ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Моделювання та розробка алгоритму зчитування Parquet-файлів для підтримки бізнес-процесів»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

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

_____ Олексій ЛЯЩЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Олексій ЛЯЩЕНКО

Керівник: Світлана ШЕВЧЕНКО
к.пед.н., доцент

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

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Лященку Олексію Володимировичу_____

1. Тема кваліфікаційної роботи: Моделювання та розробка алгоритму зчитування Parquet-файлів для підтримки бізнес-процесів

керівник кваліфікаційної роботи Світлана ШЕВЧЕНКО к.пед.н., доцент

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. № 145.

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

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, мова програмування Java 21, рішення для обробки даних Apache Spark, бібліотека для роботи з Parquet файлами Apache Parquet, бібліотека для генерації тестових даних Benerator, середовище розробки IntelliJ IDEA 2023.3.1, інтерфейс для перегляду детальної інформації про виконання JVM програм VisualVM, технічне завдання.

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

1. Аналіз предметної області, розробка технічного завдання.

2. Дослідження доступних алгоритмів перетворення внутрішньої будови сховища даних для збільшення ефективності при роботі з великими обсягами даних
3. Розробка алгоритму зчитування Parquet-файлів із вбудованою підтримкою часткової агрегації Pushdown.
4. Дослідження ефективності розробленого алгоритму.
5. Розробка вимог та рекомендацій для реалізації розробленого алгоритму

5. Перелік графічного матеріалу: *презентація*

1. Ключові етапи виконання запиту в системах обробки даних.
2. Модель запропонованого вдосконалення роботи фільтру Pushdown.
3. Змінений логічний план.
4. Отриманий фізичний план.
5. Алгоритм розділення фільтрів.
6. Модель AHP побудови фізичного плану.
7. Швидкість виконання запитів (відфільтровано 20%).
8. Швидкість виконання запитів (відфільтровано 50%).
9. Використання пам’яті (відфільтровано 20%).
10. Використання пам’яті (відфільтровано 50%).
11. Множина згенерованих фізичних планів.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Вибір і обґрунтування технологій та підходів для реалізації поставленого завдання	06.11-12.11.23	
3	Розробка алгоритму зчитування Parquet-файлів з доступом до сховища даних	13.11-19.11.23	
4	Реалізація програмного забезпечення	20.11-26.11.23	
5	Тестування алгоритму та аналіз його ефективності	27.11-30.11.23	
6	Виконання аналізу, розробка рекомендацій та висновків стосовно реалізації алгоритму	01.12-07.12.23	
7	Оформлення роботи: вступ, висновки, реферат	08.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Олексій ЛЯЩЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Світлана ШЕВЧЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра містить: 78 сторінок, 2 таблиці, 11 рисунків, 31 джерело.

Мета роботи – удосконалення бізнес-процесів внаслідок підвищення ефективності алгоритмів, що зчитують Parquet-файли.

Об'єкт дослідження – процес зчитування та обробки Parquet-файлів.

Предмет дослідження – структура алгоритму зчитування Parquet-файлів із підтримкою часткового фільтру Pushdown.

Короткий зміст роботи. Сучасні бізнес-процеси вимагають обробки величезних обсягів даних і виконання складних обчислювальних завдань, тому ефективне використання обчислювальних ресурсів стає критично важливим, що підтверджує актуальність даного дослідження. У науковій розробці проаналізовано принципи роботи алгоритму читання Parquet-файлів, запропоновано шляхи вдосконалення та адаптації алгоритму з метою підвищення його продуктивності та ефективності у відповідь на зростаючі вимоги сучасної обробки великих даних.

Реалізовано модель алгоритму з підтримкою фільтру Pushdown для підвищення ефективності обробки Parquet файлів за рахунок фільтрування на стороні файлового сховища. Здійснено тестування програмного забезпечення, що підтвердило зменшення часу на виконання запиту та пам'яті сховища для обробки даних. Алгоритм вдосконалення фільтру Pushdown може бути рекомендований як ефективне рішення для покращення продуктивності обробки великих даних в Apache Spark.

КЛЮЧОВІ СЛОВА: БІЗНЕС-ПРОЦЕСИ, BIG DATA, PARQUET, APACHE SPARK, DATA SET, PUSHDOWN.

ABSTRACT

Text part of the master's qualification work:

78 pages, 11 pictures, 2 tables, 31 sources.

The purpose of the work – improvement of business processes as a result of increasing the efficiency of algorithms that read Parquet files.

Object of research – the process of reading and processing Parquet files.

Subject of research – the structure of the algorithm for reading Parquet files with support for partial Pushdown filter.

Summary of the work: The author conducts a thorough examination of the Parquet file reading algorithm's foundational principles and puts forth strategies for its enhancement and adjustment. The aim is to augment the algorithm's capability, accelerate its processing speed, and fortify its dependability to effectively address the intensifying needs of today's sophisticated big data processing. In this context, the study introduces an advanced implementation of the Filter pushdown model.

A significant achievement of this research is the implementation of a model of the filter pushdown algorithm. It significantly improves the efficiency of processing Parquet files by enabling filtering directly at the file storage layer. The document is a substantial contribution to the fields of business processes, Big Data, Parquet, Apache Spark and pushdown filters. The research not only presents a theoretical framework but also demonstrates practical applications and improvements in data processing techniques.

KEYWORDS: BUSINESS PROCESSES, BIG DATA, PARQUET, APACHE SPARK, DATA SET, PUSHDOWN.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система

КБД – колонкові бази даних

СУБД – система управління базами даних

ПЗ – програмне забезпечення

ОС – операційна система

АНР – аналітичний ієрархічний процес

Apache Parquet – колонковий формат зберігання комп'ютерних даних, призначений для ефективного зберігання та пошуку даних

Java – об'єктно-орієнтована мова програмування

Big Data – група технологій та методів, за допомогою яких аналізують та обробляють структуровані й неструктуровані дані великих обсягів

API – програмний інтерфейс додатку

ORM – об'єктно-реляційне відображення

SQL – формат зберігання у реляційних базах даних

NoSQL – формат зберігання у нереляційних базах даних

OLAP – технологія, яка використовується для організації великих бізнес-баз даних і підтримки бізнес-аналітики

RDBMS – об'єктно-реляційна база даних

CPU – центральний процесор комп’ютера

ВСТУП

Сучасні бізнес-процеси використовують дані, які накопичуються у величезних обсягах, у зв'язку з чим зростає потреба в пришвидшенні процесу їх обробки та необхідність розробки для цього ефективних методів та інструментів.

Оптимізація обчислювальних ресурсів стає все більш критичною для підтримки швидкозмінюваних потреб бізнесу, впровадження нових технологій, забезпечення конкурентних переваг та забезпечення ефективного вирішення завдань у сучасному IT-середовищі.

Це зумовило появу кількох напрямів досліджень у сфері зберігання даних. Серед таких напрямків виділяється концепція Big Data, яка спирається на наступні аспекти: цінність, обсяг, швидкість, різноманітність, надійність та мінливість. Важливо, що головною характеристикою виступає не лише обсяг цих даних, але й здатність швидко обробляти й генерувати цінну інформацію.

Аналіз і обробка даних відіграють вирішальну роль у сучасному світі, а формат Parquet стає дедалі популярнішим для зберігання структурованих даних, тому що дає змогу ефективно та компактно зберігати інформацію в аналітиці та обробці великих даних. Традиційно процес читання Parquet-файлів вимагає вичитування всього обсягу даних та їх подальшої обробки, включаючи фільтрацію та агрегацію. Однак, ця операція може бути досить витратною з точки зору ресурсів, особливо коли мова йде про великі набори даних.

Одним з напрямків підтримки ресурсів та забезпечення стабільності в умовах високих навантажень в роботі систем є пошуки ефективного використання обчислювальних ресурсів, щоб зменшити споживання пам'яті, витрат на обладнання та операційну підтримку і обслуговування обчислювальних систем.

Актуальність дослідження зумовлена тим, що зростає складність завдань і збільшення обсягу даних, тому що сучасні бізнес-процеси вимагають обробки величезних обсягів даних і виконання складних обчислювальних завдань, тому ефективне використання обчислювальних ресурсів стає критично важливим.

Зростає необхідність пошуку нових підходів, ефективних методів й інструментів від традиційних систем для вирішення проблеми зберігання та обробки великих обсягів даних, що вимагає спеціалізованих рішень, тоді як складні технології Parquet дають змогу зберігати дані зі складними структурами, що робить його дуже затребуваним, але алгоритми їх обробки потребують вдосконалення для збільшення своєї ефективності.

Важливим напрямком залишається покращення коду для усунення витоків пам'яті та надлишкових запитів до бази даних, тому що таке використання ефективного зберігання даних дає змогу знизити навантаження на сервер і прискорити обробку запитів.

Метою дослідження є удосконалення бізнес-процесів внаслідок підвищення ефективності алгоритмів, що зчитують Parquet-файли.

У процесі дослідження вирішувалися наступні *завдання*:

1. Проаналізувати проблеми та існуючі рішення для збільшення ефективності при роботі з великими обсягами даних.
2. Дослідити доступні алгоритми обробки великих даних.
3. Розробити алгоритм зчитування Parquet-файлів із вбудованою підтримкою часткового фільтру Pushdown.
4. Дослідити ефективність модифікованого алгоритму.
5. Проаналізувати результати дослідження, розробити рекомендації для його реалізації.

Об'єкт дослідження – процес зчитування та обробки Parquet-файлів.

Предмет дослідження – структура алгоритму зчитування Parquet-файлів із підтримкою часткового фільтру Pushdown.

Методи дослідження: комплексний аналіз, порівняння, системно-структурний, інформаційно-аналітичні методи, виконання удосконалення алгоритму для читання Parquet-файлів за допомогою мови програмування Java, методи оцінки та прогнозування ефективності розробленого алгоритму.

Наукова новизна дослідження полягає в тому, що було проаналізовано принципи роботи алгоритму читання Parquet-файлів у Java та запропоновані

шляхи вдосконалення та адаптації алгоритму з метою підвищення його продуктивності, ефективності та забезпечення високої масштабованості у відповідь на зростаючі вимоги сучасної обробки великих даних.

Практична значущість роботи полягає в покращенні ефективності алгоритму обробки Parquet файлів за рахунок використання передвибірки та кешування для зниження часу читання файлів.

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані прикладні результати, що відносяться до вирішення задачі покращення ефективності алгоритму обробки Parquet файлів.

Апробація результатів та публікації. За результатами досліджень проведених в рамках магістерської роботи, зроблені доповіді:

1. Лященко О.В. Створення алгоритму ефективного кешування даних для збільшення продуктивності читання Parquet-файлів// XV Міжнародна науково-практична конференція «Distance education as the main problem of young people», 26-29 грудня 2023 р., Мадрид, Іспанія.

2. Лященко О.В. Оптимізація використання обчислювальних ресурсів в бізнес-процесах за рахунок прискорення процесу зчитування та обробки Parquet-файлів // Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ: ДУТ, 2023.

Структура роботи. Робота складається з вступу, 3 розділів, висновку, переліку посилань та додатків.

У вступі обґрунтовані актуальність теми дослідження, її значимість, визначено проблему та необхідність проведення дослідження, представлені завдання, та методи для досягнення поставленої мети.

У першому розділі роботи виконано аналіз останніх досліджень і публікацій, у яких піднімаються питання більш ефективного та раціонального використання ресурсів для обробки великих обсягів даних, визначено основні характеристики бізнес-процесів та необхідність збільшення їх продуктивності,

запропоновано вирішення поставленого завдання за рахунок прискорення процесу зчитування та обробки Parquet-файлів.

У другому розділі описано етапи проєктування та розробки та алгоритму зчитування Parquet-файлів із фільтр Pushdown для підвищення його продуктивності. Проведено вибір програмних засобів для розробки та описано особливості запропонованого алгоритму.

У третьому розділі проаналізовано ефективність алгоритму, проведено його тестування, порівняння швидкісних характеристик, наведено рекомендації до його реалізації.

У висновку описано результати роботи відповідно до поставлених завдань.

1 АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНОГО СТАНУ АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ ПРИ РОБОТИ З ВЕЛИКИМИ БАЗАМИ ТА СХОВИЩАМИ ДАНИХ

1.1 Автоматизація бізнес-процесів як засіб підвищення їх ефективності

Впровадження нових технологій має значний вплив на спосіб ведення бізнесу у сучасному світі, тому для успішного розвитку будь якого бізнес-процесу необхідна гнучкість та готовність до змін, адаптація до сучасних умов та технологій. Саме автоматизація бізнес-процесів дасть можливість прискорювати прийняття рішень, підвищувати продуктивність, залишатися конкурентоспроможним.

Зміна способів комунікації, введення онлайн-платформ, мобільних додатків впливають на способи взаємодії бізнесу з клієнтами, надають нові можливості для розвитку. Це вимагає розробки нових систем управління, використання штучного інтелекту та аналізу даних, щоб зробити бізнес-процеси більш ефективним [1].

Саме цифрова трансформація стає ключовим напрямком для багатьох компаній у бізнесі та виробництві, що потребує впровадження нових технологій, зміни бізнес-процесів та стратегій, швидкого реагування на зміни в технологічному середовищі.

За визначенням Мельник Л. Г. бізнес-процеси – це сукупність різних видів діяльності, що забезпечують вихідний продукт, які представляють цінність для внутрішнього або зовнішнього споживача [2]. Саме тому ключовими властивостями бізнес-процесів є те, що вони мають бути направлені на отримання конкретного результату заради спільних інтересів споживача та власника, а саме: інновації та зміни заради вдосконалення виробленого товару або послуги, цифрову трансформацію бізнесу для підвищення ефективності, продуктивності та конкурентоспроможності. Тож у сучасних умовах автоматизація бізнес-процесів є особливо важливою, оскільки це дає можливість швидше вирішувати проблеми споживачів, надавати якісніші послуги, отримувати більший прибуток.

Автоматизовані системи спрощують рутинні операції, знижують ризик людських помилок та збільшують точність виконання завдань, що у свою чергу дозволяє використовувати ресурси бізнесу ефективніше, і може в подальшому призвести до економії часу та зниження витрат.

Використання спеціальних інструментів та платформ для моделювання, автоматизації, моніторингу та оптимізації у бізнес-процесах дозволяє компаніям підвищити продуктивність, знизити витрати, збільшити точність та прискорити виконання завдань, що сприяє загальному покращенню діяльності організації.

Слід зазначити, що автоматизація бізнес-процесів дозволяє збирати та аналізувати великі обсяги даних, а це дозволяє краще прогнозувати тенденції та приймати більш обґрунтовані стратегічні рішення.

Для покращення обслуговування клієнтів важливі наступні оптимізації: підвищений рівень захисту та контролю над конфіденційністю даних, швидка та ефективна відповідь на запити, стабільне та швидке виконання процесів. Саме тому важливе впровадження оптимізацій програмного забезпечення особливо при роботі з великими обсягами та сховищами даних.

1.2 Аналіз проблем та існуючих рішень для збільшення ефективності при роботі з великими обсягами даних

Вивчення методів зберігання даних є одним з найважливіших напрямків розробки програмного забезпечення, метою яких є пошук найбільш оптимальних засобів для зберігання даних, а також методів зберігання даних в залежності від використовуваних програмних архітектур [3-16].

Дослідження ведуться в кількох напрямках, значну частину яких спрямовано на оцінку та аналіз NoSQL рішень та використання реляційних та NoSQL баз. Проведені дослідження показують, що реляційні бази даних мають ширші функціональні можливості як для створення вітрин даних, так і в якості інструментів для OLAP-технологій [3].

Вивчення різних форматів файлів для зберігання даних, оцінка їх продуктивності для виконання певних завдань є окремим напрямком досліджень. У проведених дослідженнях [7,8,9,10,11] представлені результати вивчення форматів зберігання великих даних, таких як Apache Avro, Apache Parquet, ORC та інших. У цих дослідженнях представлені результати вивчення різних форматів з точки зору продуктивності або вибору формату для конкретної мети. Глибокі теоретичні дослідження файлових форматів ORC та Apache Parquet [10], які пройшли тестування з використанням різних інструментів для роботи з цими форматами широко розкривають можливості їх використання, але без прив'язки до конкретної задачі.

Детальний огляд та аналіз характерних властивостей, класифікації, принципів роботи, методів і технологій великих даних представлений у дослідженнях [12]. Автори наголошують, що з появою статистичних і аналітичних баз даних, сховищ даних, колонкові бази даних (КБД) стали доволі затребуваними. Колонкова база даних NoSQL зберігає дані, які згуртовані по колонках таблиці, а не рядками, як у реляційних базах даних [13, 24]. Використання КБД може бути задіяне задля оптимізації і підвищення продуктивності процесів. Це пояснюється тим, що вони забезпечують високу ефективність стискання даних та швидкість виконання операцій пошуку, мають високу горизонтальну масштабність. Для підвищення продуктивності їх функціонування пропонуються наступні методи: стискання колонок, яке дає зменшення файлів розмірів колонок; оперування стиснутими даними з відтермінацією розпаковування [13,14,15] використання віртуального ключа та методи блокової організації даних з їх векторною обробкою, що підвищує їхню продуктивність. Автори наголошують, що сучасні КБД мають новий розширений інструментарій, адаптовані до сучасних технічних засобів та методів роботи з ними [15].

