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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження ефективності технології Apache Impala для обробки великих даних»

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

_____ Єгор Ільїн
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Єгор Ільїн
група КНДМ-63

Керівник: Микола Гніденко
науковий ступінь, вчене звання к.т.н., доцент

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

Ільїну Єгору Денисовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження ефективності технології Apache Impala для обробки великих даних

керівник кваліфікаційної роботи Микола Гніденко к.т.н., доцент,
(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: науково-технічна література, екземпляр бази даних, вимоги до обробки великих даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження методів та засобів побудови архітектури
Аналіз технологій для обробки великих даних
Розробка вимог до забезпечення відвомостійкості системи обробки великих даних

5. Перелік графічного матеріалу: *презентація*
1. Мета роботи, предмет та об’єкт дослідження.
2. Архітектура баз даних
3. СКБД
4. Основні властивості Big Data
5. Додаткові властивості Big Data
6. Різноманітність даних в області Big Data
7. Складнощі при використанні Big Data
8. Apache Spark
9. Apache Hive та Kibernetus
10. Apache Impala
11. Архітектура Apache Impala
12. Базова архітектура відмовостійкої системи
13. Архітектурна схема відмовостійкої системи обробки Big Data
14 Порівняння результатів
15. Висновки

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	вик.
2	Вивчення матеріалів для аналіху технологій обробки великих даних	05.11-12.11.23	вик.
3	Дослідження технологій для обробки великих даних	13.11-19.11.23	вик.
4	Аналіз особливостей впливу Apache impala на великі данні	20.11-25.11.23	вик.
5	Дослідження ефективності технологій для обробки великих даних	27.11-03.12.23	вик.
6	Застосування Apache Impala в обробці великих даних	04.12-10.12.23	вик.
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	вик.
8	Розробка демонстраційних матеріалів	21.12-29.12.23	вик.

Здобувач вищої освіти _____ Єгор Ільїн
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Микола Гніденко
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 69 стор., 26 рис., 35 джерел.

Мета роботи – підвищення якості існуючих архітектур програмного забезпечення відмовостійкої системи обробки великих даних.

Об'єкт дослідження – програмне забезпечення систем обробки великих даних.

Предмет дослідження – методи, моделі та засоби побудови архітектури відмовостійкої системи обробки великих даних на базі технології Apache Impala.

Короткий зміст роботи: Розроблена ефективна архітектура програмного забезпечення відмовостійких систем обробки великих даних із застосуванням технології Apache Impala.

Докладно описані програмні механізми реплікації баз даних, проведено процедуру створення екземпляру бази даних. Також запропоновано способи усунення проблем та конфліктів, які можуть виникнути при використанні розробки.

В результаті розроблено покращену архітектуру відмовостійких систем обробки Big Data на базі технології Apache Impala. Реалізована система для обробки даних, побудована із застосуванням розробленої покращеної архітектури.

КЛЮЧОВІ СЛОВА: АРХІТЕКТУРА, APACHE IMPALA, ВІДМОВОСТІЙКІСТЬ, ВЕЛИКІ ДАНІ, ВІДНОВЛЕННЯ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Text part of the master's qualification work: 69 pages, 26 pictures, 35 sources.

The purpose of the work research to improve the quality of existing software architectures of a fault-tolerant big data processing system.

Object of research – the software of big data processing systems..

Subject of research – is methods, models and tools for building the architecture of a fault-tolerant big data processing system based on Apache Impala technology.

An efficient software architecture of fault-tolerant big data processing systems using Apache Impala technology has been developed.

The software mechanisms of database replication are described in detail, the procedure for creating a database instance is carried out. Ways to eliminate problems and conflicts that may arise when using the development are also proposed.

As a result, an improved architecture of fault-tolerant Big Data processing systems based on Apache Impala technology was developed. Implemented data processing system built using developed improved architecture.

KEYWORDS: ARCHITECTURE, APACHE IMPALA, RESILIENCE, BIG DATA, RECOVERY, INFORMATION TECHNOLOGY, SOFTWARE.

ЗМІСТ

ВСТУП

На сьогодні інтелектуально-інформаційні технології (ІІТ) та інформаційно-телекомунікаційні системи (ІТС) заполонили практично всі галузі виробничої діяльності суспільства. Через цей факт все з'являється потреба у застосуванні ІІТ та ІТС у різноманітних галузях життєдіяльності. Тому в результаті отримуємо нові вразливості кіберпростору та проблеми, що пов'язані з реалізацією способів його захисту.

Головним інструментом, що примусив світове ІТ-товариство замислитися стосовно новітніх стратегій зберігання та доступу до даних, стало накопичення обсягів інформації в Інтернеті. У зв'язку з цим виникла методика ефективної роботи з великими масивами даних. Вони отримали назву Big Data.

Наразі відмовостійкі системи безпеки мають великий попит на об'єктах з підвищеною безпекою, адже вони дозволяють охороняти дані в ситуації збоїв чи при виході з ладу якогось обладнання, з якого складається вся система.

Варто зазначити, що зараз основна ідея до проектування інфраструктури високої доступності заключається в процесі розгортання надлишкових даних та таких, які мають в складі інші пристрої, комплектуючі та програмні засоби від різноманітних постачальників. Така ідея пов'язується з набагато більшими витратами, однак не забезпечує очікуваного етапу надійності через слабку імплементацію елементів, техобмежень і складності адміністративного керування. Крім того, безліч фірм проходять через процес зростання власної інфраструктури розумних технологій для росту ефективності власної справи, а тому об'єднання БД є важливою складовою даного процесу.

Виходячи наведеного вище, наше дослідження особливостей розробки та реалізації відмовостійкої системи обробки великих даних на базі технології Apache Impala з метою забезпечення високої пропускної здатності є актуальним.

1 АНАЛІТИЧНИЙ ОГЛЯД ІНФОРМАЦІЙНИХ ДЖЕРЕЛ З ТЕХНОЛОГІЙ ОБРОБКИ ВЕЛИКИХ ДАНИХ

1.1 Опис предметної області

Інформаційна система (ІС) - формальна, соціотехнічна, організаційна система, призначена для збору, обробки, зберігання та розповсюдження інформації. З соціально-технічної точки зору, інформаційна система складається з чотирьох елементів: завдань, людей, структур (чи ролей) і технологій. Інформаційні системи можна визначити як інтеграцію компонентів для збору, зберігання та обробки даних. Інформаційні системи можна визначити як інтеграцію компонентів для збору, зберігання та обробки даних, які потім використовуються для створення цифрових продуктів, що інформують, поповнюють знання та сприяють прийняттю рішень.

Існують різні типи інформаційних систем, наприклад: системи обробки транзакцій, системи підтримки прийняття рішень, системи управління знаннями, системи управління навчанням, системи управління базами даних та офісні інформаційні системи.

Критично важливим для більшості інформаційних систем є інформаційні технології, які зазвичай призначені для того, щоб дозволити людям виконувати завдання, для яких мозок не дуже підходить, наприклад: обробка великих обсягів інформації, виконання складних обчислень і управління безліччю одночасних процесів.

Комп'ютерна інформаційна система - це, по суті, ІС, яка використовує комп'ютерні технології для виконання деяких або всіх своїх завдань. Основними компонентами комп’ютерних інформаційних систем є:

1) апаратне забезпечення - це пристрої, які працюють разом для отримання, обробки та відображення даних та інформації, такі як монітори, процесори, принтери та клавіатури;

2) програмне забезпечення - це програми, які дозволяють апаратному забезпеченню обробляти дані;

3) база даних - це набір пов'язаних файлів або таблиць, що містять пов'язані дані;

4) мережа - це взаємопов'язана система, яка дозволяє різним комп'ютерам спільно використовувати ресурси;

5) процедура - це команда, яка поєднує вищезгадані компоненти для обробки інформації та досягнення кращих результатів.

Перші чотири компоненти (апаратне забезпечення, програмне забезпечення, база даних та мережа) складають так звану платформу інформаційних технологій. Працівники інформаційних технологій могли б використовувати ці компоненти для створення інформаційних систем, які стежать за заходами безпеки, ризиками та управлінням даними. Ці дії відомі як послуги інформаційних технологій.

У обчислювальній техніці база даних - це організований набір даних або тип сховища даних, заснований на використанні системи керування БД (СКБД), програмного забезпечення, яке взаємодіє з кінцевими користувачами, додатками та самою базою даних для збору та аналізу даних.

СКБД додатково включає основні засоби, призначені для адміністрування бази даних.

Сукупність бази даних, СКБД та пов’язаних із нею додатків можна назвати системою баз даних. Часто термін бази даних також широко використовується для позначення будь-якої СКБД, системи баз даних або додатка, пов’язаного з базою даних.

Невеликі БД можуть зберігатися у файловій системі, тоді як великі бази даних розміщуються у комп’ютерних кластерах або хмарних сховищах.

Проектування БД охоплює формальні методи та практичні міркування, включаючи моделювання даних, ефективне представлення та зберігання даних, мови запитів, безпеку та конфіденційність конфіденційних даних, а також питання розподілених обчислень, включаючи підтримку одночасного доступу та відмовостійкості.

Функціональність, що надається СКБД, може відрізнятися. Основна функціональність - зберігання, пошук та оновлення даних. Кодд запропонував такі функції та послуги, які має забезпечувати повноцінна СКБД загального призначення:

– зберігання, пошук та оновлення даних;

– доступний користувачеві каталог або словник даних, що описує метадані;

– підтримка транзакцій та паралелізму;

– засоби відновлення бази даних у разі пошкодження;

– підтримка авторизації доступу та оновлення даних;

– доступ до підтримки з віддалених місць;

– забезпечення дотримання обмежень задля забезпечення відповідності даних у базі даних певним правилам.

На рис. 1.1 представлено 3 види архітектури з’єднання клієнта з інформацією. Перший варіант являє однозвенную архітектуру, що окрім даних має щабель для клієнта.

Дволанкова архітектура має ще й сервер БД, який забезпечує значну частину логіки керування даними, в той час як клієнт в основному зайнятий дослідженням даних в зручному форматі.

Big Data впершу чергу відносяться до наборів інформації, які надто ємна або складна для обробляння за допомогою традиційного ПЗ для обробляння даних. Дані з великим числом записів (рядків) мають більшу статистичну силу, а дані з більш високою складністю (більше атрибутів або стовпців) можуть призвести до більш високого рівня помилкових виявлень.

Хоча інтерпретація, яка іноді використовується вільно, частково через відсутність формального визначення, здається, що найкраще описує великі дані, яка пов’язана з великим обсягом інформації, яку ми не можемо зрозуміти, якщо використовувати тільки в менших кількостях.

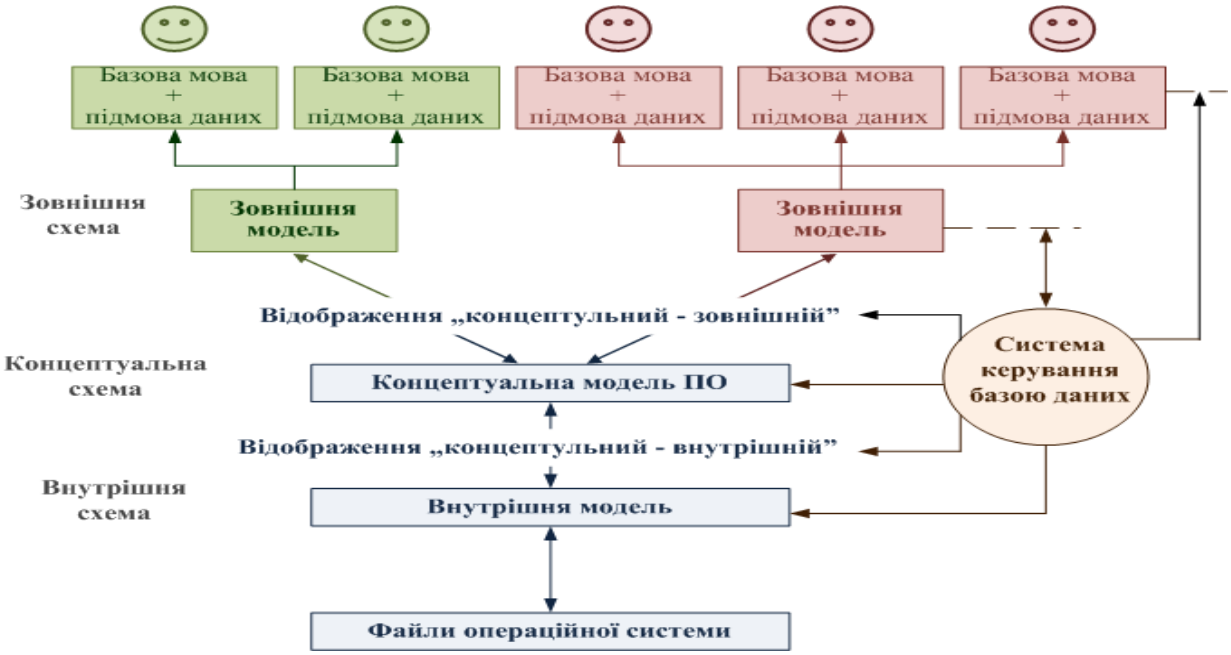

Рисунок 1.1 - Архітектура БД

Показово, що у триланкової СКБД є ще одна проміжна ланка, а саме: сервер додатків. Він призначається для того, щоб надати зв’язок клієнту з сервером БД і позбавити користувача від будь-яких проблем відносно керування даними (рис 1.2).

Рисунок 1.2 – Схема з’єднання клієнта з БД - одноланкова, двуланкова

Проблеми аналізу Big Data включають збір даних, їх зберігання, аналіз, пошук, спільне застосування, передачу, візуалізацію, запити, оновлення, конфіденційність даних та їх джерело. Спочатку Big Data асоціювалися з трьома ключовими поняттями: обсяг, різноманітність та швидкість.

При аналізі великих даних виникають проблеми з вибіркою, і таким чином раніше допускалися лише спостереження та вибірка. Таким чином, четверте поняття, істинність відноситься до якості або проникливості Big Data. Без достатніх інвестицій у досвід для забезпечення достовірності Big Data обсяг та різноманітність даних можуть призвести до витрат та ризиків, які перевищують можливості організації зі створення та вилучення цінності з Big Data.

Big Data - це різноманітні дані, що надходять із вищою швидкістю, обсяг яких постійно зростає. Таким чином, три основні властивості великих даних - це різноманітність, висока швидкість надходження та великий обсяг.

Основні властивості Big Data такі:

- Обсяг.

Кількість даних – важливий фактор. Маючи в своєму розпорядженні велику кількість, Вам потрібно буде обробляти великі обсяги неструктурованих даних низької щільності. Цінність таких даних не завжди відома. Це можуть бути дані каналів Twitter, дані відвідуваності веб-сторінок, дані мобільних додатків, мережевий трафік, дані датчиків. Деякі організації можуть надходити десятки терабайт даних, до інших - сотні петабайт.

- Швидкість.

Швидкість у цьому контексті - це швидкість прийому даних і, можливо, дій з їхньої основі. Зазвичай високошвидкісні потоки даних надходять у оперативну пам’ять, а чи не записуються на диск. Деякі "розумні" продукти, що функціонують на основі Інтернету, працюють у режимі реального чи практично реального часу. Відповідно, такі дані вимагають оцінки та дій у реальному часі.

- Різноманітність.

Різноманітність означає, що доступні дані належать до різних типів. Традиційні типи даних структуровані і можуть бути одразу збережені в реляційній БД. З появою Big Data дані почали надходити у неструктурованому вигляді. Такі неструктуровані та напівструктуровані типи даних як текст, аудіо та відео вимагають додаткової обробки для визначення їх значення та підтримки метаданих.

Ще дві якості сформувалися останні кілька років: цінність і достовірність. Дані мають внутрішньо властиву їм цінність. Однак, щоб вони приносили користь, цю цінність необхідно розкрити.

Найновіші досягнення у сфері технологій дозволили значно знизити вартість сховищ та обчислень, що дає можливість зберігати та обробляти обсяги даних, що постійно зростають. Сучасні технології дозволяють зберігати та обробляти більше даних за меншу вартість, що дозволяє Вам приймати більш точні та виважені бізнес-рішення.

Вилучення цінності з великих даних не зводиться тільки до їх аналізу (це їхня окрема перевага). Йдеться про комплексний дослідний процес за участю фахівців з глибокого аналізу, корпоративних користувачів та керівників, які будуть ставити правильні питання, виявляти шаблони, робити обґрунтовані припущення та передбачати поведінку.

У вузькому сенсі Big Data - масиви структурованих, слабоструктурованих та неструктурованих даних, обсяги яких настільки великі, що їхня обробка традиційними засобами стає неефективною або взагалі неможливою.

У широкому значенні Big Data - комплекс засобів і методів для обробки та аналізу масивів даних, що підпадають під визначення великих даних.

Спочатку з великими даними пов’язували три ключові концепції (правило «трьох V»):

1. Об’єм (volume). Дані в компанії накопичуються з багатьох джерел у величезному обсязі.

2. Швидкість зростання (velocity). Швидке зростання обсягів даних. Особливо характерно для компаній у галузі мережевої торгівлі та електронної комерції, де щодня можуть генеруватися сотні терабайт даних.

3. Розмаїття (variety). Дані з вхідного потоку можуть бути різноманітних форматів (таблиці, текст, відео, аудіо), а також бути структурованими та неструктурованими.

Поступово правило "трьох V" збагатилося додатковими елементами і трансформувалося в: "чотири V" (veracity - достовірність), "п’ять V" (viability -

життєздатність і value - цінність) і "сім V" (variability - мінливість і visualization - візуалізація).

В даний час поняття Big Data пов’язане з використанням передбачуваної та поведінкової аналітики та інших напрямів аналізу даних з метою отримання знань із величезних масивів даних.

Головними проблемами, з якими доводиться стикатися під час роботи з Big Data, є зростання обчислювальних витрат як у плані часу, так і необхідних обсягів пам’яті. Звідси випливають завдання оптимізації розміщення даних в оперативній пам’яті, кількості звернень до диска та кількості проходів за даними.

Якщо обробка даних неможлива одному комп’ютері, її алгоритм можна розділити на частини і спробувати виконати кількох машинах. Ця ідея послужила поштовхом для появи та розвитку методологій та інструментів розподіленої обробки, наприклад, MapReduce, HDFS, Hive.

Для зниження кількості ітерацій та/або проходів по набору даних при роботі аналітичних алгоритмів використовуються різні ймовірнісні модифікації. Приклад такого алгоритму є оптимальне залежне від даних хешування для наближеного пошуку найближчих сусідів.

З великими даними стикаються у багатьох сферах: наука, електронна комерція, телекомунікації, фінансовий сектор.

Крім того, для вирішення бізнес-завдань можна залучати дані зі сторонніх джерел.

1.2 Сучасні стандарти обробки та зберігання великих даних

Поява програм на базі відкритого коду, таких як Hadoop та пізніше Spark, відіграла значну роль у поширенні Big Data, оскільки ці тнхнології спрощують процеси обробляння Big Data та знижують вартість їх зберігання. За минулі роки об’єми Big Data зросли на порядки.

