ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення автоматизованих систем

Пояснювальна записка

до кваліфікаційної роботи на
ступінь вищої освіти магістр

на тему: «**ДОСЛІДЖЕННЯ МОДЕЛЕЙ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ОБРОБКИ ВЕЛИКИХ ДАНИХ**»

Виконав: студент 6 курсу, групи ІСДМ-61
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
Ніколаєнко О.М.
(прізвище та ініціали)
Керівник Полоневич О.В.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення автоматизованих систем
Ступінь вищої освіти - «Магістр»
Спеціальність підготовки - 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗАС
_____ К.П.Сторчак
“____” _____2021 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Ніколаєнко Олексію Миколайовичу
(прізвище, ім’я, по батькові)

1.Тема роботи: «Дослідження моделей архітектури інформаційних систем для обробки великих даних»
Керівник роботи: Полоневич Ольга Володимирівна, к.т.н. доцент кафедри ІПЗАС ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “___” _____року №___.
2. Строк подання студентом роботи 26 грудня 2021 року
3. Вихідні дані до роботи :
3.1. Науково-технічна література з теми магістерської роботи
3.2. Серія стандартів ISO/IEC 20547
3.3. Принципи побудови архітектур BIG DATA

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
4.1. Аналіз сукупності технології BIG DATA
4.2. Дослідити та здійснити аналіз по сукупності технологій та фреймворків обробки даних в Big Data на теперішній час
4.3. Дослідити та здійснити оцінку по основним моделям архітектур інформаційних систем для обробки великих даних.
4.4. Розробка моделі архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання

5. Перелік графічного матеріалу
1. Основні концепції технології Big Data.
2. Проблеми та перспективи технології Big Data.

3. Аспекти технології та фреймворків для роботи з Big Data.
4. Різновид моделей архітектури інформаційних систем для обробки Big Data.
5. Розробка моделей архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання.
6. Етапи виявлення ознак особи на конвеєрі машинного навчання з використанням моделей наскрізного виявлення об'єктів на базі Spark та BigDL.

6. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури		
2	Вивчення матеріалів для аналізу основних положень концепції технології Big Data		
3	Дослідження та здійснення аналізу по сукупності технологій та фреймворків обробки даних в Big Data на теперішній час		
4	Дослідження та здійснення оцінки по основним моделям архітектур інформаційних систем для обробки великих даних		
5	Розробка моделі архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання		
6	Вступ, висновки, реферат		
7	Розробка обов'язкових демонстраційних матеріалів		
8	Попередній захист роботи		

Студент _____ Ніколаєнко О.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Полоневич О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи 67 с., 1 табл., 49 рис., 21 джерела.

BIG DATA, ФРЕЙМВОРКИ, АРХІТЕКТУРА, КОНЦЕПЦІЯ, ІНФОРМАЦІЙНІ СИСТЕМИ, ОБРОБКА ВЕЛИКИХ ДАНИХ, МАШИННЕ НАВЧАННЯ

Об’єкт дослідження – моделі архітектур інформаційних систем для обробки великих даних.

Предмет дослідження – розробка архітектури великих даних для розпізнавання осіб з використанням технології машинного навчання.

Мета роботи - здійснити аналіз сукупностей технологій та фреймворків обробки даних в Big Data.

Методи дослідження - методи теорії інформації, методи наукового моделювання, методи багатокритеріальної оптимізації, обробляння та аналіз отриманих результатів.

У роботі проведено аналіз по сукупності технологіям та фреймворкам обробки даних в Big Data на теперішній час. Досліджено основні концепції систем Big Data та різновид моделей архітектур інформаційних систем для обробки великих даних. Було виявлено, що серед фреймворків обробки великих даних немає кращого варіанту, так як у кожного є свої плюси і мінуси. Тому найкращий вибір, це використання гібридних рішень з різними інструментами.

Були розглянуті технології та фреймворки по роботі з Big Data, а також виділено основні аспекти по кожному з них, що в подальшому дозволить оптимально та технічно вибрати потрібний інструмент для вирішення поставленої задачі.

Була запропонована та розроблена модель архітектури великих даних для розпізнавання осіб з використанням технології машинного навчання.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACID	Набір властивостей, що гарантують надійну роботу транзакцій бази даних (Atomicity, Consistency, Isolation)
API	Прикладний програмний інтерфейс (Application Programming Interface)
CRUD	Основні функції управління даними (CREATE, READ, UPDATE, DELETE)
DAG	Випадок орієнтовного графу (Directed Acyclic Graph)
DNN	Глибока нейромережа (Deep Neural Network)
ETL	Процес, який використовується в базах даних та сховищах даних (Extract, Transform, Load)
HBASE	Розподілена база даних
HDFS	Розподілена файлова система (Hadoop Distributed File System)
IoT	Інтернет речей (Internet of Thimgs)
JSON	Текстовий формат обміну даними між комп’ютерами (JavaScript Object Notation)
KISS	Принцип проектування системи (Keep It Simple Stupid)
KNN	Метод К-найближчих сусідів (K-Nearest-Neighbors)
ML	Машинне навчання (Machine learning)
RDD	Проста, незмінна, розподілена колекція об'єктів у фреймворку Apache Spark (Resilent Distributed Dataset)
REST	Підхід до архітектури мережевих протоколів (Representational State Transfrer)
SQL	Мова структурованих запросів (Structured Query Language)
YARN	Модуль, що відповідає за управління ресурсами кластерів та планування завдань (Yet Antother Resourse Negotiator)

ВСТУП

Актуальність. Архітектура великих даних є основою для обробки та аналізу великих даних. Це загальна система, яка використовується для управління великими обсягами даних, щоб їх можна було аналізувати для бізнес-цілей, керувати аналітикою даних і забезпечити середовище, в якому інструменти аналізу великих даних можуть витягувати життєво важливу бізнес-інформацію з неструктурованих даних. Структура архітектури великих даних слугує еталонним планом для інфраструктури логічно визначаючи, як будуть працювати рішення, компоненти, які будуть використовуватися, як буде надходити інформація. Через велику кількість необроблених даних, що генеруються з різних джерел, включаючи соціальні мережі, великі дані стали важливими для отримання, обробки та аналізу різноманітних даних з безлічі джерел для додатків реального часу.

У цій магістерській дипломній роботі пропонується розробка архітектури великих даних, що підходить для попередньої обробки та класифікації зображень, а також текстової аналітики з використанням двох ключових робочих процесів, які називаються конвеєром великих даних і конвеєром машинного навчання. Дана архітектура поєднує у собі безперервні робочі процеси очищення, інтеграції, перетворення та скорочення даних, а також різноманітні аналітичні методи з використанням відповідних методів машинного навчання.

Об'єкт дослідження – моделі архітектур інформаційних систем для обробки великих даних.

Предмет дослідження – розробка архітектури великих даних для розпізнавання осіб з використанням технології машинного навчання.

Мета – здійснити аналіз сукупностей технологій та фреймворків обробки даних в Big Data.

Завдання дослідження – в процесі дослідження вирішувалися наступні завдання:

1. Аналіз основних положень концепції технології Big Data.

2. Дослідити та здійснити аналіз по сукупності технологій та фреймворків обробки даних в Big Data на теперішній час.

3. Дослідити та здійснити оцінку по основним моделям архітектур інформаційних систем для обробки великих даних.

4. Розробка моделі архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання.

Методика дослідження – методи теорії інформації, методи наукового моделювання, методи дослідження інформаційних систем.

Наукова новизна – розроблено модель архітектури великих даних для вирішення задачі розпізнання осіб з використанням машинного навчання.

Практична значущість. Основні результати магістерської роботи можуть бути використані при розробці систем Big Data.

Апробація: Основні результати роботи опубліковано у матеріалах двох науково-практичних конференцій.

1 АНАЛІЗ СУКУПНОСТІ ТЕХНОЛОГІЇ BIG DATA

1.1 Визначення поняття BIG DATA

Як визначає Gartner «Великі дані - це великі обсяги, високошвидкісні або різноманітні інформаційні активи, які вимагають нових форм обробки, щоб забезпечити більш досконале прийняття рішень, аналітичне виявлення і оптимізацію процесів»[1]. Термін «великі дані» не вимагає пояснень – це надзвичайно великі колекції даних (набори даних), які можуть бути проаналізовані для виявлення закономірностей, тенденцій і асоціацій, особливо пов'язані з людською поведінкою та її взаємодією у мережі інтернет.

Основні джерела великих даних можна розділити на:

- Соціальні (людські) – містять в собі повідомлення в соціальних мережах, опубліковані відео і т.д.
- Машинні – дані генеруються датчиками або пристроями;
- Транзакційні;

Характеристики великих даних, відомі як 5V: обсяг (volume), різноманітність (variety), швидкість (velocity), достовірність (veracity) та цінність (value). Ці характеристики загалом прийняті як основні якості великих даних (рис.1.1).

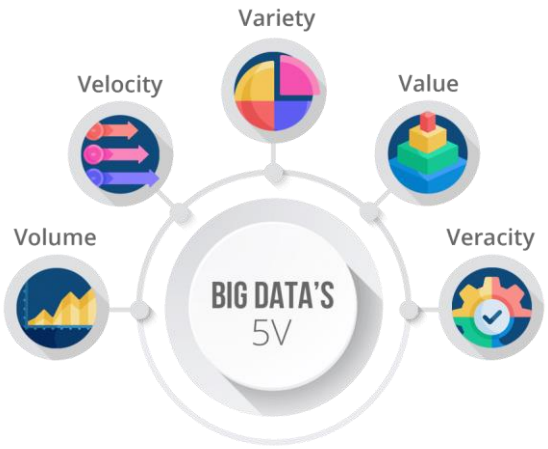


Рисунок 1.1. - Основні якості великих даних.

Обсяг

Обсяг великих даних, що зберігаються в великих компаніях, таких як Walmart (супермаркети), Apple і EBay, вимірюється кількома петабайт. Типовий диск на персональному комп'ютері (ПК) вміщує гігабайт, тому сховища великих даних цих компаній містять, принаймні, дані, які зазвичай можуть зберігатися на 1 мільйон ПК, можливо, навіть на 10-20 мільйонах ПК. Масштаб цього важко зрозуміти. Ймовірно, більш корисно розглянути типи даних, які зазвичай зберігають великі компанії:

- Роздрібні торговці (за допомогою карт лояльності, які проходять при оформленні замовлення: відомості про всі покупки, які ви робите, коли, де, як ви платите, використання купонів). Веб-сайти (кожен продукт, який ви кожен переглянули, кожна сторінка, яку ви відвідали, кожен продукт, який ви коли-небудь купували).
- Соціальні мережі (наприклад, Facebook, Instagram, TikTok, друзі та контакти, зроблені публікації, фотографії, які можна сканувати для ідентифікації та будь-які інші дані, які ви можете поширити через всесвітню мережу інтернет).
- Компанії мобільного зв'язку (номери, які ви телефонуєте, тексти, які ви відправляєте, всі місця, де колись був ваш телефон, коли він був включений з точністю до кількох метрів, ваші звички перегляду і т.д).
- Інтернет-провайдери і провайдери браузерів (кожен сайт і кожна сторінка, яку ви відвідуєте. Інформацію про всі викачування та всіх електронних листах, пошукові запити, які ви вводите).
- Банківські системи (кожна квитанція, платіж, інформація про кредитну картку (сума, дата, продавець, місцезнаходження), місцезнаходження використовуваних банкоматів).

Різноманітність

Деяку різноманітність інформації можна побачити з наведених вище прикладів. Зокрема, зберігаються такі види інформації:

- Дії при перегляді у браузері.

- Фінансові операції.
- Інтереси.
- Вподобанні звички купівлі у маркетплейсах.
- Реакція на рекламу в Інтернеті, рекламні електронні листи, банери.
- Географічні дані.
- Інформація про соціальні та ділові контакти.
- Графічна інформація (наприклад, фотографії).
- Усна інформація (наприклад, голосова пошта).
- Технічна інформація, така як аналіз вібрації та температури реактивного двигуна.

Ці дані можуть бути як структурованими, так і неструктурованими.

До структурованих даних можна віднести дані, які зберігаються в певних полях (числові, текстові, дати й т.д.), часто з певною довжиною, в межах певної записи у файлі схожих записів. Для структурованих даних потрібно модель типів і формату бізнес-даних, які будуть записуватися, а також того, як дані будуть зберігатися, оброблятися і доступні. Це називається моделлю даних. Проєктування моделі визначає та обмежує дані, які можуть бути зібрані та збережені, а також обробку, яка може виконуватися з ними.

До неструктурованих даних можна віднести ту інформацію, яка не має визначеної моделі даних. Вона буває всіх формах та розмірах, і саме ця різноманітність і нерівномірність ускладнює зберігання таким чином, щоб її можна було аналізувати, шукати або використовувати іншим чином.

Швидкість

Інформація повинна надаватися досить швидко, щоб її можна було використовувати при прийнятті рішень та управлінні продуктивністю.

Треба розуміти, що обсяг і різноманітність суперечать швидкості, тому необхідно знайти методи для обробки величезних обсягів неоднорідних, незручних даних в реальному часі.

Достовірність

Правдивість означає точність і правдивість і пов'язана з якістю даних. В контексті великих даних, щоб будь-який аналіз давав корисні результати для прийняття рішень, зібрані дані повинні бути правдивими. Щоб оцінити, наскільки правдиві зібрані дані, компанії повинні враховувати не тільки те, наскільки точним або надійним може бути набір даних, але і наскільки надійним є джерело даних.

Складність, з якою стикаються компанії, полягає в тому, що за самою своєю природою зібрані дані надходять з безлічі різних джерел. Деякі будуть більш надійними, ніж інші. Наприклад, дані, отримані від машин і транзакцій, будуть вважатися більш надійними, ніж дані, отримані від людей. Дані з транзакційних і машинних джерел буде легше перевіряти та важче маніпулювати.

Правдивість також пов'язана зі швидкістю. Щоб дані були корисні при прийнятті рішень, їх необхідно аналізувати як можна швидше. Швидкість показує, що зібрані дані швидко змінюються. Аналіз застарілих даних може привести до прийняття неправильних рішень.

Цінність

Немає сенсу витрачати сили та гроші на збір і аналіз даних, якщо це в кінцевому результаті не призведе до збільшення вартості компанії. Компаніям важливо враховувати потенціал аналітики великих даних і цінність, яку він може створити, якщо буде збирати, аналізувати та використовувати з розумом.

Приклад того, як британська група супермаркетів Tesco використовувала аналіз даних для збільшення вартості: Tesco має представництва в декількох країнах по всьому світу. В Ірландії компанія розробила систему для аналізу температури холодильників в магазинах. В холодильники були поміщені датчики, які вимірювали температуру кожні три секунди та відправляли інформацію через Інтернет в центральне сховище даних. Аналіз цих даних дозволив компанії ідентифікувати агрегати, які працювали при некоректних температурах. Компанія виявила, що деякі холодильники працювали при температурах нижче рекомендованих від -21°C до -23°C. Це явно коштувало компанії втрат енергії. Ця

інформація дозволила компанії скорегувати температуру холодильників. З огляду на, що компанія витрачала 10 мільйонів євро в рік на охолодження холодильників в Ірландії, очікуване скорочення цих витрат на 20% було значною економією[2].

Система також дозволила інженерам віддалено контролювати роботу холодильників. Коли вони визначили, що конкретний блок несправний, вони могли проаналізувати проблему, а потім відвідати магазин з потрібними деталями та замінити їх.

1.2 Обробка даних в BIG DATA та принципи доступу до даних

Щоб очистити, стандартизувати і перетворити дані з різних джерел, обробка даних повинна стосуватися кожного запису в даних, що надходять. Як тільки запис очищена і завершена, робота зроблена. Це принципово відрізняється від доступу до даних - останній призводить до повторного вилучення і доступу до однієї і тієї ж інформації з різними користувачами або додатками. Коли обсяг даних невеликий, швидкість обробки даних є меншою проблемою, ніж в порівнянні з доступом до даних, і тому зазвичай відбувається в тій же базі даних, де знаходяться остаточні дані. У міру зростання обсягу даних було виявлено, що обробка даних повинна здійснюватися поза базами даних, щоб обійти всі накладні витрати і обмеження, викликані системою баз даних, яка явно не була призначена для обробки великих даних в першу чергу. Це було, коли ETL, а потім Hadoop почали грати критично важливу роль в епоху сховищ даних і великих даних відповідно (рис.1.2).

Проблема обробки великих даних полягає в тому, що обсяг оброблюваних даних завжди знаходиться на рівні, який може вмістити жорсткий диск, але набагато більше, ніж обсяг обчислювальної пам'яті, доступної на цей момент. Основний спосіб ефективної обробки даних - розбити дані на більш дрібні частини та обробляти їх паралельно. Іншими словами, масштабованість досягається першим включенням паралельної обробки в програмі, так що при збільшенні обсягу даних кількість паралельних процесів буде збільшуватися, в той час, як кожен процес продовжує обробляти такий же обсяг даних, як і раніше; по-друге,

додаючи більше серверів з великою кількістю процесорів, пам'яті і дисків в міру збільшення кількості паралельних процесів.

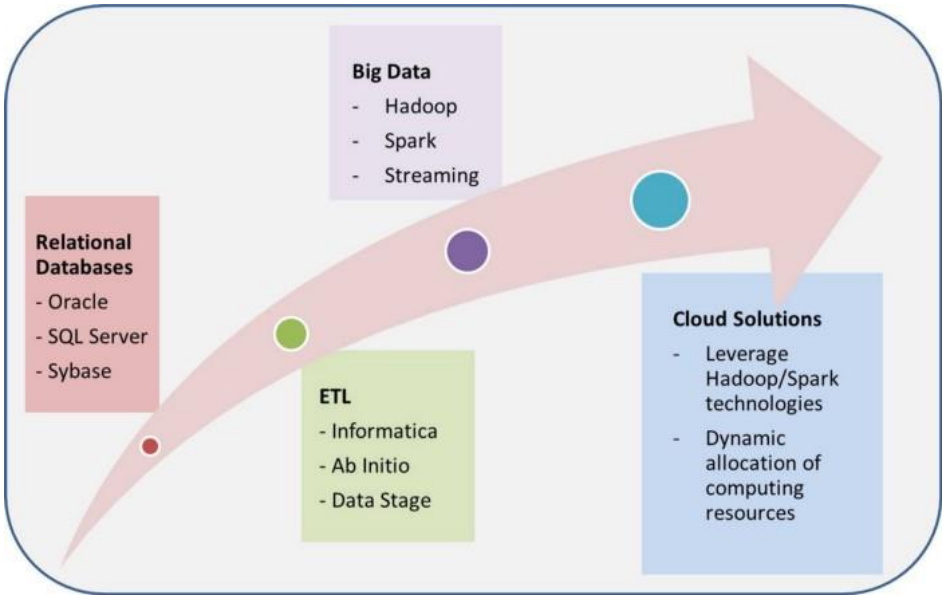


Рисунок 1.2. - Коротка історія технологій обробки даних.

Паралельна обробка великих даних вперше була реалізована за допомогою техніки розділення даних в системах баз даних і інструментах ETL. Як тільки набір даних логічно розділений, кожен розділ може оброблятися паралельно. Hadoop HDFS (високо розподілені файлові системи) застосовує той же принцип найбільш масштабованим чином. Що робить HDFS, так це розділяє дані на блоки даних, кожен з яких має постійний розмір. Потім блоки розподіляються по різним серверним вузлам і записуються сховищем метаданих в так званому вузлі імен. Коли починається процес обробки даних, кількість процесів визначається кількістю блоків даних і доступними ресурсами (наприклад, процесорами і пам'яттю) на кожному серверному вузлі. Це означає, що HDFS забезпечує масову паралельну обробку, якщо у вас досить процесорів і пам'яті з декількох серверів.

