Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи обробки даних в розподілених системах

(тема)

Виконав:

студент II курсу, групи КСМзм-22-1

Курченко С.І
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: доц. Козлов Ю.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Курченку Сергію Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи обробки даних в розподілених системах

затверджена наказом по університету від " 03 " листопада 2023 р. № 244 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024 р.

3. Вхідні дані до роботи
розподілена система
паралельність
MapReduce
Spark

4. Перелік питань, що потрібно опрацювати у роботі
Методи доступу до сховищ даних
Метод доступу до СД з використанням фільтрів
Експериментальні дослідження

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 18 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання та аналіз джерел за темою	07.11.2023–14.11.2023	
	роботи		
2	Аналіз існуючих методів	15.11.2023–26.11.2023	
3	Розробка нового або модифікація існуючого	27.11.2023–11.12.2023	
	методу		
4	Програмна реалізація розробленого або	12.12.2023–24.12.2023	
	модифікованого методу		
5	Аналіз отриманих результатів	25.12.2023–05.01.2024	
6	Оформлення пояснювальної записки та		
	документів до неї		

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Козлов Ю.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 60 с., 24 рис., 1 табл., 2 дод., 21 джерело.

MAPREDUCE, SPARK, ОБРОБКА ДАНИХ, РОЗПОДІЛЕНА СИСТЕМА, СХОВИЩЕ ДАНИХ.

Метою кваліфікаційної роботи є аналіз методів обробки та зберігання даних в розподілених системах.

У ході виконання кваліфікаційної роботи проведено порівняльний аналіз методів обробки та зберігання даних в розподілених системах. Також були розглянуті методи доступу до даних в паралельних розподілених сховищах даних на платформі MapReduce/Spark. Розроблен модифікований метод обробки даних та виконання запитів до паралельного розподіленого сховища даних на базі технології MapReduce/Spark з використанням фільтру Блума.

ABSTRACT

Master’s thesis: 60 pages, 24 figures, 1 tables, 2 appendices, 21 sources.

MAPREDUCE, SPARK, DATA PROCESSING, DISTRIBUTED SYSTEM, DATA STORAGE.

The major goal of this thesis is the analysis of data processing and storage methods in distributed systems.

In the course of the qualification work, a comparative analysis of data processing and storage methods in distributed systems was carried out. Methods of accessing data in parallel distributed data stores on the MapReduce/Spark platform were also considered. A modified method of data processing and execution of requests to a parallel distributed data store based on MapReduce/Spark technology using a Bloom filter has been developed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

ПЗ – програмні засоби

СД – сховище даних

СКБД – система керування базами даних

ВСТУП

Ні для кого не є секретом, що в сучасному світі обсяги даних стрімко зростають. Розподілені системи стають важливим інструментом для обробки великих обсягів даних. Розподілені системи – це мережі комп'ютерів, які працюють разом, об'єднані для досягнення спільної мети. Ці системи забезпечують величезну масштабованість, надійність та доступність, але також ставлять перед фахівцями нові виклики щодо обробки даних. Розподілені системи стали ключовим елементом сучасного інформаційного ландшафту. Вони забезпечують можливість обробляти великі об'єми даних з високою продуктивністю та надійністю. Вибір методу обробки залежить від конкретних потреб та вимог до системи. Паралельні розподілені системи використовуються для обробки великих об'ємів даних, забезпечуючи швидкість, надійність та масштабованість. Однак одним з ключових викликів є забезпечення ефективного доступу до даних в таких системах. В даний час спостерігається постійне збільшення обсягів збережених і аналізованих даних. IT сфера перебуває в постійному пошуку рішень для їх обробки. Поточні рішення для побудови сховищ даних (СД) з використанням багатовимірних кубів (MOLAP) і реляційних баз даних (ROLAP) мають погану масштабованість, що призводить до великих фінансових витрат для компаній на модернізацію комплексів по обробці даних. В даний час для виконання запитів до великих баз даних широко використовується технологія MapReduce. Вона передбачає паралельне виконання запитів до фрагментів бази даних, розподілених по великій кількості вузлів. Існуючі методи паралельної розподіленої обробки даних за технологією MapReduce дозволяють забезпечити доступ тільки до СД типу «зірка».

Хоча при зберіганні даних в нормалізованих таблицях сховище даних має вигляд «сніжинки». У цих методах використовуються фільтри Блума або дублювання таблиць вимірів по вузлах, але вони не дозволяють одночасно

реалізувати переваги обох підходів при виконанні одного запиту. Крім того, для існуючих методів немає математичного обґрунтування їх ефективності. Також важливим завданням є прогнозування часу виконання аналітичних запитів в паралельній розподіленої середовищі, тому що ці запити можна віднести до класу «важких». Останнім часом ця задача придбала ще більшого значення в контексті надання ХД в якості сервісу (DaaS). Постачальник DaaS повинен управляти інфраструктурними витратами, а також угодами про рівень обслуговування (SLA).

Для вирішення цього завдання використовуються вартісні прогностичні моделі. В існуючих роботах розглядаються моделі цього класу для реляційних баз даних. Тому розробка методу, що дозволяє обробляти в паралельній розподіленій системі складні запити до сховища даних довільного виду, і побудова адекватних вартісних моделей для оцінки часу виконання таких запитів, є актуальним завданням.

У роботі повинен бути у роботі аналіз існуючих методів обробки даних [1,2] в розподілених системах та методів доступу до даних в паралельних розподілених сховищах даних на платформі MapReduce. MapReduce – це модель програмування і асоційоване виконання для обробки та генерації великих наборів даних. Вперше представлений Google у 2004 році, цей підхід з того часу став фундаментом для великих систем обробки даних, таких як Hadoop. MapReduce базується на двох основних процедурах: Map та Reduce. Ці процедури приймають набір вхідних даних і конвертують його у проміжний набір даних у форматі пар "ключ-значення", а також приймають проміжний набір даних і комбінують дані за однаковими ключами. MapReduce пропонує високу масштабованість і надійність для обробки великих наборів даних. Хоча цей підхід має свої обмеження, він продовжує бути ключовою технологією в світі великих даних.

Доступ до даних в паралельних розподілених системах є складною задачею, яка вимагає виваженого підходу до архітектури та вибору технологій. Правильно підібрані методи і рішення дозволяють ефективно

працювати з даними, забезпечуючи швидкість, надійність та масштабованість.

Метою кваліфікаційної роботи є аналіз методів обробки та зберігання даних в розподілених системах.

Об’єкт досліження: методи доступу до даних в паралельних розподілених системах.

Завдання:

- аналіз і порівняння методів доступу до даних в паралельних розподілених сховищах даних на платформі MapReduce/Spark;
- розробка модифікованого методу виконання запитів до паралельного розподіленого сховища даних на базі технології MapReduce/Spark.

1 МЕТОДИ ДОСТУПУ ДО СХОВИЩ ДАНИХ

1.1 Огляд технології OLAP

Відповідно до класичним визначенням сховище даних (СД) - це «копія транзакційних даних, спеціальним чином структурована для підтримки виконання запитів і аналізу даних» [8].

У 1993 Е.Ф.Кодд ввів термін оперативного аналізу даних (OLAP – Online Analytical Processing). Основою таких систем стало СД - база даних, що містить великий обсяг історичної інформації в зручному для аналізу вигляді [9].

У 1995 році на основі вимог, сформульованих Е.Ф.Коддом, був створений тест FASMI (Fast Analysis of Shared Multidimensional Information). У назві методу фігурують основні принципи і вимоги до систем OLAP, а саме:

- швидкість – система повинна забезпечувати надання користувачу результатів запиту за прийнятний час (зазвичай не більше 5 с), нехай навіть ціною менш детального аналізу;

- аналіз – система повинна підтримувати логічний і статистичний аналіз даних, характерний для додатка, і забезпечувати збереження результатів у вигляді, доступному для кінцевого користувача;

- разподіленість – система повинна здійснювати доступ в розрахованому на багато користувачів режимі до даних з підтримкою відповідних механізмів блокувань і засобів авторизованого доступу;

- багатомірність – означає, що система повинна забезпечити багатовимірне концептуальне представлення даних, включаючи повну підтримку для ієрархій і множинних ієрархій. Багатовимірність є ключовим критерієм;

- інформація – можливість звертатися до будь-якої потрібної інформації незалежно від її обсягу і місця зберігання [10].

Таким чином, OLAP-технологія являє для аналізу дані у вигляді багатовимірних (і, отже, нереляційних) наборів даних, які називаються багатовимірними кубами (гіперкуб, куб фактів), осі якого виступають в ролі вимірювань, а осередки – залежні від них дані [3].

При тому гиперкуб є концептуальною логічною моделлю організації даних, а не фізичної реалізацією їх зберігання, оскільки зберігатися такі дані можуть і в реляційних таблицях [4,9,11].

Таким чином, багатовимірне концептуальне сховище є множинною перспективою, що складається з декількох незалежних вимірювань, уздовж яких можуть бути проаналізовані певні сукупності даних.

Одночасний аналіз по декількох вимірах визначається як багатовимірний аналіз. Осями багатовимірної системи координат є основні атрибути аналізованого бізнес-процесу (те, за чим ведеться аналіз).

Наприклад, для продажу це можуть бути тип товару, регіон, тип покупця. На перетинах осей – вимірювань (dimensions) – знаходяться дані, кількісно характеризують процес – заходи (measures), тобто агрегатні функції.

Кожен вимір включає напрямки консолідації даних, що складаються з серії послідовних рівнів узагальнення (рівнів ієрархії), де кожен вищестоящий рівень відповідає більшою мірою агрегації даних по відповідному вимірюванню (різні рівні їх деталізації).