Величезні Big Data з’являються в результаті діяльності користувачів, але тепер не лише їх.

З появою Інтернету речей (IoT) все більше пристроїв отримує підключення до Інтернету, що дозволяє збирати дані про моделі дій користувачів і роботу продуктів. А коли з’явилися технології машинного навчання, обсяг даних зріс ще більше.

Big Data мають довгу історію розвитку, проте їхній потенціал ще далеко не розкритий. Хмарні обчислення розсунули межі застосування великих даних набагато ширше. Хмарні технології забезпечують по-справжньому гнучкі можливості масштабування, що дозволяє розробникам розгортати кластери для тестування вибіркових даних на вимогу.

Крім того, також все більш значущими стають графові бази даних, що дозволяють відображати величезні обсяги даних так, щоб їх аналізувати можна було швидко і всеосяжно (рис. 1.3).

Перевагами Big Data є такі:

1. Великі дані дають можливість отримувати повніші відповіді, тому вони надають більше інформації. Більш докладні відповіді означають, що можна бути більш впевненими у достовірності даних, що забезпечує абсолютно новий підхід до вирішення завдань.

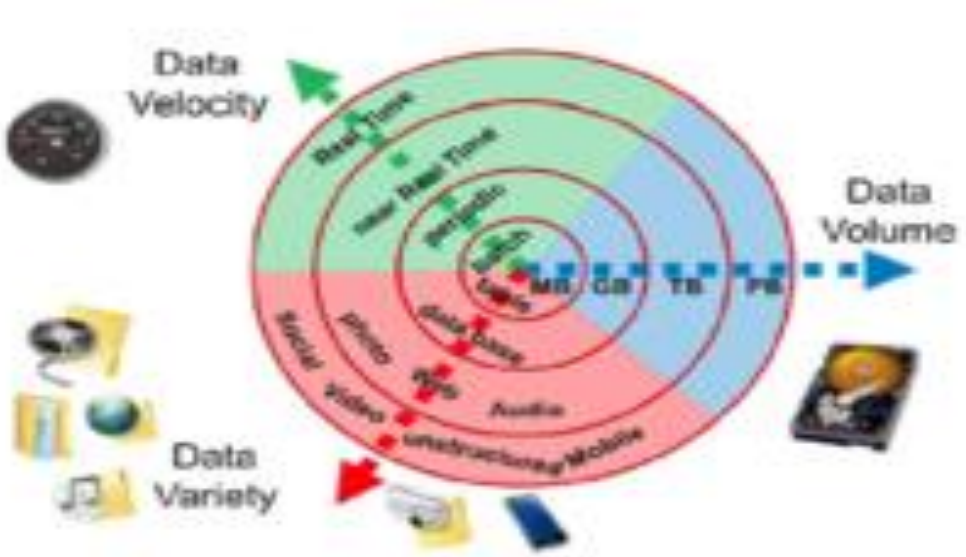

Рисунок 1.3 - Популярність основних характеристик великих даних за обсягом, швидкістю та різноманітністю

У готових рішеннях всі ці інструменти вже інтегровані між собою і супроводжуються повним пакетом програмної документації, що полегшує процеси адміністрування та підтримки інфраструктури для великих даних.

Але необхідно підкреслити, що за ці зручні фреймворки необхідно добре заплатити.

2. Інформація здатна надходити у всіх типах форматів від структурованих наборів даних, числових даних в традиційних базах даних до неструктурованих текстових документів, електронної пошти, відео, аудіо, даних біржових котирувань і фінансових транзакцій. Раніше електронні таблиці і бази даних були єдиними джерелами даних, які розглядалися більшістю додатків. У наші дні в додатках для аналізу також враховуються дані у вигляді електронних листів, фотографій, відео, пристроїв моніторингу, PDF – файлів, аудіо. Така різноманітність неструктурованих даних створює певні проблеми для зберігання, видобутку і аналізу даних. Різноманітність даних зображено на рис. 1.4.

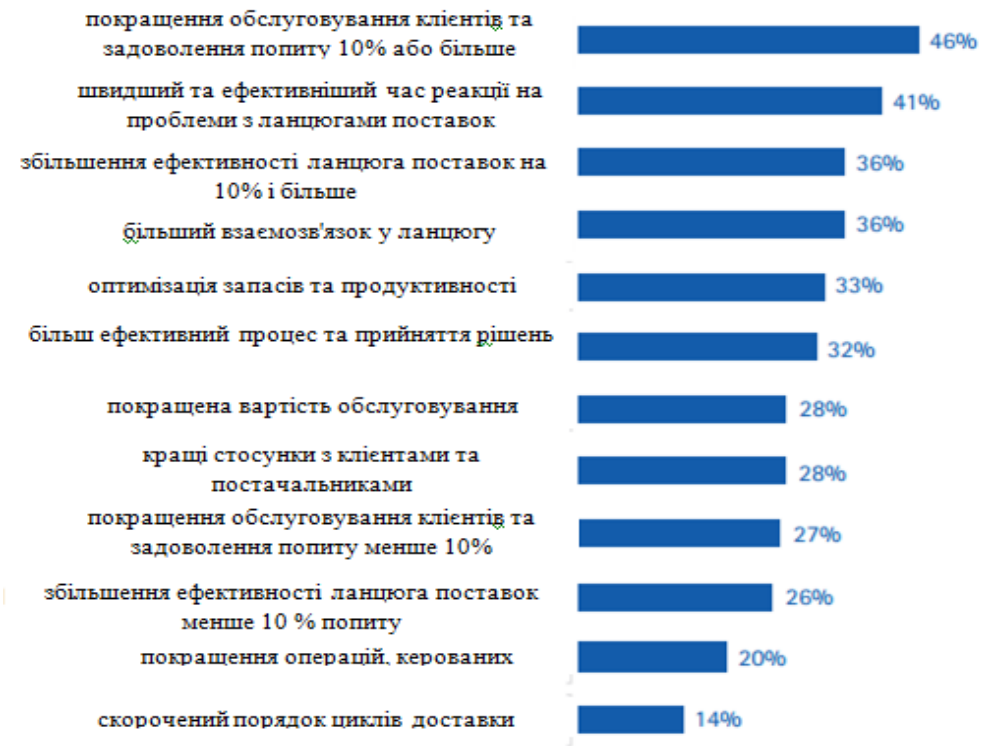

Рисунок 1.4 – Існуючі технології даних в області Big Data

Разом з тим, існуть складнощі при використанні Big Data, а саме:

1. Насамперед Big Data передбачувано займають багато місця. Хоча нові технології зберігання постійно розвиваються, обсяг даних зростає вдвічі майже кожні два роки. Організації досі стикаються з проблемами зростання обсягів даних та їх ефективного зберігання. Але недостатньо просто знайти велике сховище. Дані необхідно використовувати, щоб вони приносили зиск, і розмір цієї вигоди залежить від обробки даних.

2. Чисті дані, тобто дані, актуальні для клієнта та організовані для ефективного аналізу, потребують ретельної обробки. Фахівці з вивчення даних витрачають від 50 до 80% робочого дня на обробку та підготовку даних для використання.

3. І, нарешті, технології Big Data розвиваються семимильними кроками. Декілька років тому Apache Hadoop була найпопулярнішою технологією для роботи з великими даними. Платформа Apache Spark з’явилася у 2014 році. Сьогодні оптимальним підходом є спільне використання цих двох платформ. Щоб встигати за розвитком великих даних, потрібно докладати великих зусиль.

Основні компоненти великих даних описуються наступним чином:

- методи аналізу даних, такі як A/B-тестування, машинне навчання та обробка природної мови;
- технології Big Data, такі як бізнес-аналітика, хмарні обчислення та бази даних;
- візуалізація, така як діаграми, графіки та інші види відображення даних;

Практики процесів аналізу Big Data, як правило, вороже ставляться до повільніших загальних сховищ, віддаючи перевагу сховищу з прямим підключенням (DAS) в його різних формах від твердотільного накопичувача (SSD) до диска SATA великої ємності, захованого всередині вузлів паралельної обробки. Сприйняття загальних архітектур зберігання – мережі зберігання даних (SAN) та мережевого сховища (NAS) – таке, що вони відносно повільні, складні та дорогі. Ці якості несумісні з системами аналітики великих даних, які

процвітають за рахунок продуктивності системи, стандартної інфраструктури та низької вартості.

- доставка інформації в реальному або близькому до реального часу є однією з визначальних характеристик аналітики Big Data. Таким чином, затримки уникають завжди і скрізь, де це можливо. Дані в пам'яті або на диску з прямим підключенням в порядку, а дані в пам'яті або на диску на іншому кінці з'єднання FC SAN немає. Вартість SAN у масштабі, необхідному для аналітичних додатків набагато вище, ніж в інших методів зберігання. Big Data дозволяють отримувати нові цінні відомості, які відкривають нові можливості та бізнес-моделі.

Щоб розпочати роботу з Big Data, необхідно виконати три дії, а саме:

1. Інтеграції.

Технологія Big Data дозволяє об'єднувати дані із розрізнених джерел та додатків. Традиційні механізми інтеграції, такі як засоби для отримання, перетворення та завантаження даних (ETL), не справляються з подібними завданнями. Для аналізу наборів даних розміром в терабайт, а то й петабайт, потрібні нові стратегії та технології. Під час етапу інтеграції відбувається додавання, обробка та форматування даних, щоб корпоративним аналітикам було зручно з ними працювати.

2. Управління.

Big Data потрібне об'ємне сховище. Рішення для зберігання може бути розміщене в локальному або хмарному середовищі або там і там. Можна зберігати дані у бажаному форматі та застосовувати бажані вимоги до обробки (і необхідні механізми обробки) до наборів даних у міру необхідності. Більшість організацій обирають рішення для зберігання даних, залежно від того, де вони зберігаються в даний час. Хмарні сховища користуються зростаючою популярністю, оскільки підтримують актуальні вимоги до обчислень і дозволяють використовувати ресурси в міру потреби.

3. Аналіз.

Вкладення у Big Data окупляться сповна, коли Ви приступите до аналізу даних і почнете робити дії, виходячи з отриманих відомостей. Забезпечте новий

рівень прозорості завдяки візуальному аналізу різноманітних наборів даних. Використовуйте глибокий аналіз даних, щоб робити нові відкриття. Ділиться своїми відкриттями з іншими. Створюйте моделі даних за допомогою машинного навчання та штучного інтелекту. Застосуйте свої дані насправді. Дані самі по собі не несуть цінності, поки не перетворюються в корисну інформацію і знання, які можуть допомогти керівництву в прийнятті рішень. Для цього є ряд програмного забезпечення для роботи з Big Data, доступних на ринку. Ця програма допомагає зберігати, аналізувати, складати звіти і багато іншого.

1.3 Постановка задач розробки

Метою є розробка та практична реалізація архітектури програмного забезпечення відмовостійкої системи оброблення Big Data на базі технології Apache Impala.

Ставимо перед собою наступні задачі:

2. Дослідити наявні системи обробки Big Data на предмет застосовуваних рішень для забезпечення системної відмовостійкості.

3. Розробити архітектурне рішення, що надасть змогу максимально реалізувати принцип відмовостійкості розробленої системи Big Data.

4. Провести оцінку ефективності запропонованого проєкту.

2 МЕТОДИ ТА ЗАСОБИ ВИРІШЕННЯ ЗАДАЧІ З ПРОЕКТУВАННЯ СИСТЕМИ ОБРОБКИ ВЕЛИКИХ ДАНИХ НА БАЗІ ТЕХНОЛОГІЇ APACHE IMPALA

2.1 Огляд сучасних моделей машинного навчання, побудованих на Big Data

Розглянемо існуючі інструменти для роботи з Big Data:

- Apache Spark – це уніфікований аналітичний інструмент із відкритим вихідним кодом для великомасштабного оброблення інформації. Spark надає інтерфейс для програмування кластерів з неявним паралелізмом даних та відмовостійкістю. Спочатку розроблена в AMPLab Каліфорнійського університету в Берклі, кодова база Spark пізніше була передана в дар Apache Software Foundation, яка відтоді підтримує її (рис. 2.1).

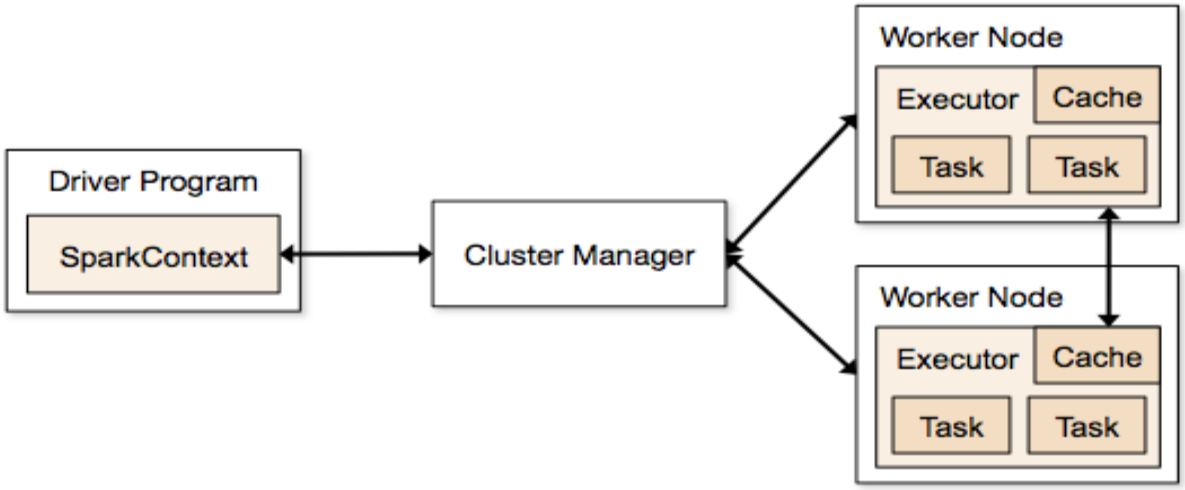

Рисунок 2.1 – Архітектура Apache Spark

На відміну від традиційного оброблювача з ядра Apache Hadoop, що представляє дворівневу концепцію MapReduce зі сховищем на диску, може використовувати спеціальні примітиви для рекурентного оброблення в

оперативній пам’яті, завдяки чому можна виграти у швидкості роботи для деяких класів завдань.

Проект надає програмні інтерфейси для мов Java, Scala, Python, R. Спочатку написаний на Scala, згодом додана істотна частина коду Java для надання можливості написання програм безпосередньо Java. Складається з ядра та кількох розширень, таких як Spark SQL (дозволяє робити SQL-запити над даними, що надходять), Spark Streaming (надбудова для обробки потокових даних), Spark MLlib (набір бібліотек МН), GraphX (призначений для розподіленої обробки графів). Може працювати як серед кластера Apache Hadoop під керуванням YARN, так і без компонентів ядра Apache Hadoop, підтримує кілька розподілених систем зберігання - HDFS, OpenStack Swift, NoSQL-СКБД Apache Cassandra, Amazon S3 (рис. 2.2).

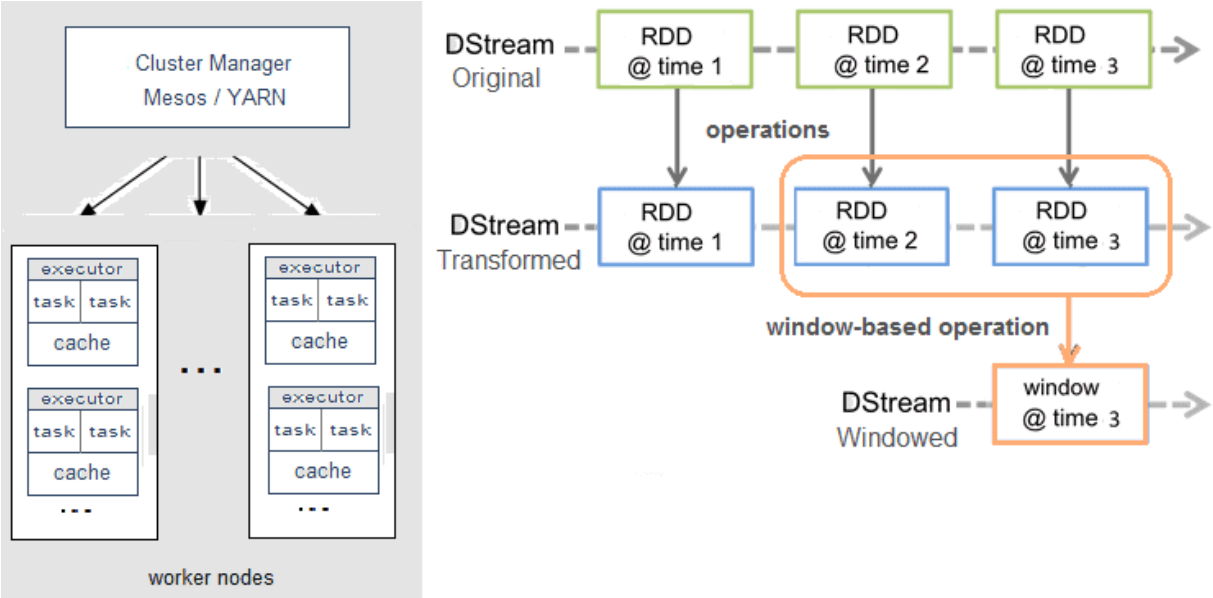

Рисунок 2.2 - Принцип роботи Apache Spark

- Elasticsearch.

Elasticsearch – це одна з найпопулярніших пошукових систем в області Big Data, масштабоване нереляційне сховище даних з відкритим вихідним кодом, аналітична NoSQL-СКБД із широким набором повнотекстових функцій пошуку.

Elasticsearch (ES) – масштабована утиліта повнотекстового пошуку та аналітики, яка дозволяє швидко в режимі реального часу зберігати, шукати та досліджувати обсяги Big Data.

ES є ядром ELK-стеку (Elastic Stack), до складу якого, крім Elasticsearch, входять такі продукти:

1)Logstash - інструмент збору, перетворення та збереження в загальному сховищі подій з різних джерел (файли, бази даних, логи та ін) в реальному часі;

2)Kibana – веб-інтерфейс для Elasticsearch, щоб взаємодіяти з даними, що зберігаються в його індексах ES через динамічні панелі моніторингу, таблиці, графіки та діаграми, які відображають зміни в ES-запитах у реальному часі;

3)FileBeat - агент на серверах для відправлення різних типів оперативних даних в ES.

У масштабних Big Data системах кілька копій Elasticsearch об’єднуються в кластер. Пошукові індекси можна розділити на сегменти, реплікувавши кожен із яких кілька разів. Це забезпечує відмовостійкість системи. На вузлі ES кластера може розміщуватися кілька сегментів. Кожен вузол кластера діє як координатор для делегування операцій правильному сегменту з автоматичним перебалансуванням та маршрутизацією.