Зараз Spark став одним з найпопулярніших швидких двигунів для великомасштабної обробки даних в пам'яті. Перш за все, Spark використовує загальний обсяг пам'яті в розподіленому середовищі з декількома вузлами даних. Однак обсягу пам'яті і раніше недостатньо, і це може виявитися дорогим, якщо будь-яка організація спробує вмістити великі дані в кластер Spark. Обробка даних

завжди починається з читання даних з диска в пам'ять і в кінці запису результатів на диски. Якщо кожен запис потрібно обробити тільки один раз перед записом на диск, що відбувається при типовій пакетній обробці, то Spark не дасть переваги в порівнянні з Hadoop. З іншого боку, Spark може зберігати дані в пам'яті на декількох етапах перетворення даних, в той час, як Hadoop не може. Це означає, що Spark пропонує переваги при багаторазовій ітеративній обробці одного і того ж фрагмента даних, що як раз і потрібно в аналітиці та машинному навчанні.

Ще одна актуальна тема в області обробки даних - потокова обробка. Вона пропонує велику перевагу в зниженні швидкості обробки, оскільки в певний момент часу потрібно обробляти тільки невеликий обсяг даних, коли вони надходять. Однак він не так універсальний, як пакетна обробка, у двох аспектах: по-перше, вхідні дані повинні надходити в «потоковому» режимі, а по-друге, що певна логіка обробки, яка вимагає агрегування за проміжок часу, все ще має оброблятися. в партії згодом.

Нарешті, хмарні рішення надають можливість більш динамічно масштабувати систему розподіленої обробки в залежності від обсягу даних і, отже, кількості паралельних процесів. Цього складно добитися на підприємстві, тому що нові сервери необхідно планувати, закладати в бюджет і купувати. Якщо потужність не спланована належним чином, обробка великих даних може бути або обмежена кількістю обладнання, яка додаткова покупка призведе до втрати ресурсів без використання. Обробка в хмарі отримує велику перевагу у вигляді еластичності інфраструктури, яка може дати більше гарантій досягнення найкращого масштабування при більш рентабельності.

Доступ до даних

У порівнянні з обробкою даних, доступ до даних має дуже різні характеристики, в тому числі:

- Структура даних сильно залежить від того, як додатки або користувачі повинні отримувати дані.

- Необхідно добре розуміти схеми вилучення даних, оскільки деякі дані можуть повторно вилучатись великою кількістю користувачів або додатків.
- Обсяг даних, що витягають кожен раз, повинен бути цільовим та, отже, повинен містити лише частина доступних даних.

З огляду на наведені вище принципи, за останні два десятиліття було кілька етапів, які відображають, як отримати доступ до постійно зростаючого обсягу даних, при цьому повертаючи запитані дані протягом декількох секунд:

- Сховище даних (уникає об'єднання таблиць, яке може бути дуже дорогим при великому обсязі даних. Тут з'являється концепція «таблиці фактів», в якій всі стовпці об'єднані без принципів нормалізації бази даних, як в реляційної базі даних).
- Колонне сховище (кожна колонка зберігається і індексується, тому доступ до неї здійснюється окремо. Це дає більш швидкий час відгуку, ніж доступ на основі рядків в звичайних реляційних базах даних, коли в рядку багато колонок, тоді як запити витягають тільки декілька стовпців за раз).
- База даних NoSQL (виключає об'єднання і реляційну структуру в цілому і адаптована для швидкого отримання інформації конкретнішим способом у форматі JSON).
- База даних в пам'яті (забезпечує високу продуктивність за рахунок зберігання всієї бази даних або всієї таблиці в пам'яті).

На (рис.1.3) нижче наведені деякі популярні приклади кожного типу бази даних, але не наводиться повний список. Зверніть увагу, що база даних може об'єднувати більше однієї технології. Наприклад, Redis - це база даних NoSQL, а також в пам'яті. Крім того, при отриманні даних зі сховищ даних і стовпчастих сховищ використовуються паралельні процеси для отримання даних, коли це може бути застосовано. Оскільки може бути багато варіантів різних типів баз даних в залежності від вмісту даних, структури даних і шаблонів пошуку користувачами або додатками, доступ до даних - це область, в якій організації необхідно швидко і постійно розвиватися. Також має бути звичайною справою мати різні типи баз даних або інструментів одночасно для різних цілей.

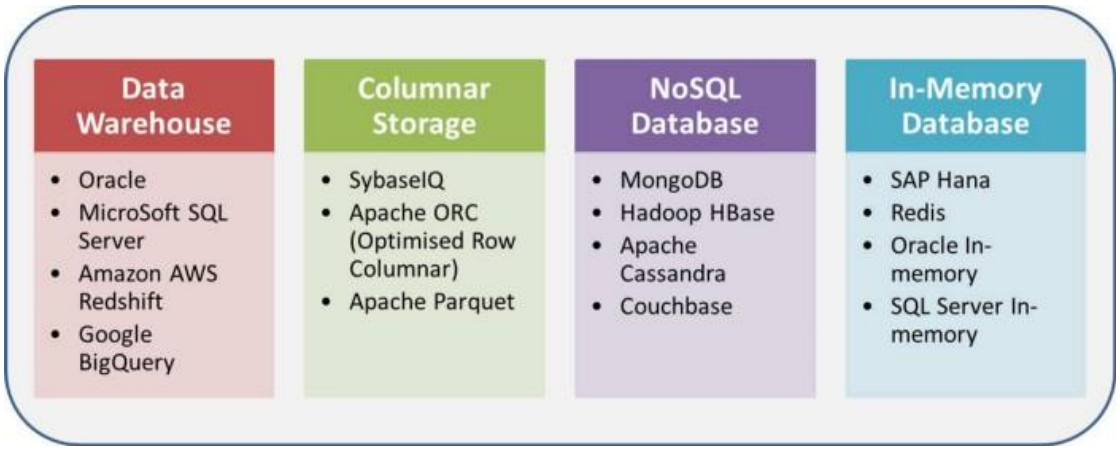


Рисунок 1.3. - Список технологій по кожному типу бази даних.

Як ми бачимо, велика різниця між обробкою даних і доступом до даних полягає в тому, що доступ до даних в кінцевому підсумку залежить від потреб клієнтів і бізнесу, а вибір правильної технології сприяє розробці нових продуктів в майбутньому і розширює можливості користувачів. З іншого боку, обробка даних є основним активом компанії, а масштабна обробка і отримання даних хорошої якості є важливим фактором, що дозволяє компанії рости разом з її даними. Багато компаній стикаються з переслідуванням їх системи обробки даних, коли обсяг даних зростає, і відновлення платформи обробки даних з нуля обходиться дорого.

Принцип паралельної обробки даних і масштабованості необхідно ретельно продумати і розробити з самого початку. Обробка даних також йде рука об руку з управлінням даними і інтеграцією даних – все це необхідно для успіху будь-якої організації. Більш того, кожна організація тепер стикається з безліччю варіантів рішень для великих даних як від спільнот з відкритим вихідним кодом, так і від сторонніх постачальників. Чітке розуміння відмінностей між обробкою даних і доступом до даних може дозволити IT-керівникам і бізнес-лідерам не тільки побудувати надійну архітектуру даних, але і приймати правильні рішення по її розширенню і модернізації в стабільних темпах.

1.3. Аналіз сучасних технологій та фреймворків для роботи з BIG DATA

Великі дані в даний час є однією з найбільш затребуваних сфер в розробці та доповненні корпоративного програмного забезпечення. Висока популярність технологій великих даних - явище, викликане швидким і постійним зростанням обсягів даних. Для забезпечення необхідної пропускної спроможності необхідно аналізувати, структурувати і обробляти масивні масиви даних. Механізми обробки даних все частіше використовуються в технічних стеках для мобільних додатків і в багатьох інших сферах.

В даний час, імовірно, немає єдиного програмного забезпечення для обробки великих даних, яке не могло б обробляти величезні обсяги даних. Для реалізації і підтримки функціональності такого програмного забезпечення були створені спеціальні платформи великих даних. Вони допомагають швидко обробляти і структурувати величезні блоки даних в реальному часі.

Прямо зараз на ринку є багато відмінних інструментів для роботи з великими даними. Однак на теперішньому розвитку фреймворків та технологій Big Data можна визначити: Apache Hadoop, MapReduce, Spark, Hive, Storm, Samza, Flink, Heron, Kudu, Presto.

Apache Hadoop

Apache Hadoop був революційним рішенням для зберігання і обробки великих даних свого часу. Велика частина програмного забезпечення для великих даних побудована на основі Hadoop або сумісна з ним. Це проект з відкритим вихідним кодом від Apache Software Foundation.

Hadoop відмінно підходить для надійних, масштабованих розподілених обчислень. Однак його також можна використовувати як файлове сховище загального призначення. Він може зберігати і обробляти петабайт даних. Це рішення складається з трьох основних компонентів:

1) Файлова система HDFS, що відповідає за зберігання даних в кластері Hadoop;
2) Система MapReduce, призначена для обробки великих обсягів даних в кластері;
3) YARN - ядро, яке відповідає за управління ресурсами.

Hadoop використовує проміжний рівень між інтерактивною базою даних і сховищем даних. Його продуктивність зростає в міру збільшення обсягу зберігання даних. Щоб збільшити його, ви можете додавати нові вузли в сховище даних. Hadoop може зберігати і обробляти багато петабайт інформації, в той час як найшвидші процеси в Hadoop працюють всього за кілька секунд. Він також забороняє будь-які зміни даних, вже збережених в системі HDFS під час обробки.

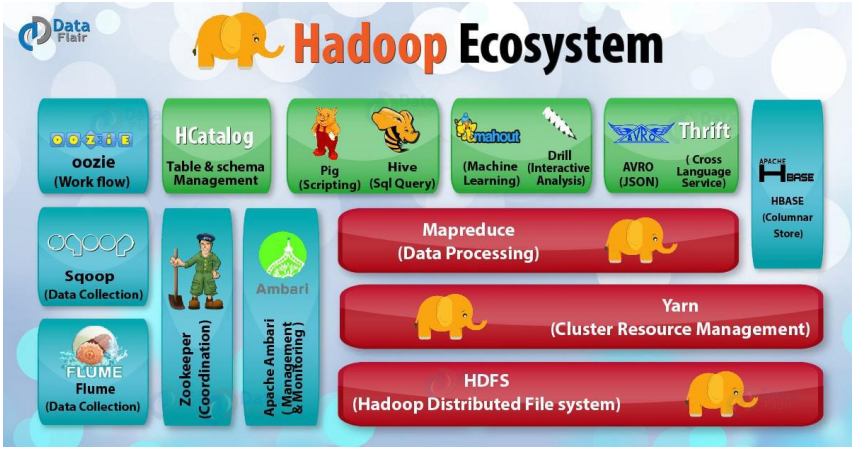


Рисунок 1.4. - Екосистема фреймворку Hadoop.

Hadoop також відмінно підходить для клієнтської аналітики, корпоративних проектів і створення озер даних. Або для будь-якої великомасштабного завдання пакетної обробки, яка не вимагає оперативності або ACID-сумісного сховища даних.

Коли він тільки з'явився, він був революційним і породив цілу індустрію. Тепер великі дані мігрують в хмару. Проте на сьогоденні Hadoop як і раніше, залишається потужним інструментом пакетної обробки, який можна інтегрувати з більшістю інших фреймворків для аналізу великих даних. Його компоненти: HDFS, MapReduce і YARN є невід'ємною частиною самої галузі. Так що не схоже, що він скоро піде. Але, незважаючи на певну популярність Hadoop, технічний прогрес

ставить нові цілі і вимоги. Поступово на ринок приходять більш просунуті його альтернативи, мова про яких буде йти далі.

MapReduce

MapReduce - це розподілена обчислювальна модель від Google, яка використовується в технологіях Big Data для паралельних обчислень дуже великих (до кількох петабайт) наборів даних у комп'ютерних кластерах, а також фреймворк для обчислення розподілених завдань на вузлах кластерів[3]. Цей механізм обробляє дані як записи і обробляє їх в три етапи:

1) Map (попередня обробка і фільтрація даних).
2) Shuffle (робочі вузли сортують дані, кожному відповідає один вихідний ключ).
3) Reduce (функція зменшення встановлюється користувачем і визначає кінцевий результат для окремих груп вихідних даних).

Більшість всіх значень повертає Reduce (функції є кінцевим результатом завдання MapReduce). MapReduce забезпечує автоматичне розподілення даних, ефективне балансування даними та безвідмовну роботу.

Він був основним продуктом галузі протягом багатьох років і використовується з іншими відомими технологіями великих даних. Але є альтернативи MapReduce, зокрема Apache Tez. Він налаштовується і працює набагато швидше. Він використовує YARN для управління ресурсами і, отже, набагато більш ефективний з точки зору ресурсів.

Apache Spark

Це платформа з відкритим вихідним кодом, створена як більш просунуте рішення в порівнянні з Apache Hadoop. Початковий фреймворк був спеціально побудований для роботи з великими даними. Основна відмінність між цими двома рішеннями - це модель пошуку даних.

Hadoop зберігає дані на жорсткому диску разом з кожним кроком алгоритму MapReduce. Поки Spark реалізує всі операції, використовуючи оперативну пам'ять.

Завдяки цьому Spark показує високу продуктивність і дозволяє обробляти великі потоки даних. Функціональні основи і основні характеристики Spark - висока продуктивність і відмовостійкість.

Він складається з п'яти компонентів: ядра і чотирьох бібліотек, які оптимізують взаємодію з великими даними(рис. 1.5).

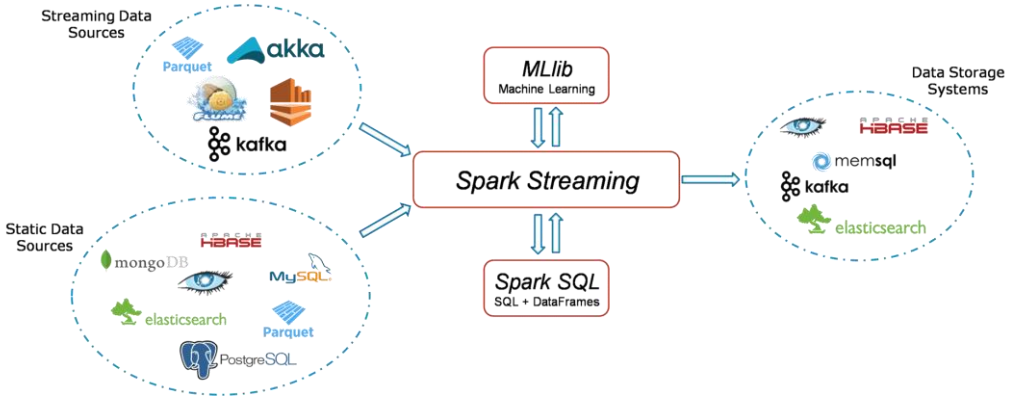


Рисунок 1.5. - Основні компоненти Apache Spark.

Spark SQL - одна з чотирьох спеціалізованих бібліотек фреймворка, яка використовується для обробки структурованих даних. Використання DataFrames і рішення запитів Hadoop Hive до 100 разів швидше.

Spark має одну з кращих реалізацій штучного інтелекту в галузі - Sparkling Water. Spark також має інструмент Streaming для обробки даних, що відносяться до потоку, в режимі реального часу. Spark більше схожий на швидкий пакетний процесор, ніж на справжній потоковий процесор, такий як Flink, Heron або Samza. І це нормально, якщо вам потрібні потокові функції в пакетному процесорі. Або, якщо вам потрібен повільний потоковий процесор з високою пропускною здатністю. Це питання перспективи.

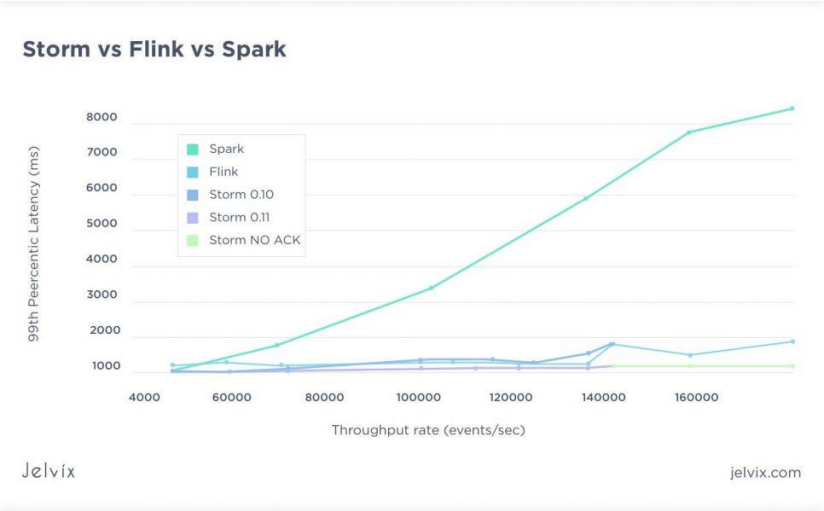


Рисунок 1.6. - Порівняння продуктивності пакетної обробки даних технології Spark з Flink та Storm.

Творці Spark заявляють, що в середньому обробка кожної мікропакції займає всього 0,5 секунди[4]. Далі йде MLib - розподілена система машинного навчання, яка в дев'ять разів швидше бібліотеки Apache Mahout. Також остання бібліотека - GraphX, використовується для масштабованої обробки даних графа. Spark часто розглядається як альтернатива Hadoop в реальному часі. Може бути, але, як і всі компоненти екосистеми Hadoop, його можна використовувати разом з Hadoop і іншими відомими фреймворками для великих даних.

Apache Hive

Apache Hive – це фреймворк для аналітики великих даних, був створений компанією Facebook, з метою об'єднати масштабованість однією з найпопулярніших платформ великих даних.

До складу Apache Hive входять такі компоненти як:

- Парсер (фільтрує SQL-запити);
- Оптимізатор (оптимізує запити для більшої ефективності);
- Виконавець (запускає завдання в фреймворку MapReduce);

Hive також може бути інтегрований з Hadoop (як серверна частина) для аналізу великих обсягів даних. Нижче наведено тест (рис.1.7), який показує швидкодію Hive в порівнянні з конкурентам, де чим нижчий показник, тим краще.

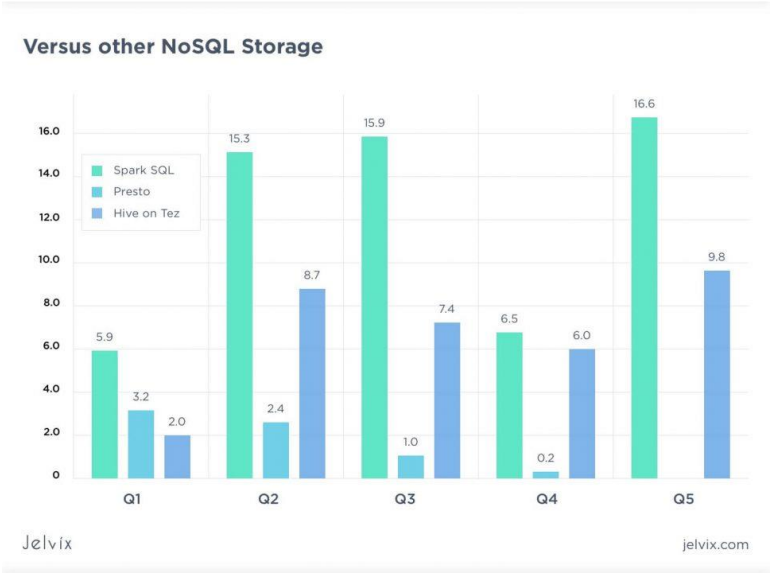


Рисунок 1.7. - Порівняння швидкодії Hive з іншими фреймворками для аналітики великих даних.

Через десять років після першого випуску Hive залишається однією з найбільш часто використовуваних платформ для аналізу великих даних.