Проблемою сучасних хмарних баз даних, що використовують архітектуру, яка розділяє управління обчисленнями та зберігання даних, є мережа, яка з’єднує обчислювальний рівень з рівнем зберігання [16]. Було досліджено два рішення для покращення поєднання процесів зберігання та обчислення. Обидва рішення

можуть зменшити обсяги даних, що передаються. Але існуючі СУБД передбачають кешування та обчислення, як ортогональні процеси, при цьому більшість систем реалізують лише один із них. Як вихід з ситуації пропонується гібридний режим виконання для хмарних СКБД, що поєднує в собі переваги кешування та витіснення, хмарної бази даних OLAP та нову політику заміни Weighted-LFUcache спеціально оптимізовано для архітектури дезагрегації. Це перевершує обидві архітектури – тільки кешування та тільки витіснення – у 2,2 рази на тесті Star Schema Benchmark. Робота представляє детальну розробку та реалізацію FPDB, хмарної OLAP СУБД з відкритим вихідним кодом на основі C++.

Актуальність пошуків підвищення ефективності роботи програмного забезпечення за рахунок оптимізації обчислювальних процесів викликана зростаючою популярністю великомасштабних аналітичних додатків, які працюють у реальному часі [17]. Як доводиться у роботі, вирішення питання необхідності ефективної обробки аналітичних та транзакційних запитів потребує розподілених систем керування даними, які можуть обробляти швидкі транзакції та аналітику одночасно. Автори пропонують вирішити це питання шляхом різних оптимізацій та зміни архітектури за допомогою Wildfire системи, яка націлена на Hybrid Transactional and Analytical.

Це дає можливість прискорити аналітичні запити з використанням стиснення та обробки кешу, автоматичного створення та зберігання по стовпцях (багатопотоковий паралелізм).

Як доводять автори, система Wildfire, яка призначена для обробки транзакції великого обсягу, виконуючи складні аналітичні запити, використовує екосистему Spark, щоб забезпечити широкомасштабну обробку даних через комплексні аналітичні запити, колонкову обробку даних та аналітику одночасно.

Результат порівняння переваг та недоліків з точки зору ємності сховища та продуктивності запитів як для VCF-файлів, так і для Apache Parquet доводить, що Apache Parquet пропонує переваги у зменшенні розміру сховища і часу виконання запитів, а також масштабується при роботі з великими наборами даних [18]. Це підтверджує необхідність їх впровадження в програмні засоби та розробки.

У статтях [18, 20] представлено детальний огляд останніх досягнень з наголосом на те, як оптимізувати витрати на використання хмарного сховища для користувачів хмари, а також є можливість залучити широку аудиторію на ринку економічно ефективних хмарних сховищ.

Порівняння та аналіз різних форматів файлів структури даних, такі як Parquet, Avro, і ORC доводять, що файли Parquet займають менше місця в пам'яті [18,22,23]. Це пов’язано з тим, що використовується двійковий формат даних і методів стиснення. Це впливає також на швидкість: формат Parquet виконується швидше і добре підтримує запити на основі стовпців, що важливо для оптимізації зберігання, продуктивності запитів до бази даних.

Глибокий порівняльний аналіз аналітичних СУБД для Big Data та пошук технологій і архітектур, призначених для отримання економічної вигоди при роботі з дуже великими обсягами даних [19,20] для забезпечення високої швидкості роботи доводить, що постійне зростання Big Data потребує збільшення швидкості обробки даних. Тому використання аналітичних обчислень всередині бази даних, які виключають переміщення даних і скорочують час обробки, стає актуальним. Розширений простір пам'яті поза межами кешу сприяє збільшенню продуктивності запитів і оптимізує зберігання даних у пам'яті.

Крім того, існує ряд досліджень, пов'язаних з розробками в області інтеграції програмного забезпечення та вибору оптимального інструменту для різних цілей з використанням різних методів прийняття рішень [21, 22, 23]. Вибір системних компонентів є важливою умовою коректної роботи програмного комплексу. Оскільки файли зберігання даних є частиною аналітичної платформи, вивчення можливостей різних форматів для розробки вітрин даних є важливим завданням для коректної роботи системи в цілому.

У сучасному інформаційному середовищі важливим напрямком досліджень є оптимізація формату збереження Parquet-файлів для зберігання даних, підтримки паралельної обробки та стиснення. Також актуальним є розробка оптимізованих алгоритмів читання та запису, які забезпечать швидку обробку великих обсягів даних, що зберігаються у форматі Parquet.

Велика увага приділяється розробці нових бібліотек та інструментів, тоді як існуючі бібліотеки та інструменти для роботи з Parquet-файлами можуть бути розширені та покращені для підтримки нових можливостей арегації Pushdown.

Проводиться розробка нових форматів даних, які будуть підтримувати процеси агрегації Pushdown та швидку обробку.

У великих корпораціях, фінансових установах, роздрібних торгових точках, телекомунікаційних компаніях, наукових дослідженнях та багатьох інших галузях використання агрегації Pushdown для обробки Parquet-файлів може значно полегшити та прискорити аналіз даних. Тому це завдання залишається актуальним і важливим для багатьох галузей.

Підсумовуючи, можна зазначити, що серед загальних тенденцій та напрямків досліджень по темі роботи, основними темами проаналізованих джерел є прискорення запитів для розробки більш ефективних алгоритмів обробки великих даних.

1.3 Обґрунтування вибору мови програмування

Java – це високорівнева, об'єктно-орієнтована мова програмування, яка має кілька інструментів та бібліотек для роботи з великими сховищами даних, такими як бази даних, розподілені системи файлів, облікові системи тощо. Java є популярною мовою завдяки своїм можливостям, системі бібліотек та фреймворків, а також великій спільноті розробників. Вона застосовується в різних сферах через свою універсальність та можливість працювати на різних платформах.

Завдяки широкому функціоналу Java може працювати з великими сховищами даних. Використання API дозволяє Java-програмам зв'язуватися з різними реляційними базами даних, такими як MySQL, Oracle, та виконувати запити до них. Завдяки своєму функціоналу вона стала дуже популярною в середовищі програмування та широко використовується у різних сферах, включаючи розробку веб-додатків, мобільних додатків, корпоративних систем, наукових досліджень та багато інших галузей.

Java відома своєю вбудованою системою безпеки, включаючи механізми безпечного виконання коду, контроль доступу і інші заходи.

Мова дозволяє створювати великі програмні системи завдяки своїм можливостям управління пам'яттю, підтримці багатопотоковості тощо.

Java має багатий набір бібліотек та фреймворків, що сприяють швидкому розвитку програмного забезпечення. Також вона користується підтримкою активної спільноти розробників та має регулярні оновлення з виправленнями помилок та вдосконаленням.

Ці інструменти та бібліотеки дозволяють Java взаємодіяти з різними системами сховищ даних, забезпечуючи можливість розробки програм, які працюють з великими обсягами даних та виконують різні операції з ними.

Недоліками мови програмування можна вважати те, що Java може вимагати більше ресурсів пам'яті порівняно з деякими іншими мовами програмування. Це може дещо впливати на повільність у порівнянні з деякими низькорівневими мовами.

Порівняно з деякими іншими мовами програмування, наприклад, Python або JavaScript, розробка в Java може вимагати більше коду для досягнення певних завдань. У порівнянні з деякими іншими мовами, синтаксис Java може вважатися менш простим та менш компактним.

Зважуючи на переваги та недоліки, обираючи мову програмування для виконання поставленого завдання, важливо враховувати потреби, специфіку завдань, які необхідно вирішити.

Мову програмування Java для виконання запланованого дослідження було вибрано через її універсальність, багатофункціональність, поширеність у програмному середовищі.

Для виконання завдання важливо, що Java широко використовується для розробки бізнес-додатків та корпоративного програмування, обробки транзакцій, роботи з базами даних та інтеграції з іншими підприємствами, підтримує роботу з різноманітними фреймворками та інструментами для обробки великих обсягів даних, такими як Apache Spark та інші

1.4 Аналіз методів Big Data

Технології обробки великих даних стають все більш популярними у бізнес-процесах усіх сфер виробництва, маркетингу, телекомунікацій, промисловості, інших найрізноманітніших сферах і галузях через постійне зростання обсягу даних та інформації, яку потрібно зберігати, обробляти, що є невід'ємним елементом сучасного бізнесу, економіки, господарства та виробництва.

Це зумовило появу кількох напрямів досліджень у сфері зберігання даних. Серед таких напрямків виділяється концепція Big Data – це технології, що необхідні для збору, організації та обробки великих обсягів даних, які характеризуються об'ємом, швидкістю збірки та різноманітністю [19, 20, 24, 25].

Технологія Big Data має великий потенціал і знаходить практичне застосування у різних аспектах: для збирання та аналізу великих обсягів даних, моніторингу, аналітики та підвищення ефективності бізнес-процесів тощо. Однією з ключових особливостей Big Data є здатність аналізувати дані для виявлення патернів, трендів, зрозуміння поведінки клієнтів, прийняття бізнес-рішень та розвитку нових стратегій на основі цих даних.

У цілому, Big Data – це потужний інструмент, який дозволяє компаніям краще розуміти потреби, ефективніше реагувати на зміни та створювати більш точні та персоналізовані стратегії маркетингу, бізнесу, виробництва тощо.

Широке використання цих технологій обумовлене тим, що Big Data спирається на наступні аспекти: цінність, обсяг, швидкість, різноманітність, надійність та мінливість. Головними принципами залишаються локальність, відмовостійкість та горизонтальна масштабованість, що означає, що обсяги інформації можуть бути величезними [19, 26].

Важливо, що головною характеристикою у Big Data виступає не лише обсяг цих даних, але й здатність швидко обробляти й генерувати цінну інформацію

Саме тому Big Data вимагає використання спеціалізованих технологій для зберігання (наприклад, Hadoop Distributed File System) та обробки (наприклад,

Apache Spark, Apache Flink) великих обсягів даних, оскільки традиційні методи можуть бути неефективними.

Робота з великими даними потребує більш складних інструментів та підходів, а саме [20, 24-26]:

- використання технологій, які призначені для роботи з великими обсягами даних, таких як Hadoop, Spark тощо;
- вибір мови програмування (наприклад, Java) та використання інструментів обробки даних (наприклад, SQL, NoSQL) для ефективної обробки та аналізу великих обсягів даних;
- використання архітектури баз даних, реляційних баз даних, а також інструментів зберігання великих обсягів даних (наприклад, HDFS, Cassandra, MongoDB).

Ці підходи дозволять ефективно працювати з великими обсягами даних та використовувати їх потенціал для аналізу та прогнозування великих обсягів даних, прийняття обґрунтованих бізнес-рішень, персоналізації та вдосконалення обслуговування клієнтів, а значить – для оптимізації бізнес-процесів.

Існують важливі відмінності між принципами обробки та аналізу невеликих, однорідних даних і великих обсягів Big Data, що вимагає спеціалізованих інструментів для обробки і технологій та підходів.

Big Data може включати дані, що надходять з високою швидкістю (наприклад, дані з датчиків IoT), потребуючи швидкої обробки та аналізу в реальному часі.

Також Big Data може зберігати не тільки структуровані, але й напівструктуровані та неструктуровані дані (такі як тексти, зображення, відео, потоки даних), що потребує спеціалізованих підходів до обробки та аналізу, здатних ефективно працювати з великими обсягами, різноманітністю та швидкими потоками даних.

Саме тому Big Data вимагає використання більш складних алгоритмів машинного навчання, обробки потоків даних та використання розподіленого обчислення для здійснення аналізу та прийняття стратегічних рішень.

Отже, основна мета використання методів Big Data – це не лише збільшення розміру даних, але й використання їх для отримання цінної інформації, що сприяє покращенню бізнесу, прийняттю обґрунтованих рішень та створенню конкурентних переваг.

Підсумовуючи, можна зазначити, що Big Data має великий потенціал для використання в різних галузях, включаючи бізнес, науку, медицину, соціальні дослідження тощо, проте ефективне управління та аналіз цих великих обсягів даних вимагає спеціалізованих підходів та інструментів.

1.5 Порівняльний огляд реляційних баз даних та Parquet

Зберігання та організація даних є ключовими аспектами в інформаційних технологіях, оскільки правильна структура даних може значно полегшити доступ, аналіз та використання цієї інформації.

Обрання відповідного методу зберігання та організації даних залежить від сфери використання або конкретних потреб бізнесу, типів даних, які використовуються, та обсягу і частоти доступу до цих даних.

Існують різні підходи до зберігання та організації даних, кожен з яких має свої особливості і використовується для різних цілей.

Одним з них є реляційна база даних (RDBMS) – це сукупність даних, організованих у вигляді таблиць з відношеннями між ними, з яких дані можуть бути використані або повторно зібрані різними способами. Цей тип бази даних ґрунтується на реляційній моделі даних, яка була запропонована Едгаром Коддом у 1970-х роках [29].

Дані в реляційних базах даних представлені у вигляді таблиць. Кожна таблиця складається з рядків (записів) та стовпців (полів), кожен стовпець має свій тип даних. Вони підтримують мову запитів SQL та дозволяють встановлювати зв'язки між різними таблицями для швидкого пошуку та обробки даних.

Реляційні таблиці мають ключі, такі як первинний ключ (Primary Key), який унікально ідентифікує кожен рядок в таблиці, та зовнішні ключі (Foreign Key), які створюють зв'язок між таблицями.

SQL – це стандартна мова запитів, яка використовується для створення, зміни та опитування даних в реляційних базах даних. SQL надає можливість виконувати різні операції, такі як додавання, видалення, зміна та вибірка даних з таблиць.

Реляційні бази даних дотримуються набору вимог до трансакційної системи принципів (ACID: Atomicity, Consistency, Isolation, Durability) для забезпечення найбільш надійної та недорогої її роботи: атомарність, цілісність даних, ізоляцію операцій та стійкість до відмов системи [29].

Реляційні бази даних широко використовуються у багатьох сферах, таких як бізнес, наука, веб-розробка та інші галузі. Вони дозволяють ефективно зберігати, керувати та отримувати доступ до даних, що робить їх надзвичайно важливим інструментом для організації і роботи з інформацією.

Другим форматом, який стає все більш популярним для організації, зберігання та роботи з даними, вважається Parquet. Він був розроблений для ефективного зберігання та обробки даних у великих системах з обробки, як Big Data.

Цей формат забезпечує ефективне зберігання даних у вигляді колонково-орієнтованого файлу, що дозволяє ефективно компресувати дані та забезпечує швидкий доступ до певних стовпців без необхідності читання всього файлу.

Parquet використовує різні методи стиснення, такі як Run-Length Encoding (RLE), Dictionary Encoding та Delta Encoding, що дозволяє зменшити обсяг даних, що зберігаються, без втрати інформації [27].

Цей формат підтримує багато типів даних, включаючи числа, рядки, дати, час та інші, що дозволяє зберігати різноманітні дані в одному файлі. Він добре інтегрується з різними розподіленими системами обробки даних, такими як Apache Hadoop, Apache Spark та інші, дозволяючи їм ефективно обробляти дані, збережені у форматі Parquet. Завдяки колонково-орієнтованій структурі та ефективному

стисненню, Parquet відмінно підходить для аналітики та обробки великих обсягів даних, забезпечуючи швидкий доступ до певних атрибутів.

Для порівняння реляційних баз даних (RDBMS) та Parquet біло обрано та проаналізовано наступні характеристики: структура зберігання даних, здатність до швидкодії та модифікацій, розподілу та паралелізму, їх функціональність та SQL-підтримка (Табл. 1.1).

Таблиця 1.1

Порівняльний огляд реляційних баз даних (RDBMS) та Parquet

	RDBMS	Parquet
Структура зберігання даних	Базуються на табличній структурі, де дані організовані у вигляді таблиць з рядками та колонками, де кожен стовпець має тип даних.	Це колонково-орієнтований формат зберігання даних, що дозволяє оптимізувати доступ до певних полів і зменшує обсяг пам'яті, необхідної для читання певних стовпців.
Швидкодія та ефективність	Призначені для транзакцій та операцій запису/читання даних, але можуть бути менш ефективними для аналітичних операцій над великими обсягами даних.	Оптимізований для аналітики та операцій читання даних, забезпечує високий рівень стиснення, що робить його ефективним для роботи з великими обсягами даних.
Розподіл та паралелізм	Можуть працювати в режимі масштабування горизонтально (за допомогою реплікації та шарування), але масштабування може бути складним для деяких систем.	Це формат файлу, який може бути оброблений різними системами, які підтримують паралельну обробку даних, такими як Apache Spark, Hadoop та інш., дозволяючи легше досягнути розподіленої обробки.