Пов’язані дані часто зберігаються в тому самому індексі з одного або декількох первинних сегментів і декількох реплік. Після створення індексу кількість первинних сегментів не можна змінити. Довгострокове зберігання індексу забезпечує шлюз, дозволяючи відновлювати індекс при збої сервера (рис. 2.3).

Завдяки широкому набору функціональних можливостей, особливо повнотекстового пошуку за багатьма мовами та аналітикою в реальному часі, Elasticsearch активно застосовується в різних Big Data системах великих та середніх компаній по всьому світу. З найбільш відомих зарубіжних користувачів варто відзначити корпорації Netflix, IBM, Facebook, Amazon, GitHub, Wikimedia, CERN, Mozilla, Adobe.

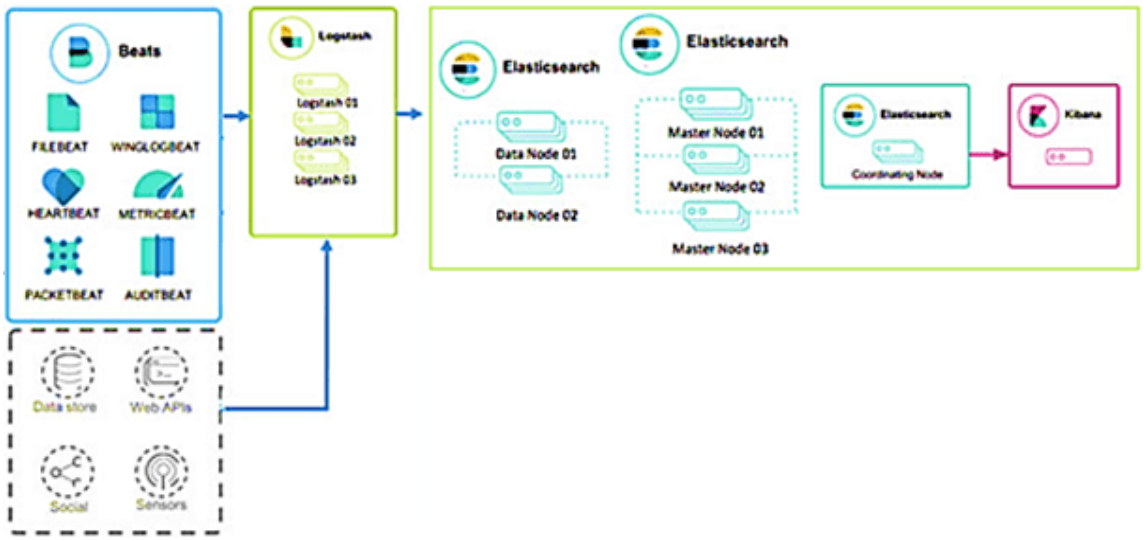

Рисунок 2.3 - Принцип функціонування ELK-стеку

- Hive

Hive – це система керування БД (СКБД) у рамках технології Hadoop для зберігання та обробляння Big Data у розподіленому середовищі. Хайв дозволяє проектувати структури Big Data (таблиці, партиції, бакети) за допомогою SQL-подібної мови, що називається HiveQL.

Apache Hive – це SQL інтерфейс доступу до даних для платформи Apache Hadoop. Хайв дозволяє виконувати запити, агрегувати та досліджувати дані із використанням SQL-синтаксису.

Дані у файловій системі HDFS використовують схему доступу на читання, що дозволяє працювати з ними як зі звичайною таблицею або реляційною базою даних; запити HiveQL є широкомовними, тобто вони можуть бути прочитані файловою системою HDFS, але не можуть бути прочитані запитами HiveQL до Java-коду завдань MapReduce (рис. 2.4).

Запити Hive створюються мовою запитів HiveQL, яка базується на мові SQL, але не має повної підтримки стандарту SQL-92. Проте ця мова дає змогу програмістам застосовувати власні запити, коли виявляється неефективним використання можливостей Hive QL. Хайв QL може бути розширений за допомогою скалярних функцій користувача (UDF), агрегацій (UDAF кодів), і табличних функцій (UDTF).

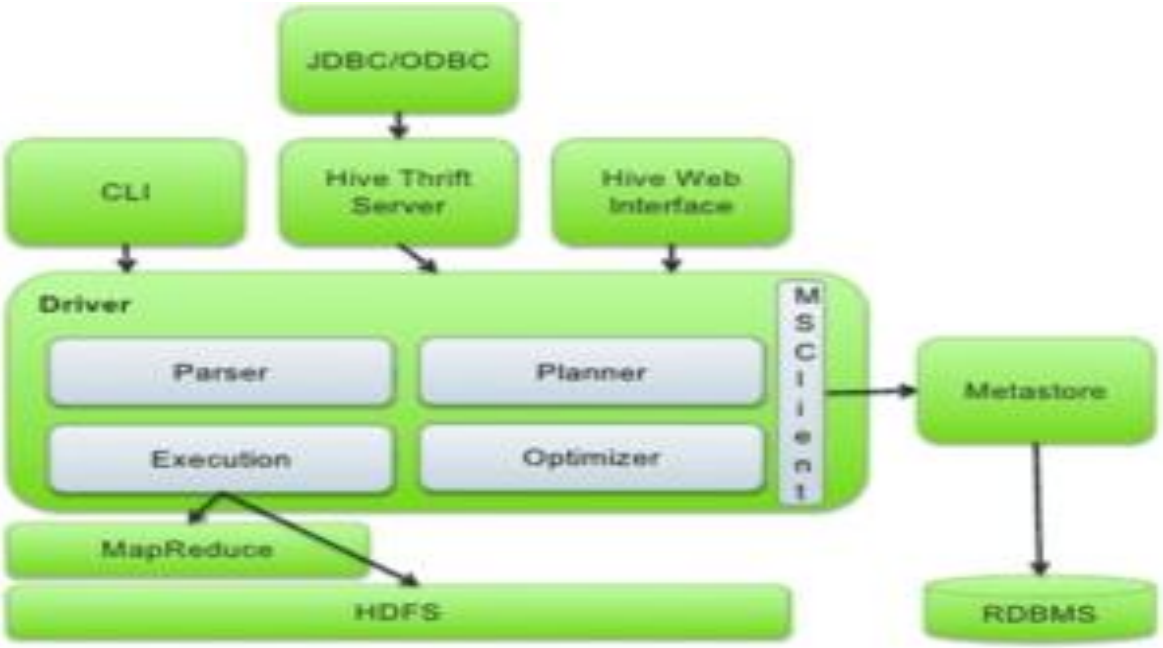

Рисунок 2.4 – Архітектура Hive

Apache Hive – це СКБД для зберігання та обробляння Big Data у розподіленому кластері екосистеми Hadoop.

СКБД Apache Hive включає такі елементи:

- HCatalog – це компонент Хайв, який відповідає за керування таблицями та сховищами. Цей компонент також забезпечує користувачів різними інструментами обробки Big Data (включаючи MapReduce та Pig) для більш простого читання та запису даних.

- WebHCat – це компонент, що дозволяє виконувати базові операції (читання, запис та видалення) з метаданими в мережі Хайв за допомогою використання інтерфейсу HTTP (Hyper Text Transfer Protocol). WebHCat також використовується як сервер для розгортання середовища Хайва.

- Dask.

Dask - це бібліотека з відкритим кодом, призначена для забезпечення паралелізму з існуючим стеком Python. Він забезпечує інтеграцію з бібліотеками Python, такими як NumPy Arrays, Pandas DataFrames та scikit-learn, щоб

забезпечити паралельне виконання на кількох ядрах, процесорах та комп'ютерах без необхідності вивчення нових бібліотек чи мов.

Dask складається з двох частин:

1. API колекцій для паралельних списків, масивів та кадрів даних для природного масштабування Numpy, NumPy, Pandas та scikit-learn для роботи в середовищах з великим обсягом пам'яті або розподілених середовищах. Колекції Dask є паралельними колекціями з базової бібліотеки (наприклад, масив Dask складається з масивів Numpy) і запускаються поверх планувальника завдань.

Планувальник завдань для побудови графіків задач та координації, планування та моніторингу завдань, оптимізований для інтерактивних робочих навантажень між ядрами ЦП та машинами (рис. 2.5).

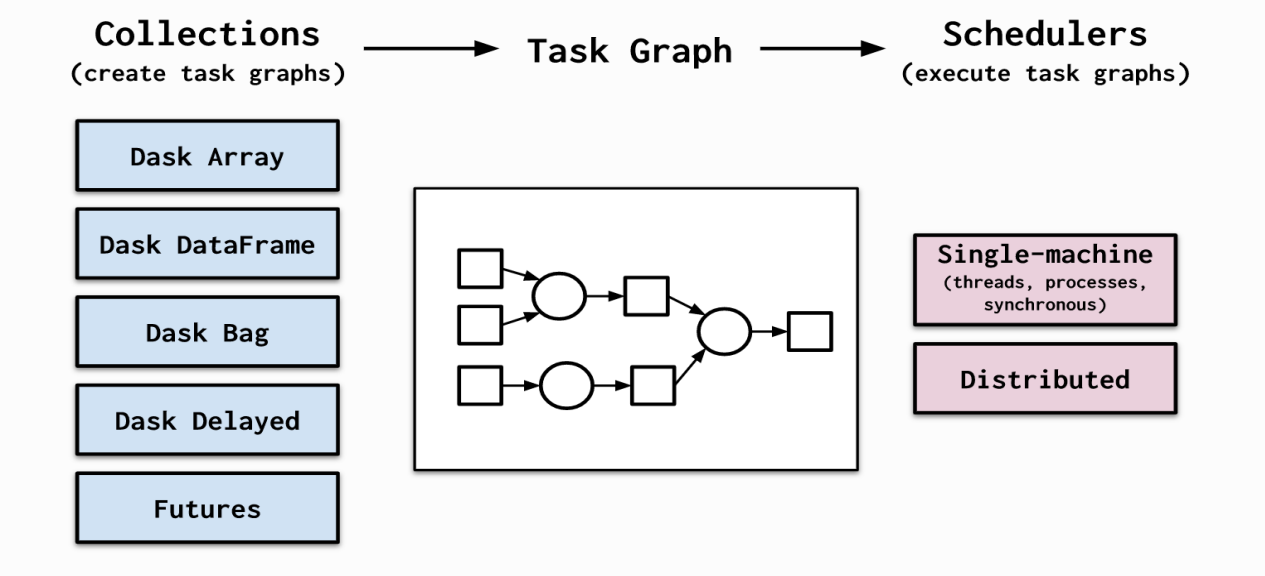

Рисунок 2.5 – Архітектура Dask

Кожна з трьох паралельних колекцій Dask - DataFrames, Bags і Arrays - може автоматично використовувати дані, розділені між ОЗП і диском, розподілені кількома вузлами в кластері, залежно від доступності ресурсів. Для завдань, які можна розпаралелити, але які погано вписуються в абстракції високого рівня, такі як Dask Arrays або DataFrames, існує відкладена функція, яка використовує декоратори Python для зміни функцій, щоб вони працювали ліниво. Це означає, що виконання затримується, а функція та її аргументи поміщаються у граф задач.

Планувальник завдань Dask може масштабуватися до кластерів із тисячі вузлів, а його алгоритми були протестовані на деяких із найбільших у світі суперкомп'ютерів. Його інтерфейс планування завдань можна налаштувати для конкретних завдань. Dask забезпечує мінімальні накладні витрати, малу затримку та мінімальну серіалізацію, необхідну для швидкості.

Зручна для користувача високорівнева мова програмування Python та бібліотеки Python, такі як NumPy, Pandas та scikit-learn, набули широкого поширення серед фахівців за даними.

Розроблені доти, як випадки використання великих даних стали настільки поширеними, ці бібліотеки мали сильного рішення для паралелізму. Python був кращим вибором для одноядерних обчислень, але користувачі змушені були шукати інші рішення для багатоядерного або багатомашинного паралелізму. Це викликало розчарування та перерву у роботі користувачів.

Ця зростаюча потреба у масштабуванні робочих навантажень у Python призвела до природного зростання Dask за останні п'ять років.

Dask - це спосіб, що легко встановлюється і швидко налаштовується, прискорити аналіз даних у Python, який не вимагає від розробників оновлення апаратної інфраструктури або переходу на іншу мову програмування. Синтаксис, що використовується для запуску завдань Dask, такий самий, як і для інших операцій Python, тому його можна інтегрувати з невеликою доробкою коду.

Також популярний серед веб-розробників Python має надійний мережевий стек, який Dask використовує для створення гнучкої, продуктивної розподіленої обчислювальної системи, здатної масштабувати широкий спектр робочих навантажень. Гнучкість Dask допомагає йому виділитися серед інших рішень для роботи з великими даними, як Hadoop або Apache Spark, а підтримка нативного коду робить його особливо зручним для користувачів Python і розробників C/C++/CUDA.

Dask був швидко прийнятий спільнотою розробників Python і вирiс разом із популярністю Numpy та Pandas, які надають цінні розширення Python для вирішення спеціальної аналітики та математичних обчислень.

Він також набагато краще масштабується, ніж Pandas, і особливо добре працює із завданнями, які легко розпаралелити, наприклад, із сортуванням даних у тисячах електронних таблиць. Прискорювач може завантажувати в пам’ять сотні кадрів даних Pandas та координувати їх за допомогою єдиної абстракції.

Сьогодні Dask керується спільнотою розробників, яка охоплює десятки установ та проектів PyData, таких як Pandas, Jupyter та Scikit-Learn. Інтеграція Dask із цими популярними інструментами привела до швидкого поширення: близько 20% серед розробників, яким потрібні інструменти Pythonic для роботи з великими

Використовуючи Pandas DataFrames, Dask може включати програми для аналізу часових рядів, бізнес-аналітики та підготовки даних. Dask-ML, бібліотеку для розподіленого та паралельного машиного навчання, можна використовувати з Scikit-Learn та XGBoost для створення масштабованого навчання та прогнозування на великих моделях та наборах даних. Розробники можуть використовувати стандартні робочі процеси Dask для підготовки та налаштування даних, а потім передавати дані в XGBoost або Tensorflow.

Відмовостійкість - це властивість стійкості, яка дозволяє системі продовжувати працювати належним чином у разі збою або серйозної дисфункції одного або кількох її компонентів.

Якщо якість її роботи взагалі знижується, то це зниження пропорційно до серйозності відмови в порівнянні з наївно спроектованою системою, в якій навіть невелика відмова може призвести до повного виходу з ладу. Відмовостійкість особливо потрібна в системах з високою доступністю, критично важливих і навіть життєво важливих системах.

Відмовостійкий дизайн гарантує, що система продовжує працювати за призначенням, коли відбувається часткова відмова системи, в деяких випадках на нижчих рівнях, замість того, щоб повністю зупинити роботу. Цей термін найчастіше використовується для опису СК, які спроектовані таким чином, щоб продовжувати працювати не на повну потужність у разі часткової відмови, хоча

це може супроводжуватися зниженням пропускної здатності і збільшенням часу відгуку.

Іншими словами, апаратні або програмні проблеми не призведуть до зупинки всієї системи. Прикладом в іншій області є автомобіль, сконструйований таким чином, щоб він продовжував керуватися навіть у разі проколу однієї з шин, або конструкція, здатна зберігати свою цілісність за наявності ушкоджень, спричинених такими причинами, як втома, корозія, виробниче ушкодження. недоліки чи вплив.

Відмовостійкість окремих систем може бути досягнута шляхом передбачення виняткових ситуацій і побудови систем, здатних впоратися з ними. Крім того, системи часто намагаються самостабілізуватися, щоб наблизитися до безвідмовного стану. Однак, якщо наслідки відмови системи є катастрофічними або вартість забезпечення належної надійності є занадто високою, найкращим рішенням може бути певна форма резервування.

У будь-якому випадку, якщо наслідки збою системи є катастрофічними, система повинна мати можливість повернутися до безпечного стану шляхом відкату. Це схоже на відновлення за допомогою відкату, але може бути виконано людиною, якщо вона присутня в циклі.

Забезпечити відмовостійку конструкцію для кожного компонента зазвичай неможливо. Супутнє резервування спричиняє ряд недоліків: збільшення ваги, розміру, енергоспоживання, вартості, а також часу на проектування, перевірку та тестування. Тому необхідно розглянути ряд варіантів, щоб визначити, які компоненти мають бути стійкими до відмови:

– наскільки критичний компонент? В автомобілі радіо не критично, тому цей компонент менше потребує відмовостійкості;

– наскільки велика можливість виходу компонента з ладу? Деякі компоненти, наприклад, приводний вал автомобіля, навряд чи вийдуть з ладу, тому відмовостійкість не потрібна;

– наскільки дорого варто зробити компонент стійким до відмов? Наприклад, потреба в резервному автомобільному двигуні, ймовірно, буде надто

дорогою як з економічної точки зору, так і з точки зору ваги та місця, щоб її можна було розглядати.

Відмовостійка архітектура з погляду інженерії - це спосіб побудови відмовостійких систем, які зберігають працездатність (можливо, зі зниженням ефективності) при відмовах елементів. Термін часто використовується у створенні комп'ютерних систем, які продовжують працювати з можливим зменшенням пропускної здатності або збільшенням часу відгуку у разі відмови частини елементів системи (проблем з апаратною або програмною частиною). Відмовостійка архітектура в ПК застосовується, наприклад, у процесі реплікації.

2.2 Розгортання системи обробки великих даних

Переваги стійких до відмови технічних рішень очевидні, але також у них існують і недоліки.

Складнощі у виявленні прихованих відмов резервованих елементів. Наприклад, водій автомобіля може не помітити, що шина проколота, якщо використовується будь-яка система відмови від стійкості.

Проблема може бути вирішена шляхом додавання спеціальної системи для виявлення відмов (у разі шини система стежить за тиском у камерах і попереджає водія, якщо воно падає).

Альтернативою може бути призначення оглядів та перевірок для виявлення та попередження прихованих відмов та пошкоджень, наприклад огляд водієм шин на кожній зупинці автотранспорту.

Складнощі у контролі множинних відмов. Відмовостійкість одного елемента може заважати виявленню відмов в іншому. Наприклад, якщо частина B виконує якусь операцію на основі даних з частини A, то працююча частина може приховати проблему, що виникла в A. Якщо надалі частина буде замінена на менш надійну, то система може раптово відмовити, при цьому здаватиметься, що проблема полягає в новій частині B. І лише після ретельної перевірки системи стане ясно, що проблема була у частині A.

Підвищення ризиків ігнорування відомих відмов. Навіть якщо оператор знає про наявність відмови резервованого елемента системи відмови від стійкості, він може зволікати з його усуненням, так як система працює. Це призведе до повної відмови системи, коли відмовлять всі елементи надмірності.

Складність перевірки. Для деяких вкрай важливих стійких до відмови систем, таких як ядерний реактор, немає простого шляху, щоб переконатися, що резервовані елементи знаходяться в робочому стані.