Hive 3 версії був випущений Hortonworks в 2018 році. Він замінив MapReduce на Tez в якості пошукової системи. Також він має можливості машинного навчання і інтеграцію з іншими популярними платформами великих даних.

Apache Storm

Apache Storm - ще одне відоме рішення, орієнтоване на роботу з великим потоком даних в реальному часі. Ключовими особливостями Storm є масштабованість і можливість швидкого відновлення після простою. Ви можете працювати з цим рішенням за допомогою Java, а також Python, Ruby, Perl, JavaScript.

У Storm є кілька елементів, які суттєво відрізняють його від аналогів. Перший - Tuple - ключовий елемент представлення даних, що підтримує серелізацію. Потім є Stream, який включає схему іменування полів в Tuple. Spout отримує дані із зовнішніх джерел, формує з них Tuple і відправляє їх Stream. Також є Bolt - обробник даних і Topology - пакет елементів з описом їх взаємозв'язку. У

сукупності всі ці елементи допомагають розробникам керувати великими потоками неструктурованих даних.

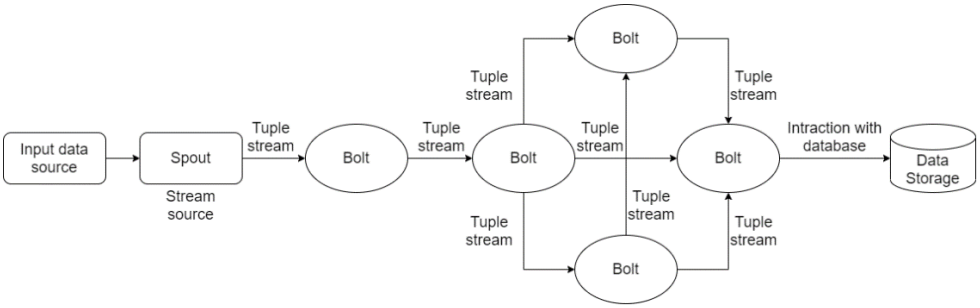


Рисунок 1.8. - Основна концепція Apache Storm.

Говорячи про продуктивність, Storm забезпечує кращу затримку, ніж Flink і Spark. Однак пропускна здатність у нього гірше. Нещодавно Twitter (провідний прихильник Storm) перейшов на новий фреймворк Heron. Storm як і раніше використовується такими великими компаніями, як Yelp, Yahoo!, Alibaba та інші. У 2021 році у нього спостерігається велика база користувачів та підтримка з боку спільноти.

Apache Samza

Apache Samza - це фреймворк для обробки великих даних з відстеженням стану, який був розроблений спільно з Kafka. Kafka забезпечує обслуговування даних, буферизацію і відмовостійкість. Це потрібно для використання там, де потрібна швидка одностадійна обробка. З Kafka його можна використовувати з низькими затримками. Samza також зберігає локальний стан під час обробки, що забезпечує додаткову відмовостійкість.

Samza була розроблена для архітектури Kappa (тільки конвеєр потокової обробки), але може використовуватися в інших архітектурах. Samza використовує YARN для узгодження ресурсів. Тому для роботи йому потрібен кластер Hadoop, а це значить, що ви можете покластися на функції, що надаються YARN (рис.1.9).

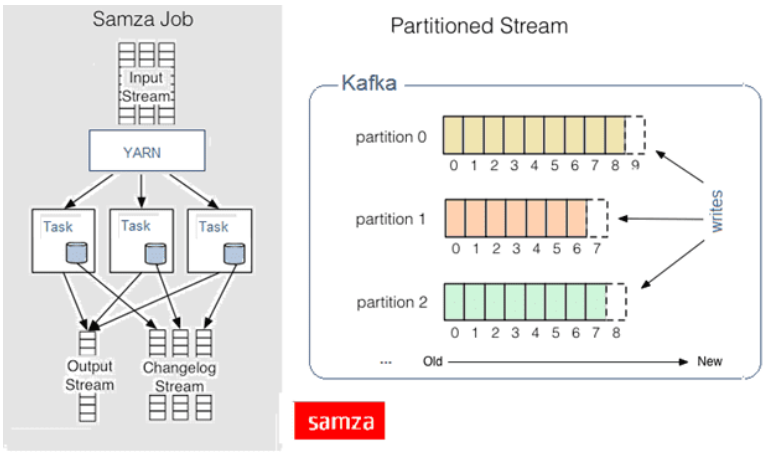


Рисунок 1.9. - Узгодження ресурсів Samza за допомогою YARN.

Ця структура обробки великих даних була розроблена для Linkedin, а також використовується eBay і TripAdvisor для виявлення шахрайства. Значна частина його коду була використана Kafka для створення конкуруючої платформи обробки даних з потоками Kafka. Загалом, Samza - грізний інструмент, який гарний у тому, для чого він створений.

Apache Flink

Apache Flink - це надійна платформа обробки великих даних для потокової і пакетної обробки. Вперше задуманий як частина наукового експерименту в 2008 році, він став відкритим в 2014 році. З тих пір він набирає популярність.

У Flink є кілька цікавих функцій і нових вражаючих технологій. Він використовує потокову обробку з відстеженням стану, таку як Apache Samza. Але він також виконує ETL і пакетну обробку з пристойною ефективністю.

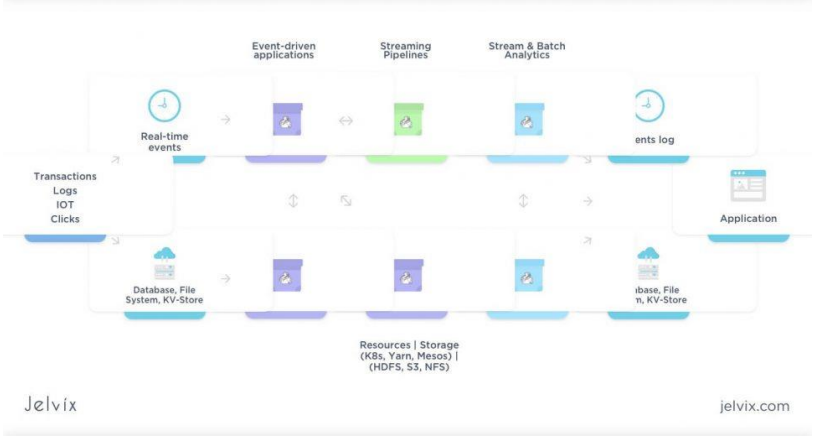


Рисунок 1.10. - Архітектура та принцип роботи Apache Flink.

Це відмінний вибір для спрощення архітектури, в якій потрібно як потокова, так і пакетна обробка. Він може витягувати тимчасові мітки з оброблених даних для створення більш точної оцінки часу і кращого кадрування аналізу потокових даних. Він також має можливість реалізації машинного навчання.

Будучи частиною екосистеми Hadoop, його можна без проблем інтегрувати в існуючу архітектуру. У нього є можливість інтеграції з MapReduce і Storm, так що на ньому є можливість запускати готові програми. Також має хорошу масштабованість для великих даних.

Flink також добре підходить для розробки подієво-орієнтованих додатків. Ви можете встановити на ньому контрольні точки, щоб зберегти прогрес в разі збою під час обробки. Flink також може підключатися до популярного інструменту візуалізації даних Zeppelin.

Компанія Alibaba використовувала Flink для спостереження за поведінкою споживачів і пошуковим рейтингом[5]. А такий фінансовий гігант ING використовував Flink для створення додатків для виявлення шахрайства та повідомлення користувачів. Крім того, у Flink є алгоритми машинного навчання. Flink безсумнівно, є однією з нових технологій обробки великих даних, яка викликає захват.

Apache Heron

Heron. Це один з новітніх механізмів обробки великих даних. Twitter розробив його як заміну Storm нового покоління. Він призначений для використання для виявлення спаму в реальному часі, завдань ETL і аналізу тенденцій.

Apache Heron повністю назад сумісний зі Storm і має простий процес міграції. Його цілі проектування включають низьку затримку, хорошу і передбачувану масштабованість і просте адміністрування. Розробники приділяють велику увагу ізоляції процесів для полегшення налагодження і стабільного використання ресурсів (рис.1.11). Тести Twitter показують значне поліпшення в порівнянні з Storm[6].

Цей фреймворк все ще перебуває в стадії розробки, тому, якщо буде потреба впровадити технологію на ранній стадії продукту, це може бути те, що вам потрібно. Володіючи відмінною сумісністю з Storm та надійною підтримкою з боку Twitter, Heron, ймовірно скоро стане наступним великим досягненням.

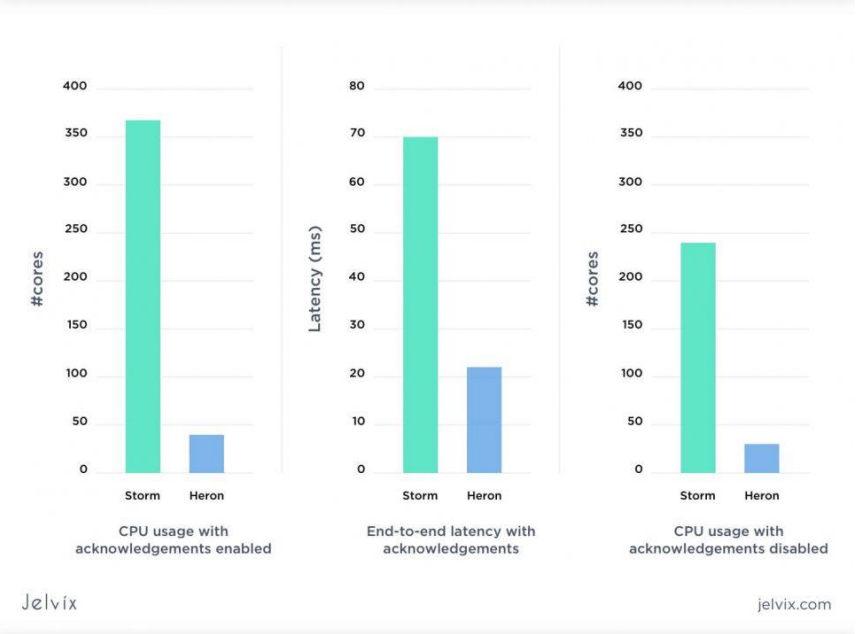


Рисунок 1.11. – Оптимальне використання ресурсів Heron порівняно з Storm.

Apache Kudu

Apache Kudu - це новий захоплюючий компонент сховища. Він розроблений для спрощення деяких складних конвеєрів в екосистемі Hadoop. Це SQL-подібне рішення, призначене для комбінації випадкових і послідовних операцій читання та запису. Також він чудово інтегрується з більшістю інших фреймворків великих даних екосистеми Hadoop, особливо з Kafka та Impala.

Kudu в даний час використовується для виявлення шахрайства з ринковими даними на Волл-стріт. Виявилося, що він особливо підходить для обробки потоків різних даних з частими оновленнями. Він також відмінно підходить для аналізу реклами в реальному часі, оскільки він досить швидкий і забезпечує відмінну доступність даних.

Kudu був обраний китайським гігантом побутової техніки Xiaomi для збору звітів про помилки. В основному через його здатності спростити та оптимізувати конвеєр даних для підвищення швидкості запитів та аналітики (рис.1.13).

Рисунок.1.13. - Архітектура Kudu для оптимізації конвеєра даних та підвищення швидкості запитів і аналітики.

Presto

Presto - швидша та гнучка альтернатива Apache Hive для невеликих завдань. Presto був випущений з відкритим вихідним кодом у 2013 році. Це адаптивний, гнучкий інструмент запитів для багатокористувацького середовища даних з різними типами сховищ.

Architecture Comparison

	Hive	Presto	Spark	BigQuery
Perfomance	Slow	Fast	Fast	Ultra Fast (using many disks)
Intermediate Storage	HDFS	None	Memory/ Disk	Colossus (?)
Data Transfer	HTTP	HTTP	HTTP	?
Query Execution	Stage-wize MapReduce	Run all stages at once (pipelining)	Stage-wise	?
Fault Tolerance	Yes	None (but, TD will retry the query)	Yes, but limited	?
Multiple Job Support	Good Can handle many jobs	Limited (- 5 concurrent queries per account in TD)	Require another resource manager (e.g. YARN, mesos)	Limited (Query queue)

Jelvix jelvix.com

Рисунок 1.14. - Характеристика фреймворку Presto у порівнянні з його конкурентами.

Presto має об'єднану структуру, велика кількість роз'ємів і безліч інших функцій. Одним з перших вимог при його розробці, це була здатність аналізувати невеликі підмножини даних (в діапазоні від 50 ГБ до 3 ТБ). Це зручно для описової аналітики такого обсягу даних.

Також мною був проведений ретельний аналіз, та можна з упевненістю сказати, що серед фреймворків обробки даних немає єдиного кращого варіанту. У кожного є свої плюси і мінуси. Крім того, результати, що надаються деякими рішеннями, суворо залежать від багатьох факторів та як показує практика, гібридні рішення з різними інструментами працюють найкраще. Але були виділені такі аспекти технологій та фреймворків для роботи з Big Data:

- Найбільш популярні (Hadoop, Storm, Hive, Spark).
- Найбільш перспективні (Flink, Heron).
- Найбільш корисні (Presto, MapReduce).
- Недооціненні (Samza, Kudu).

	Найбільш популярні			
Фреймворки / технології	**Hadoop**	**Storm**	**Hive**	**Spark**
Режим обробки даних	пакетний	пакетний / потоковий	пакетний	пакетний / потоковий
Масштабованість	горизонтальна	горизонтальна	горизонтальна	горизонтальна
Гарантії доставки повідомлень	рівно раз	принаймні, один раз	принаймні, один раз	рівно раз
Режим обчислення	на основі диска	в пам'яті	на основі диска	в пам'яті
Автоматичне масштабування	підтримує	не підтримує	підтримує	підтримує
Відмостійкість	присутня	присутня	присутня	присутня(обмежена)
Підтримка мов програмування	Java, Python	Python, Ruby, JavaScript, Perl	SQL, HiveQL	Scala, Java, Python, R

Рисунок 1.15. – Порівняння основних можливостей найбільш популярних технологій та фреймворків для роботи з Big Data.

	Найбільш перспективні	
Фреймворки / технології	**Flink**	**Heron**
Режим обробки даних	пакетний / потоковий	потоковий
Масштабованість	горизонтальна	горизонтальна
Гарантії доставки повідомлень	рівно раз	принаймні, один раз
Режим обчислення	в пам'яті	в пам'яті
Автоматичне масштабування	не підтримує	підтримує
Відмостійкість	присутня(обмежена)	присутня
Підтримка мов програмування	Java, Scala, Python, R	C++, Java, Python

Рисунок 1.16. – Порівняння основних можливостей найбільш перспективних технологій та фреймворків для роботи з Big Data.

	Найбільш корисні	
Фреймворки / технології	**Presto**	**MapReduce**
Режим обробки даних	потоковий	конвеєрний
Масштабованість	горизонтальна	горизонтальна
Гарантії доставки повідомлень	відсутня	рівно раз
Режим обчислення	в пам'яті	на основі диска
Автоматичне масштабування	підтримує	підтримує
Відмостійкість	відсутня	присутня
Підтримка мов програмування	Java, SQL	Java, Python

Рисунок 1.17. – Порівняння основних можливостей найбільш корисних технологій та фреймворків для роботи з Big Data.

	Недооціненні	
Фреймворки / технології	**Samza**	**Kudu**
Режим обробки даних	потоковий	пакетний / конвеєрний
Масштабованість	горизонтальна	горизонтальна / вертикальна
Гарантії доставки повідомлень	принаймні, один раз	відсутня
Режим обчислення	в пам'яті	в пам'яті
Автоматичне масштабування	підтримує	не підтримує
Відмостійкість	присутня	присутня
Підтримка мов програмування	Java, Scala, Samza SQL	C++, Java, Python

Рисунок 1.18. – Порівняння основних можливостей недооцінених технологій та фреймворків для роботи з Big Data.

Різноманітність пропозицій на ринку фреймворків для великих даних дозволяє технічно компанії вибрати найбільш потрібний інструмент для вирішення поставленої задачі.

1.4. Проблеми та перспективи технології big data

Великий обсяг інформації створює величезний потенціал для бізнесу, а також створює серйозні проблеми, які необхідно вирішувати в найближчому майбутньому. Кількість даних з часом сильно зростає. IDC прогнозував, що

глобальна сфера даних виросте з 33 зеттабайт в 2018 році та 175 зеттабайт до 2025 року.

Великий обсяг інформації та міграція в хмарні рішення

Таке швидке зростання обумовлене як збільшенням числа інтернет-користувачів, які роблять більше дій в Інтернеті (наприклад робота, банківська справа, покупки, соціальні мережі), так і мільярдами підключених інтелектуальних пристроїв і систем IoT. До них відносять не тільки смартфони, комп'ютери, смарт годинники і т.д, але і більші концепції розумних будинків, розумного транспорту або навіть розумних міст.

Такі величезні обсяги вимагають адекватної бази даних для зберігання і обробки інформації. До недавнього часу зібрані дані зберігалися в екосистемах з відкритим вихідним кодом, таких як Hadoop і NoSQL. В даний час все більше і більше компаній вирішують перейти на хмарні рішення, які забезпечують гнучкість, масштабованість і простоту використання.

Підвищена потреба у фахівцях з даних та аутсорсингових послуг

У міру зростання обсягу і різноманітності зібраних даних зростає потреба в найбільш підходящих системах і фахівцях по роботі з великими даними. Багато компаній стикаються з нестачею компетентного і досвідченого персоналу або мають справу з програмним забезпеченням, яке не працює належним чином. Одне з популярних рішень цієї проблеми на сьогоденні - це програмний аутсорсинг. Тобто якщо компанія не готова створити команду розробників всередині компанії, вона може найняти висококласних фахівців по роботі з великими даними в спеціалізованих аутсорсингових компаніях, де всю важку роботу будуть виконувати досвідчені фахівці та їхні команди розробників програмного забезпечення.

Інтеграція джерел

Великі дані об'єднують широкий спектр джерел даних, як онлайн, так і офлайн. Інтеграція всієї інформації у вигляді архітектурного накладення у всіх корпоративних системах має важливе значення. Це допомагає побачити загальну картину, краще зрозуміти характер зібраних даних і отримати корисні відомості,

корисні при прийнятті рішень і плануванні. Інтеграція джерел великих даних стане важливим кроком для багатьох компаній.

Проблеми з конфіденційністю

Проблеми безпеки та конфіденційності даних вже давно є серйозною проблемою. Зростаючі обсяги інформації необхідно захищати як від вторгнень, так і від кібератак. Проблема з безпекою має кілька причин:

- брак навичок безпеки (за даними журналу Cybercrime Magazine, до 2021 року кількість незаповнених посад в сфері кібербезпеки досягне 3,5 млн);
- швидкий розвиток кібератак (хакери постійно ускладнюють свої погрози);
- недотримання стандартів безпеки належним чином (багато організацій як і раніше ігнорують стандарти безпеки даних або використовують їх вибірково).

Для підтримки репутації компанії багато керівників вищої ланки і менеджери вважають конфіденційність даних своїм головним пріоритетом, поряд з безпекою і етикою даних.

Перспективи технології Big Data

Хмарні обчислення мають вирішальне значення з точки зору великих даних, оскільки компанії хочуть доставляти свої продукти і послуги швидше, ніж будь-коли. Для цього вони вже використовують сучасні архітектури мікросервісів, засновані на останніх методологіях DevOps. 2021 рік став роком розширеної аналітики. Організації та компанії гіганти вже застосовують машинне навчання, штучний інтелект і обробку природної мови на своїх платформах великих даних, щоб швидше приймати рішення та розпізнавати тенденції.