У цьому випадку стає можливим довільний вибір бажаного рівня деталізації інформації по кожному з вимірів [3, 10]. Приклад багатовимірного куба даних з декількома вимірами представлений нижче (рисунок 1.1.)

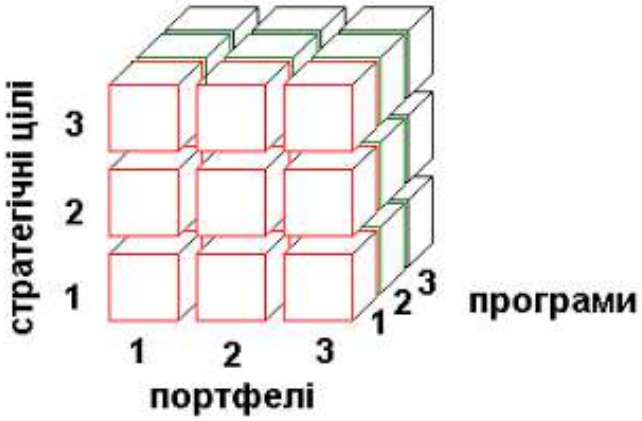

Рисунок 1.1 – Приклад багатовимірного куба даних

В [13] показано, що всі дослідження в цій області велися над розробкою методів обробки та зберігання великих обсягів даних. Також при реалізації СД виникають певні складнощі, пов'язані з їх масштабністю. Зі збільшенням обсягу оброблюваних даних різко зростає вартість фізичного обладнання.

1.2 Технологія MapReduce

Існує кілька підходів до побудови систем, що забезпечують розподілене обробку даних. Одним з найбільш популярних підходів є використання парадигми MapReduce, згідно з якою обробка даних розділяється на велику кількість елементарних завдань, що виконуються на різних вузлах кластера та, в кінцевому підсумку, зводиться в єдиний результат. Ця модель розроблена компанією Google і дозволяє обробляти петабайт даних за допомогою комп'ютерних кластерів [6, 7].

Технологія MapReduce призначена для організації розподілених обчислень на великих кластерах недорогих (і тому потенційно ненадійних) машин [14, 15].

Обчислення проводяться над множинами вхідних пар «ключ-значення», і в результаті кожного обчислення також створюється деякий

безліч результуючих пар «ключ-значення». Основний процес обробки даних складається з двох кроків: Map і Reduce. Кожен з цих двох кроків описується користувальницької функцією, при цьому складності і деталі виконання програми на кластері машин ховаються від аналітика або розробника [2].

Функція Map проводить попередню обробку даних - на вхід функції подається ключ і пов'язані з ним дані, а в результаті її роботи генеруються проміжні пари (ключ, значення). Кожен вузол кластера виконує функцію Map на своїй призначеної порції даних (в ідеалі, що знаходиться на тому ж сайті, щоб мінімізувати пересилання даних по мережі).

Крім того, на одному вузлі можуть одночасно виконуватися кілька Map-завдань (це число залежить від параметрів вузла). Проміжним етапом між Map і Reduce є етап перемішування (shuffle), в результаті якого на вхід кожної виконуваної функції Reduce надходить один з проміжних ключів і список значень, що відповідають цьому ключу (на одному вузлі можуть одночасно виконуватися кілька Reduce-задач). Функція Reduce виробляє згортку і повертає підсумковий список значень. Таким чином, застосування функції Reduce до кожного проміжного ключу і списку значень, асоційованих з цим ключем, дозволяє сформувати остаточний результат [2].

Виклики Map розподіляються по декільком вузлам кластера шляхом поділу вхідних даних на M непересічних груп (split). Вхідні групи можуть паралельно оброблятися на різних машинах. Виклики Reduce розподіляються шляхом поділу простору проміжних ключів на R частин з використанням деякої функції поділу, наприклад, функції хешування. Число розділів R і функція поділу задаються користувачем. Виконання MR-програми відбувається наступним чином: спочатку середу MapReduce розщеплює вхідний файл на M частин, розмір яких може задаватися користувачем. Потім, відразу в декількох вузлах кластера запускається основна програма MapReduce. Один з примірників цієї програми грає особливу роль і називається розпорядником (master), Решта екземплярів є виконавцями (worker), яким розпорядник призначає роботу [12].

Розпорядник повинен призначити виконавцям для виконання M завдань Map і R завдань Reduce. Виконавець завдання Map читає вміст відповідної групи, розбирає пари «ключ-значення» вхідних даних і передає кожну пару в певну користувачем функцію Map. Проміжні пари «ключ-значення», вироблені функцією Map, буферизуются в основний пам'яті. Періодично буферізованние пари, що розділяються на R областей на основі функції розподілу, записуються в локальну дискову пам'ять виконавця. Координати цих збережених на диску буферизованих пар відсилаються розпоряднику, який передає ці координати виконавцям завдання Reduce. I-ий Reduce-виконавець забезпечується координатами всіх областей буферизованих пар, вироблених усіма M Map-виконавцями.

Після отримання цих координат виконавець завдання Reduce з використанням механізму RPC переписує дані з локальних дисків виконавців завдання Map в свою пам'ять або на локальний диск. Після перепису всіх проміжних даних виконується їх сортування за значеннями проміжного ключа для освіти груп з однаковим значенням ключа. Якщо обсяг проміжних даних занадто великий для виконання сортування в основний пам'яті, використовується зовнішня сортування.

Далі Reduce-виконавець організовує цикл по відсортованим проміжним даними і для кожного унікального значення ключа викликає призначену для користувача функцію Reduce з передачею їй як аргумент значення ключа та відповідного безлічі значень. Результуючі пари функції Reduce додаються в остаточний результуючий файл даного Reduce- виконавця. Після завершення всіх завдань Map і Reduce розпорядник активізує програму користувача, що викликала MapReduce. Після успішного завершення виконання завдання MapReduce результати розміщуються в R файлах розподіленої файлової системи, імена цих результуючих файлів задаються користувачем. Зазвичай не потрібно об'єднувати їх в один файл, тому що часто отримані файли використовуються в якості вхідних для запуску наступного MR-завдання або в будь-якому іншому розподіленому додатку, яке може отримувати вхідні

дані з декількох файлів [14]. Процес виконання пар завдань Map-Reduce можна представити у вигляді послідовності дій (рисунок 1.2).

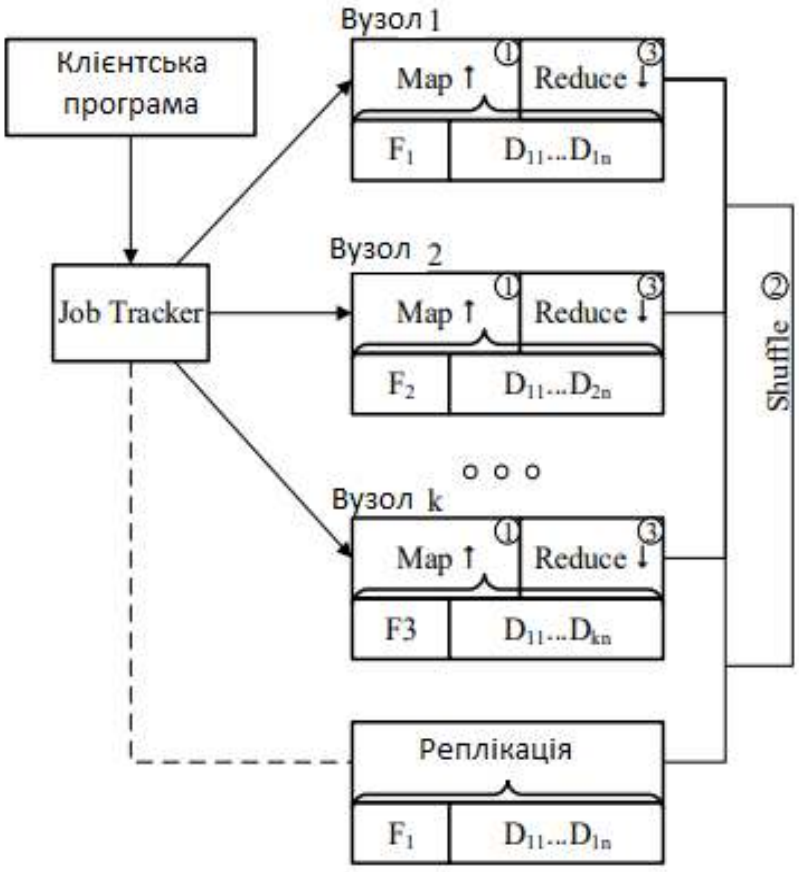

Рисунок 1.2 – Процес виконання пар завдань MapReduce

1.2.1 Переваги використання MapReduce

Одним з основних переваг технології MapReduce є горизонтальне масштабування, при якому потужність нарощується додаванням в кластер нових пристроїв (вузлів) замість заміни одного сервера на більш потужний. При горизонтальному масштабуванні обчислювальну потужність можна нарощувати практично нескінченно за рахунок великого числа дешевих машин. Так як основний напрямок застосування технології MapReduce на базі Hadoop – робота з погано структурованими даними, то використання пари ключ – значення замість структурованих таблиць дозволяє

використовувати більш гнучкі підходи до роботи з даними, а застосування функціонального програмування замість жорстко формалізованих запитів дає можливість конкретно описати кроки роботи з даними. Іншою важливою перевагою технології MapReduce є забезпечення надійності за допомогою реплікації.

Відповідно до даної парадигмі, кожен блок може бути розміщений на декількох вузлах, його розмір і коефіцієнт реплікації (кількість вузлів, на яких повинен бути розміщений кожен блок) визначаються в налаштуваннях на рівні файлу. За допомогою реплікації забезпечується стійкість розподіленої системи до локальних відмов окремих вузлів [16].