Сумно відомим прикладом є Чорнобильська аварія, коли оператори перевіряли аварійну систему охолодження шляхом відключення основної та допоміжної систем. Аварійна система не витримала, що вилилося у перегрів реактора та великий викид радіації.

Зростання витрат. Заходи в галузі відмовостійкості збільшують вартість життєвого циклу системи внаслідок зростання витрат на розробку та випробування, зростання маси та матеріаломісткості, ціни системи, витрат на додаткове технічне обслуговування та ремонт та ін. Наприклад, пілотовані космічні кораблі мають більше резервованих систем та елементів, що збільшує їх вага в порівнянні з безпілотними апаратами, які не вимагають такого рівня безпеки.

Ризик застосування елементів низької якості. Відмовостійка архітектура може дозволити використання неякісних складових частин, які інакше зробили б систему непрацюючою. Хоча ця практика може використовуватися для обмеження зростання витрат, використання кількох таких частин може знизити надійність системи та спричинити зростання непланових витрат на стадії.

Ряд методів допомагає підвищити рівень стійкості до відмови системи і підтримувати безперервну роботу, незважаючи на вихід з ладу одного або декількох компонентів.

Резервування.

Відмовостійкість на основі резервування передбачає дублювання критично важливого обладнання, систем або компонентів для забезпечення захисту від збоїв.

Активне резервування використовує кілька одиниць обладнання, що працюють одночасно. У разі виходу з експлуатації однієї частини обладнання інша може компенсувати ситуацію.

Пасивне резервування передбачає використання резервних компонентів, які починають працювати лише у разі виходу з ладу основного устаткування. Ви зіткнетеся з двома різними типами пасивного резервування.

Оперативне пасивне резервування: воно включає обладнання, що знаходиться в режимі очікування в якості гарячого резерву. Резервний компонент може працювати в умовах холостого ходу або виконувати функції, відмінні від основного обладнання. Якщо основне обладнання виходить з ладу, його функції перебирає резервне устаткування.

Неробоче пасивне резервування: ці резервні компоненти відключаються до тих пір, поки їм не потрібно замінити основне обладнання. Резервні системи можуть включатись автоматично при виході з ладу основної системи або вимагати ручного втручання.

Реплікація.

Відмовостійкість на основі реплікації передбачає копіювання даних у декілька систем. Це сприяє безперервності роботи у разі збою вузла. Чим вищий ступінь реплікації, тим вища відмовостійкість системи.

Різноманітність.

Методи підвищення стійкості до відмов, пов’язані з різноманітністю, включають впровадження в систему нового обладнання, програмного забезпечення або мережевих компонентів, щоб зробити її більш стійкою. Наприклад, використання ряду постачальників обладнання та вибір програмного забезпечення, написаного кількома мовами, може покращити різноманітність та знизити ризик збою.

Балансування навантаження.

Метод балансування навантаження для підвищення стійкості до відмови передбачає розподіл робочих навантажень вашої організації по численних

частинах обладнання. Це дозволяє розподілити діяльність по альтернативним виробничим лініям підтримки функціональності у разі збою однією лінії.

Компоненти системи відмовостійкості

Розробка системи відмови від стійкості вимагає зусиль на кожному етапі життєвого циклу обладнання. Залежно від складності система може включати деякі або всі з наступних компонентів для виявлення і реагування на збої:

– виявлення та відображення несправностей;

– діагностика та локалізація несправностей;

– маскування та компенсація несправностей;

– виявлення та відображення несправностей.

Виявлення несправності відноситься до здатності системи виявляти несправність та попереджати відповідних осіб. Це фундаментальна особливість будь-якої відмовостійкої системи. Усі інші компоненти залежить від ефективності процесу виявлення несправностей.

Наприклад, датчик у системі контролю тиску в шинах (TPMS) може виявляти переповнені або недостатньо наповнені шини та оперативно попереджати водія через панель приладів автомобіля. Виявлення та відображення - це прийнятний рівень допуску для цього типу збою. Автоматичного виправлення ситуації немає.

Діагностика та локалізація несправностей.

Для більш складних систем та обладнання до проекту включені додаткові засоби захисту, що спричиняють заходи стримування. Наприклад, розподілена система управління (РСУ) контролює параметри за допомогою набору датчиків та виконує діагностику для виявлення та локалізації несправностей.

Уявіть датчик тиску в системі, яка виявляє нафтопродукти всередині резервуара, які знаходяться в небезпечній близькості від займання або вибуху. Ці датчики можуть активувати клапан тиску, щоб активувати та направити пару під високим тиском у вихлопну трубу.

Маскування та компенсація несправностей.

Маскування несправностей часто є корисним методом захисту обладнання, яке можна відслідковувати або контролювати за допомогою технології Інтернет речей (IoT). Атаки кібербезпеки становлять постійну загрозу для такого типу обладнання. Зловмисник може спробувати зробити помилку, впровадивши в систему помилкові дані.

Невірні дані можуть означати, що системи керування та моніторингу, призначені для захисту обладнання, дійсно почнуть викликати його збої. Альтернативно, помилкові дані можуть змусити систему повірити, що несправні активи перебувають у нормальному робочому стані. Згодом активи погіршуватимуться, не викликаючи відповідних попереджень (рис. 2.6).

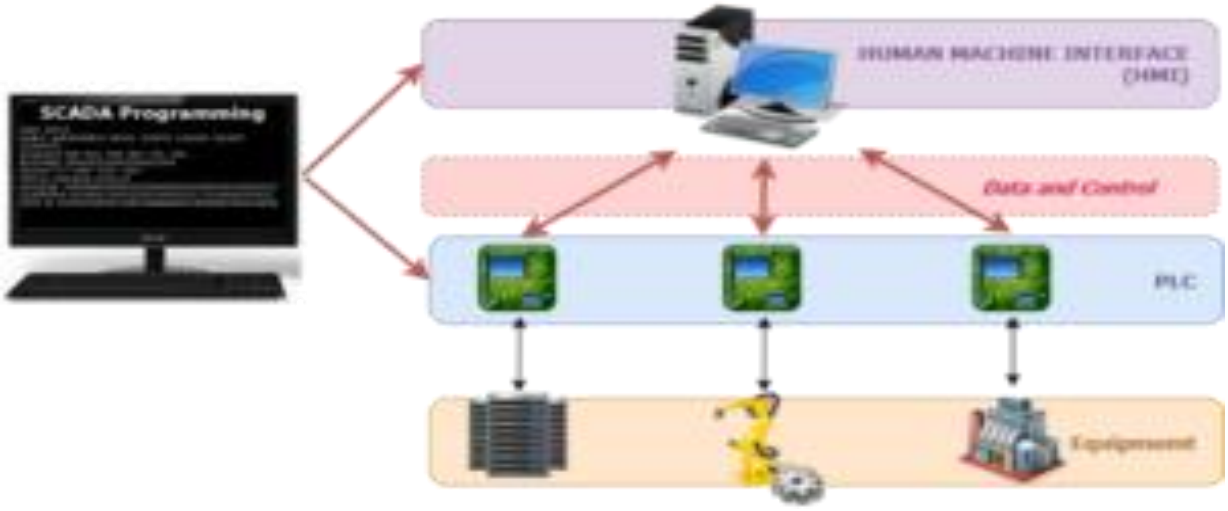

Рисунок 2.6 - Приклад системи SCADA

Маскування помилок гарантує, що системи будуть розпізнавати неправильну інформацію. Наприклад, автоматичні вимикачі в електромережах часто керуються та контролюються за допомогою системи диспетчерського керування та збору даних (SCADA).

Ці системи контролюють параметри напруги та частоти, щоб забезпечити стабільність усієї енергомережі. Якщо зловмисник спробує ввести помилкові дані, алгоритми системи дадуть відповідь введенням додаткових даних, які «маскують» зусилля зловмисника та забезпечують надійність мережі.

Kubernetes - також відомий як "k8s" або "kube" - це платформа оркестрування контейнерів для планування та автоматизації розгортання, управління та масштабування контейнерних додатків.

Kubernetes був вперше розроблений інженерами Google до того, як у 2014 році було запущено відкритий вихідний код. Він є нащадком Borg, платформа оркестрування контейнерів, що використовується всередині Google. Kubernetes по-грецьки означає "кермовий" або "пілот", звідси і "штурвал" в логотипі Kubernetes (посилання знаходиться за межами csc-ua) (рис. 2.7).

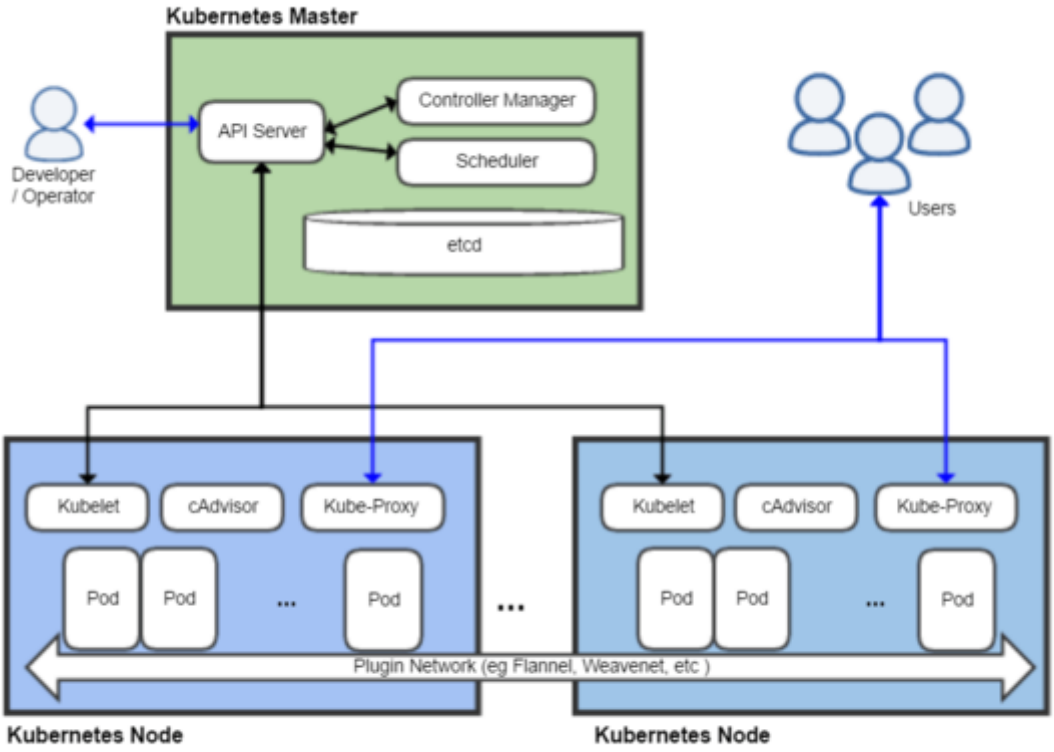

Рисунок 2.7 - Архітектура Kubernetes

Сьогодні Kubernetes і ширша екосистема контейнерів перетворюються на універсальну обчислювальну платформу та екосистему, яка конкурує, якщо не перевершує, віртуальні машини (ВМ) як основні будівельні блоки сучасної хмарної інфраструктури та додатків. Ця екосистема дозволяє організаціям надавати високопродуктивну платформу як послугу (PaaS), яка вирішує безліч пов’язаних з інфраструктурою та операціями завдань та проблем, пов’язаних з

хмарною розробкою, щоб групи розробників могли зосередитись виключно на кодуванні та інноваціях.

По мірі поширення контейнерів — сьогодні в організації їх можуть бути сотні чи тисячі — операційні групи мали планувати та автоматизувати розгортання контейнерів, створення мереж, масштабованість та доступність. Так народився ринок оркестрування контейнерів.

У той час як інші варіанти оркестрування контейнерів - в першу чергу Docker Swarm і Apache Mesos - отримали деяку підтримку на ранньому етапі, Kubernetes швидко став найпоширенішим (фактично, в якийсь момент це був найшвидший проект в історії програмного забезпечення з відкритим вихідним кодом).

Розробники вибрали (і продовжують вибирати) Kubernetes через його широту функціональності, його екосистему інструментів підтримки з відкритим вихідним кодом, що росте, а також його підтримки та переносимості між провідними хмарними провайдерами (деякі з яких тепер пропонують повністю керовані сервіси Kubernetes).

Kubernetes планує та автоматизує ці та інші завдання, пов’язані з контейнерами:

– розгортання: розгорніть вказану кількість контейнерів на вказаному хості та підтримуйте їх роботу в бажаному стані;

– впровадження: використання — це зміна розгортання. Kubernetes дозволяє запускати, зупиняти, відновлювати чи відкочувати розгортання;

– виявлення служби: Kubernetes може автоматично відкривати контейнер для Інтернету або інших контейнерів, використовуючи ім’я DNS або IP-адресу;

– надання сховища: налаштуйте Kubernetes для підключення постійного локального або хмарного сховища для контейнерів за необхідності;

– балансування та масштабування навантаження: коли трафік до контейнера різко зростає, Kubernetes може використовувати балансування та масштабування навантаження, щоб розподілити його через мережу для підтримки стабільності;

– самовідновлення для забезпечення високої доступності: при збої контейнера Kubernetes може перезапустити або замінити його автоматично; він також може розбирати контейнери, які не відповідають вашим вимогам щодо перевірки здоров'я.

Kubernetes надає безліч інструментів для інтеграції процесів проектування та розгортання ПЗ, що може працювати під управлінням даної системи.

Серед засобів, які найчастіше застосовуються в цих цілях, є такі:

- Helm – інструмент Kubernetes, функціональний аналог apt-get і yum;
- Minikube – спеціалізована програма додатку Kubernetes, яка призначається для розгортання на локальному ПК, і використовується для дослідження можливостей Kubernetes;
- Skaffold – утиліта від Google для локального моделювання у Kubernetes;
- Argo CD – утиліта для декларативного CI/CD, що засновується на Git Ops-підході;
- Kustomize – утиліта для показу yaml-маніфестів, які є вільними від шаблонів.

Для багатьох подібних засобів є інші варіанти, наприклад, замість Argo CD іноді застосовується Flux і werf (рис. 2.8).

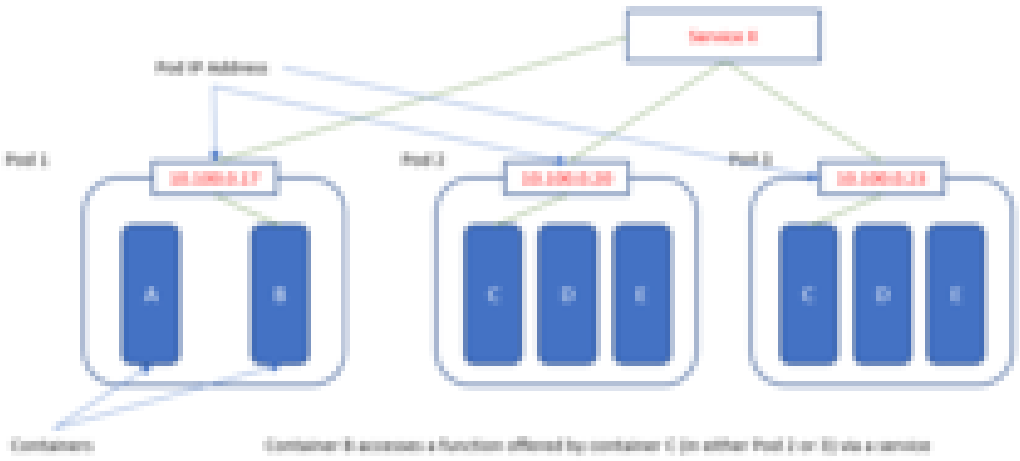

Рисунок 2.8 - Спрощене зображення того, як сервіс взаємодіє з мережею подів кластера Kubernetes

Отже, Kubernetes Service відкриває доступ до ПЗ, яке працює в кластерах за однією зовнішньою кінцевою точкою, навіть якщо робоче навантаження розподіляється між різними вузловими точками.

2.3 Розподілена обробка великих даних

Apache Spark – це розподілена система обробки данихзвідкритим вихіднимкодомдля обробки великих даних. Швидке виконання аналітичних запитів до даних будь-якого обсягу забезпечується завдяки кешування пам’яті та оптимізованого виконання запитів. Вона надає API для розробки мовами Java, Scala, Python і R, а також підтримує повторне використання коду для різних робочих навантажень - пакетної обробки, інтерактивних запитів, аналітики в реальному часі, машинного навчання та обробки графів. Її використовують організації з будь-якої галузі, у тому числі у FINRA, Yelp, Zillow, DataXU, Urban Institute та CrowdStrike.

Hadoop MapReduce – це модель програмування обробки великих наборів даних з допомогою паралельного розподіленого алгоритму. Розробники можуть писати багаторазово розпаралелені оператори, не турбуючись про розподіл роботи та стійкість до відмов.

Проте проблема MapReduce полягає у послідовному багатоетапному процесі виконання завдання.

На кожному кроці MapReduce зчитує дані з кластера, виконує операції та записує результати назад у HDFS.

Оскільки кожен крок вимагає читання та запису на диск, завдання MapReduce виконуються повільніше через затримку операції вводу-виводу на диск.

Spark був створений для усунення обмежень MapReduce за рахунок обробки в пам’яті, скорочення кількості кроків у завданні та повторного використання даних у кількох паралельних операціях.

При використанні Spark потрібно лише один крок для зчитування даних у пам’ять, виконання операцій та зворотний запис результатів, що значно прискорює виконання.

Spark також повторно використовує дані, застосовуючи кеш у пам’яті для значного прискорення алгоритмів машинного навчання, що багаторазово викликають функцію в тому самому наборі даних.

Повторне використання даних здійснюється шляхом створення DataFrames, абстракції порівняно зі стійким розподіленим набором даних (RDD), що є набором об’єктів, що кешуються в пам’яті і повторно використовуються в декількох операціях Spark.

Це значно знижує затримку, завдяки чому Spark у кілька разів швидше за MapReduce, особливо при машинному навчанні та інтерактивній аналітиці (рис. 2.9).

Незважаючи на відмінності в дизайні Spark і Hadoop MapReduce, багато організацій вважають ці інфраструктури великих даних взаємодоповнюючими та використовують їх разом для вирішення більших бізнес-задач.