Очікується, що загальнодоступні і приватні хмари вперше будуть існувати разом в 2022 році. Завдяки технології 5G, мульти-хмарним IT-стратегіям та сучасним гібридним хмарним архітектурам підприємств зможуть поліпшити управління даними, видимість в реальному часі і безпеку зібраної інформації.

В 2022 році дані також стануть швидкими й ефективними. Концепція швидких даних дозволяє обробляти потоки в реальному часі. Це означає, що інформація аналізується швидко і миттєво приносить користь, наприклад, більш швидкий процес прийняття рішень і негайне вжиття заходів. За прогнозами IDC, до 2025 року майже 30% світових даних будуть передаватися в режимі реального часу.

Незважаючи на те, що нові рішення пов'язані з новими проблемами, технологія великих даних як і раніше буде приносити велику користь. Інформація з різних джерел допомагає побачити картину в цілому. Отримавши конкретну інформацію, ми можемо просто приймати більш обґрунтовані рішення, краще знати наших клієнтів і розбиратися в деталях різних, часто складних процесів. Великі дані дозволяють нам стежити за споживчими тенденціями або навіть створювати нові. Постійне зростання ринку великих даних створить нові можливості роботи для фахівців за даними, аналітиків і менеджерів. Все більше компаній усвідомлюють, що їм потрібно використовувати дані, що дозволяють діяти. Їм буде потрібно новий професійний підхід, що виходить також від зовнішніх постачальників програмного забезпечення.

2 ДОСЛІДЖЕННЯ МОДЕЛЕЙ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ОБРОБКИ ВЕЛИКИХ ДАНИХ

2.1. Lambda архітектура

Lambda архітектура – це основна архітектура у системах великих даних(рис.2.1). Більшість архітектур - це в основному архітектура Lambda або архітектура, заснована на її варіантах. Канал даних Lambda поділено на дві частини: потокова передача в реальному часі та офлайн.

Потокова передача в реальному часі в основному залежить від більшої частини потокової архітектури для забезпечення її продуктивності в реальному часі, в той час як офлайн - це пакетна обробка для забезпечення остаточної узгодженості. Щоб гарантувати ефективність обробки потокового каналу, інкрементні обчислення є основним допоміжним посиланням, тоді як рівень пакетної обробки виконує повні обчислення даних для забезпечення їхньої остаточної узгодженості.

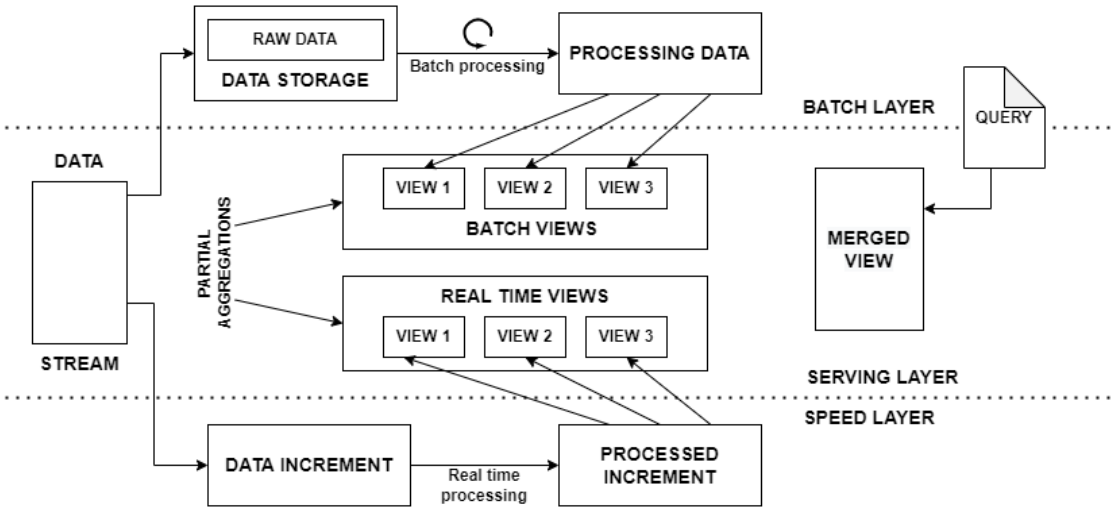


Рисунок 2.1. – Lambda Архітектура.

Щоб обробляти численні події, що відбуваються в системі Lambda, архітектура бере потік подій і поділяє / дублює його на два відносно незалежні рівні - рівень пакетної обробки та рівень швидкості.

Рівень пакетної обробки

Шар пакетної обробки відповідає за керування історичними даними та повторний розрахунок завдань для цих даних (наприклад, завдання скорочення картки або навчання моделі машинного навчання). Шар пакетної обробки приймає вхідні дані, поєднує їх з історичними даними та повторно обчислює результати шляхом ітерації по всьому набору даних.

На початку рівня пакетної обробки знаходиться сховище даних. Тут вихідні дані залишаються незмінними, це архів сирих даних. Найчастіше це дані моделі на основі фактів Data Lake на основі Hadoop, хоча, наприклад, це також відбувається у формі сховища даних на основі Vertica. Важливо, що тут залишаються незмінними - додаються лише нові дані. Навіть якщо ви втратите всі дані на рівні сервісу, ви можете відновити їх звідси.

Шар пакетної обробки працює з усім набором даних і, таким чином, дозволяє системі видавати найточніші результати. Однак результати досягаються за рахунок високої затримки через великий час обчислень.

Результатом цього шару будуть так звані пакетні перегляди. Найчастіше це денормалізовані уявлення даних, які краще підходять для бізнес-логіки.

Рівень швидкості

Рівень швидкості використовується для забезпечення результату з малою затримкою, близьким до реального часу. Рівень швидкості виконує інкрементні оновлення даних, які не були оброблені в останньому пакеті пакетного рівня. Це компенсує відмінності в релевантності даних і додає інформацію з коротким життєвим циклом (у будь-якому випадку її буде видалено при наступному запуску пакета) в окремі уявлення в реальному часі.

Завдяки цьому шару знижується вартість обробки даних – тому що нам не потрібно щоразу перераховувати всі доступні дані, а просто зосередитись на невеликій частині, яка надійшла нещодавно. Але оскільки ми не виконуємо повної повторної обробки даних, результати, ймовірно, будуть менш точними, ніж на рівні пакетної обробки.

Рівень обслуговування

Рівень обслуговування – останній компонент lambda архітектури. На цьому рівні запит спрямований на об'єднання та аналіз даних як з подання пакетного рівня, так і подання інкрементного потоку зі швидкісним рівнем.

Також часто згадується, що цей шар тісно пов'язаний із пакетним шаром. Це пов'язано з тим, що пакетний рівень постійно оновлює уявлення рівня обслуговування, і ці уявлення завжди будуть застарілими через велику затримку пакетної обробки. Але це не проблема, тому що рівень швидкості відповідатиме за будь-які дані, які ще не доступні на рівні обслуговування.

Обслуговуючий рівень індексує пакети та обробляє результати обчислень, що виконуються на попередніх рівнях, та при необхідності денормалізує їх. Через індексації та обробки інформації, що надходить, результати можуть запізнюватися в часі, це залежить від реалізації.

Тут може використовуватися практично будь-яка база даних, резидентна (у пам'яті) або постійна, включаючи спеціалізовані сховища, наприклад, для повнотекстового пошуку.

У результаті ми отримали масштабовану систему, стійку до випадкової втрати чи пошкодження даних. Він також може повторно обробляти дані, якщо вам потрібно оновити бізнес-логіку або виправити помилку. Це можливо, тому що базовий набір даних є незмінним і на його основі легко відновити стан системи.

Переваги та недоліки

Хоча архітектура Lambda надає безліч переваг, вона також привносить складність, пов'язану з необхідністю узгодження бізнес-логіки між потоковими та пакетними базами коду, що може ускладнити налагодження. Нам потрібно написати ту саму логіку в двох місцях, швидше за все, різними мовами. Це допоможе випускати нові версії та виправлення. Неважко написати його на пакетному рівні, де ми маємо всі складні інструменти, але передати його в потокову передачу може бути складно - проблеми будуть виникати навіть на рівні мовного перекладу.

Архітектура на основі Lambda також є більш стійкою та надійною завдяки розподіленій файловій системі, яка використовується для зберігання основного набору даних, головним чином тому, що вона менш схильна до людських помилок (наприклад, ненавмисне масове видалення). Кожен рівень архітектури масштабується незалежно, і Lambda-архітектуру можна легко узагальнити або розширити для великої кількості випадків використання, вимагаючи лише мінімального обслуговування. Ця архітектура забезпечує як аналіз даних у реальному часі за допомогою спеціального запиту переглядів у реальному часі, так і аналіз історичних даних.

Основна проблема архітектури Lambda полягає в підтримці синхронізації пакетного та швидкого рівнів шляхом регулярного відкидання останніх даних із рівня швидкості, щойно були передані до незмінного набору даних у пакетному рівні. Іншим обмеженням, яке слід пам’ятати, є той факт, що з рівня обслуговування можливі лише аналітичні операції; транзакційна операція неможлива. Нам також потрібно підтримувати дві схожі бази коду для рівня швидкості та для пакетного рівня, щоб виконувати однакові обчислення для різних наборів даних. Це призводить до надмірності і вимагає двох різних наборів навичок, щоб написати логіку для обох потреб.

Batch layer	HDFS (Storage) + MapReduce/PIG/Hive (Functions)
Speed Layer	Storm / S4 / Spark Streaming
Serving Layer	NoSQL database (Cassandra/ Hbase / CouchDB/ Voldemort/ MongoDB)
Queuing System	Apache Kafka / Flume

Рисунок 2.2. – Вимоги до програмного забезпечення Lambda архітектури.

Batch layer	1 replicated master node (6 cores CPU, 4 GB memory, RAID-1 storage, 64-bit operating system) 2 worker nodes (12 cores CPU, 4 GB memory, 2 TB storage, 1 GbE NIC) 1 dedicated resource manager (YARN) node (4 GB memory, and 4core)
Speed layer	Shares the Hadoop node
Serving layer	2 nodes (1TB, 4 cores, 16 GB memory)

Рисунок 2.3. – Вимоги до апаратного забезпечення Lambda архітектури.

2.2. Kappa архітектура

Архітектура Kappa спрощує архітектуру Lambda, видаляючи пакетний рівень та замінюючи його потоковим. Щоб зрозуміти, як це можливо, потрібно спочатку зрозуміти, що пакет - це набір даних з початком і кінцем (обмежений), у той час як потік не має початку та кінця і є нескінченним (необмеженим).

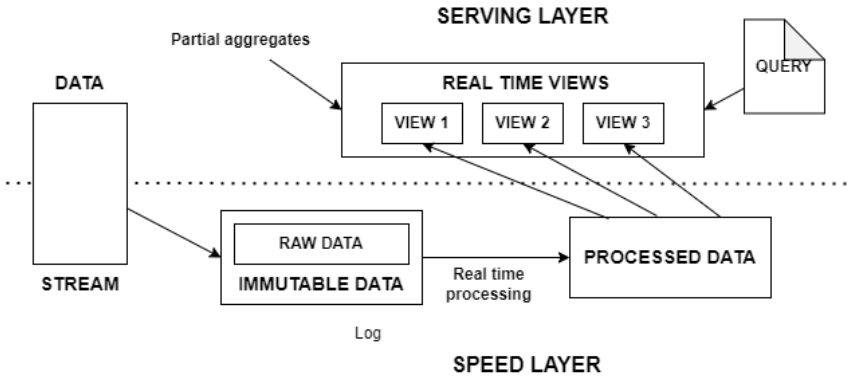


Рисунок 2.4. – Kappa Архітектура.

Оскільки пакетна обробка є обмеженим потоком, можна дійти невтішного висновку, що пакетна обробка є підмножиною потокової обробки. Отже, результати пакетної обробки Lambda також можуть бути отримані за допомогою механізму потокової передачі. Це спрощення зводить архітектуру до єдиного

механізму потокової передачі, здатного приймати необхідні обсяги даних для обробки пакетної обробки, так і обробки в реальному часі. Загальна складність системи значно зменшується із архітектурою Kappa(рис.2.4).

Також вирізняють такі основні принципи архітектури Kappa:

- Все є потоком: пакетні операції стають підмножиною потокових операцій. Отже, все можна розглядати як потік.
- Незмінні джерела даних: вихідні дані (джерело даних) зберігаються, а перегляди виводяться, але стан завжди можна перерахувати, оскільки початковий запис ніколи не змінюється.
- Єдина аналітична система: принцип коротко і просто (KISS). Потрібна єдина аналітична система. Код, обслуговування та оновлення значно зменшено.
- Функціональність відтворення: обчислення та результати можуть розвиватися шляхом відтворення історичних даних із потоку.

Щоб дотримуватись четвертого принципу, конвеєр даних повинен гарантувати, що події залишаються в порядку від покоління до надходження. Це важливо для гарантування узгодженості результатів, оскільки це гарантує результати детермінованих обчислень. Виконання одних і тих самих даних двічі за допомогою обчислень повинно привести до того ж результату.

Переваги та недоліки

Архітектура Kappa спрощує обробку даних. Вона поєднує в собі найкраще з обох світів, підтримуючи єдину базу коду, одночасно дозволяючи запитувати та аналізувати історичні дані за допомогою повторів потоків, коли код змінюється. Вона має менше рухомих частин, ніж архітектура Lambda, що також дозволяє створити просту модель програмування. Вхідні дані все ще можуть зберігатися в HDFS, але вже не так потрібно покладатися на них для виконання завдань повторної обробки історичних даних. З іншого боку, можливі лише аналітичні операції, а не транзакційні. Це не можна реалізувати з нативними хмарними сервісами, оскільки вони не підтримують потоки з тривалим терміном існування.

Крім того, дані зберігаються протягом обмеженого заздалегідь визначеного періоду часу, після чого вони відкидаються.

Ingestion tools	10 servers having each :12 physical processors, 16 GB RAM
Speed layer (Storm)	Minimum one server having : 16 GB RAM, 6 core CPUs of 2 GHz (or more) each, 4 x 2 TB, 1 GB Ethernet
Serving layer	Idem. To lambda architecture

Рисунок 2.5. – Вимоги до апаратного забезпечення Kappa архітектури.

2.3. Delta архітектура

Дивлячись на архітектури Kappa та Lambda. Ці архітектури призначені для підтримки великих обсягів даних як у режимі реального часу, так і в стані спокою. Ключовою відмінністю між цими двома архітектурами є наявність озера даних/хабу даних для консолідації всіх даних в одному місці. Lambda архітектура виглядає більш практичною, оскільки використовує дешевший носій для довгострокової пакетної обробки даних.

Однак Lambda архітектура використовує HDFS як озеро даних, і ключовою концепцією озера даних є незмінність. Це спричиняє накладні витрати на обробку даних на пакетному рівні.

Delta архітектура передбачає, що будь-які нові потокові записи обробляються як дельта (інкрементні) і не обробляються як нові записи. Концептуально цей шаблон схожий на Lambda, оскільки він заснований на швидкості та гарячому шляху. Одна велика відмінність полягає в тому, що дельта-архітектура більше не вважає озеро даних, що додається, і будь-яке пакетне перетворення може посилити пристрій даних в озері даних (дельта-запису процесу). Ця можливість полегшує обробку холодного шляху(рис.2.6).

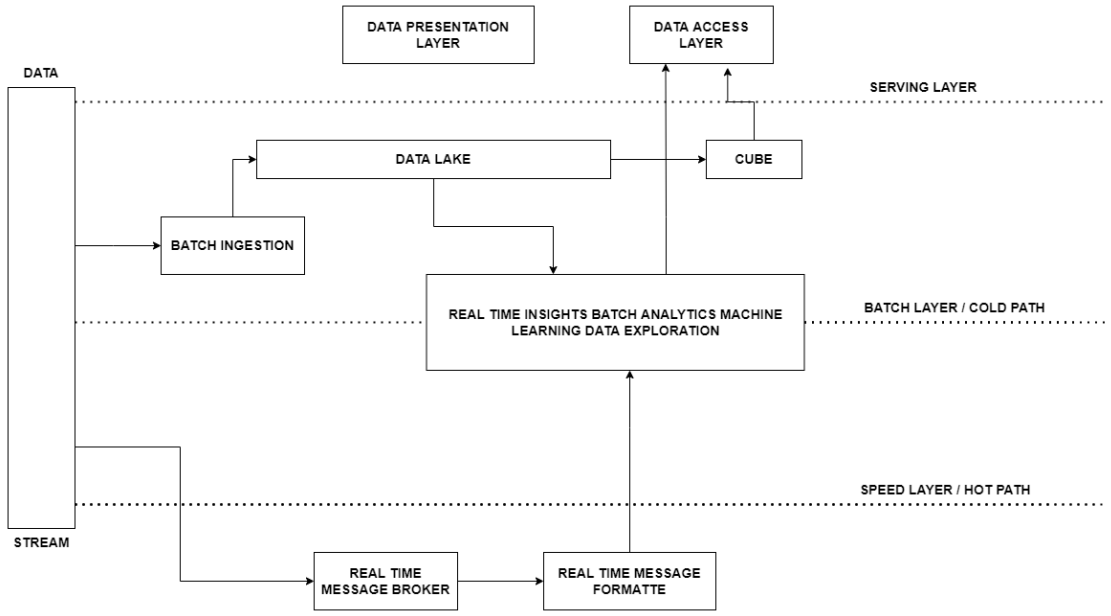


Рисунок 2.6. – Delta Архітектура.

З огляду на цю різницю, є величезна перевага. Це усуває розрив між пакетним і потоковим шаром і об’єднує їх для безперебійної обробки та менших накладних витрат. Організаціям більше не потрібно дивитися на обробку даних у розрізі, і їм не потрібно по-різному обробляти дані залежно від швидкості прийому та обробки.

Delta обробка

Delta в буквальному сенсі також використовується для позначення поступової зміни (як у Дельта в математиці). Поступові зміни в обробці даних відбуваються з будь-яких нових даних, створених/оновлених/потокових з часу останньої обробки. Вставки/оновлення даних можна об’єднати з наявними даними на рівні даних, а файли файлової системи можна оновити. Файлові системи тепер підтримують операції CRUD (Create/Read/Update/Delete) за наявною технологією, і файлову систему можна зробити сумісною з ACID (Atomicity, Consistency, Isolation, Durability). Наразі технологія Delta обробки зараз підтримується лише в Databricks Delta[7].

2.4. Data strimming архітектура

Архітектура потокових даних складається з програмних компонентів, створених і з'єднаних разом для прийому та обробки потокових даних із різних джерел. Архітектура потокових даних обробляє дані відразу після збору[8]. Обробка включає виділення їх у призначене сховище і може включати запуск додаткових етапів обробки, таких як аналітика, подальше маніпулювання даними або свого роду подальша обробка в реальному часі(рис.2.7).