1.2.2 Недоліки використання MapReduce

Для того, щоб забезпечити роботу з даними на низькому рівні абстракції з застосуванням методів функціонального програмування, необхідно пред'являти високі вимоги до навичок розробників системи. Даний фактор значно збільшує витрати компанії на заробітну плату висококваліфікованих співробітників. Крім того, зараз ринок праці перебуває в різкому дефіциті таких фахівців, час пошуку кандидата на такі вакансії може досягати півроку [17].

Відсутність підтримки звичного бізнес-аналітикам SQL є причиною того, що вимагає додаткового ускладнення взаємодії технологій і бізнесу і зазвичай передбачає два шляхи вирішення:

- поєднання ролі розробника і аналітика в рамках однієї позиції;
- постановка завдання спочатку від бізнесу аналітику, а потім від аналітика розробнику.

У першому випадку різко зростають вимоги до фахівців, що веде до складності їх пошуку на ринку праці і високим зарплатних очікуваннях. У другому випадку аналітичний функціонал переходить на дві штатні позиції,

що, по-перше, збільшує штат, а по-друге, спотворює і уповільнює ланцюжок із взаємодії бізнес-підрозділів з аналітичним підрозділом [18].

1.2.3 Додаток Hive

Hive – система управління базами даних на основі платформи Hadoop. Hive управляє даними, збереженими в файлової системі HDFS, і надає мову запитів на базі SQL, які перетворюються в послідовність завдань ядром часу виконання. Hive підтримує різні формати зберігання даних, а також може працювати безпосередньо з HDFS і Apache HBase [18], [20].

Таким чином, Hive вдає із себе движок, який перетворює SQL-запити в ланцюжки Map-Reduce завдань. Движок включає в себе наступні послідовні компоненти [18]:

- HiveQL – підтримує мову запитів Hive Query Language, який заснований на мові SQL;
- Parser – розбирає входять SQL-запити (логічний план виконання запиту);
- Optimizer – оптимізує запит для досягнення більшої ефективності;
- Planner – планує завдання на виконання;
- Executor – запускає завдання на фреймворку MapReduce. Більш детально взаємозв'язок компонентів Hive представлена на наступній схемі (рисунок 1.3).

Hive визначає просту SQL-подібну мовау запитів, названу HiveQL, що дозволяє користувачам, знайомим з SQL легко працювати з запитами до даних. У той же час, ця мова дозволяє програмістам, які знайомі зі структурою MapReduce, підключити свої власні карти для виконання більш складного аналізу, який може бути не підтримується вбудованими можливостями мови. HiveQL може бути розширений для користувача скалярними функціями (UDF), агрегації (UDAF років), і табличними функціями (UDTF) [19, 21, 22].

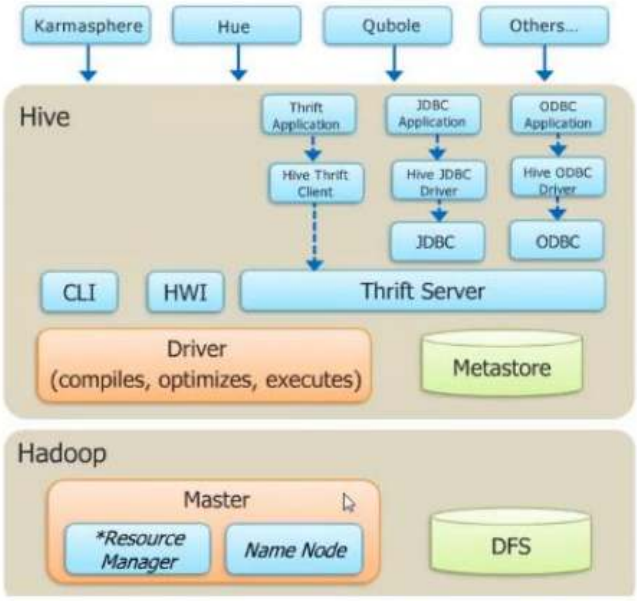

Рисунок 1.3 – Взаємозв’язок компонентів Hive

Перелічимо основні недоліки технології Hive:

- Hive не призначений для OLTP навантажень і не пропонує виконання запитів в режимі реального часу або оновлень. Кращим використанням для Hive є виконання пакетних завдань на великих обсягах оброблюваних даних (логи, журнали транзакцій);

- час виконання запитів в середовищі Hive велике. Це пов'язано з тим, що запит транслюється в послідовність завдань (Job), і все проміжні дані зберігаються на диску. Час різко зростає, якщо виконується з'єднання таблиць вимірів з великою таблицею фактів;

- незважаючи на SQL – подібний синтаксис, багато видів запитів неможливо в прямому вигляді реалізувати в Hive [20].

1.2.4 Методи MFRG та MRIJ

В рамках методу MFRJ таблиці вимірювань продубльовані по всіх вузлах системи MapReduce (MR). Вони зберігаються по рядках. Таблиця фактів ділиться на кілька груп: в 1-у групу входять стовпці (колонки)

зовнішніх ключів вимірювань (fki), стовпець кожного факту (mi) утворює окрему групу. Принцип обробки даних за допомогою технології MFRJ представлений на наступній схемі (рисунок 1.4).

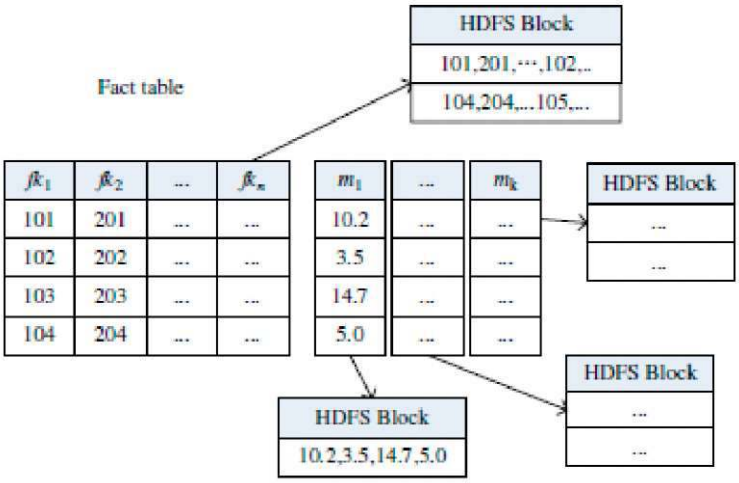

Рисунок 1.4 –Принцип обробки інформації за допомогою метода MFRJ

Метод MFRJ має такі особливості:

- для вимірювань, які беруть участь в запиті, будуються хеш-індекси в оперативній пам'яті. Вони використовуються для фільтрації записів 1-ої групи;

- на фазі Reduce виконується з'єднання записів по номеру позиції в таблиці фактів і обробка додаткових умов в запиті.

Метод MFRJ має наступні недоліки:

- всі таблиці вимірів не розподілені по вузлах, а дублюються на всіх вузлах;

- хеш-індекси вимірювань, що беруть участь в запиті, повинні зберігатися в ОП кожного вузла;

- має місце пересилання зайвих записів і рання матеріалізація вимірювань і фактів, що призводить до збільшення обсягу даних, переданих між вузлами на етапі shuffle;

- низький коефіцієнт стиснення зовнішніх ключів таблиці фактів (вони зберігаються через підрядник).

Так само в [23] розглядається метод MRIJ, в рамках якого немає необхідності дублювати всі таблиці вимірювань по всіх вузлах. Вони зберігаються по рядках і фрагментовані по вузлах. Кожен стовпець таблиці фактів зберігається у вигляді колонки, при цьому блоки таблиць вимірів і колонок таблиці фактів розподілені по вузлах довільним чином, а хеш-індекс для вимірювань будується в ОП по ключу після відбору необхідних записів. Принцип роботи MRIJ представлений нижче (рисунок 1.5).

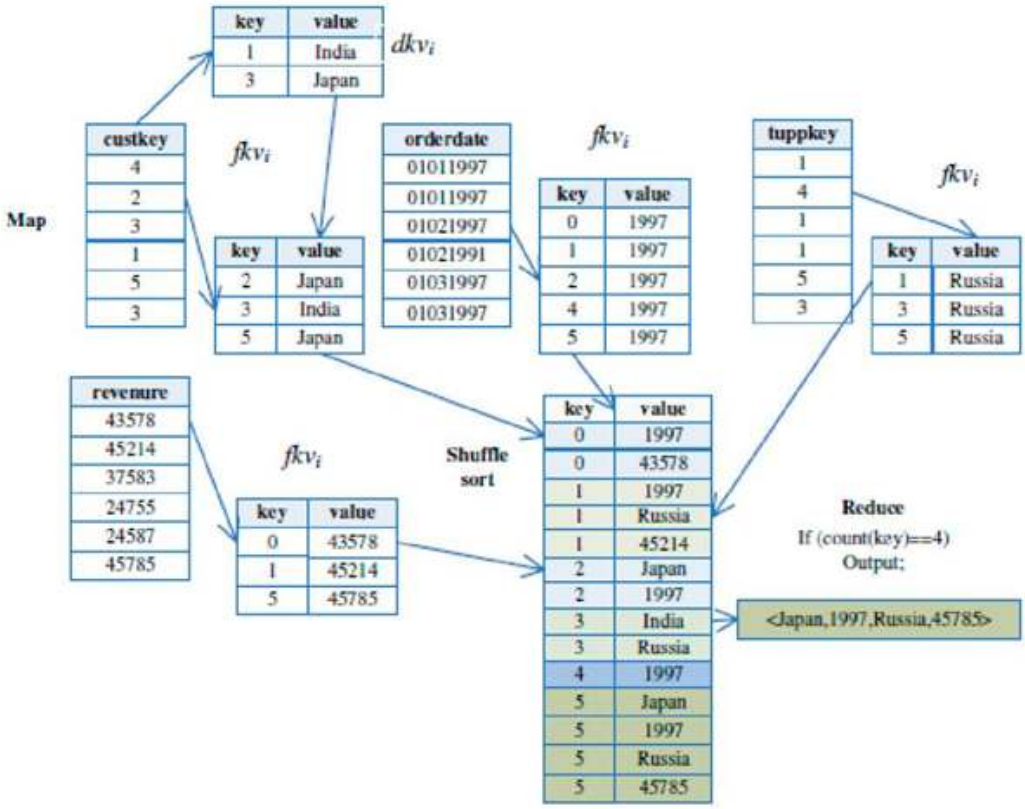

Рисунок 1.5 – Принцип обробки інформації по методу MRIJ

Недоліки методу MRIJ [24]:

- необхідно дублювати по вузлах значення вимірювань при виконанні кожного запиту. Хеш-індекси всіх вимірювань, які беруть участь в запиті, повинні повністю зберігатися в ОП;

- має місце пересилання зайвих записів і рання матеріалізація вимірювань і фактів, що призводить до збільшення обсягу даних, переданих вузлів на етапі shuffle;

- для різних позицій таблиці фактів пересилання значень вимірів дублюється, якщо в цих позиціях значення зовнішніх ключів збігаються.