Продовження таблиці 1.1

Порівняльний огляд реляційних баз даних (RDBMS) та Parquet

	RDBMS	Parquet
Використання	Широко використовується в традиційних системах управління базами даних для зберігання транзакційних даних, таких як інформація про клієнтів, фінансові й інші бізнес-дані.	Часто використовується в системах Big Data для зберігання та аналізу великих обсягів структурованих даних, таких як дані журналів, дані сенсорів тощо.
Функціональність та SQL-підтримка	Зазвичай мають повний функціонал SQL, включаючи можливість виконання складних запитів та транзакцій.	Не є СУБД, використовується для зберігання даних у вигляді файлів, вимагає окремих інструментів для роботи з ними. Хоча деякі системи, як Apache Spark SQL, надають підтримку операцій SQL з Parquet.

Порівнюючи функціонал RDBMS та Parquet, слід враховувати, що обидва вони мають свої власні особливості та сфери застосування.

Так, RDBMS надає багато можливостей для транзакцій, відносин між таблицями, використовуючи різноманітні типи даних та потужність мови SQL для роботи з даними.

У свою чергу Parquet забезпечує високу ефективність зберігання даних та прискорює аналіз великих обсягів даних, зокрема у системах Big Data, завдяки ефективному формату зберігання та читання колонок, можливості зберігати дані з вкладеними структурами.

Вибір між RDBMS та Parquet залежить від конкретного використання, вимог до швидкості, обсягу даних та характеристик проєкту. У деяких випадках може

використовуватися поєднання обох технологій для різних аспектів системи зберігання та обробки даних.

Обираючи між RDBMS та Parquet, важливо враховувати тип даних, обсяг і вимоги до операцій зберігання та аналізу даних. RDBMS найчастіше використовуються для операційної роботи з даними, тоді як Parquet стає кращим вибором для аналітичних операцій з великими обсягами даних, оскільки він більш продуктивний у порівнянні з традиційними рядковими форматами.

У багатьох випадках вони можуть використовуватися разом для різних аспектів обробки та зберігання даних.

Результати порівняння, представлені у Табл. 1.1, доводять, що Parquet є найбільш відповідним за своїми характеристиками для створення алгоритму для оптимізації бізнес-процесів. RDBMS і Parquet використовуються для різних цілей та мають різні підходи до організації та зберігання даних: RDBMS більш орієнтований на транзакційні дані, тоді як Parquet – на аналітичні операції з великими обсягами даних.

Отже, Parquet формат даних було обрано для виконання поставленого завдання через його переваги та популярність у системах обробки даних, де важливо ефективно зберігати великі обсяги інформації та забезпечити швидкий доступ до неї для різноманітних аналітичних потреб.

Висновки до 1 розділу

У першому розділі роботи виконано аналіз останніх досліджень і публікацій, у яких започатковано розв’язання питання більш ефективного використання ресурсів для обробки великих обсягів даних.

Розглянуто огляд методів обробки даних та роботи з великими обсягами даних у Big Data, що сприяє покращенню бізнесу, прийняттю обґрунтованих рішень та створенню конкурентних переваг.

Обґрунтовано, що Big Data має великий потенціал для використання в різних галузях, включаючи бізнес, науку, медицину, соціальні дослідження тощо, проте ефективне управління та аналіз цих великих обсягів даних вимагає спеціалізованих підходів та інструментів.

Було визначено основні характеристики бізнес-процесів та необхідність збільшення їх продуктивності за рахунок автоматизації та вдосконалення. У сучасних умовах економічні зміни впливають на діяльність підприємства та його подальший розвиток, тому важливо впроваджувати інновації та модифікації для підвищення ефективності, продуктивності бізнес-процесів, що дасть можливість надавати якісні послуги, залишатися затребуваними та конкурентоспроможними.

Було проаналізовано переваги та недоліки реляційних баз даних (RDBMS) та Parquet, їх можливості для підвищення ефективності використання ресурсів та збільшення швидкості обробки даних при виконанні запитів до них.

2 МОДЕЛЮВАННЯ ТА ВДОСКОНАЛЕННЯ АЛГОРИТМУ ЗЧИТУВАННЯ PARQUET-ФАЙЛІВ

У другому розділі роботи головна увага приділяється проєктуванню та розробці алгоритму для вдосконалення фільтра Pushdown в Parquet файлах. Основна мета цього розділу полягає у розробці ефективного та надійного підходу, який би підвищив продуктивність зчитування даних з Parquet файлів, використовуючи часткову підтримку фільтра Pushdown.

Завданнями цього розділу є:

- огляд та детальний опис формату файлу Parquet, його структури та особливостей, які роблять його популярним для зберігання великих обсягів даних;
- опис концепції фільтра Pushdown та його значення у контексті пришвидшення виконання запитів до даних, зокрема, у випадку використання Parquet;
- аналіз існуючих методів і технологій, що вже використовуються для реалізації фільтра Pushdown в Parquet з визначенням їхніх сильних та слабких сторін;
- обґрунтований вибір конкретних програмних засобів та мови програмування для реалізації алгоритму, враховуючи потреби проекту та доступні ресурси;
- розробити докладного плану алгоритму, включаючи його структуру, основні функції та механізми вдосконалення.

Цей розділ спрямований на створення міцної теоретичної та практичної основи для розробки алгоритму, який здатен значно підвищити продуктивність обробки даних у Parquet файлах, забезпечуючи при цьому точність і ефективність виконання запитів.

2.1 Структура та особливості Parquet формату

Зчитування та обробка Parquet-файлів із вбудованою підтримкою фільтру Pushdown є важливою задачею в області обробки та аналізу даних. Parquet – це колонково-орієнтований формат даних, який використовується для збереження великих обсягів даних з високим ступенем стиснення. Фільтр Pushdown дає можливість відфільтровувати зайві дані на рівні зчитування Parquet-файлів, що дозволяє значно зменшити кількість даних, які потрібно передавати та обробляти на стороні клієнта.

Загалом, зчитування та обробка Parquet-файлів із вбудованою підтримкою фільтру Pushdown вимагає поєднання різних методів та інструментів для досягнення найкращої продуктивності та зниження навантаження на систему обробки даних.

Таблиця 2.1

Переваги та недоліки методу зчитування та обробки Parquet-файлів із вбудованою підтримкою фільтру Pushdown

Переваги	Недоліки
Ефективність за рахунок колонкового формату	Обмежена підтримка складних фільтрів
Зменшення навантаження на мережу	Складність додавання нових фільтр операцій
Масштабованість, підтримка компресії	Відсутність універсальності
Зменшення часу на обробку даних	
Оптимізація ресурсів	

У Табл. 2.1 показані переваги методу зчитування та обробки Parquet-файлів із вбудованою підтримкою фільтру Pushdown. Parquet є колонковим форматом даних, що дозволяє ефективно стискувати дані та прискорювати операції читання та запису. Це особливо корисно для аналізу великих обсягів даних. Важливо те, що

Apache Parquet підтримує роботу з багатьма мовами програмування, включаючи Java, C++, Python, і інші. Це робить його доступним для розробників у різних середовищах.

На вибір цього формату з підтримкою фільтру Pushdown вплинула його поширеність та ефективність, яка дозволяє обробляти дані на більш низькому рівні, що зменшує кількість даних, які потрібно передавати та обробляти. Це покращує продуктивність операцій агрегації та робить їх більш швидкими.

Важливо, що усі операції фільтрації, виконані на рівні зчитування Parquet-файлів, зменшують навантаження на мережу, оскільки передаються менше даних між вузлами обробки. Це пов’язано з тим, що зберігання та обробка даних у форматі Parquet дозволяє масштабувати обробку великих обсягів даних, забезпечуючи ефективну роботу з великими даними. Це у свою чергу призводить до зменшення часу витрат на обробку даних: операції агрегації на рівні зчитування дозволяють знизити час витрат на обробку даних та прискорити аналіз.

На збільшення швидкості зчитування впливає те, що Parquet дозволяє виконувати проекції та фільтрацію колонок, а це дозволяє зчитувати лише необхідні дані і покращує продуктивність. В цілому обраний формат дає оптимізацію ресурсів за рахунок зменшення обсягу даних, які передаються та обробляються, дозволяє ефективно використовувати обчислювальні ресурси.

Серед недоліків Apache Parquet, показаних у Табл. 2.1, можна виділити складність редагування та наявність певних вимог до пам’яті. Оскільки Parquet є колонковим форматом збереження, то редагування або вставка нових даних може бути складним завданням, особливо у порівнянні з рядковими форматами. Через колонкову структуру даних, операції зчитування та обробки можуть вимагати більше пам'яті для тимчасового збереження даних. Важливо враховувати, що у деяких випадках передача Parquet-файлів може бути менш ефективною через колонкову структуру, оскільки обмін даними вимагає передачі багатьох невеликих блоків даних. Це викликано тим, що Parquet – неуніверсальний формат даних, і не завжди підходить для всіх типів даних або сценаріїв обробки [27].

Недоліками методу зчитування та обробки Parquet-файлів із вбудованою підтримкою фільтру Pushdown, можна вважати те, що цей метод може бути недоступний в деяких середовищах, адже не всі бібліотеки та системи підтримують фільтр Pushdown для Parquet-файлів. Потрібно враховувати, що цей формат має складність налаштування фільтру Pushdown, що вимагає глибокого розуміння формату Parquet та внутрішньої структури даних. Дуже складні або вкладені запити можуть викликати труднощі або навіть зробити неможливим застосування Pushdown, що знижує потенційні переваги. Можливе потенційне збільшення навантаження: хоч фільтр Pushdown має на меті зменшити обсяги переданих даних, проте він може збільшити навантаження на файлове сховище, що обробляє фільтрацію.

Підсумовуючи, слід зазначити, що Parquet широко використовується в індустрії обробки великих даних, особливо у випадках, де потрібна висока продуктивність читання та ефективне зберігання. Його часто вибирають для аналітичних застосунків, де необхідно швидко обробляти великі обсяги даних. Також, формат є популярним у бізнес-проєктах, що вимагають сумісності з різними інструментами обробки даних, оскільки він може служити універсальним форматом для зберігання даних у розподілених системах.

Усупереч деяким недолікам, метод зчитування та обробки Parquet-файлів із вбудованою підтримкою фільтр Pushdown є корисним для удосконалення агрегаційних операцій над великими обсягами даних. Однак він вимагає уважної розробки та налаштування для досягнення оптимальних результатів.

Загалом, Apache Parquet є потужним і ефективним форматом даних для обробки та збереження великих обсягів даних, але через свої обмеження вимагає розробки та розгляду в конкретних сценаріях використання.

2.2 Аналіз існуючих програмних засобів

Механізм фільтру Pushdown в контексті Parquet потрібен для оптимізації запитів, коли частина операцій фільтрації та обробки даних виконується на рівні

збереження даних у форматі Parquet. Це дозволяє обробляти лише ті дані, які є необхідними для виконання запиту, зменшуючи обсяг даних, які потрібно завантажити для подальшої обробки, тим самим поліпшуючи продуктивність та швидкість виконання операцій.

Існує кілька методів та інструментів, які застосовуються для реалізації механізму фільтру агрегації Pushdown у контексті Parquet файлів. Розуміння цих підходів дозволить нам краще оцінити поточний стан технологій та виявити потенційні напрями для інновацій та оптимізації.

1.Apache Hadoop, що використовує HDFS (Hadoop Distributed File System) для зберігання даних і має інтегровані можливості для обробки Parquet файлів. Підтримка Filter Pushdown в Hadoop може бути досягнута через використання додаткових інструментів та бібліотек, які оптимізують взаємодію з Parquet.

2.Apache Spark – один з найбільш популярних фреймворків для обробки великих даних. Він підтримує різноманітні операції з даними, включаючи Filter Pushdown для Parquet файлів. Spark автоматично застосовує оптимізації на рівні читання даних, що дозволяє ефективно відфільтровувати дані ще до їх загрузки в пам'ять.

3. Impala – інструменти, які працюють з Parquet, можуть використовувати Filter Pushdown для ефективної обробки запитів, які працюють з великими обсягами даних для забезпечення високої продуктивності при читанні та аналізі великих даних. Impala ефективно використовує механізм Filter Pushdown для оптимізації запитів.

4. Presto – це високопродуктивна, розподілена система обробки даних з підтримкою Parquet, яка використовує Filter Pushdown, оптимізуючи запити за швидкістю та ефективністю використання ресурсів.

5. Apache Drill – це розподілена система для роботи з даними, яка дозволяє виконувати запити та аналізувати дані з різних джерел даних, включаючи реляційні бази даних, NoSQL, файли JSON, Parquet, CSV та багато інших. Вона здатна виконувати Filter Pushdown, щоб підвищити продуктивність запитів.

6. Додаткові інструменти оптимізації та бібліотеки, розроблені для підвищення ефективності взаємодії з Parquet файлами. Ці інструменти можуть включати плагіни, модулі та бібліотеки, які допомагають в оптимізації запитів через механізм фільтру Pushdown.

Кожен з цих методів та інструментів, які застосовуються для реалізації механізму агрегації Pushdown у контексті Parquet файлів, має свої унікальні особливості та рішення. Вибір конкретного рішення для реалізації фільтру Pushdown у Parquet залежить від конкретних вимог проекту, включаючи масштаб даних, складність запитів та інтеграцію з іншими компонентами системи. Аналіз цих рішень дає нам змогу зрозуміти, як можна покращити або адаптувати існуючі підходи для ефективної обробки даних у Parquet файлах.

2.3 Розроблення функціональних вимог для створення алгоритму

Створений алгоритм обробки файлів має працювати за етапами виконання запиту в системах обробки даних, представленими на рис. 2.1.

Рис. 2.1. Ключові етапи виконання запиту в системах обробки великих наборів даних

Система зчитування та обробки даних має наступні ключові етапи виконання запиту. Спершу вона отримує запит на обробку даних та починає їх зчитування з відповідного джерела. Це джерело може бути файловою системою, базою даних, потоком даних тощо. Система виконує його попередню обробку та валідацію.

Після цього виконується побудова логічного плану, у нашому випадку Apache Spark конструює логічний план, який є високорівневим представленням необхідних для запиту операцій.

Під час роботи планувальник враховує різні фактори для визначення оптимального фізичного плану виконання запиту: оцінює обсяги даних, вартість різних способів виконання запиту, враховуючи фізичні операції, робить вибір оптимальної стратегії.

На цьому етапі вирішується, як та де буде виконуватися запит. Є 2 опції – виконання запиту на стороні файлового сховища та виконання решти запиту в оперативній пам'яті.

Виконання запиту на стороні файлового сховища є кращою опцією, оскільки це підвищує швидкість виконання та зменшує кількість необхідних обчислювальних ресурсів.

Для виконання решти запиту в оперативній пам'яті відбувається генерація коду та вичитка необхідних даних з файлового сховища.

Планувальник приймає рішення щодо того, як саме буде виконуватися запит, вибираючи найефективніший фізичний план виконання операцій для задоволення логічного запиту користувача до бази даних. Отже, для прискорення зчитування даних нам необхідно, щоб якомога більша частина запиту виконувалася на стороні файлового сховища.

Після виконання оптимізації та генерації коду відбувається виконання запиту на стороні файлового сховища. Виконання решти запиту буде відбуватися в оперативній пам'яті, за умови, коли повне виконання на стороні файлового сховища неможливе. Після виконаних дій відбувається повернення фінального результату користувачу. Множина згенерованих фізичних планів показана у рис. А.1 (Додаток А).

При виборі програмних засобів для вдосконалення фільтра Pushdown в Parquet файлах, були обрані наступні ключові критерії:

- сумісність з Parquet, для цього інструмент повинен мати вбудовану підтримку або легку інтеграцію з форматом Parquet;
- підтримка розширених фільтрів, яка забезпечує здатність обробляти не тільки прості, але й складні фільтри;
- масштабованість та продуктивність для обробки великих обсягів даних із високою продуктивністю.
- гнучкість у розширенні функціональності, щоб забезпечити можливість додавати користувацькі правила;
- активна спільнота розробників та наявність ресурсів для підтримки.

Після ретельного аналізу та зважуючи на зазначені критерії, було вирішено використовувати Apache Parquet, Apache Spark, фільтр Pushdown як основні інструменти для реалізації вдосконаленого алгоритму зчитування Parquet файлів для бізнес-процесів, пов'язаних з аналізом великої кількості даних. Це дозволить покращити швидкісні характеристики алгоритму зчитування Parquet-файлів із вбудованою підтримкою фільтру Pushdown, підвищити ефективність використання ресурсів, збільшити швидкість обробки даних при виконанні запитів до них.

2.4 Інструментарій для створення алгоритму

Використання спеціалізованих бібліотек, фреймворків та інструментів залежить від потреб проєкту та мови програмування, адже кожні інструменти надають різні функції та можливості для роботи з даними, обробки, аналізу та розробки алгоритмів.