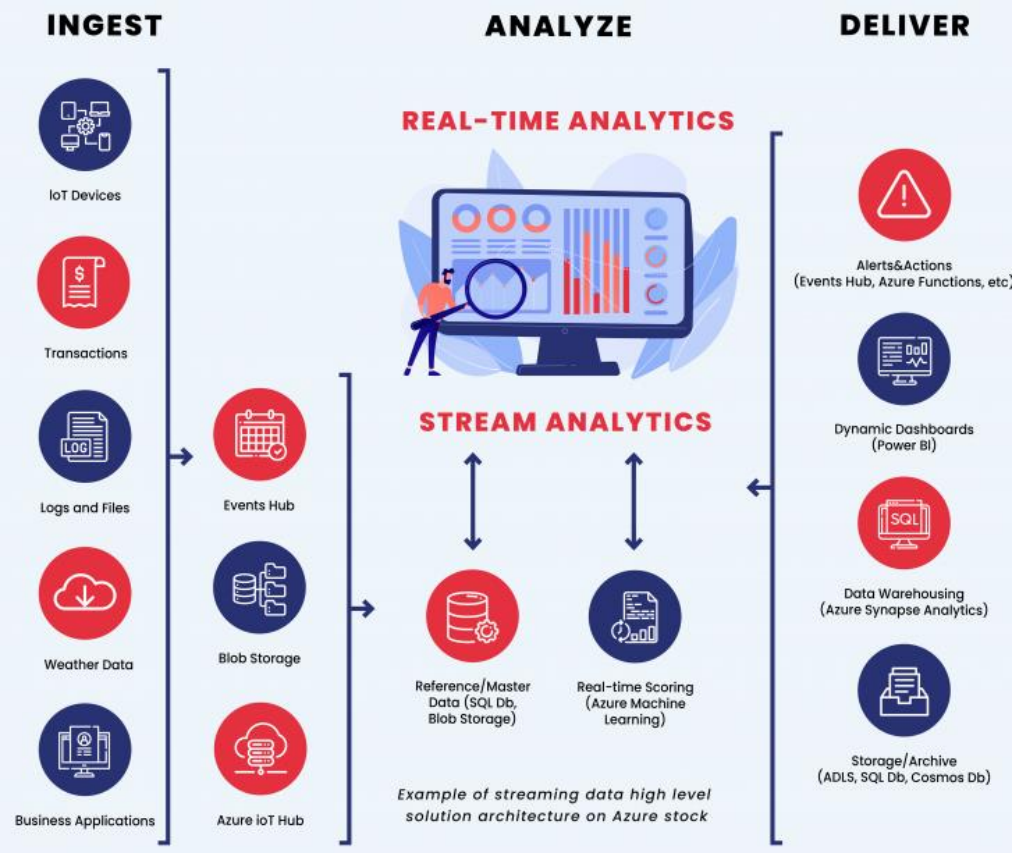


Рисунок 2.7 – Приклад архітектури потокових даних на основі хмарної платформи Microsoft Azure.

Є кілька підходів до обробки даних. По-перше, «старомодний» метод – це пакетна обробка, тобто одночасна обробка великого обсягу даних. Однак тут можна зосередитись на різниці між потоковою обробкою та операціями у реальному часі. Найпростіше пояснення полягає в тому, що операції в реальному

часі пов'язані з реакціями на дані, а потокова обробка - з діями, що робляться з даними(рис.2.8).

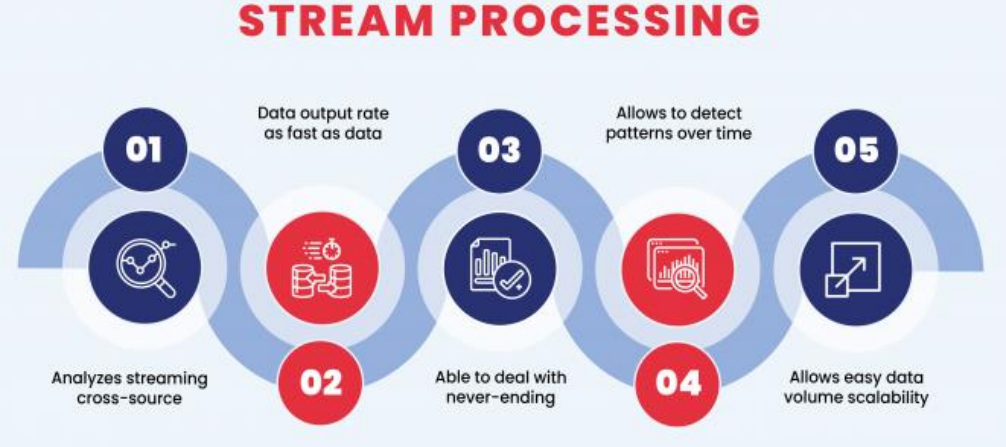


Рисунок 2.8 – Принцип потокової обробки даних.

Рішення у реальному часі гарантують виконання даних у стислі терміни після їх збору. Реакція на дані практично миттєва - обробка може зайняти хвилини, секунди або навіть мілісекунди, залежно від бізнес-вимог та застосовуваного рішення. Прикладами варіанта використання для роботи в режимі реального часу можуть бути операції купівлі/продажу на фондовому ринку, коли котирування має бути надано відразу після розміщення ордера.

Навпаки, потокова обробка - це безперервні обчислення, що відбуваються під час проходження даних через систему. Не існує обов'язкових обмежень за часом, крім потужності компонентів або технологічного рішення, що застосовується, а також стійкості бізнесу до затримок. Це означає, що немає крайнього терміну для виведення або реакції системи під час надходження даних, тому дані обробляються принаймні надходження. Успіх цієї архітектури залежить від двох речей: у довгостроковій перспективі швидкість виведення повинна бути, принаймні, дорівнює швидкості введення, і в неї має бути достатньо пам'яті для зберігання вводів, що вводяться в чергу, необхідних для обчислень. Потокова обробка корисна у разі частих подій, які слід відстежувати, або якщо події необхідно відразу виявляти та швидко реагувати. Прикладом використання може бути кібербезпека або виявлення шахрайства.

Основні компоненти архітектури потокових даних

Незалежно від платформи, архітектура потокових даних повинна включати такі основні компоненти архітектури потокових даних:

1. Посередники повідомлень

Група компонентів, яка приймає дані з джерела, перетворює їх у стандартний формат повідомлень і постійно передає їх у потоковому режимі, щоб зробити їх доступними для використання, що означає, що інші інструменти можуть прослуховувати і використовувати повідомлення, передані брокерами. . Популярні інструменти потокової обробки - це програмне забезпечення з відкритим вихідним кодом Apache Kafka або компоненти PaaS (платформа як послуга), такі як Azure Event Hub, Azure IoT Hub, GCP Cloud Pub/Sub або GCP Confluent Cloud, яка є хмарною платформою потокової передачі подій на базі Apache Kafka. Microsoft Azure також підтримує Kafka як тип кластера HDInsight, тому його можна використовувати як PaaS.

Крім прикладів, згаданих вище, є також інші доступні інструменти, такі як Solace PubSub + або Mulesoft Anypoint, які зазвичай побудовані на основі компонентів з відкритим вихідним кодом, щоб забезпечити повне середовище інтеграції з кількома хмарами, що підтримує, серед іншого, потокову передачу даних та дозволяє прибрати накладні витрати, пов'язані з налаштуванням та обслуговуванням платформи.

2. Інструменти для обробки

Потік вихідних даних від вищеописаного брокера повідомлень або потокового процесора необхідно перетворити і структурувати для подальшого аналізу з використанням інструментів аналітики. Результатом такого аналізу можуть бути якісь дії, попередження, динамічні інформаційні панелі або нові потоки даних.

Коли справа доходить до фреймворків з відкритим вихідним кодом, орієнтованих на обробку потокових даних, найбільш популярними і широко відомими є Apache Storm, Apache Spark Streaming і Apache Flink. Microsoft Azure підтримує розгортання Apache Spark та Apache Storm як типу кластера HDInsight.

На додаток до цього Azure надає власне рішення під назвою Stream Analytics, яке є чистим PaaS-компонентом Azure, що діє як аналітика в реальному часі та складний механізм обробки подій, призначений для аналізу та обробки великих обсягів даних швидкої потокової передачі з кількох джерел. одночасно. Таким чином, це підпадає під аналіз даних, а також обробку в реальному часі.

У разі Google Cloud Platform ключовими платформами для потокової обробки даних є Dataproc, який включає Spark та Flink, а також окреме пропрієтарне рішення – Dataflow.

3. Інструменти аналізу даних

Після того, як потокові дані підготовлені для використання потоковим процесором та інструментом обробки їх необхідно проаналізувати, щоб забезпечити цінність. Існує безліч різних підходів до потокової аналітики даних, але зупинимося на найвідоміших.

Apache Cassandra – це розподілена база даних NoSQL з відкритим вихідним кодом, яка забезпечує обслуговування потокових подій у додатках із малою затримкою. Потоки Kafka можна обробляти та зберігати у кластері Cassandra. Також можна реалізувати інший екземпляр Kafka, який отримує потік змін від Cassandra і передає їх іншим додаткам для прийняття рішень у реальному часі.

Інший приклад - Elasticsearch, який може безпосередньо отримувати потокові дані з тем Kafka. Завдяки формату даних Avro та реєстру схем зіставлення Elasticsearch створюються автоматично, і в Elasticsearch можна виконувати швидкий текстовий пошук чи аналітику.

Azure також надає CosmosDB з Cassandra API, тому можливості Apache Cassandra захищені в цій хмарі. GCP підтримує область з базою даних Firebase Realtime, Firestore та BigTable.

4. Зберігання потокових даних.

Вартість сховища загалом щодо невисока, тому організації зберігають свої потокові дані. Озеро даних - найгнучкіший і найдешевший варіант зберігання даних про події, але його досить складно правильно налаштувати та підтримувати. Він може включати відповідну обробку даних, поділ даних та зворотне заповнення

історичними даними, тому зрештою створення робочого озера даних може стати проблемою.

Усі постачальники хмарних послуг надають відповідні компоненти, що виступають як озера даних. Azure надає Azure Data Lake Store (ADLS), а Google Cloud Platform має хмарне сховище Google.

Інший варіант може полягати у зберіганні даних у сховищі даних або постійному сховищі вибраних інструментів, таких як Kafka, Databricks / Spark, BigQuery.

Підводячи підсумки можна сказати, що сучасна архітектура потокових даних має кілька важливих переваг, які слід враховувати при розробці конкретних рішень. Коли архітектура правильно спроектована, вона усуває необхідність у проектуванні великих даних, її продуктивність висока, її також можна швидко розгорнути, забезпечуючи високу доступність та стійкість до відмов. Ця архітектура також є гнучкою для підтримки кількох сценаріїв використання із низькою сукупною вартістю володіння ресурсами.

Для досягнення вищезгаданого компанії можуть використовувати повні рішення або розробити відповідні архітектурні схеми, щоб забезпечити швидку та надійну доставку рішень, адаптованих до потреб бізнесу.

2.5. Microservice архітектура

Використання мікросервісів є підходом, який використовується в сучасній розробці програмного забезпечення для підвищення гнучкості, великі дані дозволяють організаціям перетворити сучасний інформаційний потік на цінну інформацію. Багато з цих архітектур великих даних мають досить монолітні елементи. Однак виникає нова тенденція, в якій монолітні архітектури замінюються більш модульними, такими як мікросервіси. Ця трансформація забезпечує переваги мікросервісів, такі як модульність, еволюційний дизайн і

розширюваність, зберігаючи при цьому функціональність старого монолітного продукту. Це також справедливо для архітектур Big Data.

Система, заснована на архітектурі мікросервісів(рис.2.9), складається з набору слабо пов'язаних служб, які можуть працювати незалежно і спілкуватися один з одним через веб-сервіси REST, віддалені дзвінки або Push-повідомлення. Кожна служба реалізована за допомогою інструментів і мови програмування, які найбільше підходять для її призначення, і працює на виділеному сервері з виділеним сховищем. Основна відмінність між архітектурою мікросервісів і простою системою на основі сервіс-орієнтованої архітектури полягає в тому, що тут кожна служба зосереджена на виконанні лише одного конкретного завдання і являє собою автономним додатком.

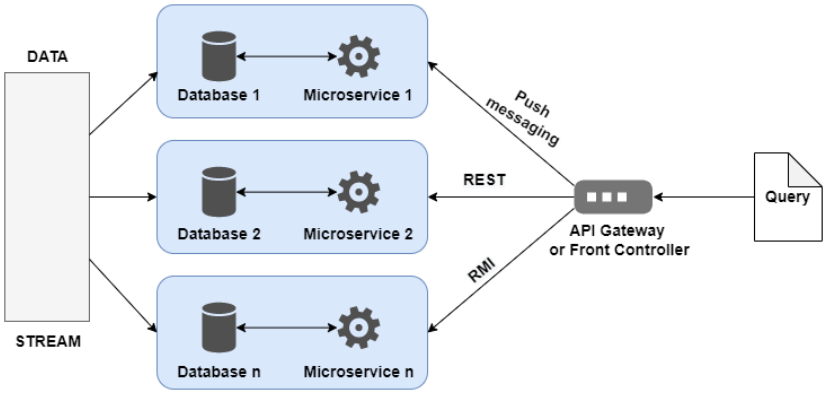


Рисунок 2.9. – Microservice архітектура.

Переваги та недоліки

У порівнянні з монолітними системами на основі мікросервісів системи дозволяють швидше розробляти, швидше здійснювати тестувати і розгортання, оскільки кожна служба невелика й незалежна від інших, таким чином, легше зрозуміти. Відмовостійкість вища, і сервіс можна (пере)записати в будь-який момент з використанням новітніх технологічних стеків без шкоди для інших служб. Різні команди можуть працювати більш ефективно, наділяючи конкретні послуги кожній. Крім того, послуги можна повторно використовувати в бізнесі, і будь-яку функцію можна масштабувати незалежно від інших. З іншого боку, розробка є

складною, і існує потреба у сильній координації команди, механізмі міжсервісного зв'язку та заходах безпеки для мережі.

Коли два сервіси, що використовують два різних технологічних стека, повинні взаємодіяти, зміни формату (упорядкування та розмаршування) також створюють накладні витрати. Кожна служба зазвичай працює у своєму власному контейнері (можливо, JVM), тому загальне споживання пам'яті набагато вище, ніж потрібно для монолітного додатка.

Container management system	**1 boot node (1+ core, 4 GB RAM, 100+ GB storage)**
	1, 3 or 5 master nodes (2+ cores, 4+ GB RAM, 151+ GB storage)
	1, 3 or 5 proxy nodes (2+ cores, 4 GB RAM, 40+ GB storage)
	1+ worker nodes (1+ cores, 4GB RAM, 100+GB storage)
	1+ optional management node (4+ cores, 8+ GB RAM, 100+ GB storage)
	2.4 GHz cores recommended
CI/CD	1 node (1+ GB RAM, 50+ GB storage)

Рисунок 2.10. – Вимоги до апаратного забезпечення мікросервісної архітектури.

Container	OpenShift (Docker based containers)
Distributed Version Control System	Git
Continuous Integration tool	Jenkins / GitLab CI, Buildbot, Drone, Concourse

Рисунок 2.11. – Вимоги до програмного забезпечення мікросервісної архітектури.

2.6. Zeta архітектура

Архітектура Zeta пропонує новий підхід, у якому технологічне рішення компанії безпосередньо інтегрується з архітектурою бізнесу/підприємства. До цієї

архітектури можна «підключити» будь-яку програму, яку потребує бізнес. Він забезпечує контейнери, які є ізольованими середовищами, в яких програмне забезпечення може запускатися та взаємодіяти разом незалежно від несумісності платформи. Вона описана на (рис.2.12).

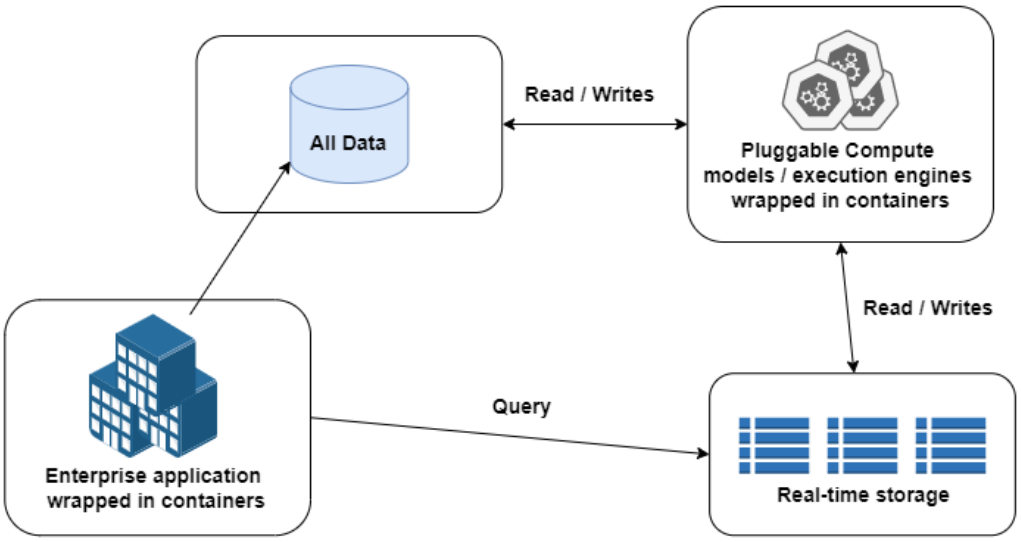


Рисунок 2.12. – Zeta архітектура.

Переваги та недоліки

Оскільки апаратне забезпечення не призначене спеціально для певного набору послуг, а є загальним для всієї системи, його краще використовувати, і його можна використовувати для задоволення найнагальніших потреб у будь-який момент.

Резервне копіювання майже в реальному часі допомагає уникнути тривалих періодів відновлення після збоїв, а діагностика проблем відбувається швидше. Створення двійкових файлів, які можна легко розгорнути в будь-якому середовищі без необхідності їх модифікації, полегшує тестування та розгортання.

Рекламна платформа, заснована на архітектурі Zeta, представлена в [21], показує, як посередники придушуються за допомогою журналів, які безпосередньо зберігаються, зчитуються та обробляються з тієї ж розподіленої файлової системи.

Distributed File System	Hadoop DFS
Real-time storage	NoSQL/NewSQL (HBase, MongoDB…)
Compute model engine	MapReduce, Spark, Apache Drill
Resource Manager	Apache Mesos, YARN
Container Management System	Docker, Kubernetes, Mesos

Рисунок 2.13. – Вимоги до програмного забезпечення Zeta архітектури.

2.7. IoT архітектура

З практичної точки зору Інтернет речей (IoT) являє собою будь-який пристрій, підключений до Інтернету. Це включає ваш комп'ютер, мобільний телефон, розумний годинник, розумний термостат, розумний холодильник, підключений автомобіль, імплантати для моніторингу серця та все інше, що підключається до Інтернету та надсилає чи отримує дані. Кількість підключених пристроїв зростає з кожним днем, як і кількість зібраних з них великих даних. Часто ці дані збираються в дуже обмежених середовищах, іноді з великою затримкою. В інших випадках тисячі або мільйони пристроїв надсилають дані із середовищ з низькою затримкою, що вимагає можливості швидкого отримання даних та відповідної обробки. Тому для вирішення цих обмежень та унікальних вимог потрібне правильне планування.

Архітектури, керовані подіями, є центральними для рішень IoT. Наступний (рис.2.14) показує можливу логічну архітектуру для Інтернету речей. На діаграмі підкреслені компоненти архітектури, що передають події.

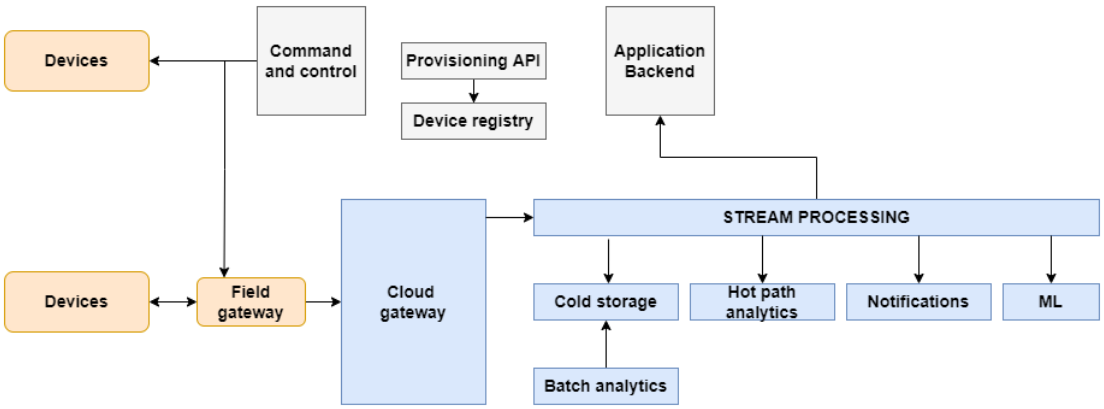


Рисунок 2.14. – IoT архітектура.