Таким чином, перераховані методи мають такими загальними недоліками:

- всі таблиці вимірювань, які беруть участь в запиті, повинні бути продубльовані на всіх вузлах системи. Для цього потрібні додаткові ресурси пам'яті і тимчасові витрати;

- хеш-індекси вимірювань, що беруть участь в запиті, повинні зберігатися в ОП вузла. Якщо в якості вимірювання виступає велика повноцінна таблиця (наприклад, «Співробітник», «хвороба», «супутник» і т.д.) і хеш-індекси великі, то можуть виникнути проблеми з оперативною пам'яттю, так як в вузлах MapReduce використовуються малопотужні станції;

- має місце пересилання зайвих записів, що призводить до збільшення обсягу даних, переданих вузлів на фазі shuffle, тобто в цих методах не вдалося уникнути пересилання значень вимірів і фактів між вузлами. З урахуванням результатів аналізу недоліків методів MFRJ і MRIJ був запропонований метод доступу до сховища даних, в якому були усунуто більшість з перерахованих проблем.

1.3 Організація доступу к розподіленим даним з використанням платформи Spark

Apache Spark – фреймворк з відкритим вихідним кодом, що поєднує в собі механізм розподілу програм по кластеру машин з моделлю для написання програм поверх нього. Spark, створений на факультеті AMPLab Каліфорнійського університету в Берклі, а потім відданий фонду Apache Software Foundation – ймовірно, перше програмне забезпечення з відкритим

вихідним кодом, по-справжньому дає дослідникам даних можливість використовувати розподілене програмування. Основні компоненти платформи Spark представлені на рисунку 1.7.

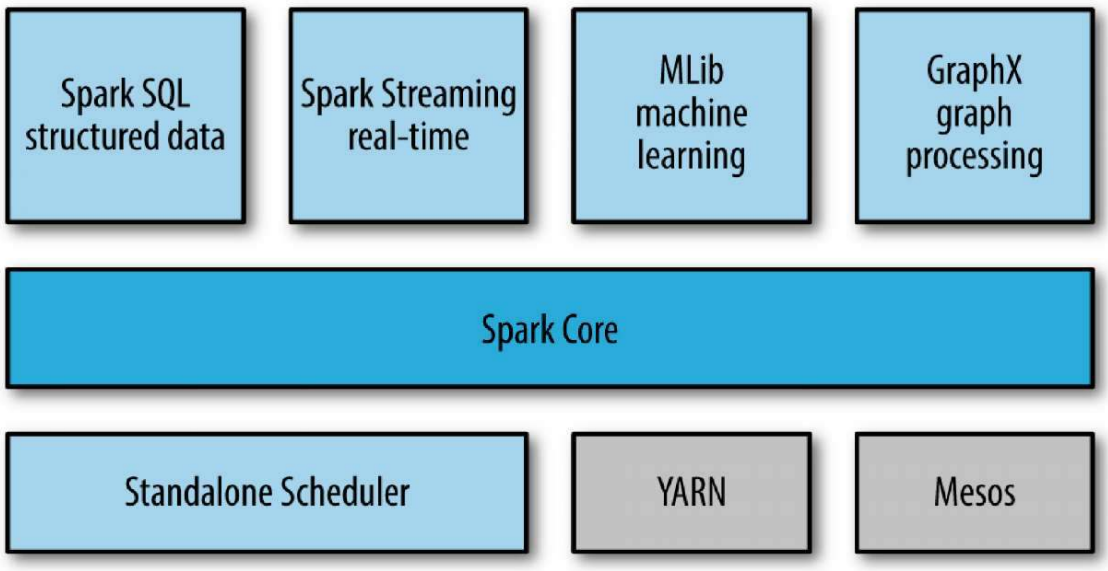

Рисунок 1.6 – Основні компоненти платформи Spark

Spark зберігає лінійну масштабованість і відмовостійкість MapReduce, додаючи нові можливості:

- універсальний ациклический граф (Directed Acyclic Graph (DAG));
- орієнтація на розробників і володіння потоковим API;
- розширені можливості завдяки обробці в ОП;
- продуктивність аналітика.

Таким чином, замість того, щоб покладатися на жорсткий формат відображення (Map) і згортки (Reduce), движок Spark може виконувати більш універсальний орієнтований ациклічний граф операторів DAG.

Це означає, що в тих випадках, коли MapReduce доводиться записувати проміжні результати в розподілену файлову систему, Spark може передати їх безпосередньо наступного кроку конвеєра. По-друге, він доповнює зазначену можливість багатим набором перетворень, що дозволяють користувачам задавати обчислення більш природним чином.

Він орієнтований насамперед на розробників і володіє потоковим API, здатним ставити складні конвеєри в декількох рядках коду. По-третє, Spark розширює можливості своїх попередників за допомогою обробки в оперативній пам'яті. Його абстракція RDD (Resilient Distributed Dataset – стійкий розподілений набір даних) дозволяє розробникам матеріалізувати будь-яке місце в оброблюваному конвеєрі в пам'яті машин кластера. Завдяки цьому, наступним крокам, які бажають працювати з тими ж даними, не потрібно буде заново їх обчислювати або зчитувати з диска. Ця можливість відкриває шлях до сценаріїв використання, колишнім раніше недоступними для механізмів розподіленої обробки.

Spark відмінно пристосований до високоітеративним алгоритмам, що вимагає множинних проходів по набору даних, так само як і до працюючих за запитом додатків, швидко відповідає на запити користувача завдяки перегляду великих наборів даних у пам'яті. В рамках перерахування переваг Spark обов'язково варто відзначити той факт, що злиття всього конвеєра, від попередньої обробки до оцінки моделі, в єдине середовище програмування може прискорити роботу аналітика. Об'єднання моделі програмування з набором аналітичних бібліотек в REPL дозволяє уникнути перемикань на IDE і назад, неминучих у фреймворками типу MapReduce, і труднощів предобработки і переміщення даних з HDFS і в неї, яких вимагають такі фреймворками, як R. Ускоряя конвеєр експериментів над даними, їх обробку і аналіз, підвищується ймовірність вилучення користі для бізнесу, своєчасного реагування на помилки і виявлення зон росту.

1.4 Розробка методу доступу до сховища даних без кешування таблиць

Розроблен метод доступу до сховища даних без кешування таблиць вимірювань в оперативній пам'яті. Для зберігання даних в запропонованому методі використовується формат RCFile. Схема зберігання даних представлена нижче (рисунок 1.7).

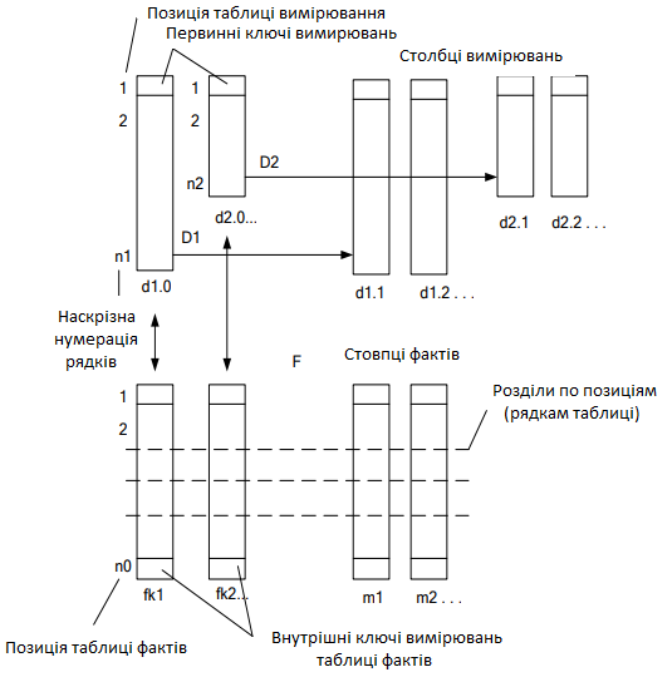

Рисунок 1.7 – Зберігання даних

Переваги методу:

- відсутне дублювання таблиць вимірів в вузлах, і тому немає необхідності передачі їх фрагментів по мережі;

- немає необхідності хешувати таблиці вимірювань в оперативній пам'яті;

- таблиці вимірювань і фактів можуть бути рівномірно розподілені по вузлах, не треба міняти політику розподілу блоків, прийняту в MapReduce за замовчуванням;

- використовується уніфікований метод зберігання таблиць (RCFile), що дозволяє виконувати декомпресію тільки тих стовпців, які використовуються в запиті.