Для створення алгоритму у Java обрано для використання наступні бібліотеки, фреймворки:

1.Parquet-бібліотека – Apache Parquet (Java). Це офіційна бібліотека для роботи з форматом Parquet, розроблена як частина проекту Apache Parquet. Вона надає можливості для читання та запису даних у форматі Parquet в Java і спеціально

створена для роботи з великими обсягами даних та оптимізації їх зберігання, що робить її популярною. Вона надає можливість зчитування, запису та оптимізації даних у форматі Parquet. Деякі реалізації Parquet, включаючи обрану, такі як Parquet-cpp, підтримують агрегацію, стиснення Pushdown.

2.Фреймворки обробки даних.

Apache Spark – це розподілена обчислювальна система та інструмент для обробки та аналізу великих обсягів даних. Вона надає ефективний функціонал для обробки даних у пам'яті, обробки потокових даних, машинного навчання, глибокого аналізу тощо.

До основних характеристик Apache Spark відноситься: швидкодія, модульність, підтримка багатьох джерел даних та складних операцій, масштабованість, простота використання, що робить його дуже ефективним та популярним інструментом.

Spark пропонує високу швидкодію завдяки використанню у пам'яті обробки даних та оптимізованій обробці завдань паралельно. Фреймворк розподілений і може працювати на кластерах серверів, дозволяючи розподілену обробку даних.

Spark має багато модулів, таких як Spark SQL для роботи з SQL-подібними запитами, MLlib для машинного навчання, GraphX для обробки графів, та Streaming для обробки потокових даних. Ці модулі дозволяють розширювати функціональність Spark у різних сферах обробки даних.

Apache Spark використовує іншу архітектуру обробки даних, яка значно покращує продуктивність порівняно з іншими фреймворками, такими як Hadoop MapReduce. Це досягається завдяки ін-меморі обробці даних, що дозволяє уникнути запису даних на диск після кожної операції [28, 29].

Цей фреймворк може працювати з різними джерелами даних, такими як Hadoop Distributed File System (HDFS), Apache HBase, Apache Cassandra та багатьма іншими, бо він надає можливості для виконання різних операцій над даними, включаючи фільтрацію, згортання (aggregation), сортування, з'єднання тощо.

Фреймворк може легко масштабуватися залежно від потреб користувача, дозволяючи працювати з великими обсягами даних, підтримує інтерактивний

режим роботи для швидкого взаємодії з даними та виконання аналізу в реальному часі.

Ці характеристики роблять Apache Spark потужним інструментом для обробки та аналізу даних у великих обсягах та в розподілених середовищах.

Недоліками Apache Spark можна вважати вимоги до ресурсів: для ефективної роботи він потребує великої кількості пам'яті та обчислювальних ресурсів. Це може бути високими витратами на обладнання та обслуговування. Відповідно запуск Spark може займати деякий час, особливо на великих кластерах, що може вплинути на час відповіді на запити. Налаштування Spark може бути складним завданням і вимагає глибокого розуміння архітектури та можливих параметрів від розробників.

Apache Spark було обрано, тому що він може працювати з даними, які зберігаються у форматі Parquet, використовуючи їх як одне з джерел даних для обробки та аналізу в своїх операціях. Загалом, Apache Spark – це потужний інструмент для обробки та аналізу даних, але він має свої обмеження та вимоги до ресурсів, які слід враховувати при його використанні.

2.5 Принципи вдосконалення фільтру Pushdown

Фільтр Pushdown є ключовим принципом в прискоренні запитів зчитування даних, особливо у контексті великих даних та розподілених обчислень. Ідея фільтру Pushdown полягає в тому, щоб відфільтровувати непотрібні дані на ранньому етапі обробки запиту, перш ніж дані будуть повністю зчитані чи передані до наступних етапів обробки. Це досягається шляхом перенесення критеріїв фільтрації ближче до джерела даних.

У контексті Parquet фільтр Pushdown означає застосування фільтрів прямо на етапі зчитування даних з файлу. Це дозволяє виключити непотрібні рядки чи колонки з подальшого аналізу, тим самим зменшуючи обсяг даних, що передаються та обробляються.

Це відіграє важливу роль в прискоренні запитів: оскільки цей фільтр дозволяє відфільтрувати нерелевантні дані на самому початку, це суттєво знижує обсяг

даних, який потрібно передавати та обробляти на сервері (рис. 2.2). Це особливо важливо в великих даних, де обсяги можуть бути дуже значними.

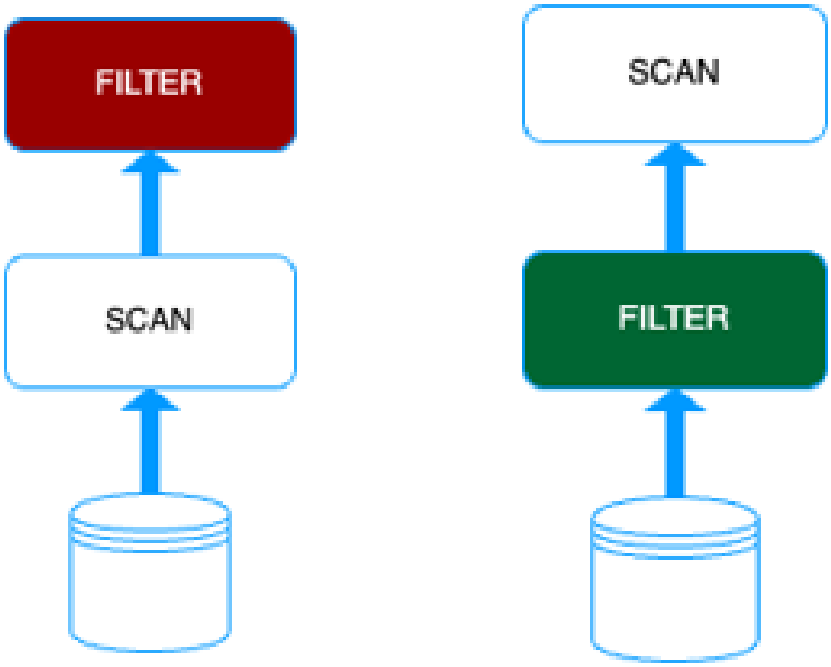


Рис. 2.2. Модель запропонованого вдосконалення роботи фільтру Pushdown

Завдяки видаленню непотрібних даних на ранньому етапі, можна досягти значного підвищення продуктивності обробки запитів. Це знижує навантаження на мережу та поліпшує час відгуку системи.

Фільтр Pushdown допомагає ефективніше використовувати обчислювальні ресурси. Замість того, щоб завантажувати великі масиви даних у пам'ять для обробки, система може зосередитись лише на релевантних даних, знижуючи навантаження на CPU та пам'ять.

Збільшення швидкості виконання запитів та зменшення часу чекання є особливо важливими в середовищах з великою кількістю одночасних запитів, таких як веб-сервіси та аналітичні платформи.

Зазначимо, що використання фільтру Pushdown обумовлене тим, що він відіграє вирішальну роль в прискоренні запитів, особливо у великих та

розподілених системах зберігання даних [28, 31]. Це дозволяє підвищити ефективність обробки даних, забезпечуючи швидкісні та ресурсні переваги.

Підтримка вдосконалення фільтру Pushdown може бути реалізована через прискорення запитів та роботи внутрішніх структур даних. Наприклад, підтримка індексів, прискорення зчитування та фільтрація колонок можуть значно покращити продуктивність та швидкодію запиту.

Важливо, що Apache Spark надає інтеграцію з Parquet і можливість використовувати фільтр Pushdown для виконання операцій фільтрації на рівні розподіленої обробки даних. Це дозволяє працювати з великими обсягами даних ефективно і допоможе у реалізації поставленого завдання.

Вдосконалення фільтру Pushdown бути важливою частиною прискорення обробки запитів. Деякі запити можуть бути переписані для зменшення обсягу даних, що передається, та більш ефективної їх обробки.

Для підтримки фільтру Pushdown, можуть використовуватися метадані, які зберігають інформацію про структуру та типи даних у Parquet-файлах. Ці метадані можуть допомагати прискорювати запити та визначати, які операції можуть бути виконані на рівні зчитування.

Використання розподілених систем обробки даних, таких як Apache Spark, дозволяє виконувати операції агрегації на рівні вузлів, що знижує навантаження на мережу та підвищує продуктивність.

Використання індексів для пошуку даних у Parquet-файлах може покращити швидкість операцій агрегації, оскільки вони дозволяють швидше знаходити необхідні дані.

Для моделювання часткового фільтру Pushdown та оцінки його ефективності можна використати наступний підхід з використанням заданих параметрів:

- N – кількість записів у Parquet файлі,
- M – кількість записів, що відповідають критеріям фільтра,
- T_{total} – загальний час обробки без удосконалення,
- $T_{total\_pushdown}$ – загальний час обробки з удосконаленням,
- T_{read} – час, необхідний для зчитування одного запису,

- $T_{process}$ – час обробки одного запису,

-$T_{process\_pushdown}$ – час обробки одного запису на стороні файлового сховища.

Розрахунок загального часу обробки без удосконалення T_{total} проводиться за формулою 2.1:

$$T_{total} = T_{read} \times N + T_{process} \times N \quad (2.1)$$

Розрахунок загального часу обробки з удосконаленням $T_{total\_pushdown}$ проводиться за формулою 2.2:

$$T_{total\_pushdown} = T_{read} \times (N - M) + T_{process} \times (N - M) + T_{process\_pushdown} \times M, \quad (2.2)$$

$$\text{де } M \leq N,\ T_{process\_pushdown} < T_{read} + T_{process}.$$

Величина M обернено пропорційна загальній швидкості роботи алгоритму.

Розрахунок оцінки ефективності (*Efficiency*) проводиться за формулою 2.3:

$$Efficiency = (T_{total} - T_{total\ pushdown}) / T_{total} \quad (2.3)$$

Цей підхід дозволяє оцінити вплив фільтру Pushdown на загальний час обробки, порівнюючи загальний час обробки даних без удосконалення з часом, що витрачається при використанні вдосконаленого алгоритму.

2.6 Проєктування алгоритму

Процес проєктування алгоритму для удосконалення фільтра Pushdown в Apache Spark включає ряд важливих вимог, які повинні бути виконані, а саме:

- забезпечення можливості обробки більш складних фільтрів, ніж стандартні можливості Spark;

- здатність ефективно перетворювати складні фільтри, такі як BETWEEN, на комбінації більш простих фільтрів, здатних до агрегації через Pushdown;
- підтримка зменшення обсягу даних, які потребують обробки, шляхом ефективного використання Pushdown механізму.
- повна сумісність з Apache Spark та його системою та інструментами;
- масштабованість та здатність забезпечувати високу продуктивність при обробці великих наборів даних.

Удосконалений алгоритм змінює логічний план, який показує послідовність операцій, які виконуються для виконання запиту, представлений у рис. 2.3. Результатом логічного плану є множина фізичних планів, яка показує можливі сценарії виконання запитів.

```
SELECT product FROM customers
WHERE regional_sales BETWEEN SQRT(sales) and 100_000
```

```
LogicalProject(expressions=[product]])
  LogicalFilter(condition=[regional_sales between(sqrt(sales), central_sales)])
    LogicalScan(file=[customers, parquet])
```

```
LogicalProject(expressions=[product]])
  LogicalFilter(condition=[>=(regional_sales, sqrt(sales))])
    LogicalFilter(condition=[<=(regional_sales, central_sales)])
      LogicalScan(file=[customers, parquet])
```

Рис. 2.3. Змінений логічний план

Планувальник у системі відповідає за перетворення логічного плану запиту у фізичний план виконання операцій над даними. Фізичний план, у свою чергу, визначає спосіб реалізації цих операцій на рівні конкретної апаратної або програмної архітектури (Рис. 2.4).

```
RamProject(expressions=[product])
  RamFilter(condition=[between(sqrt(sales), central_sales)])
    PushedProject(expressions=[product, sales, regional_sales, central_sales])
      PushedScan(file=[customers, parquet])

RamProject(expressions=[product])
  RamFilter(condition=[>=(regional_sales, sqrt(sales))])
    PushedProject(expressions=[product, sales, regional_sales])
      PushedFilter(condition=[<=(regional_sales, central_sales)])
        PushedScan(file=[customers, parquet])
```

Рис. 2.4. Отриманий фізичний план

У результаті виконання вдосконалення алгоритму, використовуючи метод оцінки вартості, буде обраний найоптимальніший варіант з точки зору часової складності, використання пам'яті, вартості витрат на операцію зчитування та мережевої передачі.

Запропонований алгоритм було структурно розроблено з використанням власного правила в Apache Spark, що модифікує процес обробки фільтрів:

- основною функцією алгоритму є переписування складного фільтру на два окремих фільтри. Для прикладу можемо розглянути BETWEEN фільтр, в результаті буде один фільтр для оператора 'більше або дорівнює' (>=) та інший для 'менше або дорівнює' (<=), що дозволяє використовувати Pushdown для частини умов, які підтримуються Parquet;
- у випадках, коли один з фільтрів не підтримує Pushdown (наприклад, через складність операцій), алгоритм залишає цю частину фільтра для обробки на більш високому рівні;
- через ефективне застосування цього правила, алгоритм прискорює процес читання даних, зменшуючи кількість непотрібних даних, які потребують обробки.

Алгоритм, розроблений для вдосконалення агрегації Pushdown, представлений на рис. 2.5. Він базується на додаванні користувацького правила (rule), яке спрямоване на використання фільтрів у Parquet. Основною метою цього правила є переписування складних фільтрів на декілька простих.

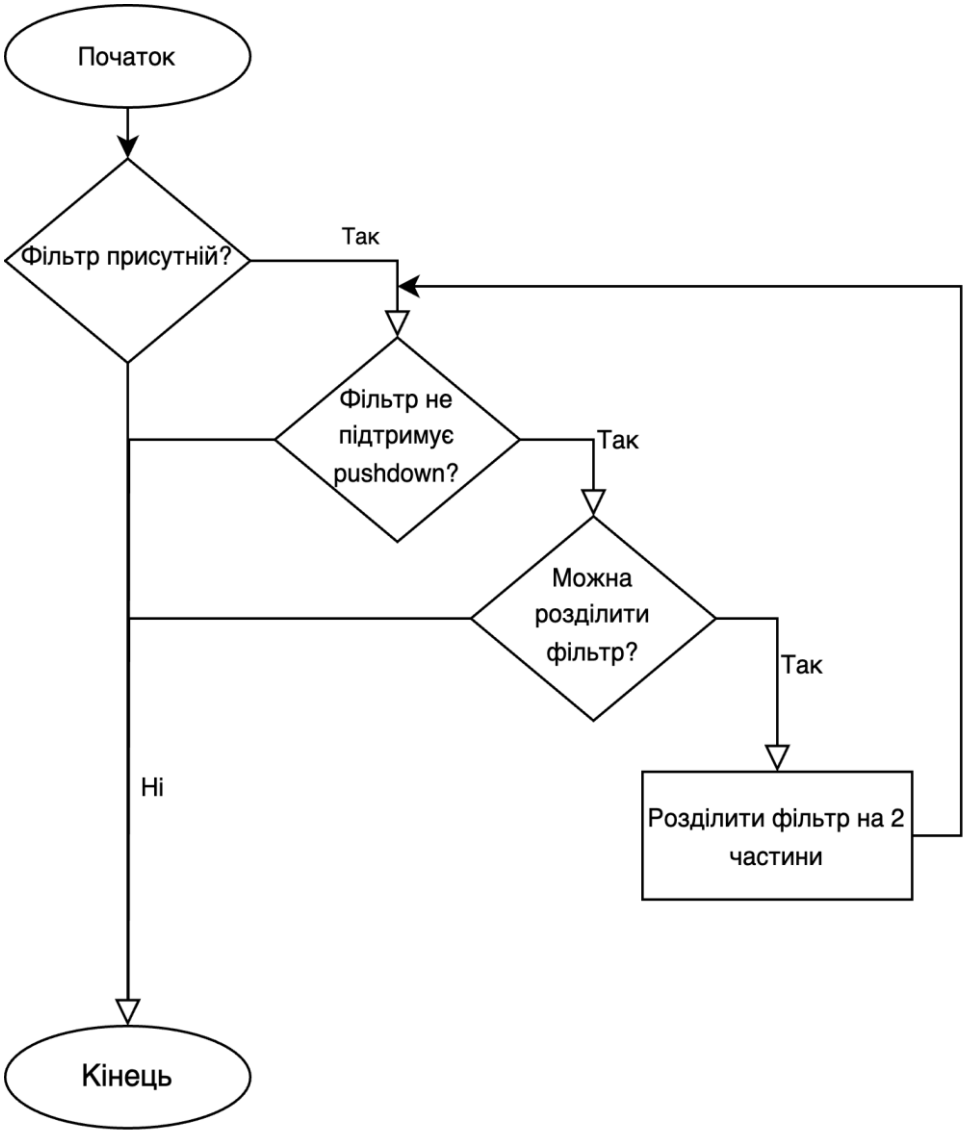


Рис. 2.5. Алгоритм розділення фільтрів

Це розбиття дозволяє покращити використання Filter Pushdown, оскільки Apache Spark ефективніше обробляє ці простіші умови.