Хмарний шлюз приймає події пристрою на межі хмари, використовуючи надійну систему обміну повідомленнями з низькою затримкою.

Пристрої можуть надсилати події безпосередньо на хмарний шлюз або через польовий шлюз. Польовий шлюз — це спеціалізований пристрій або програмне забезпечення, зазвичай спільно з пристроями, яке отримує події та пересилає їх до хмарного шлюзу. Польовий шлюз також може попередньо обробляти вихідні події пристрою, виконуючи такі функції, як фільтрація, агрегація або перетворення протоколу.

Після прийому події проходять через один або кілька потокових процесорів, які можуть направляти дані (наприклад, у сховище) або виконувати аналітику та іншу обробку.

Нижче наведено деякі поширені типи обробки:

- Запис даних про події в холодне сховище, для архівування або пакетної аналітики.
- Аналітика гарячого шляху, аналізуючи потік подій у реальному часі, щоб виявляти аномалії, розпізнавати закономірності за вікнами часу або ініціювати сповіщення, коли в потоці виникає певна умова.
- Обробка особливих типів нетелеметричних повідомлень від пристроїв, таких як сповіщення та нагадування.
- Машинне навчання.

Полі, затінені сірим кольором, показують компоненти системи Інтернету речей, які не мають прямого відношення до потокової передачі подій:

- Реєстр пристроїв — це база даних наданих пристроїв, включаючи ідентифікатори пристроїв і зазвичай метадані пристроїв, наприклад, розташування.
- API надання є загальним зовнішнім інтерфейсом для надання та реєстрації нових пристроїв.
- Деякі рішення IoT дозволяють надсилати на пристрої командні та контрольні повідомлення.

Вимоги до апаратного та програмного забезпечення

Рівень MQ/SP (черга повідомлень і обробка потоку) може бути реалізований за допомогою Apache Kafka або fluentd для збору даних і Apache Spark або Storm для їх обробки. Інтерактивний рівень зберігання може бути реалізований за допомогою будьякої бази даних NoSQL разом із такими інструментами, як Apache Drill для взаємодії з нею. Рівень DFS може використовувати HDFS разом із Hive та Apache Mahout для машинного навчання над основним набором даних. Вимоги до обладнання для кожного з компонентів цієї архітектури вже були описані в попередніх архітектурах.

2.8. Загальна оцінка основних моделей архітектур інформаційних систем для обробки великих даних

У цій магістерській роботі мною було представлено загальну оцінку основних моделей архітектур інформаційних систем для обробки великих даних в цілому(2.15). Дана робота може бути додатково розширена за допомогою більшої кількості архітектур за умови, що будуть доступні дані щодо їх продуктивності та вимоги у реальних випадках використання. Тим не менш, навіть на цьому етапі

можна сподіватися, що дана магістерська дипломна робота вкладе вагомий внесок щодо в подальшій ефективності розробки систем Big Data.

Архітектури	Lambda	Kappa	Delta	Data strimming	Microservice	Zeta	IoT
Режим обробки	пакетний / у реальному часі	у реальному часі	пакетний / у реальному часі	у реальному часі	пакетний / у реальному часі	пакетний / у реальному часі	пакетний / у реальному часі
Методологія обробки даних	запит і звітність	запит і звітність	запит і звітність	запит і звітність	запит і звітність / аналітичний / прогнозний аналіз	запит і звітність	запит і звітність / аналітичний / прогнозний аналіз
Частота передачі даних	постійно у реальному часі	безперервна	за запитом	постійно у реальному часі	за запитом	за запитом	за запитом

Рисунок 2.15. – Оцінка основних моделей архітектур інформаційних систем для обробки великих даних.

3 РОЗРОБКА АРХІТЕКТУРИ ВЕЛИКИХ ДАНИХ ДЛЯ РОЗПІЗНАВАННЯ ОСІБ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ МАШИННОГО НАВЧАННЯ

3.1. Розпізнавання обличчя за допомогою машинного навчання

Навчання складного розпізнавання обличчя на зображеннях та відео може зайняти години, дні або навіть тижні. Найчастіше однієї машини з кількома графічними процесорами достатньо навчання великих моделей за розумний час. Однак для більш вимогливих робочих навантажень розпізнавання осіб у реальному часі розподіл обчислювальних навантажень між кількома машинами може значно скоротити час навчання, забезпечуючи швидке ітеративне експериментування та прискорюючи розгортання глибокого навчання. Таким чином, архітектури обробки великих даних та платформи паралельної обробки, такі як MapReduce та Spark, відіграють ключову роль у таких областях.

Існує два основних підходи до покращення розпізнавання обличчя, а саме паралельність моделі та паралельність даних. Архітектура великих даних спирається на розподілені кластерні обчислення для завдань попередньої обробки та класифікації неоднорідних і розрізнених потоків великих даних, що виходять з різних джерел даних. У цьому контексті паралелізму даних можна досягти за допомогою звичайного стеку Hadoop MapReduce, як це пропонується в нашому технічному стеку. Однак, коли справа доходить до паралельності моделей, фреймворки пам'яті відіграють ключову роль. Таким чином, ми пропонуємо використовувати обробку Spark в пам’яті для досягнення паралельності моделі навчання.

Для підтримки паралельності даних і моделей ми використовуємо загальний вибір технологій з відкритим вихідним кодом, а саме розподілену файлову систему Hadoop та HBase для збору даних з різнорідних джерел даних у запропонованій нами архітектурі великих даних. Наш підхід полягає у використанні гібриду

паралельності даних і паралельності моделей, показаних на (рис.3.1), де паралелізму можна досягти за допомогою системи MapReduce Hadoop.

MapReduce - це модель програмування для обробки великих наборів даних, які використовуються в різноманітних реальних завданнях. Розробники можуть вказати обчислення в термінах карти та функції зменшення, а базова система часу виконання автоматично паралелізує обчислення між великомасштабними кластерами машин, обробляє збої машин і планує між машинний зв'язок для ефективного використання мережі та диски[9]. Однак дослідження в минулому зосереджувалися лише на пакетній реалізації розпізнавання обличчя на основі функцій за допомогою MapReduce[10]. Однак з міркувань продуктивності було вибрано Spark замість MapReduce для алгоритмів розпізнавання обличчя, реалізованих у цій роботі.

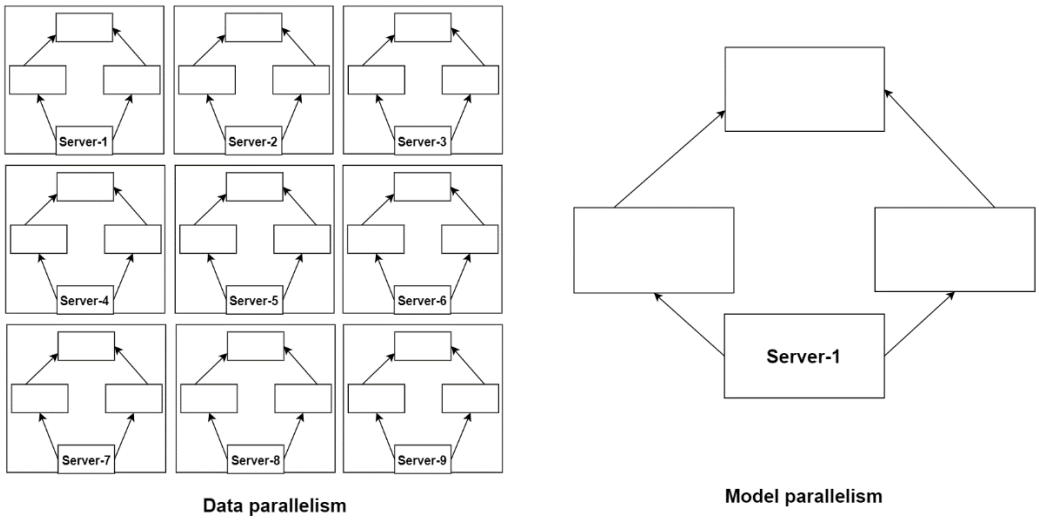


Рисунок 3.1. – Порівняння паралелізму даних з паралелізмом моделі.

Apache Spark забезпечує високу продуктивність як пакетних, так потокових даних, використовуючи сучасний планувальник DAG (Directed Acyclic Graph), оптимізатор запитів і механізм фізичного виконання. Дослідження проведене Hazarika[11] вказує, що Spark може вирішувати проблеми навчання великих наборах даних зображень, використовуючи бібліотеки машинного навчання з ітеративним кешуванням у пам'яті, які працюють поверх Spark Data Frames. У деяких випадках Spark перевершує Hadoop MapReduce за швидкістю обчислень в

10 разів у ітеративних завданнях машинного навчання та до 20 разів у ітеративних додатках. Це пов'язано з тим, що Spark використовує обробку пам'яті в порівнянні з MapReduce, який повинен читати та здійснювати запис на диск.

3.2. Алгоритми класифікації великих даних для інтелектуального аналізу

K-найближчі сусіди (k-NN)

Класифікатор k-NN(k-nearest neighbors algorithm) повільно і лінивий навчається, оскільки він використовує весь навчальний набір щоразу, коли виконується тестування прикладу. Під час навчання класифікатора k-NN він демонструє часову складність O(1). Коли новий екземпляр класифікується за навчальним набором з m екземплярів і n атрибутами, він демонструє тимчасову складність O(mn+mlog(m)), де O(mn) – це часова складність для обчислення косинусоподібної подібності між тестом, а O(mlog(m)) – це часова складність для сортування подібності під час пошуку k-найближчих сусідів нового прикладу. Підводячи підсумок, зі збільшенням розміру даних час обробки k-NN буде зростати експоненціально.

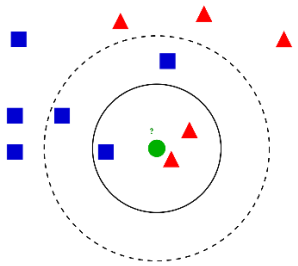


Рисунок 3.2. – Приклад роботи алгоритму k-NN.

Розподілена обробка Hadoop з реалізацією MapReduce класифікатора k-NN (MR-KNN) була запропонована шляхом відображення навчальних прикладів із наступним зменшенням кількості прикладів, найближчих до тестового прикладу[12]. Повідомлялося, що MR-KNN прискорився з 16 до 149 разів під час тестування на наборі даних з 1 мільйоном прикладів. Прискорення запропонованої реалізації залежало головним чином від кількості використаних карт і k сусідів. Ці

результати були додатково покращені за допомогою реалізації Iterative Spark (kNN-IS), подібної до фреймворку MR-KNN, але з використанням кількох редукторів для прискорення процесу агамаегації результату[13].

Серед популярних методів класифікації в запропонованій моделі архітектури великих даних буде використовуватися алгоритм k-NN, оскільки він найбільш підходить як для аналізу зображень, так і для тексту в порівнянні з іншими алгоритмами ML, такими як Naive Bayes, Support Vector Machines, які зміщені для одного формату даних над іншим. Крім того, алгоритм k-NN використовує підхід лінивого навчання, який імітує навчання шляхом запам’ятовування, коли всі навчальні дані завантажуються в простір ознак. Прогноз виводиться шляхом ітераційного порівняння нової точки даних із усім простором об’єктів, щоб наблизити її найближчого сусіда (передбачуваний клас). Цей підхід контрастує з типом алгоритмів, що користуються ентузіазмом учнів, які будують модель на основі минулих спостережень (навчальних даних). Він також здатний узагальнити в одну гіпотезу, яка може охопити весь простір ознак для розпізнавання обличчя.

Основною проблемою під час використання k-NN є ненадійні обчислення через великі обсяги пам’яті, необхідні для завантаження та обчислення всього простору функцій у пам’яті. Розробка моделі розпізнавання обличчя, яка може всебічно представляти різні функції, включаючи словниковий запас слів для асоціації текстової інформації та інших метаданих, зазвичай доступних із зображеннями обличчя, може стати інтенсивною обчислювальною роботою. Такі нові міркування багатогранних функцій із різних колекцій великих даних, включаючи соціальні мережі, можуть покращити класифікацію зображень. Таким чином, проблема великого сліду пам’яті k-NN стає набагато складнішою для додатків великих даних, таких як розпізнавання обличчя. У цьому контексті, серед різних алгоритмів класифікації, k-NN принесе більше переваг для досягнення бажаної обчислювальної продуктивності в реальних додатках.

Кластеризація K-середніх

Однією з найпростіших методик ML для поділу набору даних є k-means, типовий неієрархічний алгоритм кластеризації. Його мета — розділити вибірку на заздалегідь визначену кількість (k) кластерів, що не перекриваються, щоб кластери були максимально однорідними щодо використовуваних вимірювань.

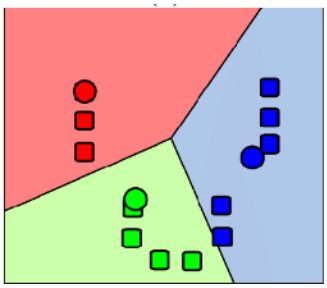


Рисунок 3.3 – Приклад роботи алгоритму k-means.

Алгоритм k-середніх є дуже ефективним і, можливо, найшвидшим алгоритмом кластеризації, який може обробляти як довгі (багато записів), так і широкі набори даних (багато полів введення). На основі кластеризації навчального набору з використанням алгоритму кластеризації k-середніх, нещодавня робота запропонувала використовувати орієнтирну спектральну кластеризацію (LC-KNN) для збільшення швидкості k-NN[14]. Після звуження найбільш релевантного кластера послідовний k-NN був застосований з меншого набору прикладів до тестового прикладу. LC-KNN було оцінено на дев’яти величезних наборах даних, що показують розумне наближення. В іншому дослідженні запропоновано процес збільшення кластера (cKNN) для прискорення швидкості k-NN[15]. Повідомлена середня точність для різних наборів даних була в діапазоні від 83% до 90%, залежно від кількості використовуваних кластерів. Крім того, ці результати були покращені, коли для вивчення репрезентативних ознак для класифікації використовувалися глибокі нейронні мережі (DNN).

Дерево рішень

Алгоритм дерева рішень належить до сімейства алгоритмів навчання з наглядом. На відміну від інших алгоритмів навчання з наглядом, алгоритм дерева

рішень також можна використовувати для вирішення проблем регресії та класифікації.

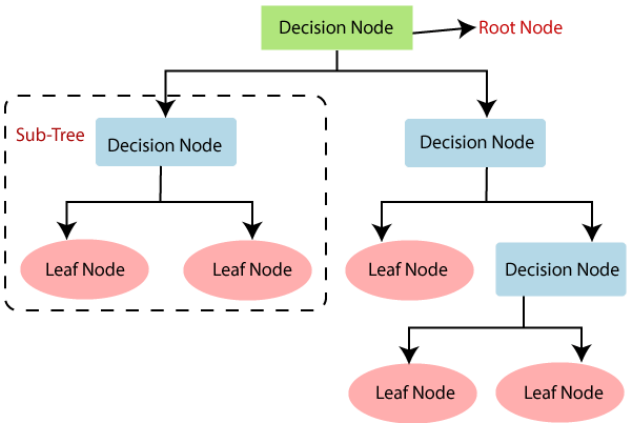


Рисунок 3.4 – Приклад роботи алгоритму дерева рішень.

Метою використання дерева рішень є створення навчальної моделі, яка може використовуватися для прогнозування класу або значення цільової змінної шляхом вивчення простих правил прийняття рішень, виведених з попередніх даних (навчальних даних).

У деревах рішень для передбачення мітки класу для запису ми починаємо з кореня дерева. Ми порівнюємо значення кореневого атрибута з атрибутом запису. На основі порівняння ми йдемо по гілці, що відповідає цьому значенню, і переходить до наступного вузла.

Типи дерев рішень базуються на типі цільової змінної, яку ми маємо. Він може бути двох видів:

- Дерево рішень категорійних змінних (дерево рішень, яке має категоріальну цільову змінну, потім воно називається деревом рішень категорійних змінних).
- Дерево рішень безперервних змінних (дерево рішень має безперервну цільову змінну, тоді воно називається деревом рішень безперервних змінних).

3.3. Запропонована модель архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання

Запропонована нами система обробки великих даних досить потужна, щоб ідентифікувати чи перевіряти обличчя чи розуміти вираз обличчя цифровими зображеннями і відео, що зазвичай називаємо розпізнаванням осіб.

Ця система працює шляхом порівняння найбільш поширених і помітних рис особи на даному зображенні (зібраному в реальному часі з різних зовнішніх джерел) з особами, що зберігаються у базі даних. Система розпізнавання осіб також здатна розпізнавати закономірності та варіації на основі текстури та форми особи людини, щоб однозначно розпізнавати людину.

На (рис.3.5) представлена архітектурна схема запропонованого нами рішення обробки великих даних. На ньому показана архітектура наскрізного прототипу для збору, очищення, обробки та візуалізації результатів розпізнавання осіб для отримання практичних відомостей (зображення розглядається як неструктуровані великі дані. У нашому випадку використання).

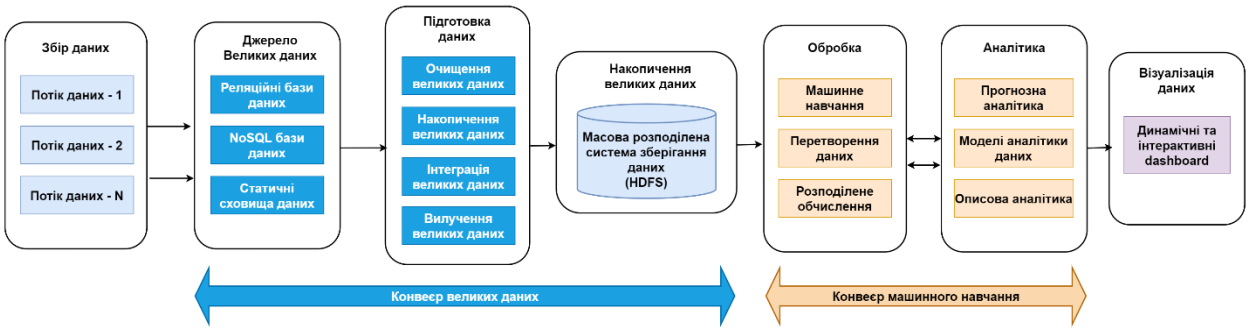


Рисунок 3.5. – Модель архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання.

Пропонована нами архітектура поєднує та автоматизує рівні прийому, зберігання, обробки та спеціального аналізу. Прототип складається з двох основних робочих процесів для ефективної обробки різних потоків даних, а саме:

1) Конвеєр великих даних;
2) Конвеєр машинного навчання;

Конвеєр великих даних

Основна роль конвеєра великих даних полягає в автоматизації переміщення величезних даних з різними типами даних із зовнішніх джерел потоків даних до озера даних для подальшого аналізу. Крім переміщення даних, ми також використовуємо дані з традиційних реляційних баз даних, баз даних NoSQL та статичних сховищ даних, таких як файли: jpg, mp3, mp4, json, щоб їх можна було ефективніше аналізувати наступними процесами в озері даних. Ключові інструменти та методи реалізації конвеєра великих даних для запропонованої нами архітектури великих даних полягають у тому, що він може відповідати наступним цілям:

- для підключення декількох типів даних та сховищ файлів (наприклад, реляційних баз даних та баз даних NoSQL, сховищ статичних файлів для неструктурованих та напівструктурованих даних);
- для забезпечення підтримки сценаріїв, що настроюються, для реалізації перевірки якості даних і попередньої обробки на етапі прийому даних.