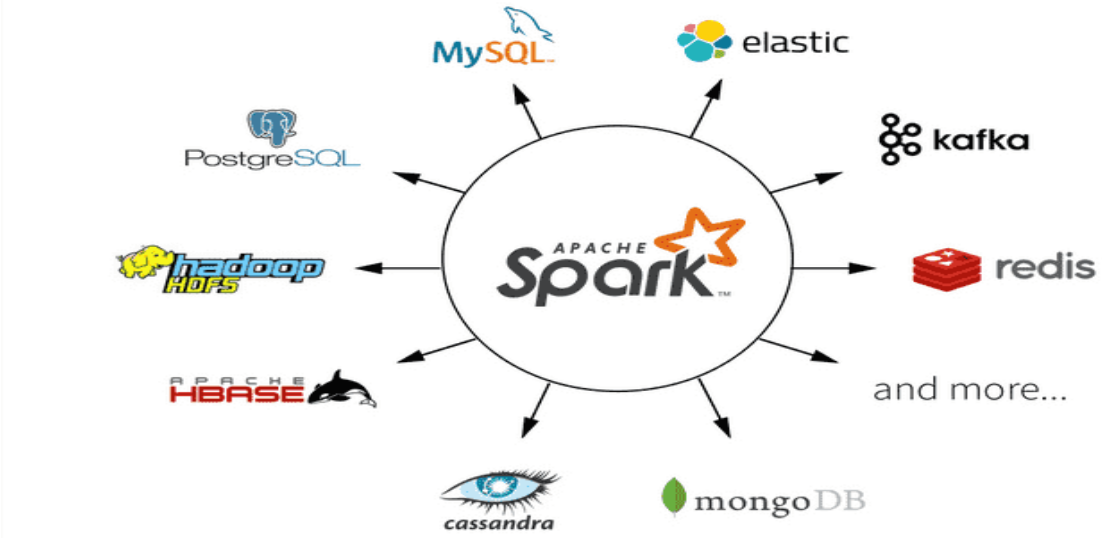

Рисунок 2.9 – Функціональна схема сумісності Apache Spark

Hadoop – це платформа з відкритим вихідним кодом, що використовує розподілену файлову систему Hadoop (HDFS) як сховище, YARN як спосіб управління обчислювальними ресурсами, використовуваними різними додатками, та реалізацію моделі програмування MapReduce як двигун виконання. У типовій реалізації Hadoop також використовуються різні механізми виконання, такі як Spark, Tez та Presto.

Spark – це платформа з відкритим вихідним кодом, орієнтована на інтерактивні запити, машинне навчання та робочі навантаження у реальному часі. Вона не має власної системи зберігання, але вона виконує аналітику в інших системах зберігання даних, таких як HDFS, або в інших популярних сховищах, таких як Amazon Redshift, Amazon S3, Couchbase, Cassandra та інші. Spark на Hadoop використовує YARN для спільного використання кластера та набору даних, як і інші двигуни Hadoop, забезпечуючи стабільний рівень обслуговування та відгуку.

Apache Spark має безліч переваг, які роблять його одним з найактивніших проектів в екосистемі Hadoop. Приклади наведені нижче.

Швидкість.

Spark може забезпечувати швидке виконання аналітичних запитів до даних будь-якого обсягу завдяки кешування в пам’яті та оптимізованого виконання запитів.

Зручність для розробників.

Apache Spark за замовчуванням підтримує Java, Scala, R та Python, що дозволяє вибирати мову для написання програм. Ці API полегшують роботу розробників, оскільки приховують складність розподіленої обробки за простими високорівневими операторами, що значно скорочує обсяг необхідного коду.

Підтримка кількох робочих навантажень.

Apache Spark підтримує кілька робочих навантажень, включаючи інтерактивні запити, аналітику в реальному часі, машинне навчання та обробку графів. Одна програма може легко комбінувати кілька робочих навантажень.

Платформа Spark включає:

- Spark Core як основа платформи;
- Spark SQL для інтерактивних запитів;
- Spark Streaming для аналітики у режимі реального часу;
- Spark MLlib для машинного навчання;
- Spark GraphX для обробки графів.

Spark Core є основою платформи - двигуном. Він відповідає за керування пам’яттю, усунення несправностей, планування, розподіл та моніторинг завдань, а також за взаємодію із системами зберігання даних. Spark Core доступний через інтерфейс прикладного програмування (API), створений Java, Scala, Python і R. Ці API приховують складність розподіленої обробки за простими високорівневими операторами.

2.4 Обробка потоків великих даних

Apache Impala є відкритим джерелом, масивно-паралельним обробленням (MPP) SQL Query Engine для кластера комп’ютера під керуванням Apache Hadoop. Impala описується як еквівалент Google F1 з відкритим кодом на виході, який надихнув його на розвиток у 2012 році.

Impala пропонує технологію масштабованої паралельної бази даних у Hadoop, забезпечуючи користувачам реалізацію системи запитів SQL із низькою затримкою до даних, що зберігаються в HDFS і Apache HBase, без необхідності переміщати або перетворювати дані. Impala є вбудованим у Hadoop і можна застосовувати той самий файл і формати даних, метадані, системи безпеки сфери та ресурсів, що застосовуються MapReduce, Apache Hive, Apache Pig та інше ПЗ Hadoop.

Impala призначена для аналітиків та фахівців з обробки даних, що дозволяють виконувати аналіз даних, що зберігаються в Hadoop, за допомогою SQL або інструментів бізнес-аналітики. В результаті великомасштабна обробка даних (через MapReduce) та інтерактивні запити можуть виконуватися в одній

системі з використанням одних і тих же даних та метаданих, що усуває необхідність перенесення наборів даних до спеціалізованих систем та/або власні формати просто для виконання аналізу.

Особливості використання Apache Impala включають в себе:

– підтримує сховища HDFS, S3, ABFS, Apache HBase та Apache Kudu;

– читає формати файлів Hadoop, включаючи текст, LZO, SequenceFile, Avro, RCFile, Parquet та ORC;

– підтримує безпеку Hadoop (автентифікація Kerberos, Ldap);

– деталізована авторизація на основі ролей за допомогою Apache Sentry та Apache ranger;

– Використовує метадані, драйвер ODBC та синтаксис SQL з Apache Hive.

Impala піднімає планку продуктивності SQL-запитів в Apache Hadoop, зберігаючи при цьому знайомий інтерфейс користувача. За допомогою Impala можна запитувати дані, що зберігаються в HDFS або Apache HBase, включаючи SELECT, JOIN і агрегатні функції, в режимі реального часу.

Більш того, Impala використовує ті ж метадані, синтаксис SQL (Hive SQL), драйвер ODBC і інтерфейс користувача (Hue Beeswax), що і Apache Hive, забезпечуючи знайому та уніфіковану платформу для пакетних запитів або запитів у реальному часі. (З цієї причини користувачі Hive можуть використовувати Impala з мінімальними витратами на налаштування).

Отже, Apache Impala - це популярний двигун MPP з відкритим вихідним кодом і широким спектром можливостей у Cloudera Distribution Hadoop (CDH) та CDP. Impala заслужила довіру ринку завдяки low-latency highly interactive SQL-запитам.

Можливості Impala дуже широкі, Impala не тільки підтримує Hadoop Distributed File System (HDFS – розподілену файлову систему Hadoop) із Parquet, Optimized Row Columnar (ORC – оптимізований вузол зберігання), JavaScript Object Notation (JSON), Avro та текстові формати, але також має вбудовану підтримку Kudu, Microsoft Azure Data Lake Storage (ADLS) та Amazon Simple Storage Service (S3).

Impala має високий рівень безпеки за допомогою either sentry або ranger і, як відомо, може підтримувати тисячі користувачів із кластерами із сотень вузлів на багатопетабайтних датасетах.

Давайте розглянемо загальну архітектуру Impala (рис. 2.10).

Для перевірки працездатності кластера Impala використовує StateStore. Якщо вузол Impala з якоїсь причини переходить у режим "оффлайн", StateStore передасть повідомлення про це по всіх вузлах і пропустить недоступний вузол. Служба каталогу Impala керує метаданими для всіх інструкцій SQL всіх вузлів кластера.

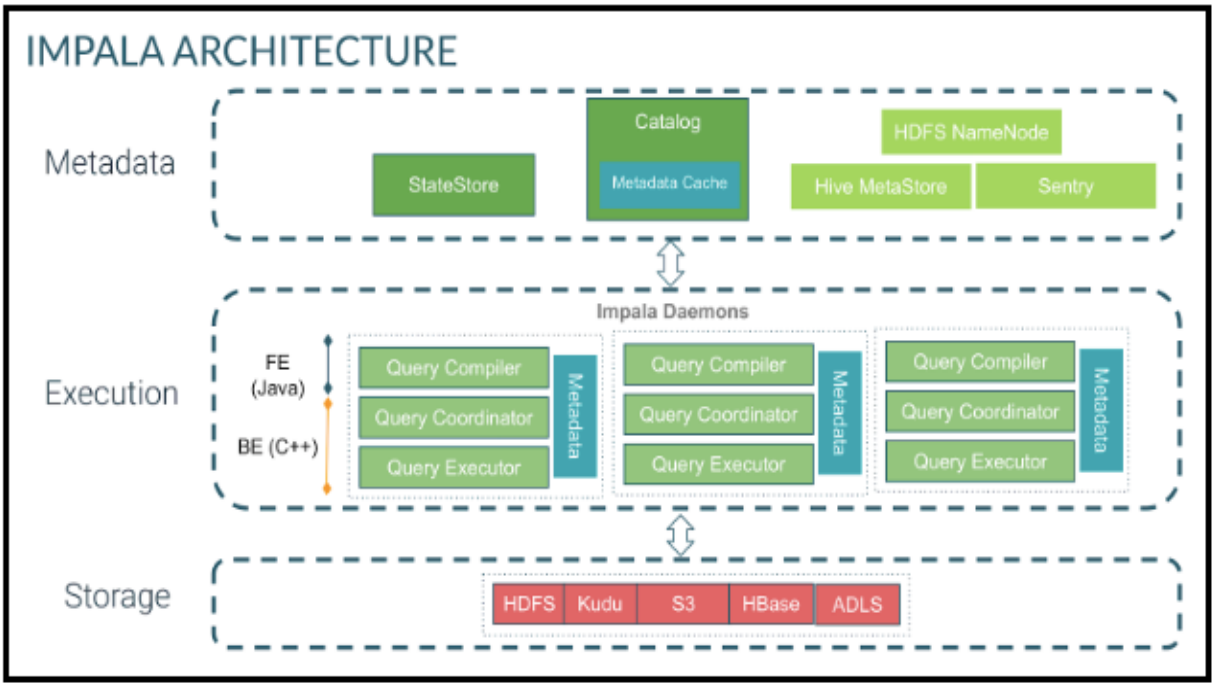

Рисунок 2.10 – Архітектура Apache Impala

StateStore та служба каталогів обмінюються даними зі сховищем Hive MetaStore для розміщення блоків та файлів, а потім передають метадані робочим вузлам. При надходженні запиту він передається одному з численних програм погодження, де виконується компіляція та ініціюється планування. Фрагменти плану повертаються і програма узгодження організує його виконання. Проміжні результати передаються між службами Impala, а потім повертаються.

Така архітектура ідеально підходить для випадків, коли нам потрібні вітрини даних для бізнес-аналітики для отримання відповідей на запити з низьким

часом затримки, як це зазвичай буває у випадках з використанням ad-hoc, self-service і discovery types.

За такого сценарію ми маємо клієнтів, які повідомляють нам відповіді на складні запити від менше однієї секунди до п’яти секунд.

Для даних Internet of Things (IoT) та пов’язаних з ними сценаріїв, Impala разом зі streaming рішеннями, такими як NiFi, Kafka або Spark Streaming, та відповідними сховищами даних, такими як Kudu, може забезпечити безперервну конвеєрну обробку з часом затримки менш ніж десять секунд.

Завдяки вбудованим функціям читання/запису на S3, ADLS, HDFS, Hive, HBase та багато інших, Impala є чудовим SQL-движком для використання при запуску кластера до 1000 вузлів, і більше 100 трильйонів рядків у таблицях або датасетах розміром 50BP і більше.

Щоб уникнути затримок, Impala обходить MapReduce для прямого доступу до даних через спеціалізований механізм розподілених запитів, який дуже схожий на ті, що використовуються у комерційних паралельних СКБД. В результаті продуктивність на порядок вища, ніж у Hive, залежно від типу запиту та конфігурації (рис. 2.11).

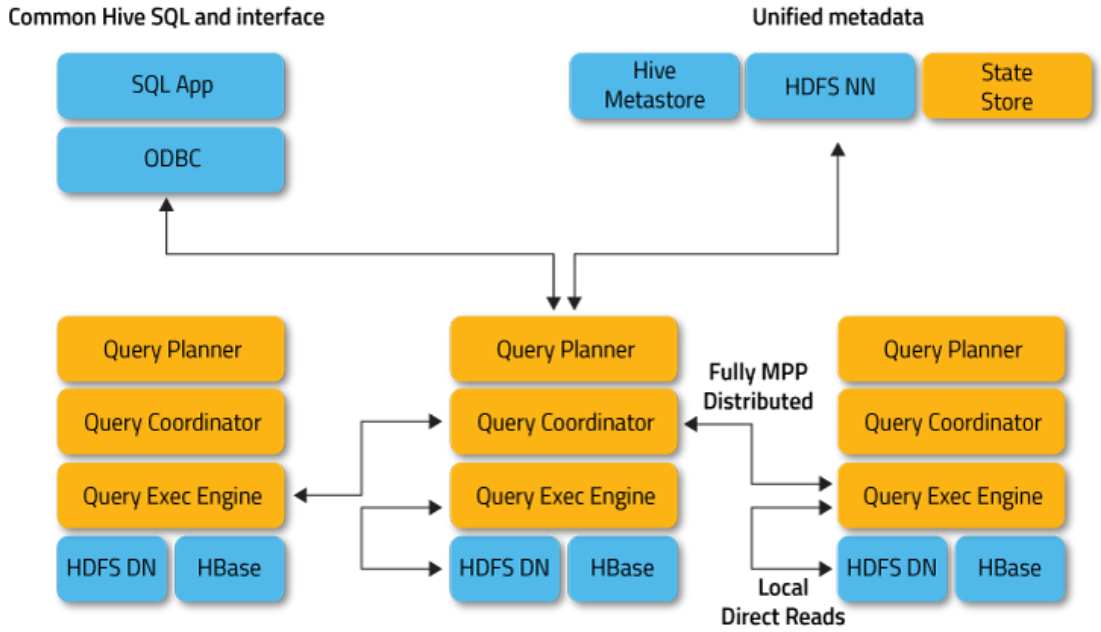

Рисунок 2.11 – Схема запитів Apache Impala

Цей підхід має багато переваг порівняно з альтернативними підходами до запиту даних Hadoop, у тому числі:

– завдяки локальній обробці на вузлах даних можна уникнути вузьких місць у мережі;

– можна використовувати єдине відкрите та уніфіковане сховище метаданих;

– дороге перетворення формату даних не потрібно і, отже, немає жодних накладних витрат;

– всі дані доступні для запиту негайно без затримок для ETL;

– все обладнання використовується для запитів Impala, а також MapReduce;

– для масштабування потрібен лише один пул машин.

Підсумовуючи результати аналізу, можна констатувати таке:

1. Проаналізовані способи вирішення задачі з розробки системи архітектури для системи оброблення Big Data на базі технології Apache Impala.

2. Досліджено способи проектування архітектури у вигляді засобів для розгортання складних систем, обробляння потоків Big Data і засобів для розподіленого обробляння Big Data.

3 РЕАЛІЗАЦІЯ СИСТЕМИ З ВИКОРИСТАННЯМ РОЗРОБЛЕНОЇ АРХІТЕКТУРИ

3.1 Розробка архітектури

Архітектура для обробки великих даних дозволяє приймати, обробляти та аналізувати дані, які є надто об’ємними або надто складними для традиційних систем баз даних.

Рішення для обробки великих даних зазвичай призначені для одного або кількох з наступних типів робочого навантаження:

– пакетне оброблення джерел неактивних великих даних;
– обробка великих даних у динаміці як реального часу;
– інтерактивне вивчення великих даних;
– прогнозна аналітика та машинне навчання (рис. 3.1).

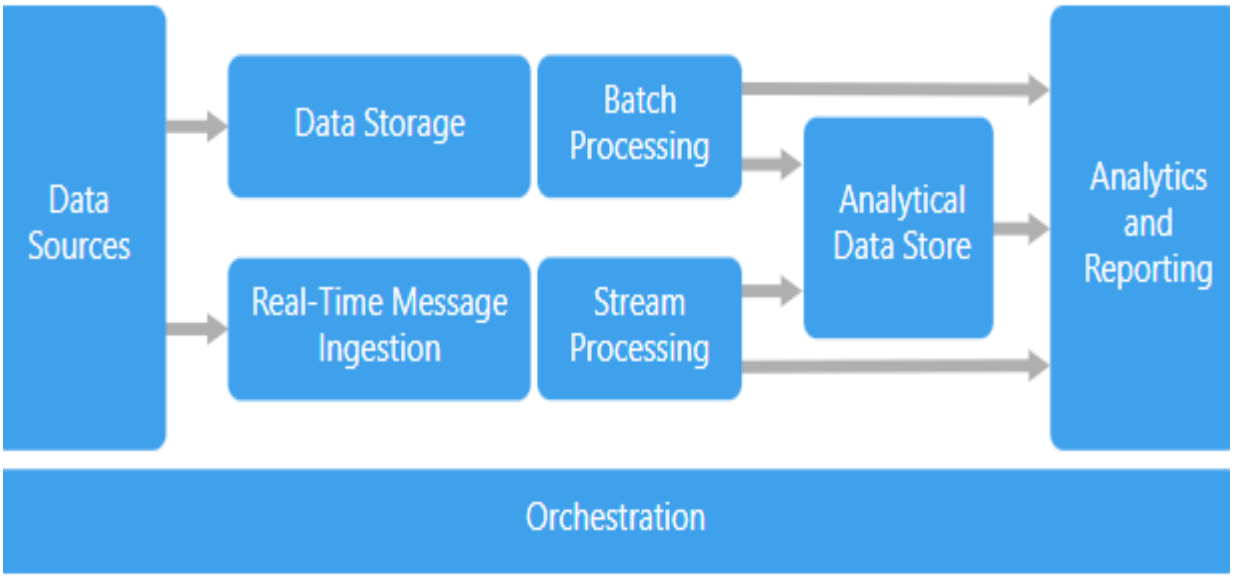

Рисунок 3.1 – Приклад класичної архітектури обробки Big Data

Більшість архітектур для обробки великих даних включають деякі або всі наведені нижче компоненти:

– джерела даних. Всі рішення для обробки великих даних починаються з одного або кількох джерел даних. Ось деякі приклади;

– сховища даних додатків, наприклад, реляційні бази даних;

– статичні файли, які створюються програмами, наприклад, файли журналу веб-сервера;

– джерела даних із передачею в режимі реального часу, наприклад, пристрої Інтернету речей;

– сховище даних. Дані пакетної обробки зазвичай зберігаються в розподіленому сховищі файлів, де можуть міститися значні обсяги великих файлів у різних форматах. Цей тип сховищ часто називають озером даних. Таке сховище можна реалізувати за допомогою Azure Data Lake Store або контейнерів великих двійкових об’єктів у службі сховища Azure;

– пакетне оброблення. Оскільки набори даних дуже великі, часто у вирішенні обробляються пакетні завдання. Їх виконується фільтрація, статистична обробка та інші процеси підготовки даних до аналізу.