1.5 Порівняння Hadoop та Spark

Порівняємо два розглянутих підходу, Hadoop і Spark, по продуктивності, стійкості до збоїв і простоті програмування:

- запити в Spark виконуються швидше. У Hadoop проміжні дані зберігаються і на локальному диску (перед shuffle), і в файлової системі HDFS. В Spark проміжні дані зберігаються на локальному диску тільки перед shuffle («перетасуванням»). Результати виконання операції join зберігаються в ОП (при достатньому обсязі оперативної пам'яті). При цьому витрата ВП набагато вище;

- стійкість до збоїв в Hadoop вище. Якщо станеться збій вузла, то Hadoop перезапустить функцію (map або reduce), яка виконує тільки частину роботи, на іншому вузлі. В Spark весь запит виконується з початку, якщо все проміжні дані зберігаються в ОП;

- в Spark простіше програмувати складні процеси обробки (відмінні від простих запитів Select).

В Spark досить послідовно закодувати необхідні операції map, join, save і реалізувати функції map на мові високого рівня. У Hadoop необхідний кодувати функції map і reduce для кожного завдання на мові Java. При цьому система Spark відмінно масштабується, обробка на кожному етапі ведеться паралельно на багатьох серверах. Це дозволяє забезпечити високу ефективність обробки вхідного потоку даних і запитів операторів.

2 МЕТОД ДОСТУПУ ДО СД З ВИКОРИСТАННЯМ ФІЛЬТРІВ

Для більшості сховищ даних (СД) найбільш ефективним способом моделювання N-мірного куба фактів є схема «зірка». Ця схема зазвичай представляється як таблиця фактів, оточена декількома таблицями розмірності (або вимірювань) в формі зірки. Кожна таблиця розмірності відповідає будь-якій колонці в таблиці фактів. Таблиці розмірності використовуються як основа для аналізу даних в таблиці фактів. Кінці зірки утворюються таблицями вимірювань (Dimension Tables), а їх зв'язки з таблицею фактів (Fact Table), розташованої в центрі, утворюють промені.

Багато схем баз даних можна представити у вигляді схеми «сніжинка» з додатковими зв'язками. Наприклад, схему бази даних з TPC-H можна представити в наступному вигляді. Це не зовсім «сніжинка», тому що таблиця Nation є консольною для двох таблиць: Supplier і Customer.

2.1 Схеми сховищ даних

Для нормалізації таблиць вводять багаторівневі вимірювання, тобто використовуються різні розширення схеми «зірка», наприклад, схема «сніжинка» (snowflake schema). Це розширення може проявлятися в двох різновидах [3, 11]:

- у разі великого числа складних атрибутів в таблиці вимірювання деякі атрибути можуть бути деталізовані в окремих таблицях цього виміру. Іншими словами, окремий вимір міститься не в одній, а в декількох пов'язаних між собою таблицях. Додаткові таблиці вимірювання в такій схемі, зазвичай відповідні верхніх рівнів ієрархії вимірювання і перебувають у відношенні «один до багатьох» з головною таблицею вимірювання, іноді називають консольними таблицями;

- інша розширення пов'язано зі створенням окремих таблиць фактів для поєднань різних вимірів.

Схема бази даних має наступні кущі типу «зірка» з одним або декількома вимірами.

Кущ – це таблиця фактів і всі пов'язані з нею таблиці вимірювань. Остання таблиця в описі кожного приведеного вище куща – це таблиця фактів цього куща.

Тест TPC-H оцінює продуктивність систем підтримки прийняття рішень. Він складається з набору складних бізнес-орієнтованих запитів ad-hoc.

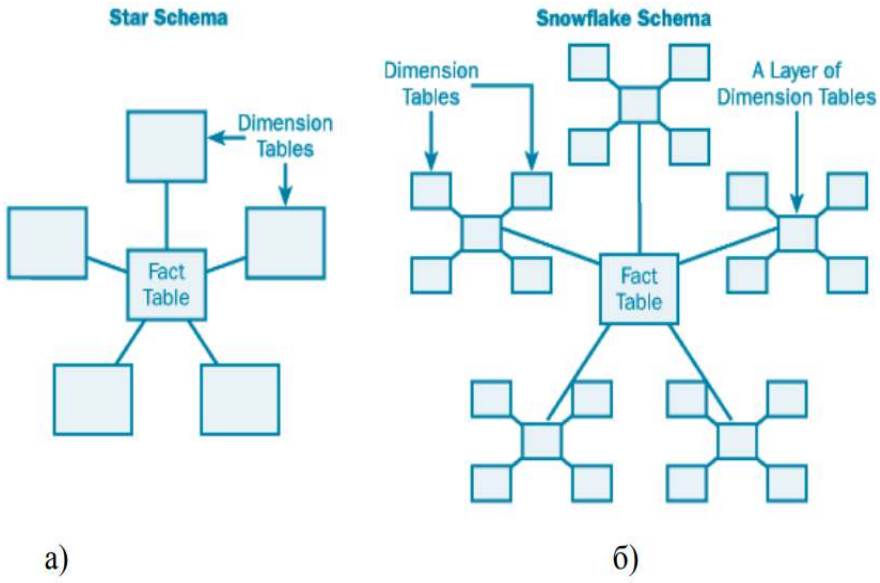

Рисунок 2.1 – Схеми поширених СД

Кущ – це таблиця фактів (куща) і всі пов'язані з нею таблиці вимірювань. Остання таблиця в описі кожного приведеного вище куща – це таблиця фактів цього куща, зліва – це вимірювання куща. Як видно з опису, таблиця фактів куща може виступати в якості вимірювання в іншому кущі. Наприклад, таблиця фактів Partsupp в кущі «Part, Supplier – Partsupp» є виміром в кущі «Partsupp, Orders – Lineitem». Кущі «Nation-Supplier» і «Nation-Customer мають загальну таблицю вимірювання Nation. Таблиця, яка

не є фактом ні в одному кущі, назвемо кінцевої. Кінцевими таблицями (рисунок 2.2) є таблиці Part і Region. Пронумеруємо таблиці фактів в кущах довільним чином рисунок 2.2).

Таблицю фактів i-го куща позначимо через F_i, i = 1 ... NF. Також довільно пронумеруємо кінцеві таблиці. Позначимо їх через D_i, i = 1 .ND.

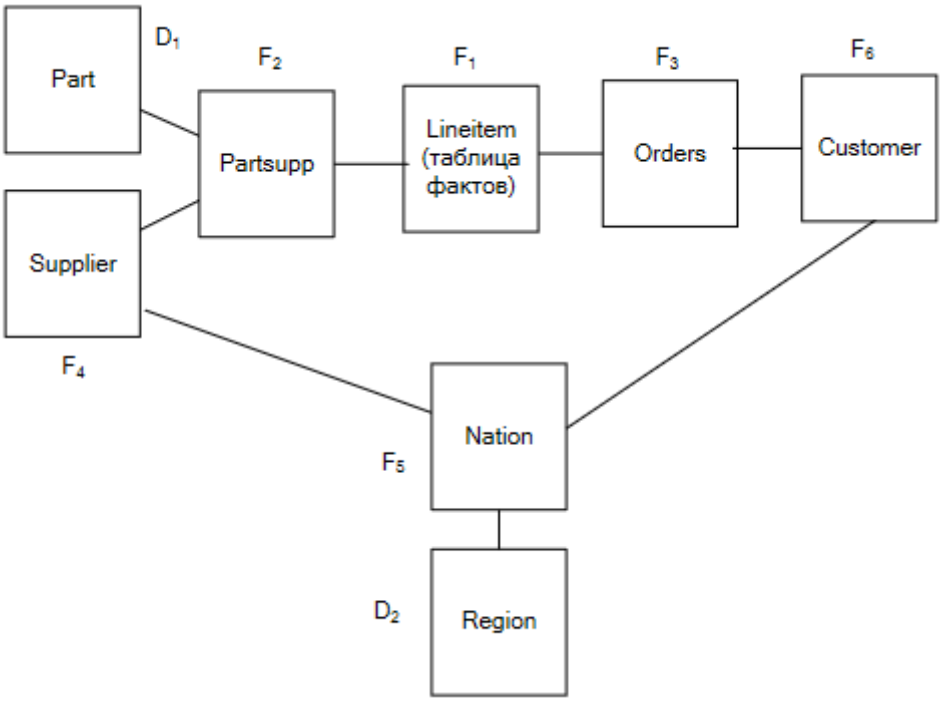

Рисунок 2.2 – Схеми поширених СД

Тест TPC-H оцінює продуктивність систем підтримки прийняття рішень (СППР). Він складається з набору складних бізнес-орієнтованих запитів ad-hoc. Дані в таблицях і запити підібрані так, щоб відображати деяку усереднену по індустрії бізнес-активність. Типові запити складені так, щоб відповідати основним типам запитів в СППР: ціноутворення і знижки, управління прибутком, дослідження переваг покупців, дослідження ринку і т.п.

Крім того, параметри атрибутів, їх селективність і потужність в конкретних запитах тесту добре відомі, що полегшує створення математичної моделі і її калібрування. Завдяки детально описаним в документації

властивостям схеми бази даних і запитів, дані тесту TPC-H були обрані для постановки тестового експерименту на віртуальному кластері і послужили основою для створення тваринницьких ферм математичної моделі.

Схема сховища тесту TPC-H представлена на рисунку 2.3. Схема сховища тесту TPC-H представлена на рисунку 2.3.