Для впровадження цієї логіки, розроблено спеціалізоване правило у мові програмування, яка використовується в Apache Spark. Це правило втручається у

процес виконання запитів, визначаючи можливість застосування переписаних фільтрів для конкретних сценаріїв читання даних.

Обраний підхід дозволяє не тільки підвищити продуктивність обробки запитів у Spark, але й забезпечує гнучкість у використанні більш складних фільтрів, які традиційно не підтримуються стандартними механізмами Pushdown. У результаті це відкриває можливості для більш ефективної роботи з великими наборами даних, значно підвищуючи загальну продуктивність системи.

Особливості реалізації розробленого вдосконаленого алгоритму обробки Parquet файлів:

- максимальне прискорення обробки фільтрів, що робить процес читання даних з Parquet файлів більш ефективним, забезпечує значне зниження обсягу даних, які потрібно обробити, і підвищує швидкість запитів;
- гармонійна інтеграція в існуючу архітектуру Apache Spark, не вимагаючи великих змін у стандартній конфігурації чи інфраструктурі;
- створення правила таким чином, щоб його можна було легко адаптувати або розширити для подальших оптимізацій, забезпечуючи високу гнучкість у використанні алгоритму для різних сценаріїв обробки даних;
- мінімізація впливу на загальну продуктивність, залишаючи цю частину для обробки на рівні двигуна Spark, у випадках, коли частина фільтра не підтримує Pushdown,

Виконуючи вказані умов, розроблений алгоритм відіграє ключову роль у підвищенні ефективності та продуктивності роботи з великими наборами даних у Parquet, роблячи процес читання та обробки даних значно швидшим і ефективнішим.

2.7 Теоретичні аспекти методу оцінки вартості з використанням AHP

Використовуючи модель оцінки вартості ресурсів, ми можемо оцінити часові витрати, мережеві, витрати пам’яті та витрати на операцію зчитування. Щоб обрати найефективніший фізичний план серед кількох фізичних планів, які генерує Apache

Spark, необхідно використати математичну модель для отримання кінцевої вартості, ка може бути представлена формулою 2.4:

$$\text{Cos t}\,(T) = f\,(\,O(T),\ S(T), P(T), D(T)\,) \qquad (2.4)$$

де $O(T)$ – вартість часових витрат,

$S(T)$ – вартість витрат пам'яті,

$P(T)$ – вартість мережевих витрат,

$D(T)$ – вартість витрат на операції зчитування.

Spark використовує модель AHP, з метою обрання найефективнішого з точки зору вартості фізичного плану. Для оцінювання альтернативних фізичних планів обрано часові витрати, мережеві, витрати пам’яті та витрати на операцію зчитування. Загальна схема AHP показана на рис. 2.6:

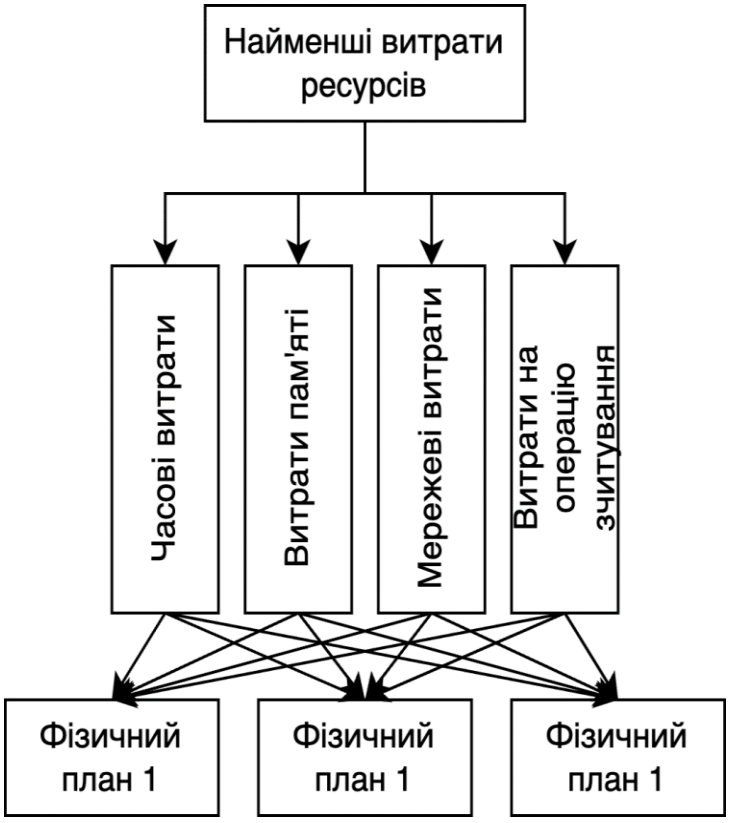


Рис. 2.6. Модель AHP побудови фізичного плану

Для початку, щоб побудувати модель AHP, потрібно скласти матрицю A для попарних порівнянь згідно зі шкалою, запропонованою Thomas L. Saaty [30]. Кожен

елемент матриці α_{iy} приймає значення від 1 до 9. Показник α_{iy} відображає важливість *i*-го виміру у порівнянні з *j*-м виміром.

Потім можна використовувати метод сумування для підрахунку власних векторів U_A матриці A за формулами 2.5, 2.6:

$$U = (u_1, u_2, \ldots, u_n), \tag{2.5}$$

$$u_k = \frac{\sum_{j=1}^{n} a_{kj}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}} \tag{2.6}$$

Після цього вартість може бути представлена як сума вартостей за кожним виміром кожного фізичного плану $L_1, L_2, L_3, \ldots, L_n$. Матрицю порівнянь слід сконструювати відповідно для кожного виміру. Наприклад, матриця порівнянь вартості часової складності X_0 має вигляд, представлений формулою 2.7 [30]:

$$X_0 = \begin{matrix} x_{11}, & x_{12}, & \ldots, & x_j \\ x_{21}, & x_{22}, & \ldots, & x_{2j} \\ \vdots & & & \\ x_{i1}, & x_{i2}, & \ldots, & x_{ij} \end{matrix} \tag{2.7}$$

де x_{ij} представляє значення порівняння фізичного плану *i* з фізичним планом *j* у вимірі часових витрат.

Аналогічно необхідно побудувати матрицю порівняння Xs витрат пам'яті, матрицю порівняння X_P мережевих витрат та матрицю порівняння X_D витрат на операцію зчитування. Далі отримуємо власні вектори U_O матриці порівняння вартості відповідно до формул (2.8 – 2.11):

$$U_O = (u_{o1}, u_{o2}, \ldots, u_{on}), \tag{2.8}$$

$$U_S = (u_{s1}, u_{s2}, \ldots, u_{sn}), \tag{2.9}$$

$$U_P = (u_{p1}, u_{p2}, \ldots, u_{pn}), \tag{2.10}$$

$$U_D = (u_{d1}, u_{d2}, \ldots, u_{dn}) \tag{2.11}$$

Вартість кожного фізичного плану $W_L = (w_{L1}, w_{L2}, w_{L3}, \ldots, w_{Ln})$ може бути отримана шляхом підставляння власних векторів U_A, U_O, U_S, U_P, U_D через формулу 2.12:

$$W_{L_i} = \sum_{j=1}^{n} u_{oj} a_{ij} + \sum_{j=1}^{n} u_{sj} a_{ij} + \sum_{j=1}^{n} u_{pj} a_{ij} + \sum_{j=1}^{n} u_{dj} a_{ij} \quad (2.12)$$

Найдешевший фізичний план обирається наступною формулою (2.13). Це і є фінальне представлення запиту перед його виконанням.

$$L_i = min(W_{L_1}, W_{L_2}, W_{L_3}, \ldots, W_{L_n}) \quad (2.13)$$

Проведені розрахунки показують, що алгоритм для визначення оптимального фізичного плану в Spark SQL використовує модель оцінювання вартості. Використання методу аналітичного ієрархічного процесу (AHP) дозволяє систематично порівнювати різні фізичні плани з урахуванням часової та просторової складності, а також витрат на мережеву передачу та дискові операції.

Завдяки цьому підходу забезпечується покращення продуктивності запитів, що сприяє оптимальному використанню системних ресурсів та зниженню загальних витрат обробки даних.

Висновки до 2 розділу

У другому розділі було виконано ретельне проєктування та вдосконалення алгоритму фільтр Pushdown для Parquet файлів у контексті Apache Spark.

Основна увага була зосереджена на створенні ефективного механізму для обробки складних фільтрів, та їхньому переписуванні на множину простих фільтрів, що дозволяють частково використовувати можливості Pushdown.

Розроблений алгоритм відкриває нові можливості для прискореного читання даних із Parquet файлів, забезпечуючи більш ефективне використання ресурсів та зниження часу відгуку на запити.

Алгоритм було розроблено таким чином, щоб він гармонійно інтегрувався в екосистему Apache Spark, не потребуючи істотних змін у стандартній інфраструктурі та архітектурі.

Завдяки удосконаленню фільтра Pushdown, алгоритм забезпечує значне підвищення продуктивності при роботі з великими наборами даних, особливо у складних аналітичних запитах.

Розроблений механізм має високий потенціал для подальшого розширення та адаптації до інших типів складних фільтрів, що робить його цінним інструментом для розвитку оптимізаційних стратегій в обробці великих даних.

3 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМУ

3.1 Методологія проведення оцінки та тестування

У третьому розділі роботи основна увага приділяється аналізу ефективності розробленого алгоритму фільтру Pushdown для Parquet файлів у Apache Spark.

Метою запланованих тестувань є глибоке дослідження та оцінка ефективності алгоритму, його впливу на продуктивність обробки даних та можливості його практичного застосування в реальних сценаріях.

Для цього потрібно зробити детальний аналіз ефективності розробленого алгоритму, визначивши його вплив на швидкість та продуктивність обробки запитів в Apache Spark, порівнявши розроблений алгоритм зі стандартними підходами, щоб визначити його переваги та обмеження.

Методологія проведення оцінки та тестування алгоритму включає в себе ряд кроків та підходів, спрямованих на виявлення помилок, визначення роботи алгоритму та забезпечення відповідності вимогам. Для цього було продумано детальний план тестування, включаючи обрання тестових сценаріїв, тестових випадків, визначення ресурсів та графіку проведення тестування. Для детального аналізу результатів планується провести серію тестів алгоритму на реальних наборах даних для оцінки його продуктивності в різних умовах. Це дасть змогу визначити, наскільки ефективно алгоритм покращує час обробки запитів, зокрема в умовах великих обсягів даних.

Після проведення тестових процедур згідно з розробленими сценаріями планується відстеження знайдених помилок або дефектів в програмному забезпеченні до їх виправлення та підтвердження виправлень. Після цього проводиться повторне тестування, виконується аналіз результатів тестування для прийняття рішення про готовність програмного забезпечення та розробляються рекомендації до його реалізації.

Для визначення ефективності необхідно порівняти результати розробленого алгоритму з результатами, отриманими з використанням стандартних методів у Apache Spark, та надати рекомендації щодо практичного застосування розробленого алгоритму, враховуючи його сильні та слабкі сторони.

Цей розділ надає комплексний погляд на дієвість та ефективність розробленого алгоритму, дозволяючи оцінити його практичну цінність та потенціал для використання у різноманітних сценаріях обробки даних.

3.2 Підготовка до тестування

Підготовка до тестування удосконаленого алгоритму фільтру Pushdown в Apache Spark включає ретельний підбір тестових даних. Тестові дані відіграють критичну роль у оцінці ефективності та надійності алгоритму, а також у виявленні потенційних недоліків або обмежень. Ключові аспекти вибору тестових даних включають:

- різні типи та структури, від простих до складних, що дозволяє оцінити роботу алгоритму в різних сценаріях для отримання всебічної оцінки;
- реальні дані або дані, що імітують реальні умови, що є необхідним та важливим для забезпечення валідності тестування;
- набори даних, які повинні відображати різні розміри та обсяги, включаючи великі Data set, щоб перевірити масштабованість та продуктивність алгоритму.

Методологія тестування була обрана з метою забезпечення ефективної та всебічної оцінки алгоритму:

- юніт-тестування для перевірки окремих компонентів алгоритму для забезпечення їх правильної роботи та взаємодії;
- інтеграційне тестування для оцінки взаємодії розробленого алгоритму з різними компонентами Apache Spark та з тестовими даними;

- тестування продуктивності за рахунок вимірювання часу виконання запитів та порівняння з показниками до впровадження алгоритму, включаючи тестування на різних обсягах даних;
- стрес-тестування для перевірки алгоритму в умовах високого навантаження для визначення його стабільності та масштабованості;
- порівняльний аналіз з використанням контрольних груп даних та сценаріїв для порівняння роботи алгоритму зі стандартними методами обробки в Apache Spark.

Такий підхід до підготовки та тестування забезпечує глибоке розуміння роботи алгоритму та його ефективності.

3.3 Тестування швидкісних характеристик

Тестування швидкісних характеристик розробленого удосконалення алгоритму для фільтра Pushdown в Apache Spark було зосереджене на оцінці його впливу на продуктивність читання та обробки даних.

Для процедури тестування було налаштоване контрольоване тестового середовище зі стабільними параметрами апаратного забезпечення та конфігурації системи для забезпечення послідовності тестів.

Виконання запитів до даних Parquet файлів виконувалися з використанням стандартного механізму Apache Spark та з удосконаленим алгоритмом, щоб оцінити вплив останнього на швидкість обробки.

Збір метрик продуктивності відбувався за рахунок вимірювання часу відповіді на запити, часу зчитування даних з файлів, а також загального часу виконання обробки. Тести були повторені декілька разів для забезпечення надійності та точності результатів.

Аналіз результатів тестування швидкісних характеристик дозволив отримати наступні висновки:

1. Покращення часу відповіді. Спостерігалося значне зменшення часу відповіді на запити при використанні удосконаленого алгоритму у порівнянні зі

стандартними методами. Це вказує на підвищену ефективність фільтрації даних на ранньому етапі.

2. Ефективність обробки великих наборів даних. Особливо помітне покращення продуктивності було виявлено при роботі з великими наборами даних, що підтверджує практичну користь удосконаленого алгоритму.
3. Консистентність результатів: повторні тести підтвердили покращення продуктивності, вказуючи на стабільність та надійність модифікації.
4. Аналіз впливу на різні типи запитів. Тестування також включало аналіз різних типів запитів, що дозволило оцінити універсальність та ефективність алгоритму в різних сценаріях використання.

Ці результати тестування підкреслюють значний вплив розробленого алгоритму на швидкість та ефективність обробки даних у Apache Spark, особливо у контексті великих наборів даних зі складними фільтрами. Такий позитивний вплив відкриває шлях для його ширшого застосування у різноманітних областях обробки великих наборів даних.

3.4 Порівняльний аналіз роботи алгоритму

Для оцінки відносної ефективності розробленого удосконалення алгоритму фільтра Pushdown було проведено порівняльний аналіз з поточною версією Apache Spark 3.5.0. Основні точки порівняння включали стандартні методи в Apache Spark: оцінка різниці у продуктивності між розробленим алгоритмом та вбудованими можливостями фільтрації в Apache Spark, а також порівняння з альтернативними методами та інструментами, призначеними для покращення продуктивності читання з Parquet файлів, такими як спеціалізовані плагіни або інші фреймворки обробки даних.

За результатами проведеного порівняльного аналізу можна зробити наступні висновки:

1. Розроблений алгоритм перевищує стандартні можливості Apache Spark у плані швидкості обробки та ефективності читання даних, особливо у випадках складних запитів та великих обсягів даних.
2. У порівнянні з іншими існуючими рішеннями, розроблений алгоритм демонструє конкурентоспроможну продуктивність, особливо у контексті вирішення складних задач фільтрації.
3. Розроблений алгоритм не лише підвищує швидкість обробки даних, але й забезпечує високий рівень універсальності, ефективно справляючись із різними типами запитів.

На рис. 3.1, 3.2 показано відмінності у швидкості виконання запитів. Для побудови графіків використовувалися дані в мілісекундах по осі У, та кількість рядків по осі Х. Було проведено порівняння по двом сценаріям, коли фільтр прибирає 20 відсотків від загальної кількості даних (для першого сценарію) та 50 відсотків (для другого сценарію). Графіки демонструють, що ефективність у другому випадку більша, а час на виконання запиту зменшився на 22 відсотки. Теоретично, час може зменшитися до 50 відсотків, оскільки лише половина рядків буле вичитана, але на практиці частина часу йде на ініціалізацію, планування та інші накладні витрати.