Зазвичай ці завдання входить читання вихідних файлів, їх обробка і запис вихідних даних у нові файли. Варіанти: виконання завдань U-SQL в Azure Data Lake Analytics, використання завдань користувача Hive, Pig або Map/Reduce в кластері HDInsight Hadoop і застосування програм Java, Scala або Python в кластері HDInsight Spark.

Отримання повідомлень у режимі реального часу. Якщо рішення містить джерела в режимі реального часу, в архітектурі має бути передбачений спосіб збирання та збереження повідомлень у режимі реального часу для потокової обробки. Це може бути просте сховище даних з папкою, в яку вхідні повідомлення розміщуються для обробки.

Але для прийому повідомлень багатьом рішенням потрібне сховище, яке можна використовувати як буфер. Таке сховище має підтримувати обробку з горизонтальним масштабуванням, надійну доставку та іншу семантику черги повідомлень. Варіанти: Центри подій Azure, Центри Інтернету речей та Kafka.

Поточний обмін повідомленнями. Після запису повідомлень у режимі реального часу у вирішенні потрібно виконати їхню фільтрацію, статистичну обробку та інші процеси підготовки даних до аналізу. Потім оброблені потокові дані записуються у вихідний приймач.

Azure Stream Analytics надає керовану службу потокової обробки на основі постійного виконання запитів SQL для неприв'язаних потоків. Ви також можете використовувати відкритий код технології потокової передачі Apache, такі як потокова передача Spark в кластері HD Insight.

Сховище аналітичних даних. У багатьох рішеннях обробки великих даних дані готуються до аналізу. Потім оброблені дані структуруються відповідно до формату запитів для засобів аналітики.

Сховище аналітичних даних, яке використовується для обробки таких запитів, може бути реляційною базою даних типу Kimball, як можна побачити в більшості традиційних рішень бізнес-аналітики (BI).

Крім того, дані можна представити за допомогою технології NoSQL з низькою затримкою, як-от HBase або інтерактивна база даних Hive, яка надає абстракцію метаданих для файлів даних у розподіленому сховищі.

Azure Synapse Analytics – це керована служба для зберігання великих обсягів даних у хмарі. HDInsight підтримує Interactive Hive, HBase і Spark SQL, які також можна використовувати для надання даних для аналізу.

Аналіз та створення звітів. Більшість рішень для обробки великих даних дозволяють отримати уявлення про дані за допомогою аналізу та звітів. Щоб розширити можливості аналізу даних, можна включити в архітектуру шар моделювання, наприклад, модель таблиці або багатовимірного куба OLAP в Azure Analysis Services.

Також можна включити підтримку самостійної бізнес-аналітики з використанням технологій моделювання та візуалізації Microsoft Power BI або Microsoft Excel. Аналіз та створення звітів також може виконуватися шляхом інтерактивного вивчення даних фахівцями з їхнього аналізу та обробки.

Для таких сценаріїв багато служб Azure підтримують функції аналітичного блокнота, наприклад Jupyter, який дозволяє користувачам застосовувати свої навички роботи з Python або R. Для великомасштабного вивчення даних можна використовувати Microsoft R Server (окремо або зі Spark).

Оркестрація. Більшість рішень для обробки великих даних складаються з повторюваних робочих процесів, під час яких перетворюються вихідні дані, дані переміщаються між декількома джерелами і приймачами, оброблені дані завантажуються в сховища аналітичних даних або результати передаються безпосередньо у звіт або на панель моніторингу.

3.2 Розробка підсистеми обробки потоків даних

Щоб автоматизувати ці робочі процеси, можна використовувати технологію оркестрації, таку як фабрика даних Azure або Apache Oozie і Sqoop.

Azure пропонує багато служб, які можна використовувати в архітектурі для обробки великих даних. Їх можна умовно поділити на дві категорії:

Керовані служби, включаючи Azure Data Lake Store, Azure Data Lake Analytics, Azure Synapse Analytics, Azure Stream Analytics, Центри подій Azure, Центр Інтернету речей Azure та Фабрика даних Azure.

Технології з відкритим кодом на основі платформи Apache Hadoop, включаючи HDFS, HBase, Hive, Spark, Oozie, Sqoop та Kafka. Ці технології доступні в Azure у службі Azure HDInsight (рис. 3.2).

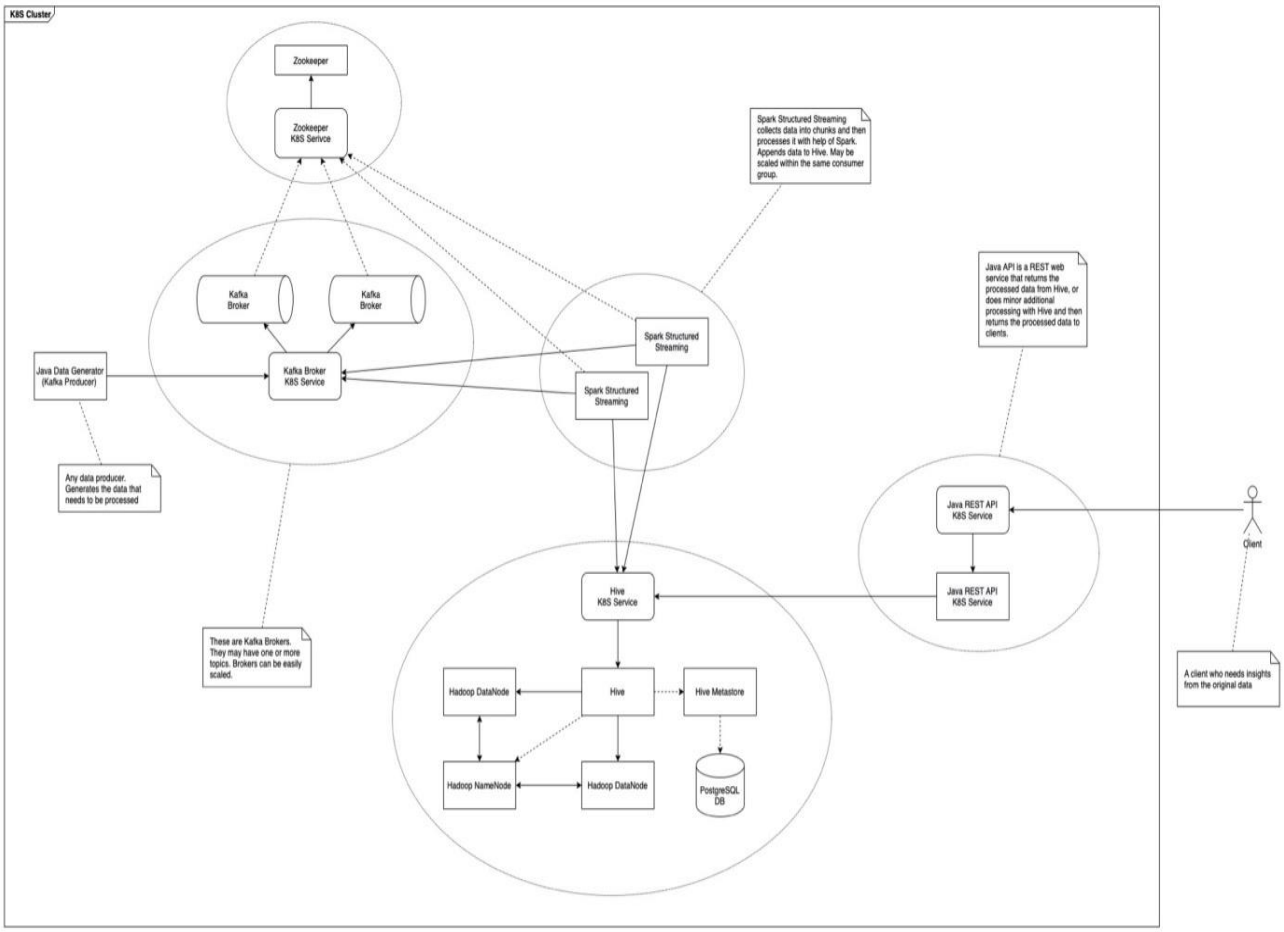

Рисунок 3.2 – Архітектурна схема відмовостійкої системи обробки Big Data.

Використовуйте цю архітектуру для наступних сценаріїв:

- зберігання та обробка даних в обсягах, надто великих для традиційної бази даних;
- перетворення неструктурованих даних для аналізу та створення звітів;
- запис, обробка та аналіз неприв’язаних потоків даних у режимі реального часу або з низькою затримкою;

Використовуйте машине навчання Azure або Azure Cognitive Services.

На рис. 3.3 представимо розроблену підсистему обробки Big Data.

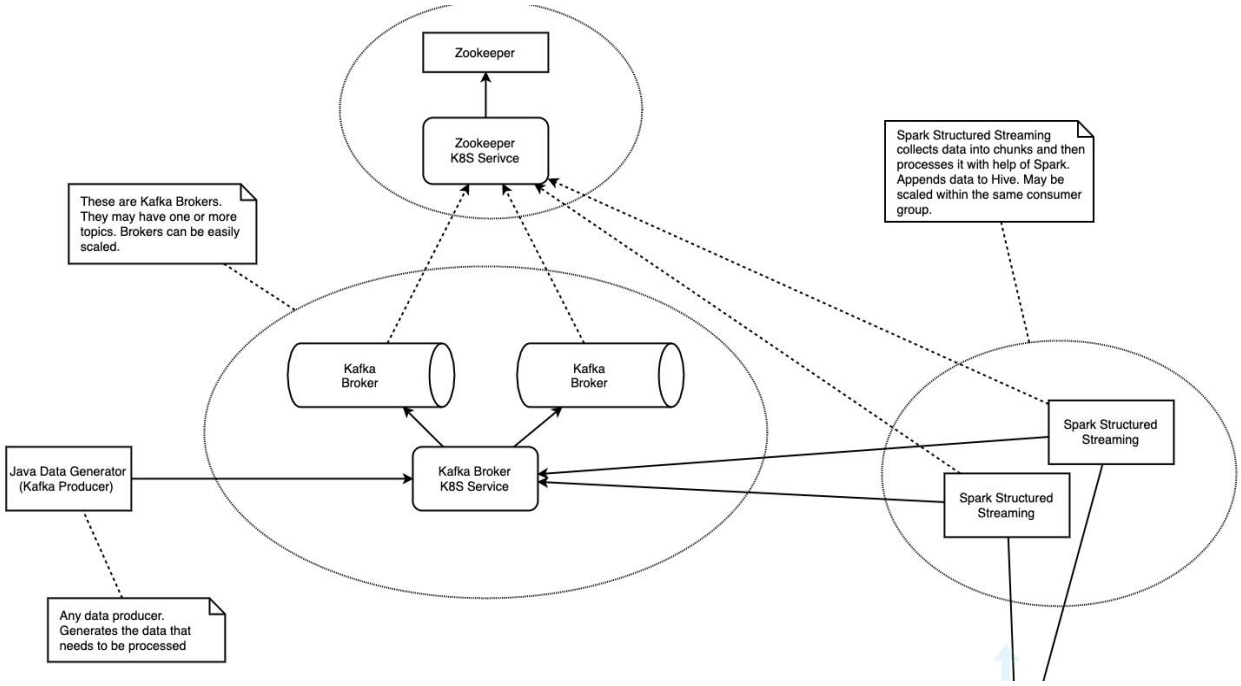

Рисунок 3.3 - Підсистема обробки Big Data

3.3 Розробка підсистеми зберігання та надання доступу до даних

За відмовостійку генерацію і оброблення масиву Big Data в розробленій системі відповідає підсистема оброблення потоків Big Data (рис. 3.4).

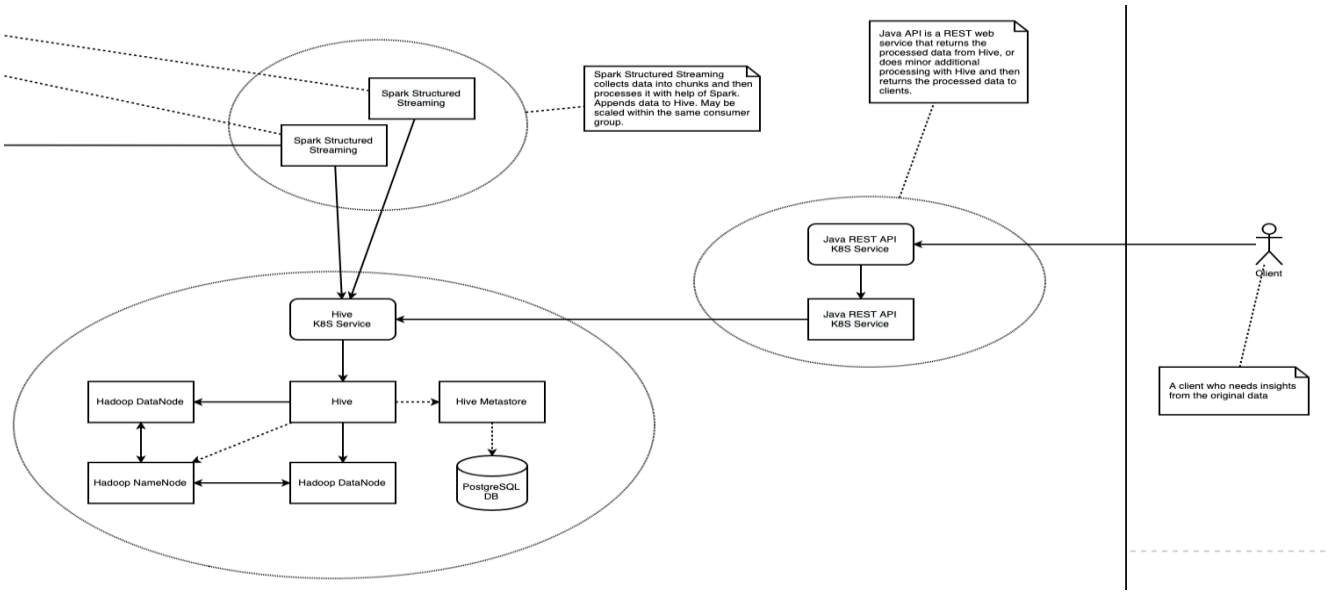

Рисунок 3.4 – Схема зберігання та надання доступу до Big Data

Hive застосовує Hadoop та Hadoop FileSystem для зберігання та обробки Big Data. Повну архітектуру Hive представимо на рис. 3.5.

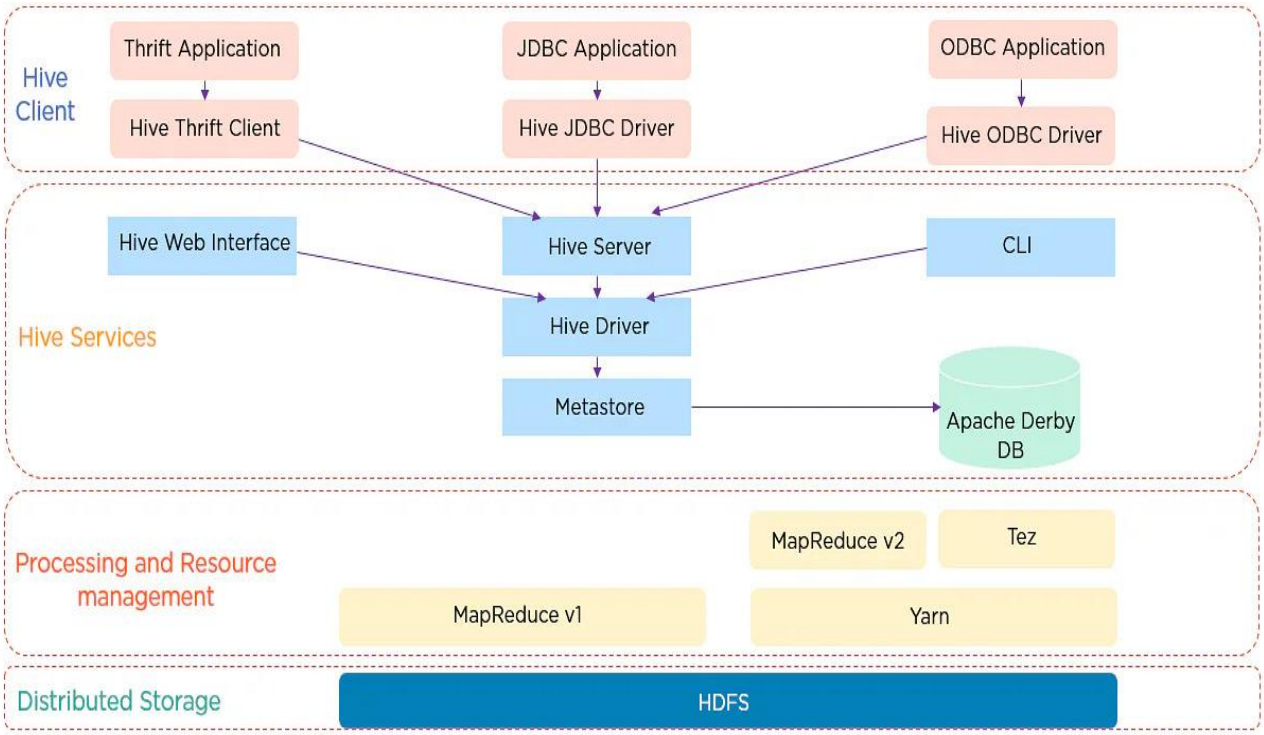

Рисунок 3.5 – Приклад повної архітектури Hive

Переваги:

Можливість вибору технологій. Можна комбінувати та зіставляти керовані служби Azure та технології Apache у кластерах HD Insight, щоб з максимальною вигодою застосовувати існуючі навички та інвестувати у технології.

Підвищення продуктивності з допомогою паралелізму. У рішеннях обробки великих даних використовується перевага паралелізму, що дозволяє застосовувати високопродуктивні рішення, які можуть масштабуватися до роботи з великими обсягами даних.

Еластичне масштабування. Всі компоненти архітектури для обробки великих даних підтримують горизонтальне масштабування, щоб ви могли адаптувати рішення для малих і великих робочих навантажень і платити лише за ресурси, які використовуєте.

Взаємодія з рішеннями. Компоненти архітектури для обробки великих даних також використовуються у відповідних рішеннях Інтернету речей та корпоративних рішеннях бізнес-аналітики, що дозволяє створювати інтегровані засоби для робочих навантажень обробки даних.

3.4 Розгортання системи

Система розгорнута на платформі Kubernetes. Використовуючи Kubernetes, всі машини в кластері розглядаються як єдиний пул ресурсів. Kubernetes виконує важливу функцію розподіленої операційної системи, ефективно керуючи плануванням та розподілом ресурсів, моніторингом справності інфраструктури та навіть забезпеченням бажаного стану інфраструктури та робочих навантажень. Крім того, Kubernetes може працювати в різних кластерах та інфраструктурах у хмарних сервісах і приватних центрах обробки даних, що робить його потужною операційною системою для запуску сучасних програм.

Як і інші зрілі розподілені системи, Kubernetes має два рівні: майстер-вузли та робочі вузли. Головний вузол зазвичай керує площиною управління, яка відповідає за планування робочого навантаження та управління життєвим циклом. Робочі вузли діють як робочі конячки, які виконують додатки. Сукупність головних і робочих вузлів є кластером.