Конвеєр машинного навчання

Основна роль конвеєра машинного навчання – допомогти автоматизувати машинне навчання та налаштування параметрів робочих процесів, оскільки машинне навчання відноситься до шаблонів навчання даних. Іншими словами, машинне навчання може вивести закономірність або нетривіальні відносини між набором спостережень та бажаною відповіддю. Ключові інструменти та методи реалізації конвеєра машинного навчання для пропонованої нами архітектури полягають у тому, що він може відповідати наступним цілям:

- для підтримки розподілених обчислень, щоб отримати масштабований конвеєр;
- для надання функцій та API для реалізації аналізу функцій та ітеративних алгоритмів машинного навчання;

Конвеєр великих даних, показаний на (рис.3.6), збирає дані з різних джерел в один потік і готується до подальшого тестування в наступному конвеєрі ML. Наприклад, попередня обробка розпізнавання обличчя повинна мати можливість отримати область інтересу, а потім виявити особливості за допомогою гістограми орієнтованих градієнтів. Потім дані зображення обличчя кодуються для значень RGB, а обмежуючий прямокутник малюється на межі для кожної області обличчя, виявленої для візуалізації в поточному кадрі.

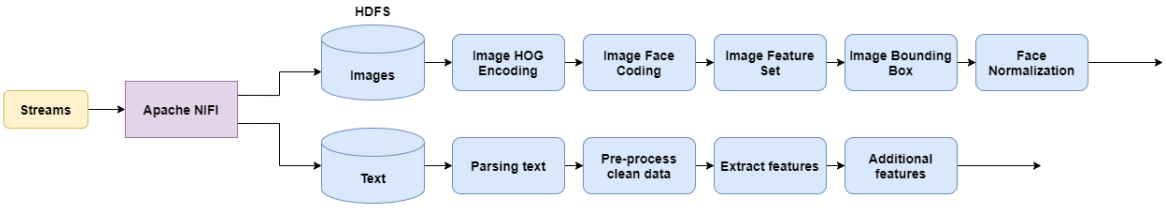


Рисунок 3.6. – Принцип роботи конвеєру великих даних.

Згодом реалізується нормалізація для сприйняття обличчя за різних умов освітлення за допомогою техніки вирівнювання гістограми. Спочатку зображення RGB перетворюється на колірний простір YUV для роз'єднання яскравості (Y) і хроматичної площини (UV). Потім вирівнювання гістограми застосовується до каналу яскравості Y. Формула для вирівнювання гістограми H_k задається таким чином:

$$H_k = \left[\frac{cdf_k - cdf_{min}}{t - cdf_{min}}(d - 1)\right], k \epsilon 0, \dots d-1$$

де cdfk – набір кумулятивної функції щільності, t – загальна кількість пікселів для зображення обличчя, а d – глибина кольору в каналі Y. Вирівнювання гістограми дозволяє виділити більше рис обличчя. Потім вирівняне зображення обличчя перетворюється назад на зображення RGB для подальшої обробки в конвеєрі машинного навчання.

Для фактичного запропонованого конвеєра машинного навчання (рис.3.7) наше ключове рішення було зосереджено на його реалізації за допомогою технології Spark. Запропонована архітектура буде використовувати API Pipelines від Spark та налаштовувати відповідні програми для автоматизації завдань

машинного навчання, необхідних для процесу виконання функцій на основі зображень та етапів аналізу. Також буде використовуватись робочий процес ML шляхом інтеграції Spark Transformers та Estimators як послідовності конвеєрних програм за допомогою Spark ML. У той час як Transformer включає в себе методи трансформації ознак та вивчені моделі, Estimator включає в себе підготовку моделі навчання та навчання на попередньо оброблених даних обличчя (рис.3.8). Процес робочого процесу та аналіз зображень облич у конвеєрі ML – надає робочий процес етапу процесу та аналізу в конвеєрі ML для випадку використання розпізнавання обличчя.

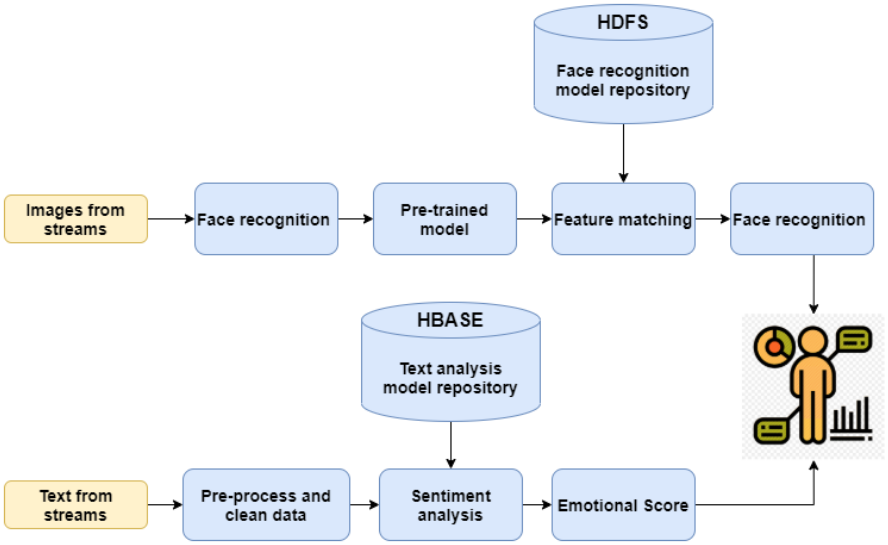


Рисунок 3.7. – Принцип роботи конвеєру машинного навчання.

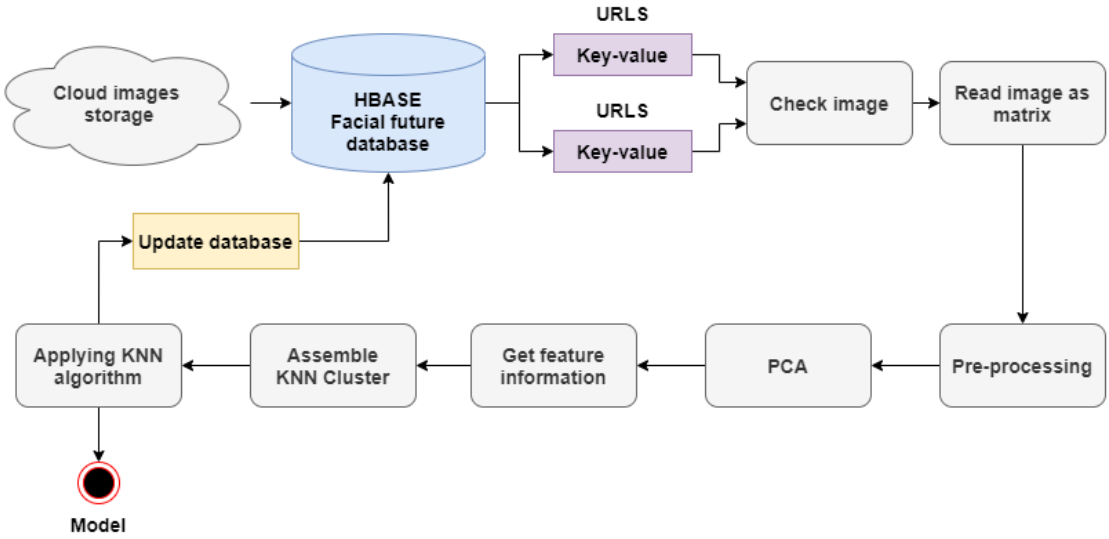


Рисунок 3.8. – Етапи процесу підготовки моделі за допомогою Estimator у контексті Spark ML.

Використовуючи вищезгадані конвеєри, запропонована архітектура великих даних є достатньо загальною, щоб використовувати різні вхідні джерела великих даних для збору інформації з потоків даних, обробляти, аналізувати великі масиви даних у реальному часі.

3.4. Принципи побудови моделі ML для розпізнавання осіб за допомогою розподіленої системи глибокого навчання BigDL

BigDL – це розподілена система глибокого навчання для платформ та робочих процесів великих даних. Вона реалізована як бібліотека поверх Apache Spark та дозволяє користувачам писати свої програми глибокого навчання як стандартні програми Spark, які працюють безпосередньо на існуючих кластерах великих даних (Apache Hadoop або Spark). Вона підтримує API, подібний до Torch і Keras [16] для побудови моделей нейронної мережі (рис.3.9). Вона також підтримує як широкомасштабне розподілене навчання, так і висновки, використовуючи масштабну архітектуру базової платформи Spark (яка ефективно працює на сотнях або тисячах серверів).

```
#distributed data processing
spark = SparkContext(appName="text_classifier", ...)
input_rdd = spark.textFile("hdfs://...")
train_rdd = input_rdd.map(lambda x: read_text_and_label(x))
                     .map(lambda data: decode_to_ndarrays(data))
                     .map(lambda arrays: to_sample(arrays))

#distributed training
model = Sequential().add(Recurrent().add(LSTM(...)))
                    .add(Linear(...)).add(LogSoftMax())
optimizer = Optimizer(model=model, training_rdd=train_rdd,
                      criterion=ClassNLLCriterion(),
                      optim_method=Adagrad(), ...)
trained_model = optimizer.optimize()

#distributed inference
test_rdd = ...
prediction_rdd = trained_model.predict(test_rdd)
```

Рисунок 3.9. – Програмний код конвеєру машинного навчання для побудови моделі розпізнавання тексту (включаючи завантаження, обробку, навчання, передбачення і т.д) на Spark ML та BigDL.

BigDL забезпечує виразну, «інтегровану з аналітикою даних» модель програмування глибокого навчання; в рамках єдиного уніфікованого конвеєра аналізу даних користувачі можуть ефективно обробляти дуже великий набір даних за допомогою API Spark (наприклад, RDD [17], Spark SQL, конвеєр ML тощо), передавати розподілений набір даних до нейронної мережі моделювати та виконувати розподілене навчання чи висновки поверх Spark. Всупереч загальноприйнятій думці спільноти машинного навчання (що тонкий доступ до даних і оновлення на місці є вирішальними для ефективного розподіленого навчання [18]), BigDL забезпечує широкомасштабне паралельне навчання даних безпосередньо поверх функціональної обчислювальної моделі (з операціями копіювання при записі та грубими операціями).

Модель виконання BigDL

Так як BigDL підтримує широкомасштабне розподілене навчання поверх Apache Spark. Хоча він прийняв стандартну практику (наприклад, паралельне навчання даних для масштабованого навчання, ключовою новинкою BigDL є те, як ефективно реалізуйте ці функції на функціональній, грубозернистої обчислювальної моделі Spark.

Звичайна думка спільноти машинного навчання полягає в тому, що тонкий доступ до даних і мутація даних на місці мають вирішальне значення для підтримки високоефективного сервера параметрів, AllReduce і розподіленого навчання [19]. З іншого боку, системи великих даних (такі як Spark) зазвичай використовують зовсім іншу, функціональну модель обчислень, де набір даних є незмінним і може бути перетворений лише в новий набір даних без побічних ефектів (наприклад, копіювання при записі); крім того, перетворення є грубозернистими операціями (тобто застосування однієї операції до всіх елементів даних одночасно).

Модель виконання Spark

Модель виконання Spark Подібно до інших систем великих даних (наприклад, MapReduce), кластер Spark складається з одного вузла драйвера і

кількох робочих вузлів, як показано на (рис.3.10). Драйвер відповідає за координацію завдань у завданнях Spark (наприклад, планування та диспетчеризація завдань), тоді як працівники відповідають за фактичні обчислення. Щоб автоматично паралелізувати обробку даних у кластері у відмовостійкий спосіб, Spark надає паралельну, функціональну модель обчислень. У завданні Spark дані представлені як Resilient Distributed Dataset (RDD), який є незмінною колекцією записів, розділених на кластері, і може бути перетворена лише для отримання нових RDD (тобто копіювання під час запису) за допомогою функціональних операторів, такі як map, filter і reduce (наприклад, див. рядки 4 - 6 на рис.3.9); крім того, ці операції є паралельними даними (тобто застосовуються до окремих розділів даних паралельно різними завданнями Spark), і грубозернистими (тобто застосовують одну і ту ж операцію до всіх елементів даних одночасно).

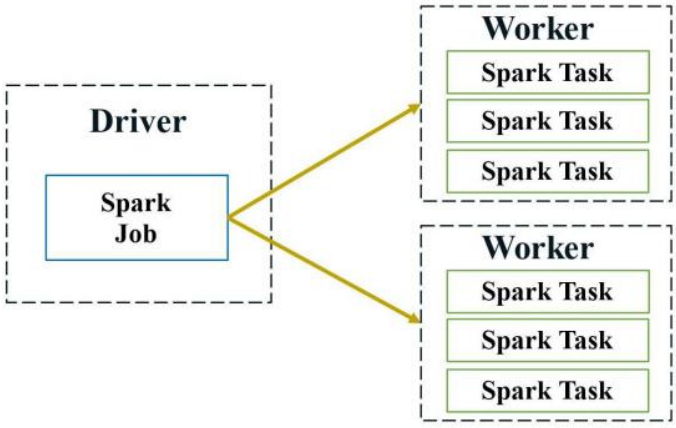


Рисунок 3.10. – Структура кластера Spark.

Паралельне навчання даних у BigDL

Побудований на основі паралельної функціональної обчислювальної моделі Spark, BigDL забезпечує синхронне навчання паралельних даних для навчання моделі глибокої нейронної мережі в кластері, яка досягає кращої масштабованості та ефективності (з точки зору часу до якості) порівняно з до асинхронного навчання. Зокрема, розподілене навчання в BigDL реалізується як ітераційний процес, як на (рис.3.11), де кожна ітерація виконує кілька завдань Spark, щоб спочатку обчислити градієнти за допомогою поточного міні-пакету (за допомогою

завдання «модель вперед-назад»), а потім зробити одне оновлення параметрів моделі нейронної мережі (за допомогою «параметра завдання синхронізації»).

```
1: for i = 1 to M do
2:     //"model forward-backward" job
3:     for each task in the Spark job do
4:         read the latest weights;
5:         get a random batch of data from local Sample partition;
6:         compute local gradients (forward-backward on local model
           replica);
7:     end for
8:     //"parameter synchronization" job
9:     aggregate (sum) all the gradients;
10:    update the weights per specified optimization method;
11: end for
```

Рисунок 3.11. – Алгоритм паралельного навчання даних у BigDL.

Навчальні дані в BigDL представлені у вигляді RDD зразків (див. рядок 6 на рис.3.9), які автоматично розподіляються на кластер Spark. Крім того, для реалізації паралельного навчання даних BigDL також створює RDD моделі, кожна з яких є копією оригінальної моделі нейронної мережі. Перед навчанням як модель, так і зразок RDD кешуються в пам'яті, розподіляються на частини та розміщуються в кластері, як показано на (рис.3.12).

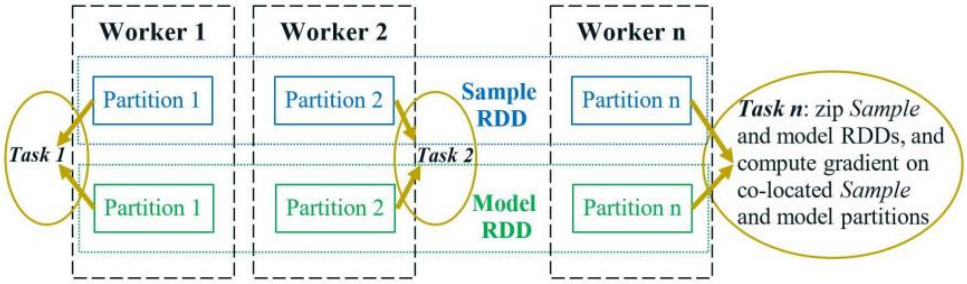


Рисунок 3.12. – Завдання Spark "модель вперед-назад" для паралельного обчислювання локальних градієнтів для кожної репліки моделі.

Отже, на кожній ітерації навчання моделі одне завдання Spark "модель вперед-назад" може просто застосувати функціональний оператор zip до спільно розташованих розділів двох RDD (без додаткових витрат) і обчислити локальні градієнти в паралельно для кожної репліки моделі (використовуючи невелику партію даних у спільно розташованому розділі зразка), як показано на (рис.3.12).

BigDL не підтримує паралельність моделі (тобто відсутність розподілу моделі між різними працівниками). На практиці це не є обмеженням, оскільки BigDL працює на серверах процесора Intel Xeon, які зазвичай мають великий об'єм пам'яті (100 ГБ) і можуть легко вмістити дуже великі моделі.

Синхронізація параметрів у BigDL

Синхронізація параметрів є важливою операцією для навчання паралельної розподіленої моделі даних (з точки зору швидкості та масштабованості). Щоб підтримувати ефективну синхронізацію параметрів, існуючі фреймворки глибокого навчання зазвичай реалізують сервер параметрів або AllReduce, використовуючи такі операції, як дрібнозернистий доступ до даних та зміна даних на місці. На жаль, ці операції не підтримуються функціональною обчислювальною моделлю систем великих даних (наприклад, Spark).

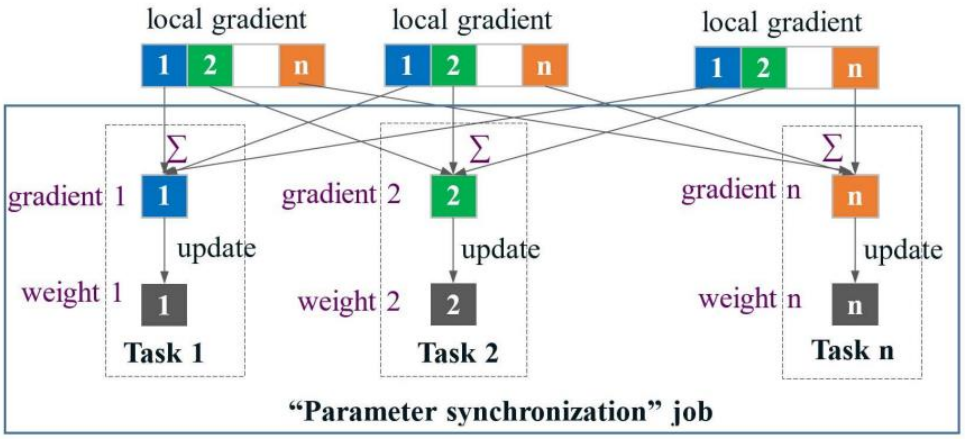


Рисунок 3.13. – Синхронізація параметрів у BigDL.

BigDL застосував зовсім інший підхід, який безпосередньо реалізує ефективну операцію, подібну до AllReduce[20], використовуючи наявні примітиви в Spark (наприклад, перемішування, трансляція, кеш у пам'яті тощо), щоб імітувати функціональність архітектури сервера параметрів (рис.3.13), а саме:

- Кожному з n – завдань, присвоюється унікальний ідентифікатор від 1 до n.
- Далі запускається ще одне завдання синхронізації параметрів; кожне завдання n цього завдання відповідає за управління n-ний розподіл параметрів (як

показано на рис.3.14), як це робить сервер параметрів. Зокрема, n-ний розділ локальних градієнтів (обчислений попереднім завданням «моделювання вперед-назад») спочатку перетасовується до завдання n, яке поєднує ці градієнти та застосовує оновлення до n-го розділу ваг, як показано на (рис.3.13).

- Після цього кожна задача n у завданні «синхронізація параметрів» транслює n-ний розділ оновлених ваг; тобто, завдання «моделювання вперед-назад» наступної ітерації можуть зчитувати останнє значення всіх вагів доти, як розпочнеться наступний крок навчання.
- Операції перемішування та широкомовного розсилання на стороні задачі, описані вище, реалізовані поверх розподіленого сховища в пам'яті в Spark: відповідні завдання просто зберігають локальні градієнти та оновлені ваги у сховищі в пам'яті, які потім можуть бути прочитані віддалено за допомогою завдання Spark з надзвичайно низькою затримкою.