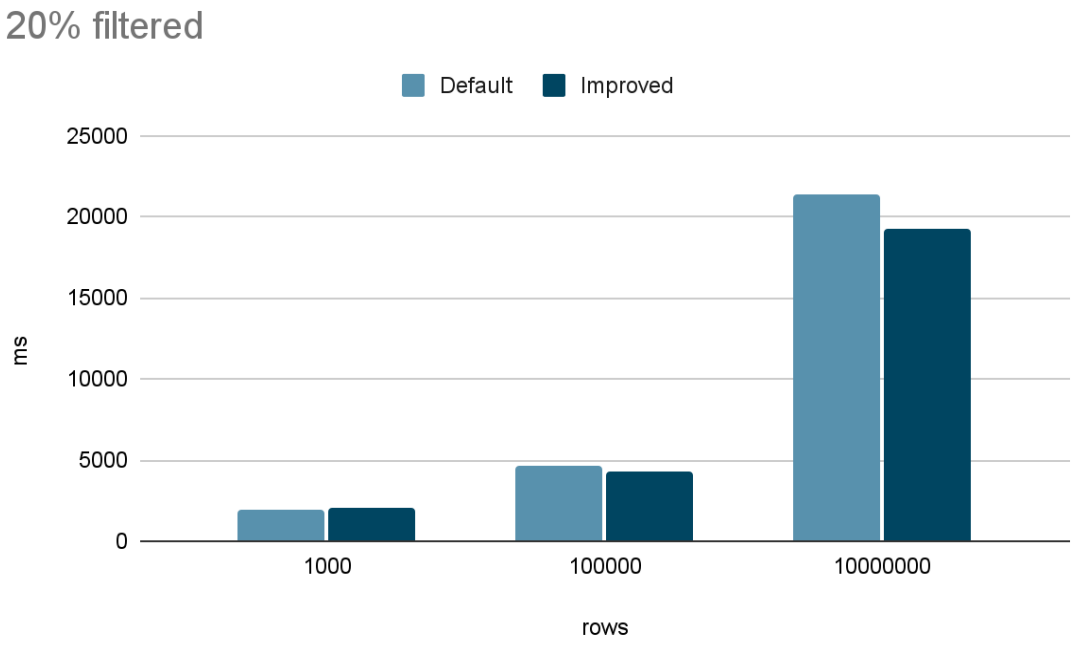


Рис.3.1. Швидкість виконання запитів (відфільтровано 20%)

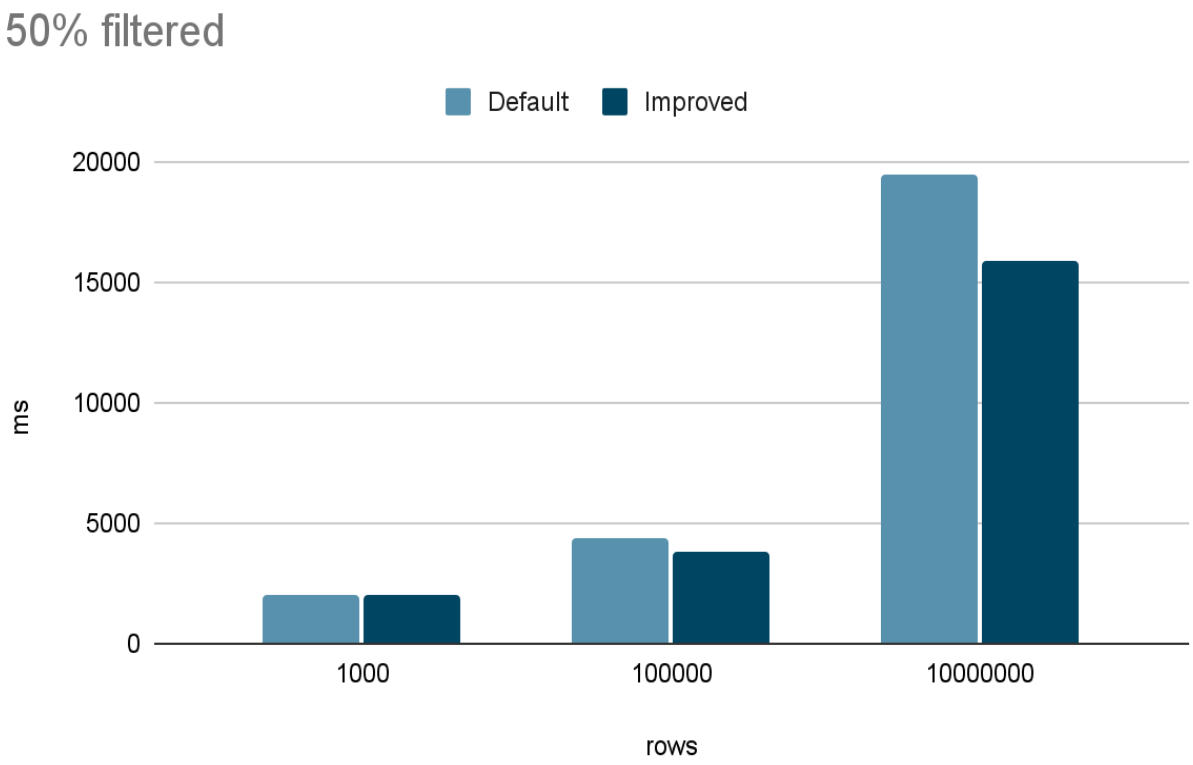


Рис. 3.2. Швидкість виконання запитів (відфільтровано 50%)

За результатами проведеного аналізу можна зазначити, що алгоритм показав високу ефективність у реальних сценаріях використання, що робить його цінним доповненням до інструментарію обробки великих даних.

Було виявлено значні переваги удосконаленого алгоритму у порівнянні з існуючими рішеннями, що підтверджує його потенціал для покращення загальної продуктивності систем, що працюють з великими обсягами даних.

3.5 Оцінка ресурсної ефективності алгоритму

У рамках оцінки ресурсної ефективності розробленого удосконаленого алгоритму фільтра Pushdown у Apache Spark, було здійснено детальний аналіз споживання ключових системних ресурсів, таких як пам'ять і час роботи CPU. Цей аналіз має важливе значення для визначення вартісної ефективності алгоритму, особливо при роботі з великими обсягами даних.

Для цього було виміряно обсяг пам'яті, який використовується під час виконання запитів з використанням розробленого алгоритму порівняно зі стандартними методами обробки в Apache Spark. Метою цього тестування було визначити, чи призводить виконане удосконалення фільтра до зниження споживання пам'яті за рахунок ефективнішого відсіву нерелевантних даних на ранньому етапі.

Також для оцінки використання CPU було здійснено аналіз навантаження на процесор при використанні алгоритму. Основними показниками були час CPU, витрачений на обробку запитів, та його порівняння з використанням CPU при стандартних методах. Це дозволило оцінити, чи підвищує розроблений алгоритм ефективність використання обчислювальних ресурсів.

На рисунках 3.3, 3.4 показано порівняння використаної пам’яті при роботи з великими обсягами даних. На великому Data set є незначне покращення, яке доходить до 7 %. Можна побачити, що відсутня велика різниця у використаній пам’яті, є незначне покращення. Це пояснюється тим, що Apache Spark має вбудовані оптимізації і не вивантажує весь Data set у пам’ять одночасно.

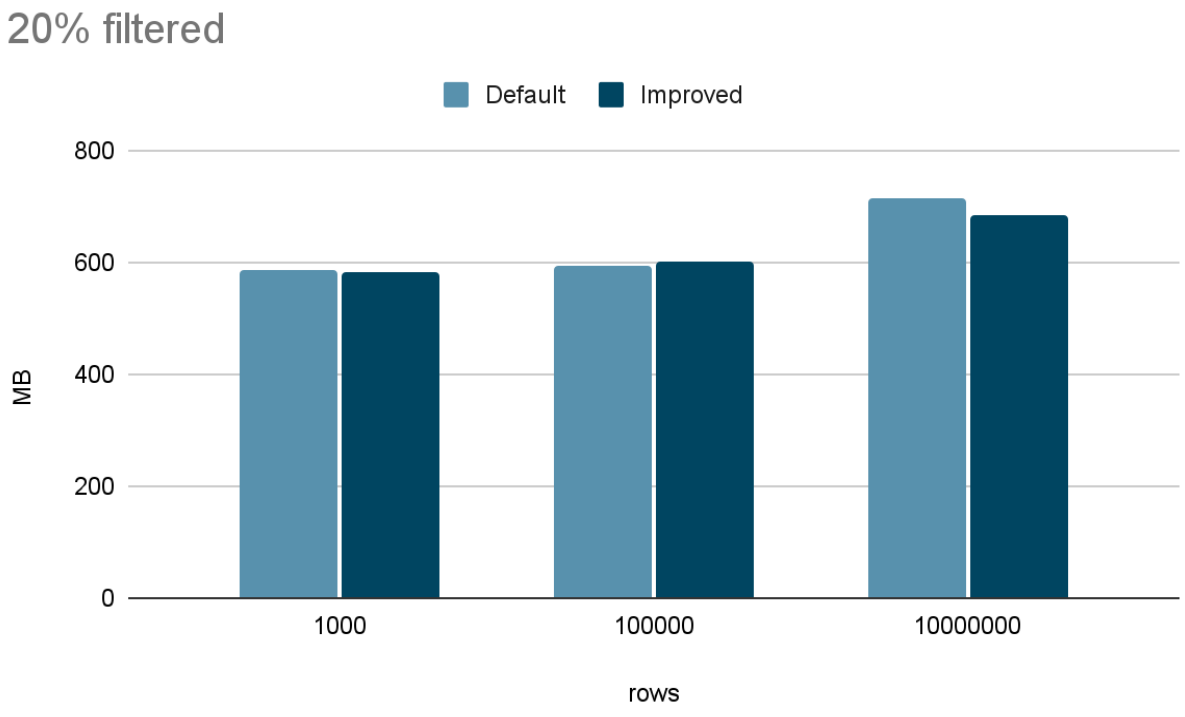


Рис. 3.3. Використання пам’яті (відфільтровано 20%)

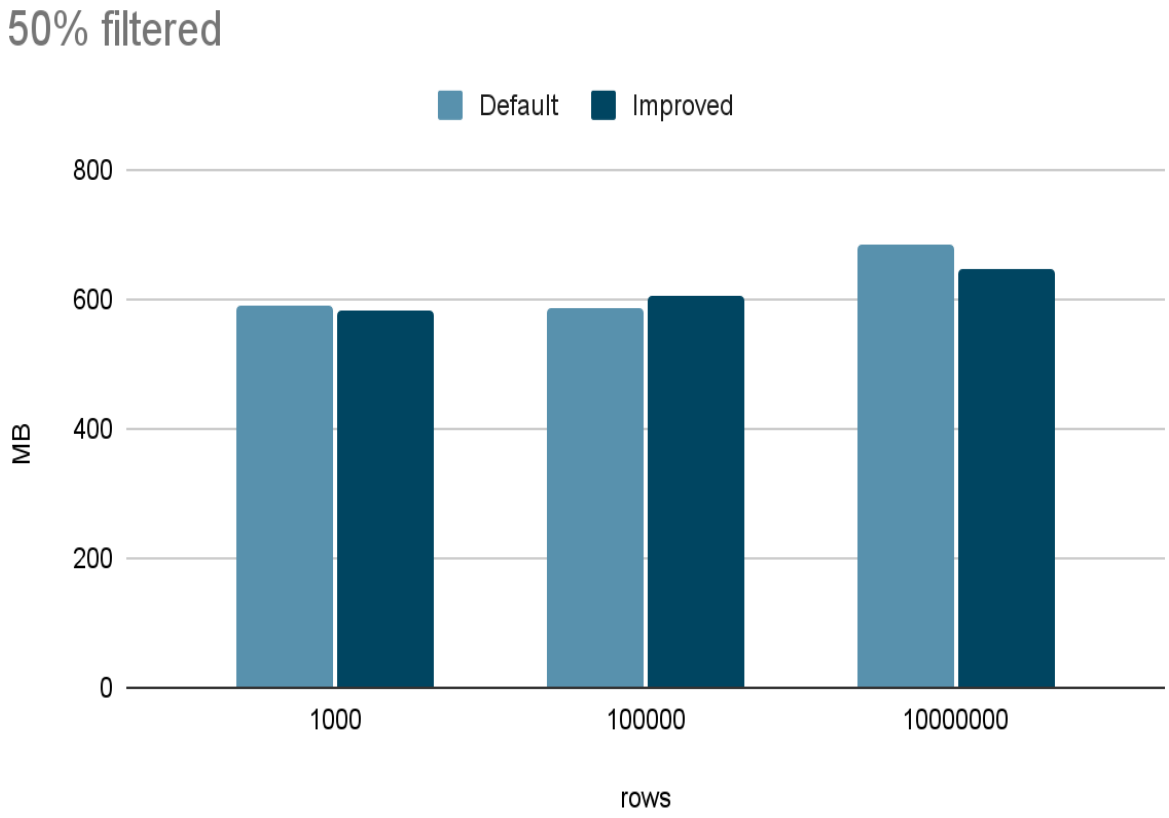


Рис. 3.4. Використання пам’яті (відфільтровано 50%)

З аналізу споживання ресурсів можна зробити наступні висновки:

1. Зниження споживання пам'яті. Результати показали, що удосконалений алгоритм зазвичай використовує менше пам'яті, порівняно з традиційними методами, що пояснюється більш ефективною фільтрацією даних на ранніх етапах обробки.
2. Підвищення ефективності використання CPU. Алгоритм продемонстрував зниження часу роботи CPU, необхідного для обробки запитів, що вказує на підвищену ефективність у використанні обчислювальних ресурсів.
3. Загальне покращення ресурсної ефективності. Отримані дані свідчать про те, що розроблений алгоритм забезпечує загальне покращення ресурсної ефективності при обробці великих даних, що робить його більш ефективним рішенням.

Отримані при тестуванні результати підкреслюють значення розробленого алгоритму не тільки з точки зору підвищення продуктивності, але й у плані більш

ефективного використання системних ресурсів, що є критично важливим для роботи з великими обсягами даних.

3.6 Усунення виявлених проблем

Під час тестування розробленого алгоритму фільтра Pushdown в Apache Spark була виявлена ключова проблема, яка полягала в тому, що збільшується час роботи планера правил, що відповідає за оптимізацію запитів. Ця проблема особливо стала помітною при обробці запитів, які не містили фільтрів, що підтримуються алгоритмом. Для усунення цієї проблеми було вирішено впровадити наступні зміни:

1. Модифікувати алгоритм таким чином, щоб користувацьке правило активувалося лише тоді, коли в запиті виявлено фільтри, які підтримуються алгоритмом. Це дозволило зменшити непотрібні обчислення та аналізи, що виконувалися планером правил, коли оптимізація не була необхідною.
2. Удосконалити механізм виявлення фільтрів, які можуть бути виконані на стороні файлового сховища, з метою забезпечення швидшого та більш точного визначення сценаріїв, в яких повинен активуватися алгоритм.

Після внесення модифікацій було проведено додаткове тестування для перевірки, що внесені зміни не впливають негативно на загальну функціональність алгоритму та зменшують час роботи планера правил.

Після впровадження зазначених змін було відзначено значне поліпшення у часі роботи планера правил, особливо у випадках, коли запити не містили підтримуваних фільтрів. Це покращення допомогло уникнути зайвого навантаження на систему та забезпечило більш ефективну роботу алгоритму в різних сценаріях запитів. Таким чином, вирішення виявленої проблеми збільшення часу роботи планера правил сприяло оптимізації загальної продуктивності системи та підвищило ефективність розробленого алгоритму.

3.7 Рекомендації щодо реалізації алгоритму

На основі отриманих результатів та досвіду, набутого під час розробки та тестування удосконаленого алгоритму фільтра Pushdown у Apache Spark, можна надати наступні рекомендації для його впровадження:

1. Перед впровадженням алгоритму необхідно проаналізувати специфічні потреби та характеристики системи, з якою він буде інтегрований. Це дозволить визначити, чи є цей алгоритм підходящим рішенням для конкретного випадку.
2. Важливо оцінити та переконатися, що алгоритм сумісний з існуючою архітектурою та інфраструктурою, особливо якщо використовуються специфічні версії Apache Spark або існують конкретні налаштування системи.
3. Впроваджувати алгоритм рекомендується поступово, починаючи з тестового середовища, та проводити ретельне тестування перед його застосуванням у продуктивній системі. Це дозволить виявити та усунути потенційні проблеми, не впливаючи на стабільність загальної системи.
4. Після впровадження алгоритму важливо здійснювати постійний моніторинг його роботи, аналізувати ефективність та при необхідності вносити корективи для подальшого удосконалення.
5. При плануванні впровадження алгоритму важливо враховувати можливі майбутні оновлення та зміни в Apache Spark, щоб забезпечити його сумісність та актуальність на тривалий час.

Враховуючи розроблені рекомендації, можна ефективно інтегрувати та використовувати розроблений алгоритм, щоб підвищити продуктивність та ефективність обробки даних у системах, які використовують Apache Spark для роботи з великими обсягами даних.

Висновки до 3 розділу

У третьому розділі проаналізовано ефективність алгоритму вдосконалення фільтра Pushdown у Apache Spark, проведено його тестування, порівняння швидкісних характеристик, наведено рекомендації до його реалізації.

Проведені тестування та аналіз показали, що алгоритм підвищує ефективність обробки даних, зокрема при роботі з великими наборами даних. Це демонструє його важливість та практичність у сучасних сценаріях обробки великих даних.