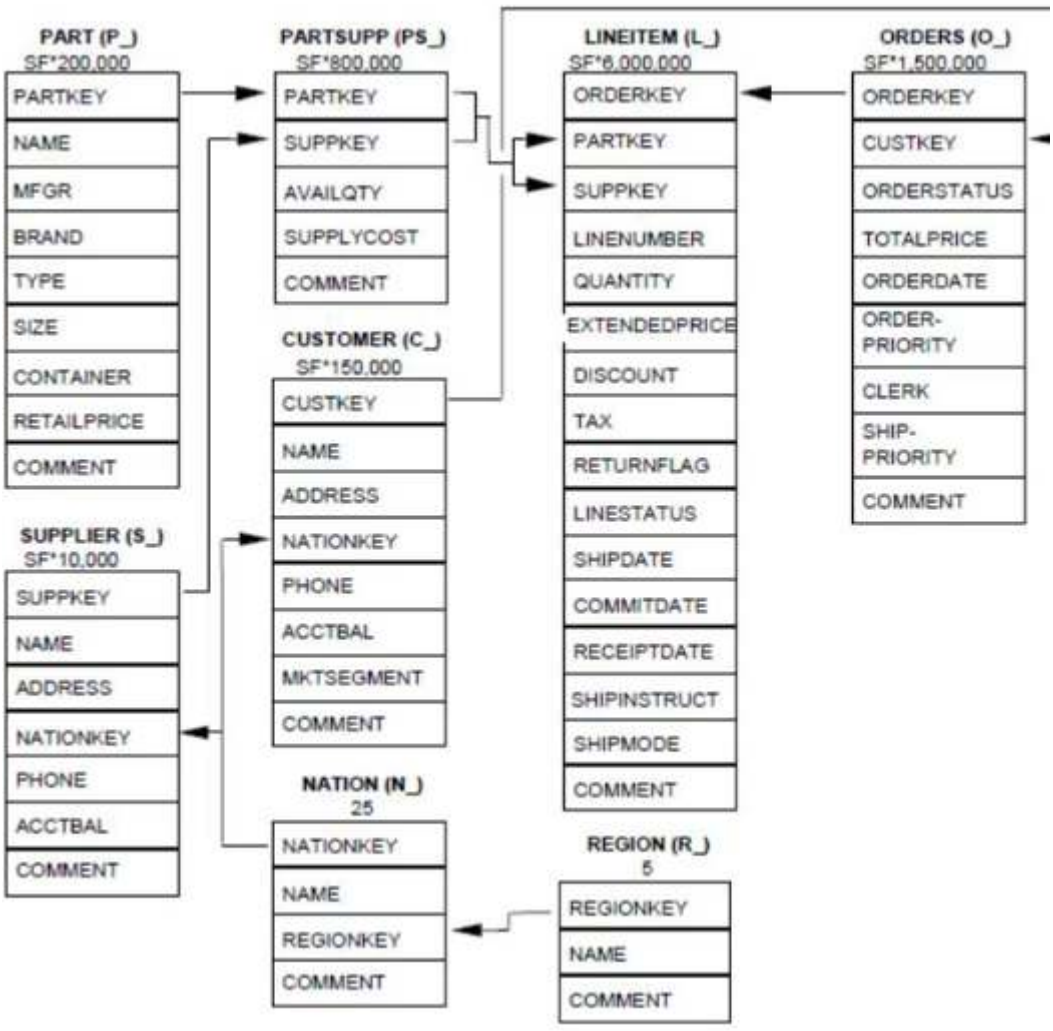

Рисунок 2.3 – Інфологічна схема СД

2.2 Представлення запитів

Таблиці, які беруть участь в запиті, можна також зв'язати і представити у вигляді вкладених кущів. Назвемо отриману структуру схемою запиту.

Подання вихідного запиту у вигляді кущів дозволяє описати вихідний запит як послідовність підзапитів і з'єднань, кожне з яких пов'язане з будь-яким кущем запиту. Розглянемо приклади перетворення вихідного запиту в послідовність 5 підзапитів і з'єднань.

```
select    l_orderkey
, sum(l_extendedprice*(1-l_discount)) as revenue
, o_orderdate
, o_shippriority
from          customer, orders, lineitem
where         c_mktsegment = '[SEGMENT]'
and           c_custkey = o_custkey
and           l_orderkey = o_orderkey
and           o_orderdate < date '[DATE]'
and           l_shipdate > date '[DATE]'
group by      l_orderkey, o_orderdate, o_shippriority
order by      revenue desc, o_orderdate;
```

Рисунок 2.3 – Запит Q3 из тесту TPC-H

Q3 являє собою «запит пріоритету доставки». Він витягує пріоритет доставки та потенційний дохід, який визначається як сума l_extendedprice * (1-l_discount), з замовлень, що мають найбільший дохід серед тих, які не були надіслані на певну дату. Замовлення перераховуються в порядку убування доходу. Якщо існує більше 10 відвантажених замовлень, виводяться тільки 10 замовлень з найбільшим доходом.

Схема запиту Q3 має вигляд, представлений на рисунку 2.4. У дужках вказані умови фільтрації вихідних таблиць. Ідентифікатори вихідних таблиць наведені на рисунку 2.2.

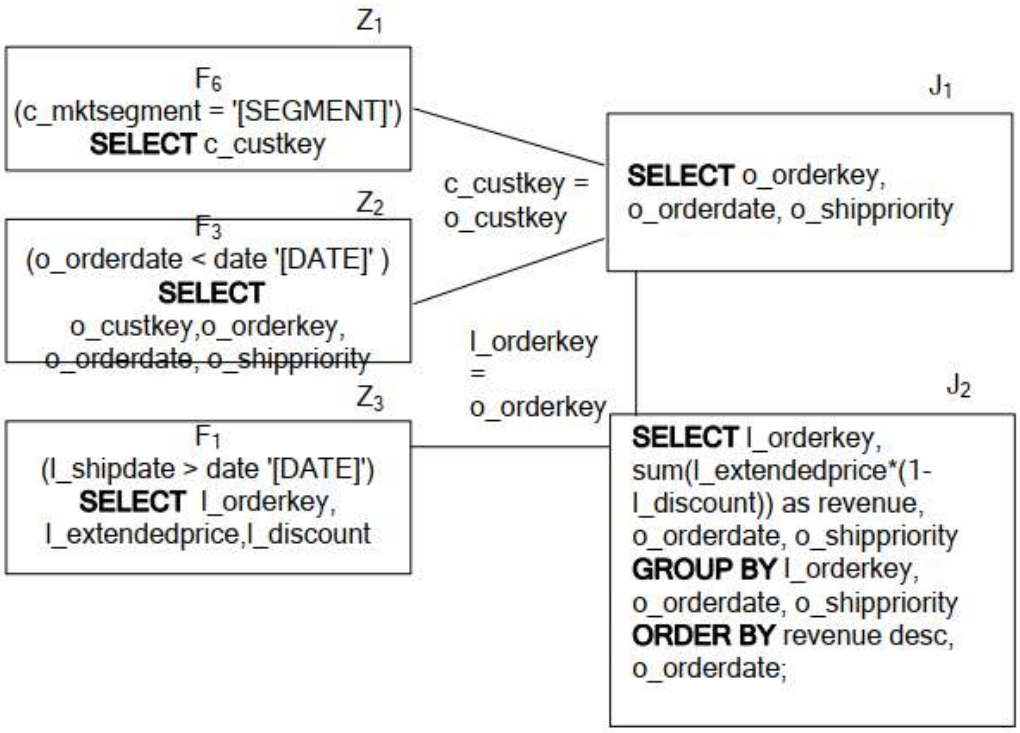

Рисунок 2.4 – Схема запиту Q3 тесту TPC-H

Тут можна виділити два кущі Z1, Z2-J1і Z3, J1-J2. В описі кожного куща зліва наведені таблиця вимірювання і таблиця фактів, а праворуч - оператор їх сполуки. Послідовність підзапитів {Zi} і з'єднань {Ji} має такий вигляд, який представлен на рисунку 2.5.

Подання вихідного запиту у вигляді кущів дозволяє описати вихідний запит як послідовність підзапитів Zi і з'єднань Jj. Це можна зробити практично для будь-якого SQL-запиту. Для цього достатньо вкладені підзапити SELECT представити у вигляді проміжних таблиць і включити їх у FROM вихідного запиту. Далі наведені схеми формування проміжних таблиць для різних підзапитів SELECT мови SQL. У дужках показані імена відповідних запитів з тіста TPC-H. В Spark створення проміжних таблиць може бути реалізовано за допомогою DataFrame / DataSet.

Z_1:

```
select c_custkey
from Customer F6
where c_mktsegment = '[SEGMENT]' ;
```

Z_2:

```
select o_custkey,o_orderkey, o_orderdate, o_shippriority
from Orders F3
where o_orderdate < date '[DATE]';
```

J_1:

```
select o_orderkey, o_orderdate, o_shippriority
from Z1, Z2
where c_custkey = o_custkey;
```

Z_3:

```
select l_orderkey, l_extendedprice, l_discount
from Lineitem F1
where l_shipdate > date '[DATE]'
```

J_2:

```
select l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o_shippriority
from Z3, J1
where l_orderkey = o_orderkey
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate;
```

Рисунок 2.5 – Послідовність підзапитів

2.3 Розробка модифікованого методу

Розроблений метод включає наступні кроки.

Крок 1. Розробити схему запиту.

Крок 2. Визначити підзапити і з'єднання, де формуються і використовуються фільтри Блума. Включити в алгоритм 1 після пропозицій select оператори побудови або використання фільтрів Блума. Це дозволяє істотно зменшити обсяг пересилаються по мережі даних на фазі перетасовки (shuffle) і зменшити час з'єднання таблиць на стадії Reduce.