1: **for** each task n in the "parameter synchronization" job **do**
2: **shuffle** the n^{th} partition of all gradients to this task;
3: aggregate (sum) these gradients;
4: updates the n^{th} partition of the weights;
5: **broadcast** the n^{th} partition of the updated weights;
6: **end for**

Рисунок 3.14. – Алгоритм синхронізації параметрів у BigDL.

Етапи виявлення ознак особи на конвеєрі машинного навчання з використанням моделей наскрізного виявлення об’єктів на базі Spark та BigDL:

- Конвеєр зчитує сотні мільйонів зображень із розподіленої бази даних HBase у Spark, як стійкі розподілені набори даних (RDD).
- Потім він використовує BigDL для попередньої обробки цих зображень, для завантаження моделі SSD (включаючи зміну розміру, нормалізацію та пакетну роботу). BigDL надає бібліотеку попередньої обробки зображень на основі OpenCV, яка підтримує звичайні трансформації та доповнення.

- Потім він генерує RDD цільових зображень (шляхом збереження об’єкта з найвищим балом у якості цілі та обрізання вихідного зображення на основі координат цілі) і подальшу попередню обробку RDD цільових зображень.
- Далі він використовує BigDL, для завантаження моделі DeepBit для розподіленого вилучення цільових зображень у Spark і зберігає результати RDD вилучених ознак об’єктів у розподіленій файловій системі Hadoop HDFS.

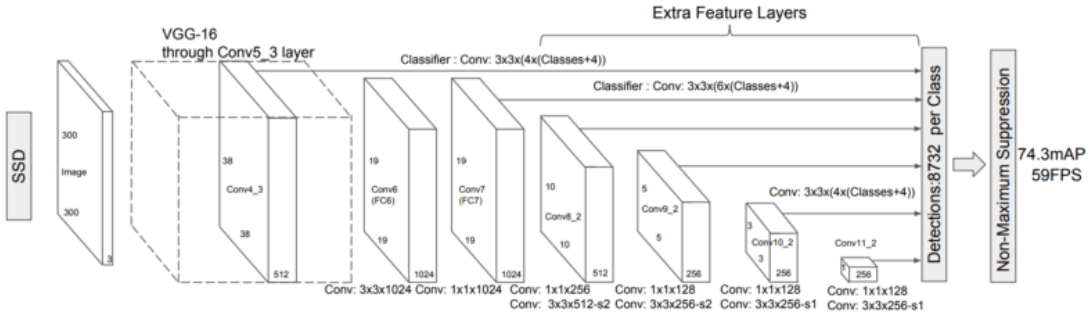


Рисунок 3.15. – Single shot detector.

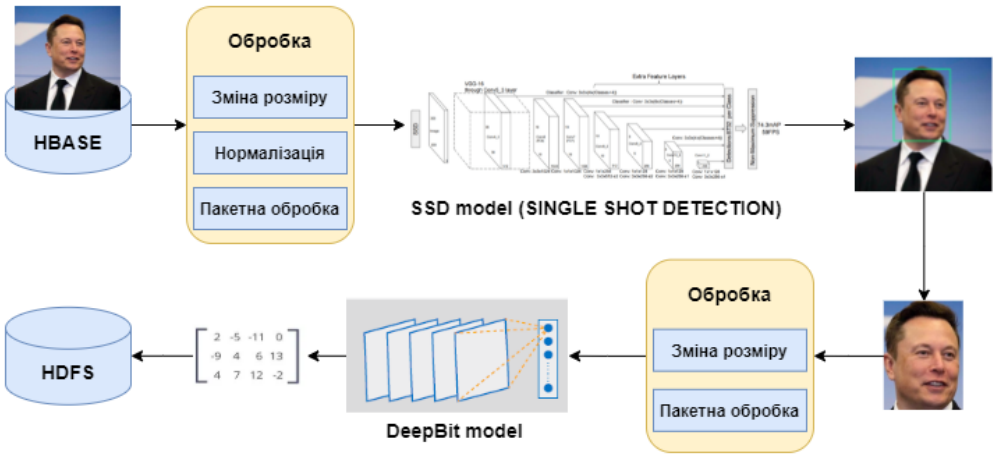


Рисунок 3.16. – Етапи виявлення ознак осіб конвеєру машинного навчання з використанням моделей наскрізного виявлення об’єктів на базі Spark та BigDL.

ВИСНОВОК

У ході виконання магістерської дипломної роботи було досліджено основні концепції систем Big Data, різновид моделей архітектур інформаційних систем для обробки великих даних. Також мною був проведений ретельний аналіз по фреймворкам по роботі з великими даними, можна з упевненістю сказати, що серед фреймворків обробки даних немає єдиного кращого варіанту. У кожного є свої плюси і мінуси. Крім того, результати, що надаються деякими рішеннями, суворо залежать від багатьох факторів. Гібридні рішення з різними інструментами працюють найкраще.

Також мною були виділені такі аспекти технологій та фреймворків для роботи з Big Data: найбільш популярні (Hadoop, Storm, Hive, Spark), найбільш перспективні (Flink, Heron), найбільш корисні (Presto, MapReduce).

Різноманітність пропозицій на ринку фреймворків для великих даних дозволяє технічно компанії вибрати найбільш потрібний інструмент для вирішення поставленої задачі.

Була запропонована мною та розроблена модель архітектури великих даних для розпізнавання осіб з використанням технології машинного навчання. Ця система працює шляхом порівняння найбільш поширених і помітних рис особи на зображенні (зібраному в реальному часі з різних зовнішніх джерел) з особами, що зберігаються у базі даних. Система розпізнавання осіб також здатна розпізнавати закономірності та варіації на основі текстури та форми особи людини, щоб однозначно розпізнавати людину.

ПЕРЕЛІК ПОСИЛАНЬ

1. Definition of Big Data - IT Glossary | Gartner [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.gartner.com/en/information-technology/glossary/big-data (дата звернення 12.10.2021) – Назва з екрана.

2. IBM SPSS Case Study Retail - Tesco Ireland [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.spssanalyticspartner.com/case-study-tesco-ireland/ (дата звернення 12.10.2021) – Назва з екрана.

3. MapReduce - Википедия [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://ru.wikipedia.org/wiki/MapReduce (дата звернення 12.10.2021) – Назва з екрана.

4. Structured Streaming Programming Guide - Spark 3.2.0 Documentation [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html (дата звернення 13.10.2021) – Назва з екрана.

5. A Year of Blink at Alibaba: Apache Flink in Large Scale Production - DATAVERSITY [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.dataversity.net/year-blink-alibaba/ (дата звернення 13.10.2021) – Назва з екрана.

6. Top 10 Big Data Frameworks In 2021 - Jelvix [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://jelvix.com/blog/top-5-big-data-frameworks (дата звернення 15.10.2021) – Назва з екрана.

7. Databricks Delta: A Unified Data Management System for Real-time Big Data [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://databricks.com/blog/2017/10/25/databricks-delta-a-unified-management-system-for-real-time-big-data.html (дата звернення 13.11.2021) – Назва з екрана.

8. Streaming Data Architecture – what it is, and what are the benefits of it? [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://dsstream.com/streaming-data-architecture/ (дата звернення 17.11.2021) – Назва з екрана.

9. MapReduce: simplified data processing on large clusters [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://dl.acm.org/doi/10.1145/1327452.1327492 (дата звернення 25.11.2021) – Назва з екрана.

10. 3D Face Reconstruction From Volumes of Videos Using a Map reduce Framework [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://ieeexplore.ieee.org/document/8821354 (дата звернення 25.11.2021) – Назва з екрана.

11. Performance comparision of Hadoop and spark engine [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://ieeexplore.ieee.org/document/8058263 (дата звернення 27.11.2021) – Назва з екрана.

12. A MapReduce-Based k-Nearest Neighbor Approach for Big Data Classification [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://ieeexplore.ieee.org/document/7345490 (дата звернення 27.11.2021) – Назва з екрана.

13. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0950705116301757?via%3Dihub (дата звернення 28.11.2021) – Назва з екрана.

14. Efficient kNN classification algorithm for big data [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0925231216001132?via%3Dihub (дата звернення 28.11.2021) – Назва з екрана.

15. Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0031320317303898?via%3Dihub (дата звернення 28.11.2021) – Назва з екрана.

16. Image classification from scratch [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://keras.io/examples/vision/image_classification_from_scratch/ (дата звернення 10.12.2021) – Назва з екрана.

17. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf (дата звернення 10.12.2021) – Назва з екрана.

18. Amazon SageMaker [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://aws.amazon.com/sagemaker/ (дата звернення 10.12.2021) – Назва з екрана.

19. Ryaboy, D.: Scaling Big Data Mining Infrastructure: The Twitter Experience. SIGKDD Explorations [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.researchgate.net/publication/262251948_Ryaboy_D_Scaling_Big_Data_Mining_Infrastructure_The_Twitter_Experience_SIGKDD_Explorations_142_6-19 (дата звернення 11.12.2021) – Назва з екрана.

20. MPI Reduce and Allreduce [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/ (дата звернення 11.12.2021) – Назва з екрана.

21. Zeta architecture [Електронний ресурс]: – [Веб-сайт]. Режим доступу: https://www.waitingforcode.com/general-big-data/zeta-architecture/read (дата звернення 15.10.2021) – Назва з екрана.

ДЕМОСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення автоматизованих систем

Дослідження моделей архітектури інформаційних систем для обробки великих даних

Підготував студент групи: ІСДМ-61
Ніколаєнко Олексій Миколайович
Керівник роботи: к.т.н., доцент каф. ІПЗАС, Полоневич Ольга Володимирівна

Мета роботи: здійснити аналіз сукупностей та фреймворків обробки великих даних в Big Data.

Об'єкт дослідження: моделі архітектур інформаційних систем для обробки великих даних.

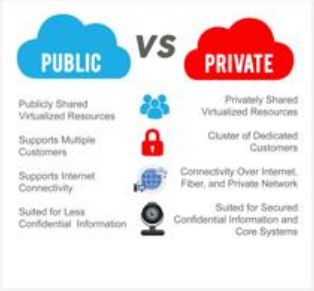
Предмет дослідження: розробка архітектури великих даних для розпізнавання осіб з використанням технології машинного навчання.

Завдання на магістерську роботу:

- Провести аналіз основних положень концепції технології Big Data.
- Провести аналіз по сукупності технологій та фреймворків обробки даних в Big Data на теперішній час.
- Провести оцінку по основним моделям архітектур інформаційних систем для обробки великих даних.
- Розробити модель архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання.
- Дослідити етапи виявлення особи на конвеєрі машинного навчання з використанням моделей наскрізного виявлення об'єктів на базі Spark та BigDL.

Основні концепції технології Big Data

Проблеми та перспективи технології Big Data



Аспекти технологій та фреймворків для роботи з Big Data

Найбільш популярні технології та фреймворки

	Найбільш популярні			
Фреймворки / технології	**Hadoop**	**Storm**	**Hive**	**Spark**
Режим обробки даних	пакетний	пакетний / потоковий	пакетний	пакетний / потоковий
Масштабованість	горизонтальна	горизонтальна	горизонтальна	горизонтальна
Гарантії доставки повідомлень	рівно раз	принаймні, один раз	принаймні, один раз	рівно раз
Режим обчислення	на основі диска	в пам'яті	на основі диска	в пам'яті
Автоматичне масштабування	підтримує	не підтримує	підтримує	підтримує
Відмостійкість	присутня	присутня	присутня	присутня(обмежена)
Підтримка мов програмування	Java, Python	Python, Ruby, JavaScript, Perl	SQL, HiveQL	Scala, Java, Python, R

Найбільш перспективні технології та фреймворки

	Найбільш перспективні	
Фреймворки / технології	**Flink**	**Heron**
Режим обробки даних	пакетний / потоковий	потоковий
Масштабованість	горизонтальна	горизонтальна
Гарантії доставки повідомлень	рівно раз	принаймні, один раз
Режим обчислення	в пам'яті	в пам'яті
Автоматичне масштабування	не підтримує	підтримує
Відмостійкість	присутня(обмежена)	присутня
Підтримка мов програмування	Java, Scala, Python, R	C++, Java, Python

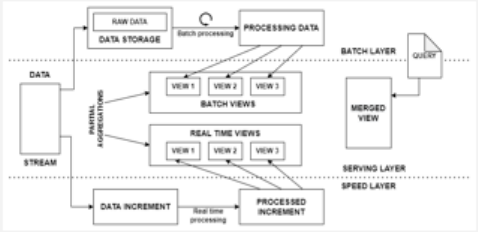
Найбільш корисні технології та фреймворки

	Найбільш корисні	
Фреймворки / технології	**Presto**	**MapReduce**
Режим обробки даних	потоковий	конвеєрний
Масштабованість	горизонтальна	горизонтальна
Гарантії доставки повідомлень	відсутня	рівно раз
Режим обчислення	в пам'яті	на основі диска
Автоматичне масштабування	підтримує	підтримує
Відмостійкість	відсутня	присутня
Підтримка мов програмування	Java, SQL	Java, Python

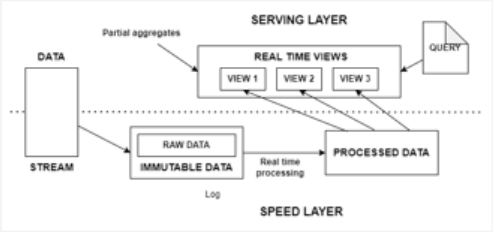
Найбільш недооцінені технології та фреймворки

	Недооцінені	
Фреймворки / технології	**Samza**	**Kudu**
Режим обробки даних	потоковий	пакетний / конвеєрний
Масштабованість	горизонтальна	горизонтальна / вертикальна
Гарантії доставки повідомлень	принаймні, один раз	відсутня
Режим обчислення	в пам'яті	в пам'яті
Автоматичне масштабування	підтримує	не підтримує
Відмостійкість	присутня	присутня
Підтримка мов програмування	Java, Scala, Samza SQL	C++, Java, Python

Різновид моделей архітектур інформаційних систем для обробки великих даних



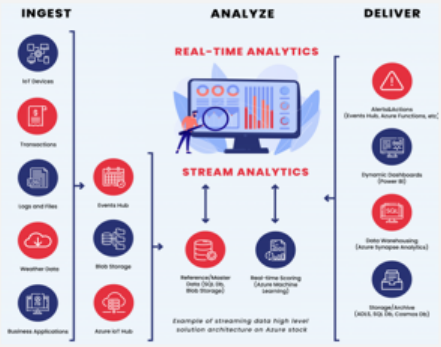
Lambda архітектура



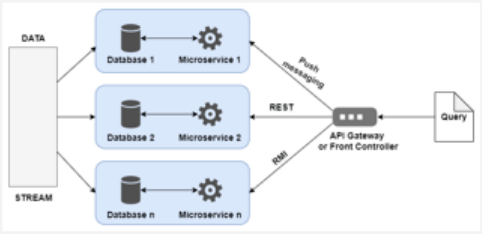
Kappa архітектура

Delta архітектура

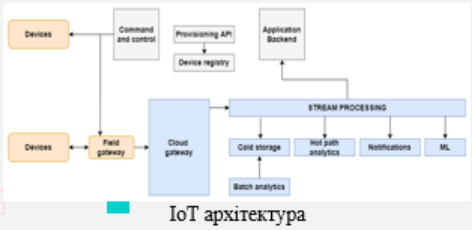
Різновид моделей архітектур інформаційних систем для обробки великих даних



Strimming архітектура



Microservice архітектура

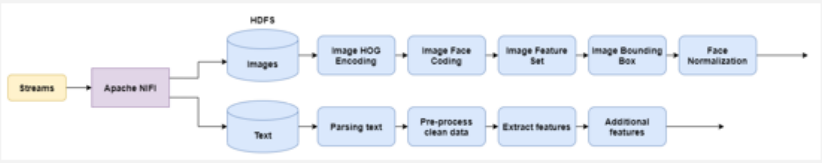


IoT архітектура

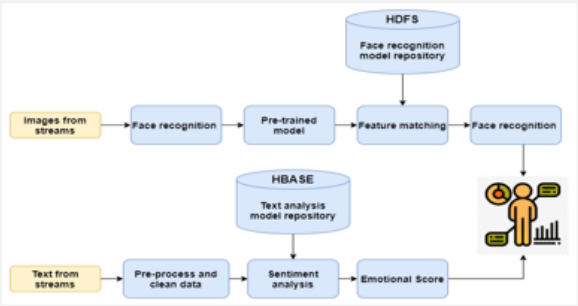
Zeta архітектура

Розроблена модель архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання

Розроблена модель архітектури великих даних для вирішення задачі розпізнавання осіб з використанням технології машинного навчання

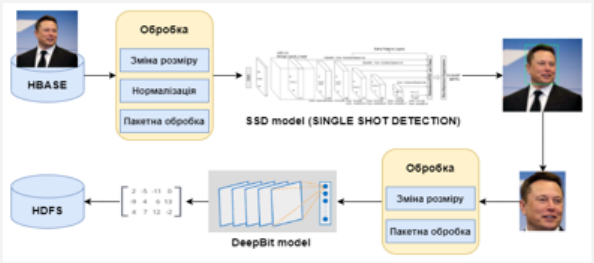


Принцип роботи конвеєру великих даних



Принцип роботи конвеєру машинного навчання

Етапи виявлення ознак особи на конвеєрі машинного навчання з використанням моделей наскрізного виявлення об'єктів на базі Spark та BigDL



Етапи:

1. Зчитування даних із HBASE
2. Попередня обробка
3. Генерація RDD цільових зображень
4. Розподілене вилучення цільових зображень
5. Зберігання результатів

Висновок

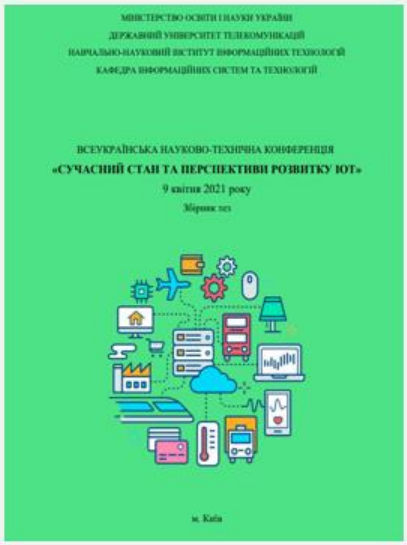
У ході виконання магістерської дипломної роботи було досліджено основні концепції систем Big Data, різновид моделей архітектур інформаційних систем для обробки великих даних. Також мною був проведений ретельний аналіз по фреймворкам по роботі з великими даними, можна з упевненістю сказати, що серед фреймворків обробки даних немає єдиного кращого варіанту. У кожного є свої плюси і мінуси. Крім того, результати, що надаються деякими рішеннями, суворо залежать від багатьох факторів. Гібридні рішення з різними інструментами працюють найкраще.

Також мною були виділені аспекти технологій та фреймворків для роботи з Big Data, що в подальшому дозволить технічно вибрати найбільш потрібний інструмент для вирішення поставленої задачі.

Була розроблена модель архітектури великих даних для розпізнавання осіб з використанням технології машинного навчання та виділено основні етапи виявлення ознак особи на конвеєрі машинного навчання з використанням моделей наскрізного виявлення об'єктів на базі Spark та BigDL.

Апробація

Опубліковано тезу «Дослідження застосування хмарної платформи AIOPS в управлінні it інфраструктурою», 79 ст.



Опубліковано тезу «ТЕХНОЛОГІЯ INTERNET OF NANO THINGS», 128 ст.

Опубліковано статтю «Використання фреймворку MLOPS для збільшення рівня життєвого циклу машинного навчання в IoT», 49 ст.

Дякую за увагу!