Доведено, що алгоритм сприяє більш ефективному використанню системних ресурсів, таких як пам'ять і час роботи CPU, що робить його ефективним рішенням.

Після тестування для усунення виявлених проблем, таких як збільшення часу роботи планера правил, було проведено модифікацію, що дозволить уникнути зайвого навантаження на систему та забезпечило більш ефективну роботу алгоритму в різних сценаріях запитів.

Було розроблено рекомендації, які охоплюють аспекти впровадження, тестування, моніторингу та удосконалення алгоритму в реальних умовах, забезпечуючи його ефективне використання.

Результати аналізу підкреслюють потенціал алгоритму для подальшого розвитку та адаптації до змінних умов та вимог сучасних систем обробки даних.

Враховуючи проведені тестування, удосконалений алгоритм фільтра Pushdown може бути рекомендований як ефективне рішення для покращення продуктивності обробки великих даних в Apache Spark. Його впровадження та правильне використання дозволить значно підвищити ефективність роботи систем, що займаються аналітикою та обробкою великих обсягів даних.

ВИСНОВКИ

У ході виконання дослідницької роботи було проведено аналіз предметної області, проведено дослідження сучасного стану останніх існуючих рішень, у яких піднімається питання розв’язання проблеми прискорення та збільшення ефективності обробки великих обсягів даних.

Для цього було визначено основні характеристики бізнес-процесів, пов'язаних з аналізом великої кількості даних, та обґрунтовано необхідність збільшення їх швидкодії за рахунок вдосконалення при роботі з великими обсягами даних. Це сприятиме вирішенню стратегічних завдань, прийняттю обґрунтованих рішень, покращенню бізнесу та створенню конкурентних переваг.

Були проаналізовані існуючі проблеми при роботі з великими обсягами даних та сформовано основні завдання для розробки вдосконаленого алгоритму фільтрації даних на стороні файлового сховища.

Для втілення вдосконалень було виконано ретельне проєктування та розробка вдосконаленого алгоритму з фільтром Pushdown для Parquet файлів у контексті Apache Spark.

Було проведено тестування для оцінки ефективності розробленого засобу, які показують, що за наявності підтримуваного фільтру в запиті обробка Parquet файлів з удосконаленим алгоритмом фільтру Pushdown зменшує час виконання запиту та обсяг використаної оперативної пам’яті.

Практично було доведено, що кількість відфільтрованих даних прямо пропорційна загальній ефективності зчитування Parquet файлів. При цьому застосування удосконаленої фільтрації має сенс на великих обсягах даних, коли час на виконання займає більшу частину самого запиту.

Після проведеної модифікації було розроблено рекомендації, які охоплюють аспекти впровадження, тестування, моніторингу та поліпшення алгоритму в реальних умовах, забезпечуючи його ефективне використання.

Враховуючи результати тестування у поточній реалізації алгоритму, рекомендовані його подальші дослідження з метою розширення його універсальності та ефективності.

Завдяки виконаному вдосконаленню алгоритм може бути використаний для подальшого розвитку та адаптації до змінних умов та вимог сучасних систем обробки даних як ефективне рішення для покращення продуктивності обробки Big Data в Apache Spark.

Впровадження та правильне використання вдосконаленого алгоритму дозволить значно підвищити ефективність роботи систем, що займаються аналітикою та обробкою великих обсягів даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Латишева О.В, Карлаш Ю.Д. Сутність та особливості впровадження моделей бізнес-процесів у системі управління на підприємствах в Україні./ Латишева О.В, Карлаш Ю.Д. // Економіка та управління підприємствами. – 2019. – Випуск 29. – С. 211-218. – [Електронний ресурс]. – Режим доступу: http://www.market-infr.od.ua/journals/2019/29_2019_ukr/33.pdf
2. Мельник Л. Г., Таранюк Л. М. Сутність і природа бізнес-процесу при реалізації реінжинірингу на промислових підприємствах. / Мельник Л. Г., Таранюк Л. М. // Бізнесінформ. –2012. – № 10. – С.88-91. – [Електронний ресурс]. – Режим доступу: http://www.business-inform.net/pdf/2012/10_0/88_92.pdf
3. Палагін О.В. Функціонально-орієнтований підхід у дослідницькому проектуванні. // Кібернетика та системний аналіз. – 2017. – №6. – С. 185–192.
4. Ali W.; Shafique M.U.; Majeed M.A.; Raza A. Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics. Asian J. Res. Comput. Sci. 2019, 4, 1–10. –[Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=Comparison+between+SQL+and+NoSQL+Databases+and+Their+Relationship+with+Big+Data+Analytics&author=Ali,+W.&author=Shafique,+M.U.&author=Majeed,+M.A.&author=Raza,+A.&publication_year=2019&journal=Asian+J.+Res.+Comput.+Sci.&volume=4&pages=1%E2%80%9310&doi=10.9734/ajrcos/2019/v4i230108.
5. Bicevska, Z.; Oditis, I. Towards NoSQL-based Data Warehouse Solutions. Procedia Comput. Sci. 2017, 104, P. 104–111. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=Towards+NoSQL-based+Data+Warehouse+Solutions&author=Bicevska,+Z.&author=Oditis,+I.&publication_year=2017&journal=Procedia+Comput.+Sci.&volume=104&pages=104%E2%80%93111&doi=10.1016/j.procs.2017.01.080.

   6. Hamoud, A.K.; Ulkareem, M.A.; Hussain, H.N.; Mohammed, Z.A.; Salih, G.M. Improve HR Decision-Making Based On Data Mart and OLAP. J. Phys. Conf. Ser. 2020, 1530, 012058. – [Електронний ресурс]. – Режим доступу:

https://scholar.google.com/scholar_lookup?title=Improve+HR+Decision-Making+Based+On+Data+Mart+and+OLAP&author=Hamoud,+A.K.&author=Ulkareem,+M.A.&author=Hussain,+H.N.&author=Mohammed,+Z.A.&author=Salih,+G.M.&publication_year=2020&journal=J.+Phys.+Conf.+Ser.&volume=1530&pages=012058&doi=10.1088/1742-6596/1530/1/012058.

7. Wang X.; Xie Z. The Case for Alternative Web Archival Formats to Expedite the Data-To-Insight Cycle. In Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, Virtual Event, China, 1–5 August 2020; pp. 177–186. – – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=The+Case+for+Alternative+Web+Archival+Formats+to+Expedite+the+Data-To-Insight+Cycle&conference=Proceedings+of+the+ACM/IEEE+Joint+Conference+on+Digital+Libraries&author=Wang,+X.&author=Xie,+Z.&publication_year=2020&pages=177%E2%80%93186
8. Ahmed, S.; Ali, M.U.; Ferzund, J.; Sarwar, M.A.; Rehman, A.; Mehmood, A. Modern Data Formats for Big Bioinformatics Data Analytics. Int. J. Adv. Comput. Sci. Appl. 2017, 8. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=Modern+Data+Formats+for+Big+Bioinformatics+Data+Analytics&author=Ahmed,+S.&author=Ali,+M.U.&author=Ferzund,+J.&author=Sarwar,+M.A.&author=Rehman,+A.&author=Mehmood,+A.&publication_year=2017&journal=Int.+J.+Adv.+Comput.+Sci.+Appl.&volume=8&doi=10.14569/IJACSA.2017.080450.
9. Ramírez, A.; Parejo, J.A.; Romero, J.R.; Segura, S.; Ruiz-Cortés, A. Evolutionary composition of QoS-aware web services: A many-objective perspective. Expert Syst. Appl. 2017, 72, 357–370. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=Evolutionary+composition+of+QoS-aware+web+services:+A+many-objective+perspective&author=Ram%C3%ADrez,+A.&author=Parejo,+J.A.&author=Romero,+J.R.&author=Segura,+S.&author=Ruiz-

Cort%C3%A9s,+A.&publication_year=2017&journal=Expert+Syst.+Appl.&volume=72&pages=357%E2%80%93370&doi=10.1016/j.eswa.2016.10.047 .

10. Plase, D.; Niedrite, L.; Taranovs, R. A Comparison of HDFS Compact Data Formats: Avro Versus Parquet. Moksl. Liet. Ateitis 2017, 9, 267–276. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=A+Comparison+of+HDFS+Compact+Data+Formats:+Avro+Versus+Parquet&author=Plase,+D.&author=Niedrite,+L.&author=Taranovs,+R.&publication_year=2017&journal=Moksl.+Liet.+Ateitis&volume=9&pages=267%E2%80%93276&doi=10.3846/mla.2017.1033.
11. Raevich, A.; Dobronets, B.; Popova, O.; Raevich, K. Conceptual model of operational-analytical data marts for big data processing. E3S Web Conf 2020, 149. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=Conceptual+model+of+operational-analytical+data+marts+for+big+data+processing&author=Raevich,+A.&author=Dobronets,+B.&author=Popova,+O.&author=Raevich,+K.&publication_year=2020&journal=E3S+Web+Conf&volume=149&doi=10.1051/e3sconf/202014902011.
12. Aditi Prakash.A Deep Dive into Parquet: The Data Format Engineers Need to Know. – June 21, 2023. – [Електронний ресурс]. – Режим доступу: https://airbyte.com/data-engineering-resources/parquet-data-format.
13. Abadi D.J., Myers D.S., DeWitt D.J., Madden S.R. Materialization strategies in a column-oriented DBMS. In Proceedings of the International Conference on Data Endineering (ICDE), P. 466–475, 2007.
14. Zukowski M., Heman S., Nes N., Boncz P. Super-Scalar RAM-CPU Cache Compression. // In Proceedings of the 22nd International Conference on Data Endineering (ICDE), 2006. – P. 59-71.
15. Ab adi D.J., Boncz P., Harizopoulos S., Idreos S., Madden S. (2013), “The Design and Implementation of Modern Column-Oriented Database Systems”, Foundations and Trends® in Databases: Vol. 5: No. 3, – P. 197-280.

16. Yifei Yang, Matt Youill, Matthew Woicik, Yizhou Liu, Xiangyao Yu, Marco Serafni, Ashraf Aboulnaga, Michael Stonebraker. FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. // «Proceedings of the VLDB Endowment». №11 (14), 2021. – Pp. 2101-2112. – [Електронний ресурс]. – Режим доступу: https://pages.cs.wisc.edu/~yxy/pubs/fpdb.pdf.
17. Рибачок Д.О. Комп'ютерно-математичні моделі мережевих структур і галузей та їх застосування на реальних великих даних // 318 Збірник матеріалів X Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2022». – Київ, 15 листопада 2022 р.). К.: КОМПРИНТ, 2022. – 294 с. – С. 73–78.
18. Ashok Shivayogappa, Supreeth Shivashankar. A Comparison of HDFS File Formats: Avro, Parquet and ORC // «International Journal of Advanced Science and Technology», № 29(4). June 2020. pp:4665-4675. : DOI:10.5281/zenodo.7027910. – pp. 4665-4675.
19. Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, Samir Belfkih. Big Data technologies: A survey. // Journal of King Saud University – Computer and Information Sciences. – Volume 30, Issue 4, October 2018, P. 431-448. – [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S1319157817300034?via%3Dihub.
20. Khan, N., Yaqoob, I., Hashem, I.A.T., Inayat, Z., Mahmoud Ali, W.K., Alam, M., Shiraz, M., Gani, A., Big data: survey, technologies, opportunities, and challenges. Sci. World. 2014. – [Електронний ресурс]. – Режим доступу: https://www.hindawi.com/journals/tswj/2014/712826/ .
21. Rasheed, Y.; Qutqut, M.H.; Almasalha, F. Overview of the Current Status of NoSQL Database. Int. J. Comput. Sci. Netw. Secur. 2019, 19, 47–53. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=Overview+of+the+Current+Status+of+NoSQL+Database&author=Rasheed,+Y.&author=Qutqut,+M.H.&author=Alm

asalha,+F.&publication_year=2019&journal=Int.+J.+Comput.+Sci.+Netw.+Secur.&volume=19&pages=47%E2%80%9353r].

22. Wang, X.; Xie, Z. The Case for Alternative Web Archival Formats to Expedite the Data-To-Insight Cycle. In Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, Virtual Event, China, 1–5 August 2020; pp. 177–186. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=The+Case+for+Alternative+Web+Archival+Formats+to+Expedite+the+Data-To-Insight+Cycle&conference=Proceedings+of+the+ACM/IEEE+Joint+Conference+on+Digital+Libraries&author=Wang,+X.&author=Xie,+Z.&publication_year=2020&pages=177%E2%80%93186 .
23. Plase, D.; Niedrite, L.; Taranovs, R. A Comparison of HDFS Compact Data Formats: Avro Versus Parquet. Moksl. Liet. Ateitis 2017, 9, 267–276. – [Електронний ресурс]. – Режим доступу: https://scholar.google.com/scholar_lookup?title=A+Comparison+of+HDFS+Compact+Data+Formats:+Avro+Versus+Parquet&author=Plase,+D.&author=Niedrite,+L.&author=Taranovs,+R.&publication_year=2017&journal=Moksl.+Liet.+Ateitis&volume=9&pages=267%E2%80%93276&doi=10.3846/mla.2017.1033.
24. An Introduction to Big Data Concepts [Електронний ресурс] // Sagara Technology Idea Lab, 2020. – Режим доступу: https://sagaratechnology.medium.com/ig-an-introduction-to-big-data-concepts-c4cac70a7f03 .
25. S. J. Samuel, K. Rvp, K. Sashidhar, and C. R. Bharathi, “a Survey on Big Data and Its Research Challenges,” ARPN Journal of Engineering and Applied Sciences, vol. 10, no. 8, 2015.
26. Томас Єрл, Ваджид Хаттак, Пол Булер Основи Big Data: Концепції, алгоритми та технології /Пер.з англ. Анатолія Гладуна; За наук. ред. Олексія Найди. Дніпро: «Баланс Бізнес Букс», 2018. 320 с.
27. Apache Parquet. Apache Parquet. URL: https://parquet.apache.org/ (дата звернення: 11.11.2023).

28. Braams B. Predicate Pushdown in Parquet and Apache Spark. / Computer Science. – 2018. – December, – [Електронний ресурс]. – Режим доступу: https://homepages.cwi.nl/~boncz/msc/2018-BoudewijnBraams.pdf .
29. Резніченко В.А. 60 років базам даних (четверта частина). / Резніченко В.А. // Проблеми програмування. – 2022. – № 2. – С. 57-95. – [Електронний ресурс]. – Режим доступу: http://doi.org/10.15407/pp2022.02.057 .
30. Xin Lian and Tianyu Zhang. The Optimization of Cost-Model for Join Operator on Spark SQL Platform // International Conference on Smart Materials, Intelligent Manufacturing and Automation. – 2018. – 173, 01015. – [Електронний ресурс]. – Режим доступу: https://www.matec-conferences.org/articles/matecconf/pdf/2018/32/matecconf_smima2018_01015.pdf
31. Jason Lam. Applying Sampling and Predicate Pushdown in an Interactive Data Exploration System. / Jason Lam .// Department of Electrical Engineering and Computer Science. – May 12, 2020. – [Електронний ресурс]. – Режим доступу: https://doi.org/10.1051/matecconf/201817301015.