Крок 3. Використовуючи алгоритм 1, згенерувати програму на мові Scala для виконання запиту в середовищі Spark. У наведених у п. 2.1 прикладах 1 і 2 в кожному кущі була присутня одна таблиця вимірювань і одна таблиця фактів. З алгоритму 1, слід, що кожне з'єднання Jr відповідає деякому куща, в якому можуть з'єднуватися кілька таблиць вимірів {Di} і таблиця фактів F. Реалізацію підзапитів, зазначених в кущі схеми вихідного запиту, і їх з'єднання за допомогою каскадного використання фільтра Блума (КІФБ) можна описати у вигляді послідовності перетворень і дій, представлених на рисунку 2.6.

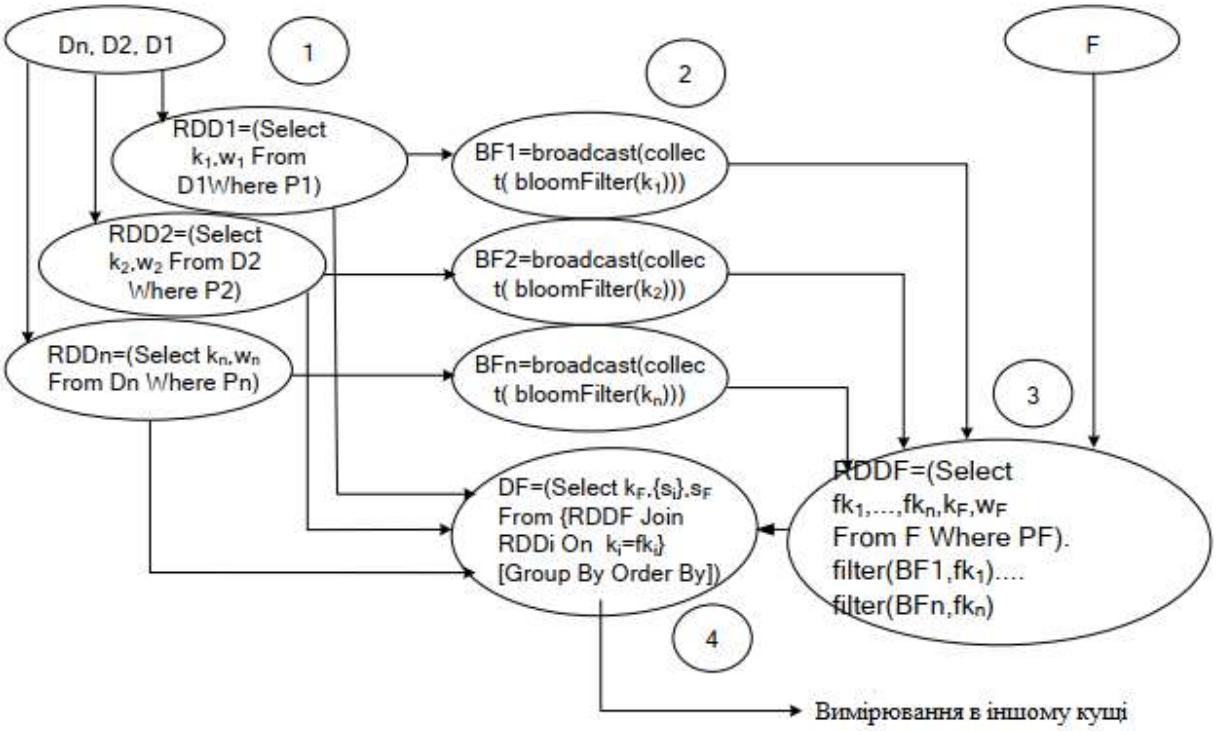

Рисунок 2.6 – Загальна схема реалізації куща вихідного запиту

Послідовність перетворень і дій утворює план DAG (Directed Acyclic Graph) виконання запиту. У загальному випадку план може бути складним і мати вигляд ациклического графа. Він працює як конвеєр, паралельно «проштовхуючи» розділи RDD (фрагменти таблиць, які зберігаються на станціях кластера) між вузлами графа. На Малюнок 2.6 приведений план

куща запиту з декількома таблицями вимірювань і таблицею фактів: (D1; ..., Dn, F).

Результат їх з'єднання може виступати в якості вимірювання в іншому кущі запиту. Трек просування розділів RDD має такий вигляд (див. рисунок 2.6).

Етап 1. Читання таблиць змін Di, фільтрація записів, отримання проекції (ki, wi).

Етап 2. Побудова фільтрів Блума для розділів таблиць RDDi в ОП (по ключу ki для кожного вимірювання), збірка кожного фільтра Блума на драйвер (collect), широкомовлення фільтрів Блума по всім Виконавцям (broadcast), де виконується фільтрація таблиці фактів (F) .

Етап 3. Читання таблиці фактів, фільтрація записів зі умові PF і за допомогою фільтрів Блума, отримання проекції ({fki}, kF, wF).

Етап 4. Послідовне з'єднання відфільтрованої таблиці фактів з відфільтрованими таблицями вимірювань (RDDF Join RDDi). Групування і сортування, якщо це передбачено для даного куща. Перехід до наступної куща.

2.4 Розробка моделі виконання запитів

Оцінимо виграш в обсязі переданих по мережі даних при використанні методу КІФБ. Розглянемо два випадки. Випадок 1. K-1 кущів (Ti, Ti + i) з одним виміром в кожному (рисунок 2.7). Будемо припускати, що таблиці Ti і Ti + 1 пов'язані ставленням 1: M, i = 1 ... K-1. Таблиця Ti1 відрізняється від вихідної таблиці Ti тим, що вона може включати колонки інших таблиць як результат попередніх з'єднань, але при цьому | Ti1 | = | Ti | (В силу зв'язку 1: M). На рисунку прийняті наступні позначення: BFi- це фільтр Блума, Vi і Vi1 – це обсяги (в байтах) тих колонок таблиць Ti і Ti1, які вказані в запиті SELECT, pi – ефективна селективність (частка записів таблиці Ti, що задовольняють умові підзапиту по цій таблиці).

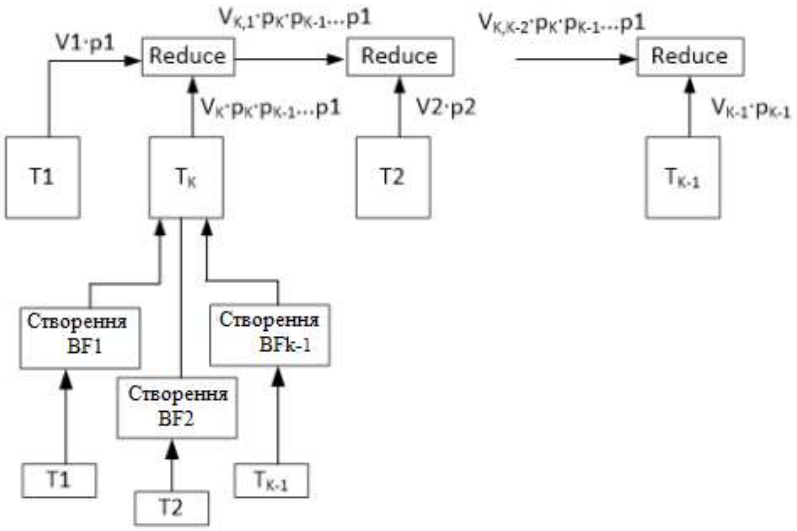

Рисунок 2.7 – Схема з'єднання K-1 таблиць вимірів з таблицею фактів з використанням фільтрів Блума

Щоб не використовувати конкретні числові значення в якості індексів, будемо вважати такі позначення еквівалентними: T1 = T1, p1 = p1, V1 = V1, V21 = V21 і т.д. Пояснимо використання розробленого методу КІФБ. Створюється фільтр Блума BF1 по первинному ключу z1 відфільтрованої таблиці T1 (будемо позначати її як T1 (p1)). Таблиця T2 (p2) додатково фільтрується по BF1 і з'єднується с T1 (p1), виходить таблиця T21 (p2p1). Далі створюється новий фільтр Блума BF2 по первинному ключу z2 таблиці T21 (p2p1). Попередній фільтр BF1 видаляється. Таблиця T3 (p3) додатково фільтрується по BF2 і з'єднується с T21 (p2p1). І т.д. Розглянемо з'єднання таблиць Ti1 (pipi-1 ... p1) і Ti + 1 (pi + 1pi ... p1), i = 1 ... K-1, T11 = T1.

За аналогією отримаємо оцінку обсягу shuffle для методу КІФБ (то, що пересилається на Reduce плюс фільтри Блума):

$$\sum_{I=1}^{K-1} V_I P_I + \sum_{I=1}^{K-2} V_{K,I} * \prod_{J=K}^{1} P_J + V_K \prod_{J=K}^{1} P_J + \sum_{I=1}^{K-1} V_{BLI}$$

Це обсяг всіх що беруть участь в з'єднанні таблиць, починаючи з T2, без обсягів фільтрів Блума (їх обсяг невеликий). Тобто ефект може бути більшим. Це важливо, тому що в задачах прийняття рішень частовзапрос включають багато таблиць.

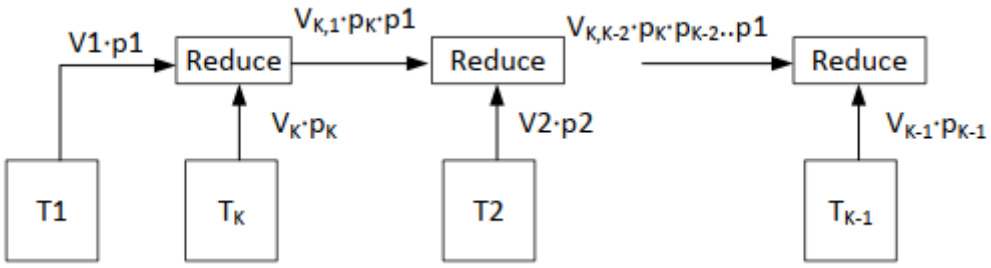

Рисунок 2.8 – Метод КІФБ

На рисунку 2.9 представлений процес обробки запиту Q3 в середовищі Spark, отриманий після аналізу результатів роботи монітора.