Існує кілька варіантів розробки та локального розгортання: docker-compose та minikube.

Обидва варіанти були розглянуті при проектуванні системи; варіант розгортання docker-compose був швидко розроблений, оскільки він вимагає менше ресурсів для запуску і є більш простим варіантом розгортання.

Після запуску скрипту система розгортається на платформі Kubernetes і всі її функції стають доступними. Екран інформаційної панелі minikube з переліком сервісів показано на рис. 3.6.

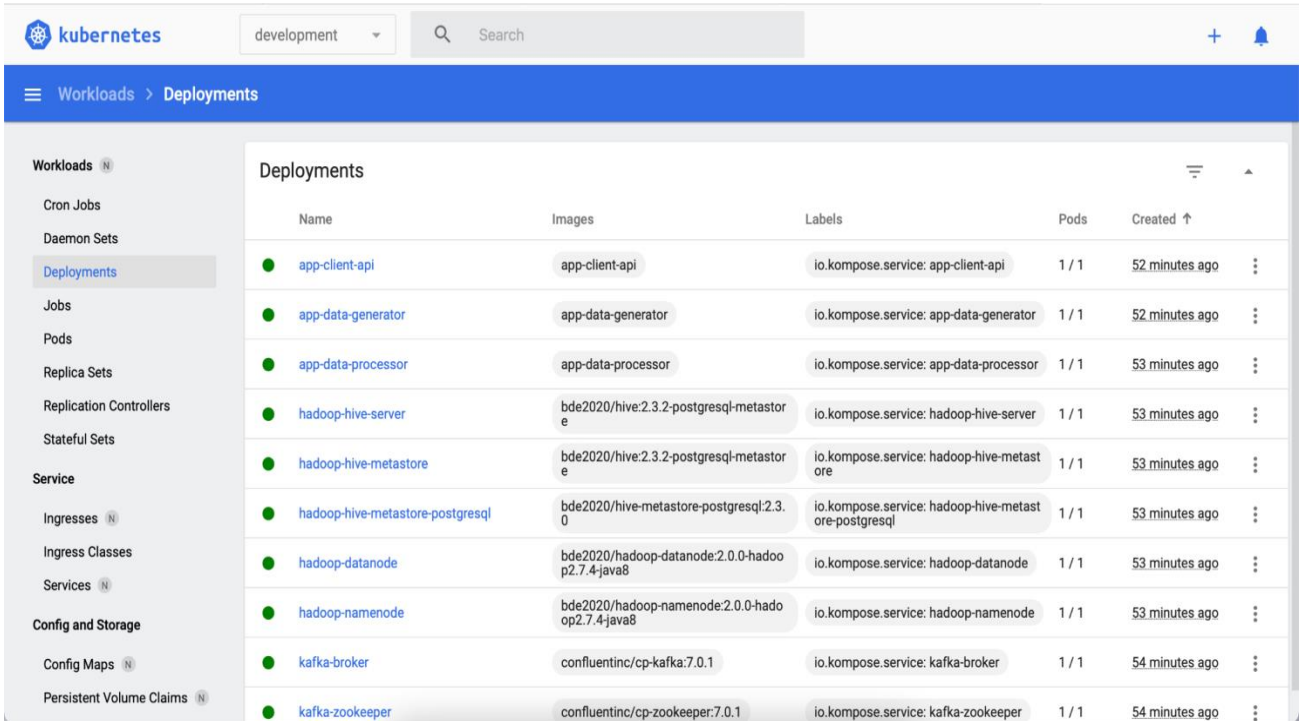

Рисунок 3.6 – Список сервісів розгорнутих на Kubernetes

В панелі керування можна побачити статус сервісу та кількість реплік (актуальних і запитаних).

Після налаштування система починає працювати: дані генеруються, обробляються та зберігаються в HDFS.

Функціональність системи можна перевірити, звернувшись до відкритого порту кластера. Цей порт надає доступ до сервісів користувацького інтерфейсу, які ми запитували.

На рис. 3.7 запити на обслуговування інтерфейсу користувача та відповіді на них можна побачити як агреговані дані у форматі JSON.

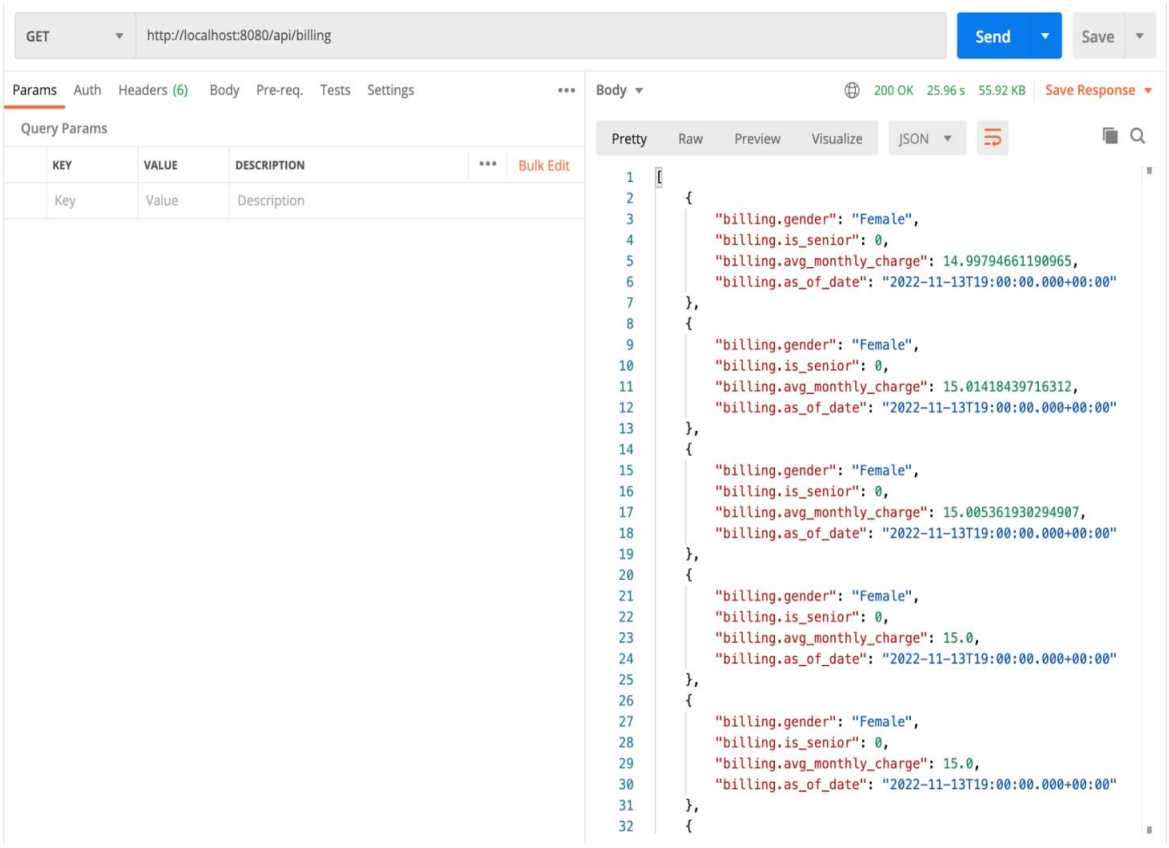

Рисунок 3.7 - Приклад взаємодії між клієнтом і кластером

3.5 Відмовостійкість та продуктивність розробленої архітектури

Як зазначалося раніше, розроблена архітектура спрямована на забезпечення максимальної відмовостійкості при обробці великих обсягів даних. Тому доцільно порівняти розроблену архітектуру, а точніше систему, побудовану за розробленою архітектурою, з базовою системою, яка забезпечує відмовостійкість лише за рахунок розгортання сервісів та копій сховища, як показано на рис. 3.8.

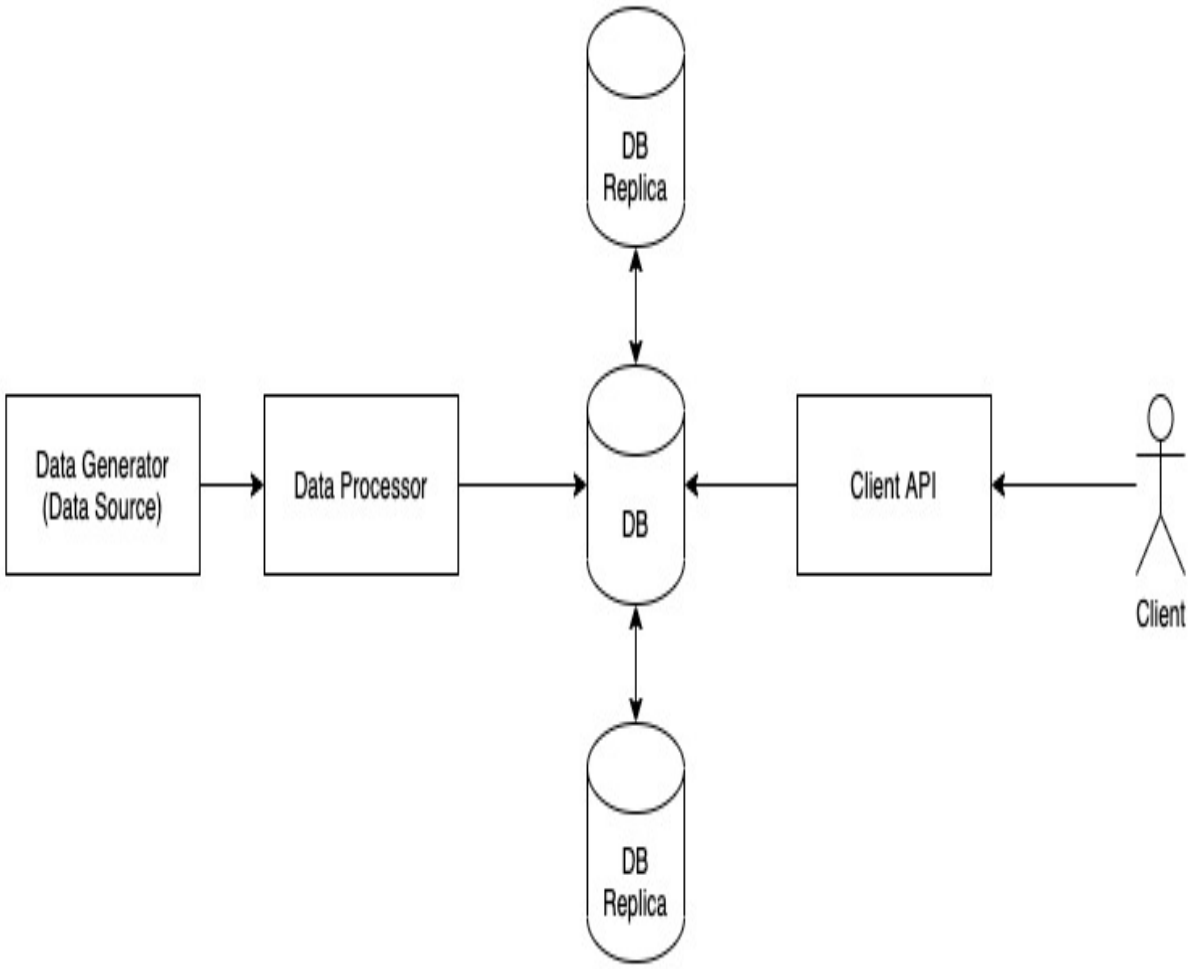

Рисунок 3.8 - Базова архітектура, побудована за допомогою реплікації елементів, відмовостійкість яких треба реалізувати

Базова архітектура, побудована за допомогою реплікації елементів, має на меті реалізацію відмовостійкості. Це означає, що система буде продовжувати працювати навіть у випадку відмови окремих компонентів. Реплікація дозволяє створити копії елементів системи, що забезпечує збереження функціональності та доступності даних навіть при виникненні проблем з окремими компонентами.

Для тестування продуктивності було використано набір даних з більш ніж 1 800 000 записів.

Першим тестом було порівняння аналітичних запитів, результати якого виявилися наступними на рис. 3.9 та 3.10.

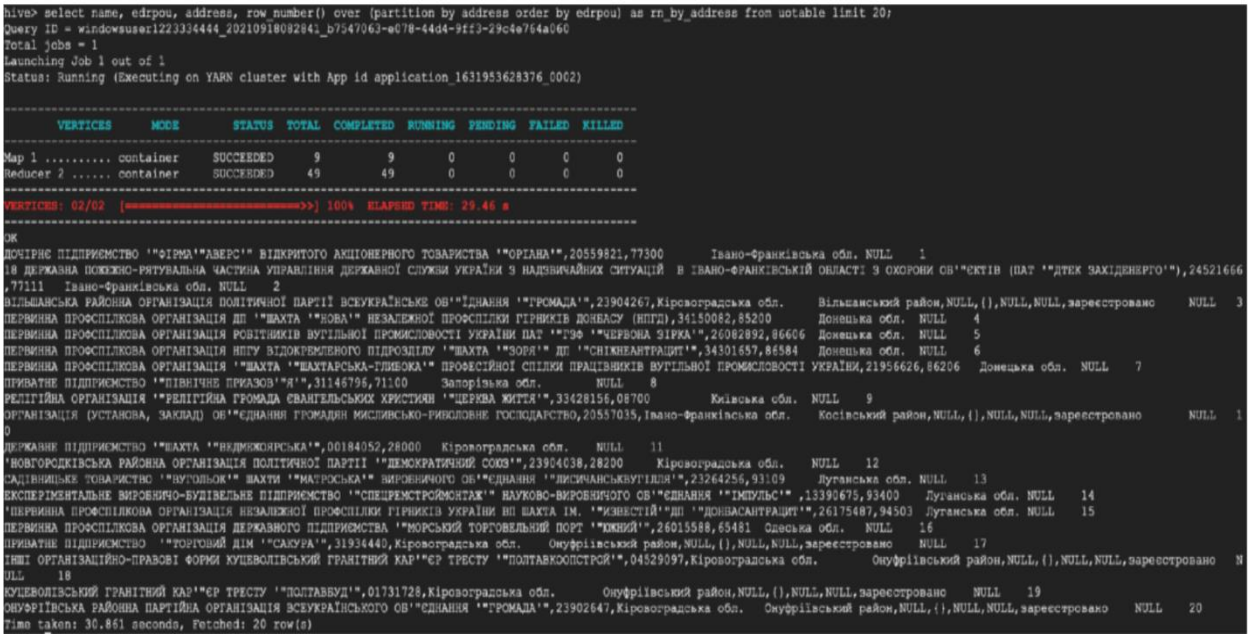

Рисунок 3.9 - Аналітичний запит Hive

Рисунок 3.10 - Аналітичний запит Postgre SQL

Час виконання аналітичного запиту з використанням Hive становить 30,861 секунди порівняно з 42,926 секундами для аналогічного запиту в Postgre SQL. Це означає, що для набору даних в 1,8 мільйона запитів Hive є швидшим для базових операцій аналізу.

Для більших наборів даних PostgreSQL потребує дуже багато часу на обробку, іноді кілька годин. З іншого боку, у Hive таке збільшення не призводить до дуже помітного збільшення часу виконання запитів.

У наступному порівнянні продуктивності порівнюється час виконання запиту з оператором JOIN, тобто об'єднанням таблиць. Результати порівняння можна побачити на рисунках 3.11, 3.12.