ДОДАТОК А

ГЕНЕРАЦІЯ ФІЗИЧНИХ ПЛАНІВ

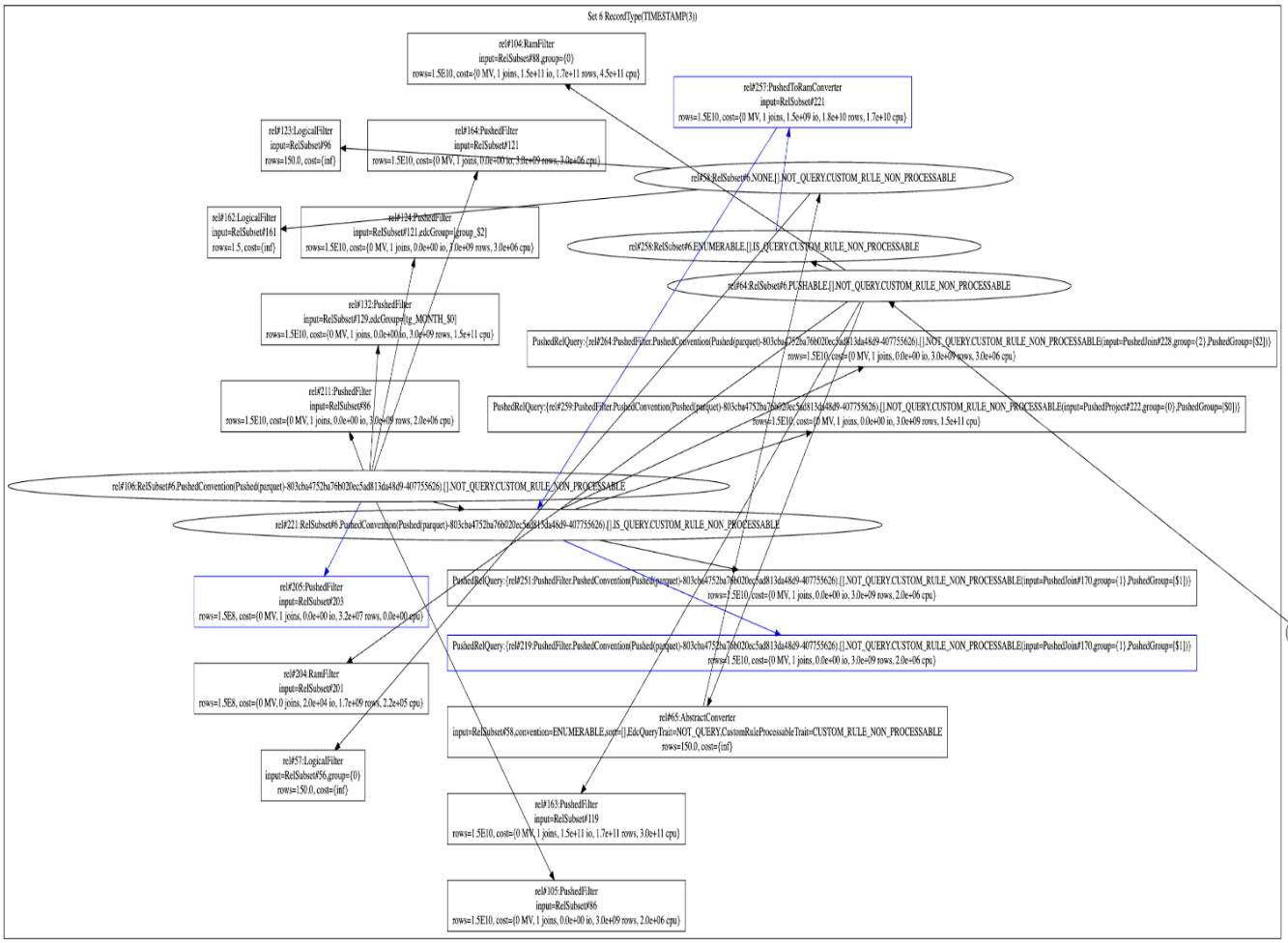


Рис. А.1. Множина планів

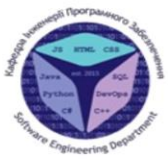
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Магістерська робота

«МОДЕЛЮВАННЯ ТА РОЗРОБКА АЛГОРИТМУ ЗЧИТУВАННЯ PARQUET-ФАЙЛІВ ДЛЯ ПІДТРИМКИ БІЗНЕС-ПРОЦЕСІВ»

Виконав: студент групи ПДМ-62 Лященко Олексій Володимирович

Керівник: к.п.н., доц., доцент кафедри ІПЗ Шевченко С. М.

Київ - 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: удосконалення бізнес-процесів внаслідок підвищення ефективності алгоритмів, що зчитують Parquet-файли.

Об'єкт дослідження: процес зчитування та обробки Parquet-файлів.

Предмет дослідження: структура алгоритму зчитування Parquet-файлів із підтримкою часткового фільтру Pushdown.

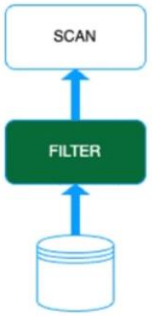
КЛЮЧОВІ ЕТАПИ ВИКОНАННЯ ЗАПИТУ В СИСТЕМАХ ОБРОБКИ ВЕЛИКИХ НАБОРІВ ДАНИХ

3

PUSHDOWN УДОСКОНАЛЕННЯ

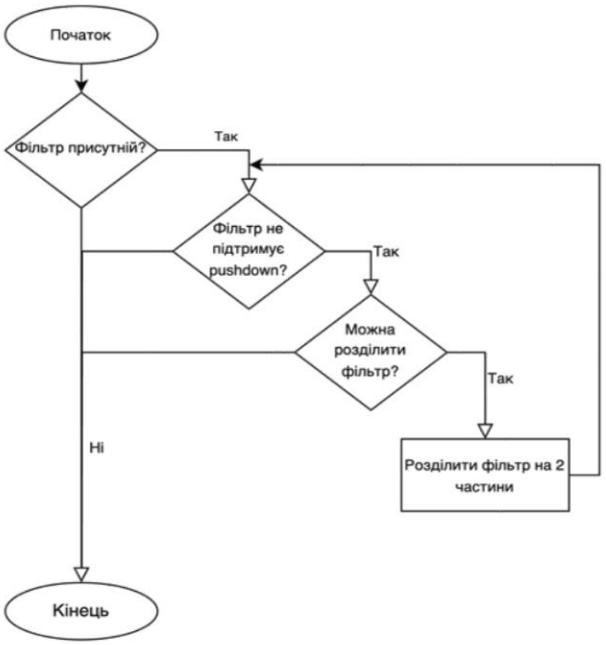
Pushdown удосконалення — це техніка в області обробки даних і баз даних, яка полягає в перенесенні певних операцій обробки даних на нижчий рівень, зазвичай ближче до рівня зберігання даних.
Фільтр Pushdown є ключовим принципом в прискоренні запитів зчитування даних, особливо у контексті великих даних та розподілених обчислень.
Це удосконалення використовується для покращення продуктивності запитів, зменшення часу обробки та використання ресурсів.

- Зменшення обсягу даних
- Підвищення швидкості запитів
- Ефективне використання ресурсів
- Зменшення навантаження на мережу



4

АЛГОРИТМ РОЗДІЛЕННЯ ФІЛЬТРІВ



Алгоритм, розроблений для вдосконалення фільтру Pushdown, базується на додаванні користувацького правила.

Основною метою цього правила є переписування складних фільтрів на декілька простих.

5

ЗМІНЕНИЙ ЛОГІЧНИЙ ПЛАН

```
SELECT product FROM customers
WHERE regional_sales BETWEEN SQRT(sales) and 100_000
```

```
LogicalProject(expressions=[product]])
  LogicalFilter(condition=[regional_sales  between(sqrt(sales), central_sales)])
    LogicalScan(file=[customers, parquet])
```

```
LogicalProject(expressions=[product]])
  LogicalFilter(condition=[>=(regional_sales, sqrt(sales))])
    LogicalFilter(condition=[<=(regional_sales, central_sales)])
      LogicalScan(file=[customers, parquet])
```

6

ОТРИМАНИЙ ФІЗИЧНИЙ ПЛАН

```
RamProject(expressions=[product])
  RamFilter(condition=[between(sqrt(sales), central_sales)])
    PushedProject(expressions=[product, sales, regional_sales, central_sales])
      PushedScan(file=[customers, parquet])

RamProject(expressions=[product])
  RamFilter(condition=[>=(regional_sales, sqrt(sales))])
    PushedProject(expressions=[product, sales, regional_sales])
      PushedFilter(condition=[<=(regional_sales, central_sales)])
        PushedScan(file=[customers, parquet])
```


МОДЕЛЮВАННЯ ЧАСТКОВОГО ФІЛЬТРУ PUSHDOWN ТА ОЦІНКИ ЙОГО ЕФЕКТИВНОСТІ

1. Розрахунок загального часу обробки без удосконалення T_{total}

$$T_{total} = T_{read} \times N + T_{process} \times N$$

2. Розрахунок загального часу обробки з удосконаленням $T_{total\_pushdown}$

$$T_{total\_pushdown} = T_{read} \times (N - M) + T_{process} \times (N - M) + T_{process\_pushdown} \times M,$$

$$\text{де } M \leq N,\ T_{process\_pushdown} < T_{read} + T_{process}.$$

3. Розрахунок оцінки ефективності (*Efficiency*)

$$Efficiency = (T_{total} - T_{total\ pushdown}) / T_{total}$$

де - N – кількість записів у Parquet файлі,
- M – кількість записів, що відповідають критеріям фільтра,
- T_{total} – загальний час обробки без удосконалення,
- $T_{total\_pushdown}$ – загальний час обробки з удосконаленням,
- T_{read} – час, необхідний для зчитування одного запису,
- $T_{process}$ – час обробки одного запису,
- $T_{process\_pushdown}$ – час обробки одного запису на стороні файлового сховища.

ШВИДКІСНЕ ПОРІВНЯННЯ ВИКОНАННЯ ЗАПИТУ

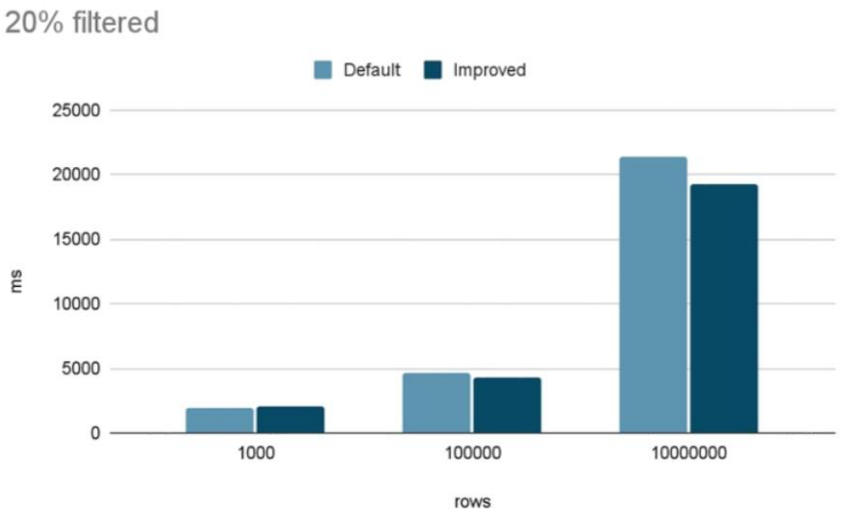


Рисунок 1. Швидкість виконання запитів (відфільтровано 20%).

На Рис. 1, 2 показані відмінності у швидкості виконання запитів. Було проведено порівняння по двом сценаріям, коли фільтр прибирає 20 відсотків від загальної кількості даних (для першого сценарію) та 50 відсотків (для другого сценарію).

9

ШВИДКІСНЕ ПОРІВНЯННЯ ВИКОНАННЯ ЗАПИТУ

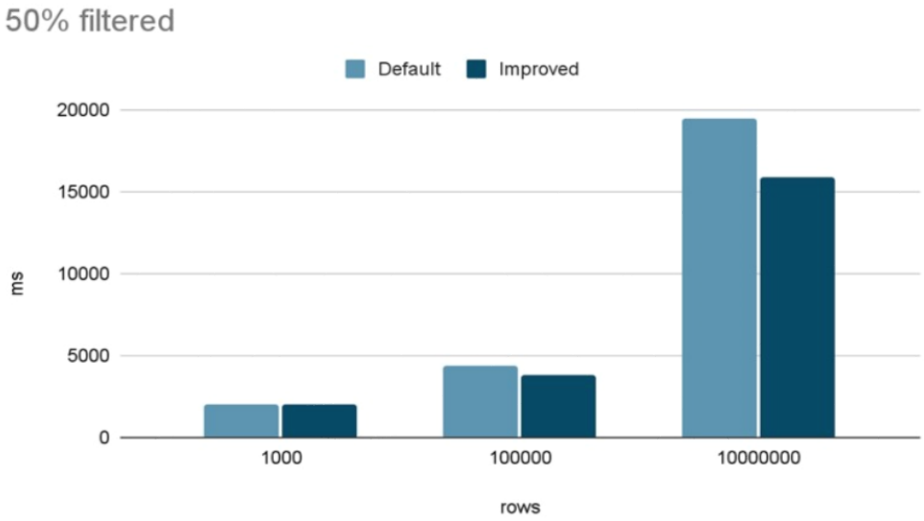


Рисунок 2. Швидкість виконання запитів (відфільтровано 50%)

Графіки демонструють, що ефективність у другому випадку більша, а час на виконання запиту зменшився.

10

ПОРІВНЯННЯ ВИКОРИСТАНОЇ ПАМ'ЯТІ

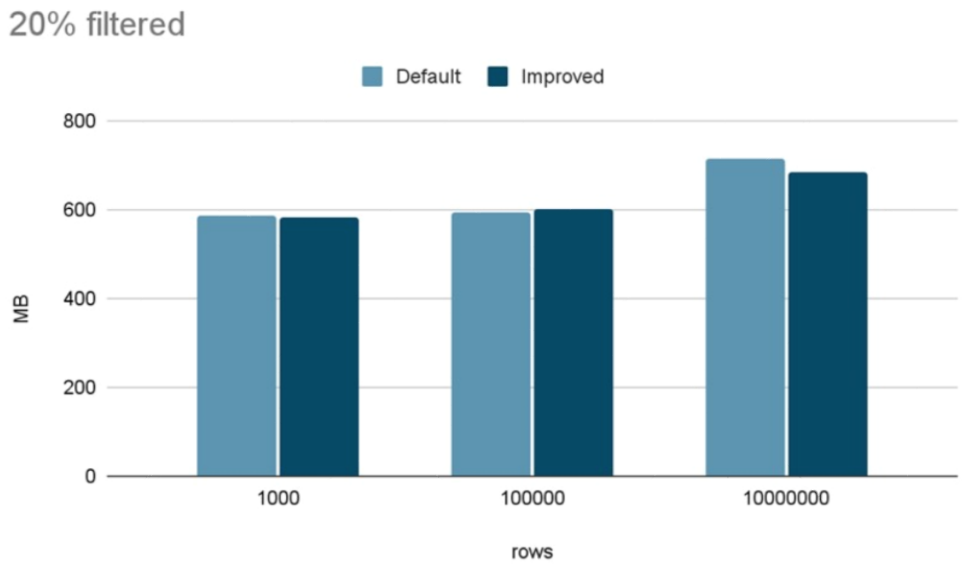


Рисунок 3. Швидкість виконання запитів (відфільтровано 20%).

11

ПОРІВНЯННЯ ВИКОРИСТАНОЇ ПАМ'ЯТІ

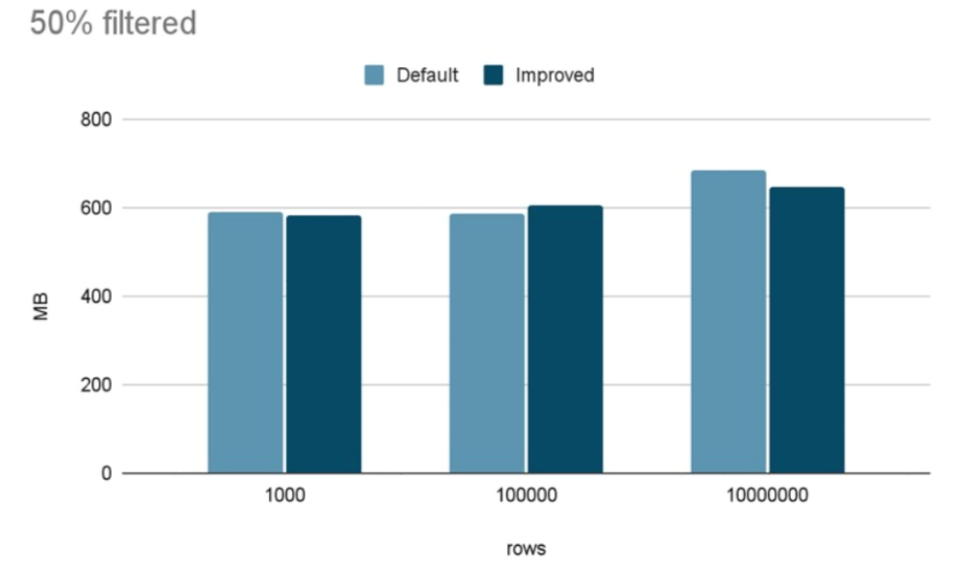


Рисунок 4. Використання пам'яті (відфільтровано 50%).

На Рис. 3, 4 можна побачити, що відсутня велика різниця у використаній пам'яті, є незначне покращення. Це пояснюється тим, що Apache Spark має вбудовані оптимізації і не вивантажує весь датасет у пам'ять одночасно.

12

ВИСНОВКИ

1. Проаналізовані існуючі рішення та алгоритми при роботі з великими обсягами даних, визначено актуальність та важливість проблеми для бізнес-процесів.
2. З'ясовано особливості Parquet-файлів та алгоритми їх зчитування.
3. Розроблено та теоретично обґрунтовано алгоритм зчитування Parquet-файлів із вбудованою підтримкою часткового фільтру Pushdown.
4. Досліджено ефективність модифікованого алгоритму та представлені рекомендації для його реалізації.

13

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Лященко О.В. Створення алгоритму ефективного кешування даних для збільшення продуктивності читання Parquet-файлів// XV Міжнародна науково-практична конференція «Distance education as the main problem of young people», 26-29 грудня 2023 р., Мадрид, Іспанія.

2. Лященко О.В. Оптимізація використання обчислювальних ресурсів в бізнес-процесах за рахунок прискорення процесу зчитування та обробки Parquet-файлів // І Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ: ДУТ, 2023.

14

ДЯКУЮ ЗА УВАГУ!