Spark Streaming та Apache Flink. Ознайомилися з широкою екосистема Spark та інтеграція з Kafka, що виділяють його в конкретних випадках використання. Були розглянуті алгоритми обробки даних для обробки повідомлень, що прийшли із запізненням, алгоритми відновлення обробки потоку даних після переривання, методи покращення швидкості обробки потоку, зменшення затримок.

Використання Spark та його аналогів - це не просто вибір, це стратегічний імператив для тих, хто прагне використати потенціал великих даних у сучасному

динамічному середовищі. Постійна актуальність Spark підкреслює його значення як інструменту для інновацій та вирішення проблем у світі аналітики даних, що постійно розвивається.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1.Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia

."Learning Spark: Lightning-Fast Big Data Analysis". Link: Learning Spark on O'Reilly. 2015p.

2. Matei Zaharia, Reynold Xin, Patrick Wendell, Tathagata Das, and Sean Owen. "Mastering Apache Spark" .Mastering Apache Spark on Jupyter Notebooks 2015-2021p.

3. Bill Chambers and Matei Zaharia. "Spark: The Definitive Guide". Spark: The Definitive Guide on Amazon. 2018p

4. Holden Karau, Boris Lublinsky, et al."High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark". High Performance Spark on O'Reilly. 2017p

5. Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills."Advanced Analytics with Spark: Patterns for Learning from Data at Scale". Advanced Analytics with Spark on O'Reilly. 2015p

6. Muhammad Asif Abbasi. "Big Data Processing with Apache Spark". Big Data Processing with Apache Spark on Springer. 2017p

7. Holden Karau and Boris Lublinsky."Learning Spark SQL". Learning Spark SQL on O'Reilly. 2019p

8.Ali Ghodsi, Matei Zaharia, et al."Big Data Analysis with Scala and Spark"

Big Data Analysis with Scala and Spark on Coursera. Online Course (Coursera)

9. Holden Karau and Rachel Warren. "High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark". Spark on O'Reilly. 2017

10. Michael Armbrust, Reynold Xin, et al.Big Data with Apache Spark and Python".Taming Big Data with Apache Spark and Python on Udemy. Online Course (Udemy)