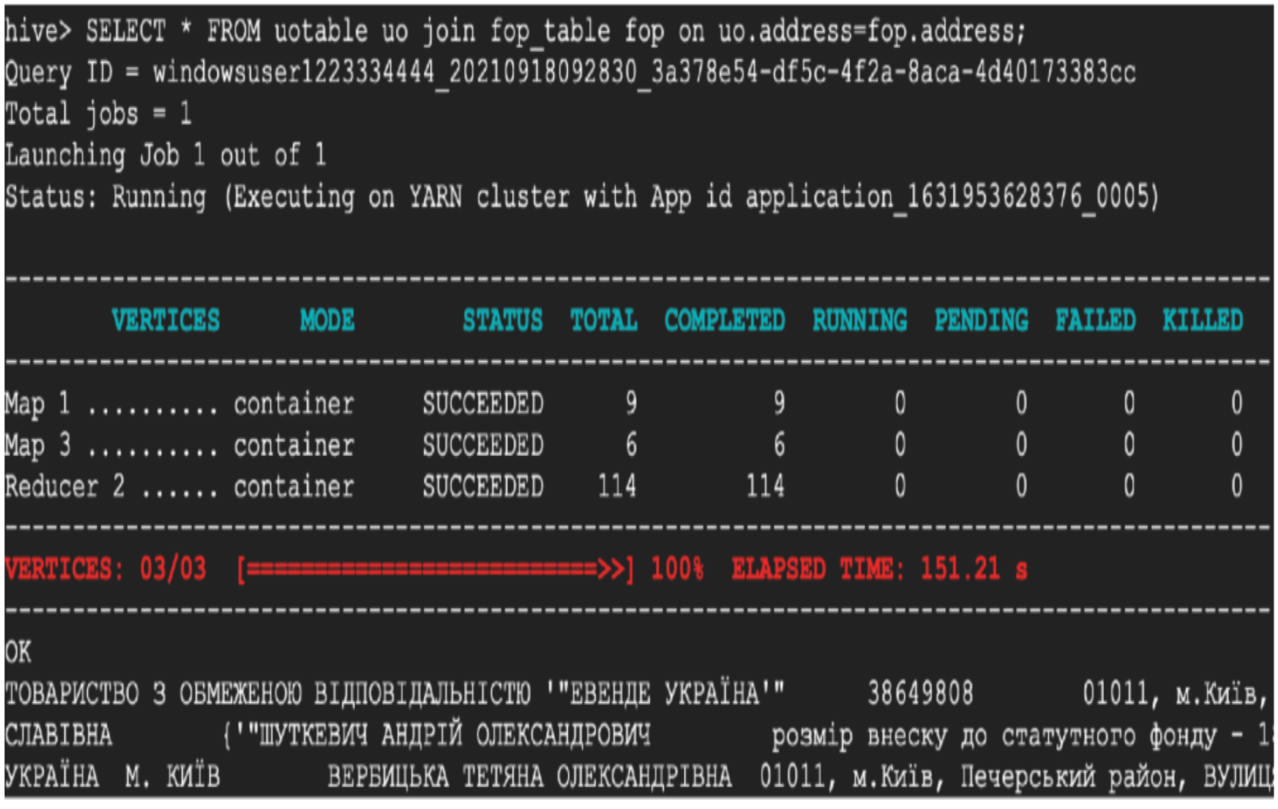

Рисунок 3.11 - Запит з JOIN Hive

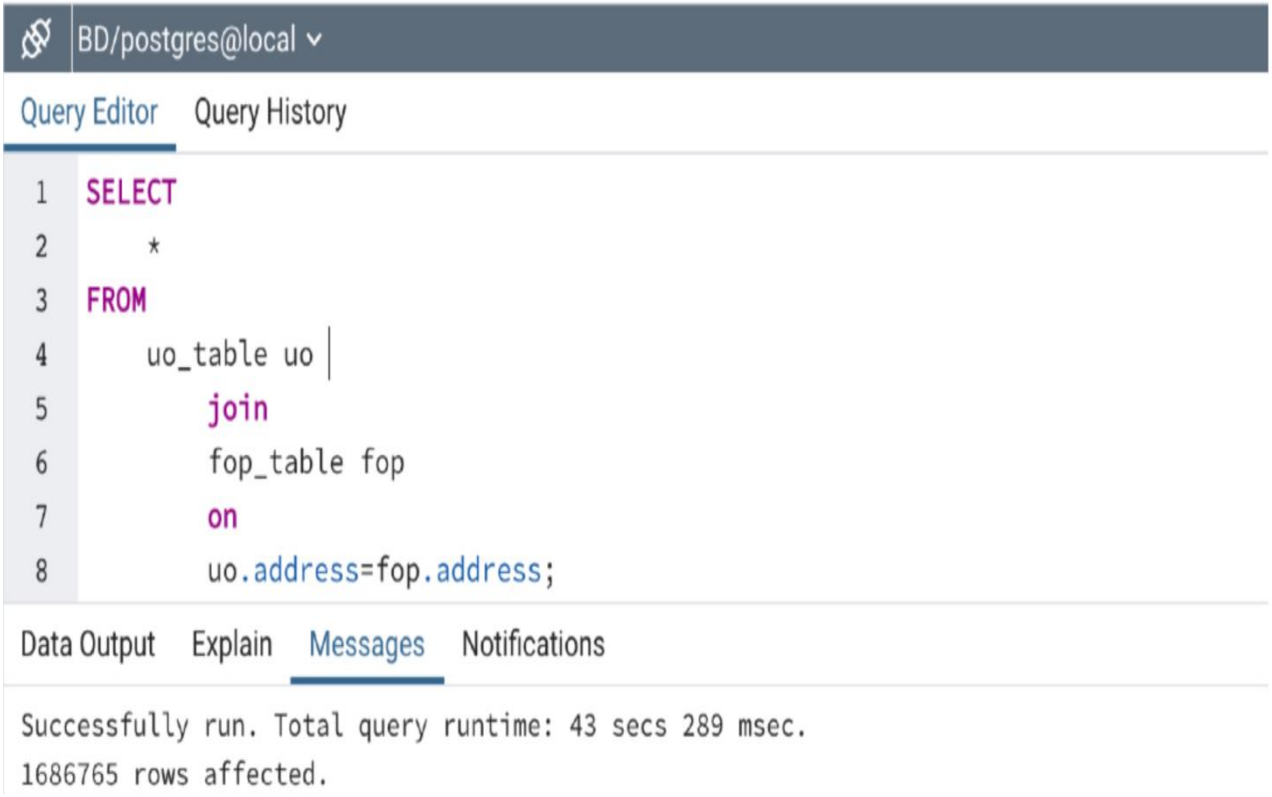

Рисунок 3.12 - Запит з JOIN Postgre SQL

Запит на об'єднання таблиць в Hive зайняв 30 861 секунду, в той час як в Postgre SQL - 43 289 секунд. Це показує, що використання Hive для великих обсягів даних є більш ефективним, ніж використання стандартної бази даних.

Складнощі.

Складність. Рішення обробки великих даних можуть бути дуже складними і містити безліч компонентів для прийому даних з декількох джерел. Створення, тестування та усунення несправностей процесів обробки великих даних може стати непростим завданням.

Понад те, у кількох системах може бути велика кількість параметрів конфігурації для оптимізації продуктивності.

Набір навичок. Багато технологій для обробки великих даних є вузькоспеціалізованими. Вони використовуються платформи і мови, які є стандартними більш загальних архітектур додатків. З іншого боку, технології обробки великих даних сприяють розвитку нових інтерфейсів API з урахуванням більш традиційних мов.

Наприклад, мова U-SQL в Azure Data Lake Analytics побудована на комбінації Transact-SQL і C#. Аналогічно, API на основі SQL доступні для Hive, HBase та Spark.

Зрілість технологій. Багато технологій, що використовуються для обробки великих даних, знаходяться в розвитку. Основні технології Hadoop, наприклад Hive та Pig, вже сформовані. Але нові технології, такі як Spark, з кожним випуском вносяться значні зміни і вдосконалення.

Керовані служби, такі як Azure Data Lake Analytics та фабрика даних Azure, є відносно молодими в порівнянні з іншими службами Azure і, швидше за все, з часом змінюватимуться.

Безпека. У рішеннях обробки великих даних всі статичні дані зазвичай зберігаються у централізованому озері даних. Захист доступу до цих даних — це непросте завдання, особливо якщо дані повинні прийматися та використовуватися кількома програмами та платформами.

Рекомендації.

Використання паралелізму. У більшості технологій обробки великих даних робоче навантаження розподіляється між кількома одиницями обробки. Тому статичні файли даних створюються та зберігаються у форматі, доступному для розбивки.

Розподілені файлові системи, такі як HDFS, можуть забезпечити оптимізацію продуктивності читання та запису. У цьому фактична обробка паралельно виконується кількох вузлах кластера. Це скорочує загальний час виконання завдань.

Секціонування даних. Пакетна обробка зазвичай виконується в регулярному розкладі, наприклад, щотижня або щомісяця. Секціоновані файли та структури даних, такі як таблиці, ґрунтуються на темпоральних періодах, які відповідають розкладу обробки. Це спрощує прийом даних, планування завдань та усунення помилок. Крім того, таблиці розділів, які використовуються у запитах Hive, U-SQL або SQL, можуть значно підвищити їхню продуктивність.

Застосування семантики схеми під час зчитування. Використання озера даних дозволяє об’єднувати сховище для файлів у декількох форматах, чи то структуроване, чи напівструктуроване, чи неструктуроване. Використовуйте семантику схеми під час зчитування, яка проектує схему на дані при обробці, а не під час зберігання.

Це підвищує гнучкість рішення та запобігає утворенню вузьких місць під час прийому даних в результаті перевірки даних та типів.

Обробка даних дома. У традиційних рішеннях бізнес-аналітики для переміщення даних до сховища часто використовується процес вилучення, перетворення та завантаження (ETL).

Для великих обсягів даних і різноманітних форматів у рішеннях обробки великих даних зазвичай використовуються різні варіації ETL, наприклад перетворення, вилучення і завантаження (TEL). За такого підходу дані обробляються в розподіленому сховищі даних. Вони перетворюються на необхідну структуру перед переміщенням до сховища аналітичних даних.

Регулює витрати при тарифікації на основі обсягу та часу використання. Для завдань пакетної обробки дуже важливо враховувати два фактори: витрати на одиницю обчислювальних вузлів та щохвилинна вартість використання цих вузлів для виконання завдання. Наприклад, виконання пакетного завдання може тривати вісім годин під час використання чотирьох вузлів кластера.

Але може виявитись, що всі чотири вузли використовуються для завдання лише протягом перших двох годин, а після цього достатньо двох вузлів.

У такому разі виконання всього завдання на двох вузлах збільшить загальний час, але не подвоїть його. Тому сукупна вартість буде меншою. У деяких бізнес-сценаріях більш тривалий час обробки може бути кращим для використання ресурсів кластера, що не використовуються.

Поділ кластерних ресурсів. При розгортанні кластерів HDInsight зазвичай можна підвищити продуктивність, підготувавши окремі кластерні ресурси кожного типу робочого навантаження. Наприклад, кластери Spark включають Hive, але при масштабній обробці з використанням Hive та Spark рекомендуємо

розгорнути окремі кластери Spark і Hadoop. Аналогічно, при використанні HBase та Storm для потокової обробки з низькою затримкою та Hive для пакетної обробки рекомендуємо розгорнути окремі кластери для Storm, HBase та Hadoop.

Оркестрація прийому даних. Іноді існуючі бізнес-програми можуть записувати файли даних для пакетної обробки безпосередньо в контейнери великих двійкових об’єктів у сховищі Azure, де вони можуть використовуватись службами HDInsight або Azure Data Lake Analytics.

Тим не менш, часто потрібно виконувати оркестрацію прийому даних з локального або зовнішнього джерела в озері даних. Найбільш прогнозований та централізовано керований підхід для цього – робочий процес або конвеєр оркестрації, наприклад, підтримуваний фабрикою даних Azure або Oozie.

Очищає конфіденційні дані на ранній стадії. При прийомі даних необхідно очищати конфіденційні дані на ранній стадії, щоб не зберігалися в озері даних.

ВИСНОВКИ

1. Визначено, що великі дані впершу чергу відносяться до наборів даних, які надто великі або складні для обробки за допомогою традиційного програмного забезпечення для обробки даних. Дані з великою кількістю записів (рядків) мають більшу статистичну силу, а дані з більш високою складністю (більше атрибутів або стовпців) можуть призвести до більш високого рівня помилкових виявлень. Хоча інтерпретація, яка іноді використовується вільно, частково через відсутність формального визначення, здається, що найкраще описує великі дані, яка пов’язана з великим обсягом інформації, яку ми не можемо зрозуміти, якщо використовувати тільки в менших кількостях.

2. З’ясовано, що проблеми аналізу Big Data включають збір даних, зберігання даних, аналіз даних, пошук, спільне використання, передачу, візуалізацію, запити, оновлення, конфіденційність інформації та джерело даних.

3. Проаналізовано, що основна ідея до проектування інфраструктури високої доступності заключається в процесі розгортання надлишкових даних та таких, які мають в складі інші пристрої, комплектуючі та програмні засоби від різноманітних постачальників. Така ідея пов’язується з набагато більшими витратами, однак не забезпечує очікуваного етапу надійності через слабку імплементацію елементів, техобмежень і складності адміністративного керування. Крім того, безліч фірм проходять через процес зростання власної інфраструктури розумних технологій для росту ефективності власної справи, а тому об’єднання БД є важливою складовою даного процесу.

4. Проаналізовано, що на відміну від традиційного оброблювача з ядра Apache Hadoop, що представляє дворівневу концепцію MapReduce зі сховищем на диску, може використовувати спеціальні примітиви для рекурентного оброблення в оперативній пам’яті, завдяки чому можна виграти у швидкості роботи для деяких класів завдань. Apache Hive – це SQL інтерфейс доступу до даних для платформи Apache Hadoop. Хайв дозволяє виконувати запити, агрегувати та досліджувати дані із використанням SQL-синтаксису.

5. Розглянуто, що Apache Impala пропонує технологію масштабованої паралельної БД у Hadoop, забезпечуючи користувачам виконання запитів SQL із низькою затримкою до даних, що зберігаються в HDFS і Apache H Base, без необхідності переміщати або перетворювати дані. Impala є вбудованим у Hadoop і дозволяє вам використовувати той самий файл і формати даних, метадані, системи безпеки сфери та ресурсів, що використовуються Map Reduce, Apache Hive, Apache Pig та інше програмне забезпечення Hadoop. Для уникнення затримок, Impala обходить Map Reduce для прямого доступу до даних через спеціалізований механізм розподілених запитів, який дуже схожий на ті, що використовуються у комерційних паралельних СКБД. В результаті продуктивність на порядок вища, ніж у Hive, залежно від типу запиту та конфігурації

6. В результаті розроблено покращену архітектуру відмовостійких систем обробки Big Data на базі технології Apache Impala. Реалізована система для обробки даних, побудована із застосуванням розробленої покращеної архітектури. Описано всі елементи системи та обґрунтовано їх застосування, із врахуванням специфіки. Також була проаналізована відмовостійкість систем, створених із застосуванням покращеної архітектури на базі технології Apache Impala. Було виконане порівняння продуктивності систем за рахунок порівняння продуктивності найменш швидких компонентів цих систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бабичєв, С. Г. Основи сучасної криптографії [Текст] / С. Г. Бабичєв. - М. : ДІАЛОГ-МІФІ, 2011. – С. 35-39.

2. Береза А. М., Основи створення інформаційних систем. Навчальний посібник. 2-ге видання. – К.: КНЕУ, 2001р. – 156 с.

3. Бурячок В. Л., Толюпа С. В., Аносов О. О., Козачок В. О., Лукова-Чуйко Н. В. Системний аналіз і прийняття рішень в інформаційній безпеці: учебник. / В. Л. Бурячок,С. В. Толюпа, О. О. Аносов, В. О. Козачок, Н. В. Лукова-Чуйко / – К.:ДУТ, 2015. – 345 с.

4. Бурячок В. Л. Інформаційний та кіберпростір: проблеми безпеки, методи і способи боротьби. [Підручник]. / В. Л. Бурячок, Г. М. Гулак, В. Б. Толубко. – К. : ТОВ «СІК ГРУП УКРАЇНА», 2015. – 449 с.

5. Вороновський Г. К., Махотило К. В., Петрашев С. Н., Сергєєв С. А. Генетичні алгоритми, штучні нейроні мережі і проблеми віртуальної реальності. - Замовне. - Х.: ОСНОВА, 1997. - С. 112.

6. Глушков В. М., Амосов М. М., Артеменко І. А. Енциклопедія кібернетики. Том 2. Київ, - 1974. – С. 33-54.

7. Про Основні засади розвитку інформаційного суспільства [Електронний ресурс]. - [S. l. : s. n.]. – Режим доступу: http://zakon4.rada.gov.ua/laws/ show/537-16.

8. Растригін Л. А., Ейдук Я. Ю. Адаптивнні методи багатокритеріальної оптимизації // Автоматика і телемеханіка. 1985. - № 1. - С. 5-26.

9. Сорокін С. М. Метод виявлення атак типу «відмова в обслуговуванні» на WEB-додатку / С. М. Сорокін // Прикладна дискретна математика. - 2014. - № 1(23). - С. 55 - 64.

10. Скулиш Є. Д. Засоби аналізу та оцінка ризиків інформаційної безпеки / Є. Д. Скулиш, О. Г. Корченко, Ю. І. Горбенко, С. В. Казмирчук // Інформаційна безпека. Людина, суспільство, держава – 2011. – №3 (7). – С. 31-48.

11. Татарчук М.І. Корпоративні інформаційні системи: Навч. посібник.– К.: КНЕУ, 2005. 291 с.

12. Тарасов О. В., Федько В. В. Клієнт-серверні технології СУБД Oracle. Мова SQL Oracle. Навчальний посібник для самостійної підготовки студентів з навчальної дисципліни «Організація баз даних та знань». Харків: Вид. ХНЕУ ім. С. Кузнеця, 2015. – 384 с.

13. Тархов Д. А. Нейронні мережі. Моделі і алгоритми. – М.: Радіотехника, 2010. – С. 65-70.

14. Тимчук М.І. Особливості використання Oracle GoldenGate для розробки відмовостійкої архітектури баз даних. Інформаційні моделі, системи та технології: Праці VIII наук.-техн. конф. (Тернопіль, 09-10 грудня 2020 р.) Тернопіль, 2020. – С. 121.

15. Типове положення про службу захисту інформації в автоматизованій системі [Текст] : НД ТЗІ 1.4-001. - 2000. - Чин. 2000.12.04. - К. : ДСТСЗІ СБ України, 2000. - С. 4-30.

16. Уосермен Ф. Нейрокомп’ютерна техніка: Теорія і практика. Переклад І. Ю. Юрчак, 2001. – С. 88-94.

17. Федько В. В. Організація баз даних та знань : навч.-прак. посібн. / В. В. Федько, О. В. Тарасов, М. Ю. Лосєв. – Х.: Вид. ХНЕУ, 2013. – 200 с.

18. Ясницький Л. Н. Введення в штучний інтелект. - 1-е. – Видавничий центр «Академія», 2005. - С. 170-176.

19. Augspurger T. Dask for Machine Learning. Dask. URL: https://examples.dask.org/machine–learning.html.

20. Brownlee J. Why Use Ensemble Learning?. Machine Learning Mastery. URL: https://machinelearningmastery.com/why–use–ensemble–learning.

21. Dask is a flexible library for parallel computing in Python. Dask. URL: https://docs.dask.org/en/latest.

22. Daubechies I. The wavelet transform, time-frequency localization and signal analysis / I. Daubechies // IEEE Transactions Information Theory. - 1990. - Vol. 36. - P. 961–1005.

23. Image and Video Coding – Emerging Standards and Beyond. // IEEE Transactions on Circuits and Systems for Video Technology. 1997. V. 8. № 7. P. 814–837.

24. Hank Tullis, Paul Elbow, Chaitanya R. Geddam. Beginning Oracle GoldenGate. Apress. 2015. - 400 p.

25. Vettigli G. Cambridgespark. From simple regression to multiple regression with decision trees. URL: https://info.cambridgespark.com/latest/from–simple–regression–to–multiple–regression–with–decision–trees.

26. Kurzweil R. 15 Big Data Problems You Need to Solve. SolveXia. URL: https://www.solvexia.com/blog/15–big–data–problems–you–need–to–solve.

27. Loeffler C., Ligtenberg A., Moschytz G. Practical Fast 1-D DCT Algorithms with 11 Multiplications. Proc. Int'l. Conf. on Acoustics, Speech, and Signal Processing 1989 (ICASSP '89). – 991p.

28. Machhi C. PySpark – SparkContext. tutorialspoint. URL: https://www.tutorialspoint.com/pyspark/pyspark_sparkcontext.htm

29. Mallat S. Understanding image transform codes / S. Mallat, F. Falzon // Proc. SPIE Aerospace Conf. – 1997. – P. 56-62.

30. Salamon J. A Dataset and Taxonomy for Urban Sound Research / J. Salamon, C. Jacoby, J. Bello. // 22nd ACM International Conference on Multimedia,Orlando USA. – 2014 – P.13-19.

31. Singh A. Ultimate guide to handle Big Datasets for Machine Learning using Dask (in Python). Analytics Vidhya. URL: https://www.analyticsvidhya.com /blog/2018/08/dask–big–datasets– machine_learning–python.

32. Sen J. A Robust Mechanism for Defending Distributed Denial OF Service Attacks on Web Servers / J. Sen // International Journal of Network Security & Its Applications (IJNSA). - 2011, March. - Vol. 3, N 2. - P. 162–179.

33. RIEDEL M. Big Data – Definition, Importance, Examples & Tools. Research Data Alliance. URL: https://www.rd–alliance.org/group/big–data–ig–data–development–ig/wiki/big–data–definition–importance–examples–tools.

34. Tsvetovat M. top 5 problems with big data – and how to solve them. Piesync. URL: https://www.piesync.com/blog/top–5–problems–with–big–data–and–how–to–solve–them/

35. Liu, H., Tang, B., Zhang, J., Deng, Y., Zheng, X., Shen, Q., Yan, X., Zeng, D., Mao, Z., Zhang, C. and You, Z., 2022, June. GHive: A Demonstration of GPU-Accelerated Query Processing in Apache Hive. In Proceedings of the 2022 International Conference on Management of Data (pp. 2417-2420).