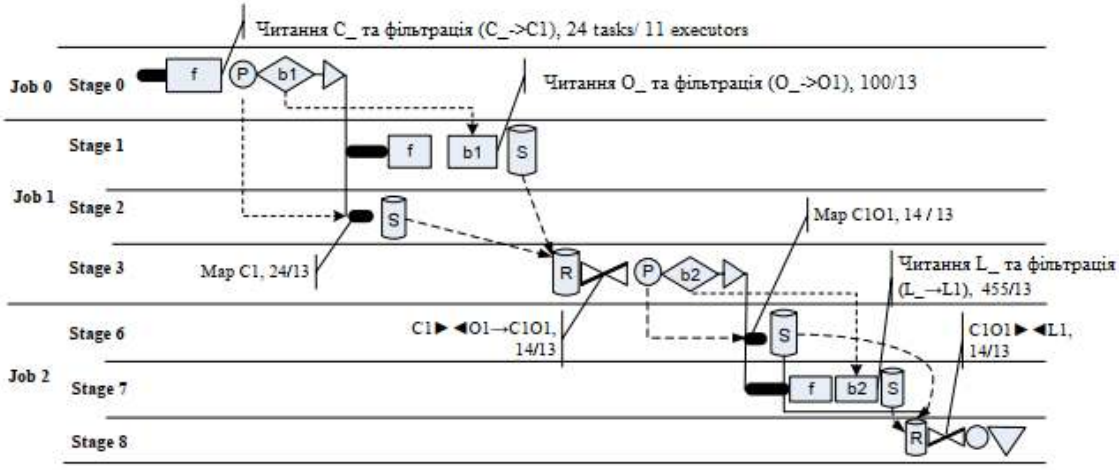

Рисунок 2.9 – Процес виконання запиту Q3 у середовищі Spark

З рисунка 2.10 випливає, що можна виділити два куща з'єднання таблиць. Перший кущ пов'язаний зі стадією Stage 3. Тут виконується

з'єднання проміжних таблиць (C1 і O1), отриманих в результаті обробки вихідних таблиць C_ і O_ (стадії Stage 0 і Stage 1). Перед з'єднанням таблиці C1 і O1 серіалізуются, зберігаються на локальних дисках вузлів (операції S), а потім виконується їх «перетасування (shuffle – читання з дисків і передача їх по мережі - операція R на рисунку 2.10).

Другий кущ пов'язаний зі стадією Stage 8. На цій стадії виконується з'єднання проміжних таблиць (C1O1 і L1), отриманих в результаті з'єднання C1 і O1 і обробки вихідної таблиці L_ (стадії Stage 3 і Stage 7). Тут також виконується «перетасування» з'єднуються таблиць. Були раніше наведені запит Q17 (з корельованим підзапитом) з тіста TPC-H (1-й стовпець) і його декомпозиція на підзапити (пошук в одній таблиці) і з'єднання результатів виконання підзапитів (пошук в двох таблицях) (2-й стовпець). Візуалізація цих розбиття була продемонстрована на рисунок 2.5.

Розроблена вартісна модель процесів виконання запитів в Spark, основу якої складають подмодели пов'язаних паралельних процесів. У цій моделі враховується можливість використання при реалізації запитів додаткових конструкцій: фільтрів Блума і дублювання невеликих таблиць по вузлах системи. Модель дозволяє прогнозувати час виконання запитів в розподіленої паралельної системі, коли потрібна оцінка часу (в рамках сервісу DaaS) або коли будується оптимальний план реалізації запиту на етапі проектування.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Для проведення експериментального порівняння розробленого методу виконання запитів до сховища даних (ХД) з каскадним використанням фільтра Блума (КІФБ) і методу без використання фільтра Блума (БІФБ) було розгорнуто віртуальний кластер. Кластер складався з 8 вузлів, на одному з яких були інстальовані керуючі служби HDFS, Hive, Spark 2, Yarn.

Основні характеристики кожного віртуального вузла були наступними: один двоядерний процесор, 8 GB оперативної пам'яті, 200 GB SSD диск, ОС Ubuntu 14.16. Як дистрибутива Hadoop використовувалася Cloudera. Так як кластер для експерименту було розгорнуто у віртуальному середовищі, час виконання запиту є випадковим і може відрізнятися від експерименту до експерименту. Але тут важливі не абсолютні значення, а ставлення часу для різних методів реалізації запитів.

Обсяги переданих даних і число записів залежать від значення параметра SF наповнення бази даних і не залежать від того, чи є вузли віртуальними або фізичними. Експерименти проводилися для запитів Q3 (поєднання трьох таблиць) розглянутих в попередньому розділі. Схеми виконання цих запитів за структурою однакові: вони включають підзапити (select), з'єднання (join) і фільтри Блума.

Таблиця 3.1 – Характеристики таблиць

Назва таблиці	Кількість записів
CUSTOMER	SF*150000
ORDERS	SF*1500000
LINEITEM	SF*6001215
PART	SF*200000

Варто врахувати, що в Hadoop/Spark використовується ефективний механізм стиснення даних. Фактичні параметри стиснення таблиць, використовуваних в експериментальних запитах Q3 і Q17, представлені на рисунку 3.1.

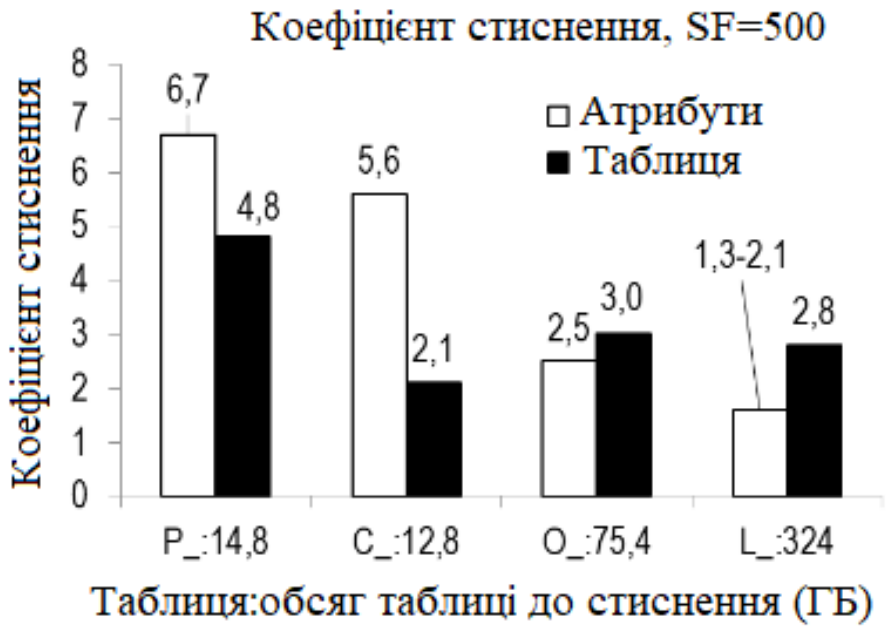

Рисунок 3.1 – Експериментальні показники стиснення таблиць тесту TPC-H

Варто врахувати, що в Hadoop / Spark використовується ефективний механізм стиснення даних. Фактичні параметри стиснення таблиць, використовуваних в експериментальних запиті Q3, представлені на рисунку 3.1.

Запит Q3 був реалізований за допомогою методу КІФБ (запит був приведений раніше в Таблиці 2, стовпець 1) і без використання фільтра Блума (БІФБ) в середовищі Spark SQL. На рисунку 3.2 представлена програма на мові Scala, що реалізує специфікації, представлені в Таблиці 3.1, стовпець 2.

1	import java.util.Calendar import breeze.util.BloomFilter import org.apache.spark.storage.StorageLevel import org.apache.spark.sql.functions
2	val sqlContext = new org.apache.spark.sql.SQLContext(sc) sqlContext.setConf("spark.sql.shuffle.partitions", "14")
3	val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc) hiveContext.setConf("spark.sql.orc.filterPushdown", "true") hiveContext.sql("use tpch_flat_orc_100")
4	val time = Calendar.getInstance().getTime()
5	val customer = hiveContext.sql("SELECT c_custkey FROM customer where c_mktsegment = 'BUILDING'").persist(StorageLevel.MEMORY_AND_DISK)
6	val bloom1 = customer.stat.bloomFilter($"c_custkey", 15000000, 0.01) val broadbloom1 = sc.broadcast(bloom1)
7	val orders = hiveContext.sql("SELECT o_custkey as c_custkey, o_orderkey as a_id, o_orderdate, o_shippriority FROM orders where o_orderdate< '1995-03-15'").filter(line=>broadbloom1.value.mightContain(line(0)))
8	val join1 = customer.join(orders,Seq("c_custkey")).persist(StorageLevel.MEMORY_AND_DISK)
9	val bloom2 = join1.stat.bloomFilter($"a_id", **15000000**, 0.01) val broadbloom2 = sc.broadcast(bloom2)
10	val lineitem = hiveContext.sql("SELECT l_orderkey as a_id, l_extendedprice, l_discount FROM lineitem where l_shipdate> '1995-03-15'").filter(line=>broadbloom2.value.mightContain(line(0)))
11	val join2 = lineitem.join(join1,Seq("a_id"))
12	val join2_r = join2.withColumn("revenue", (-join2("l_discount")+1)*join2("l_extendedprice")) val aggr = join2_r.groupBy("a_id","o_orderdate","o_shippriority").agg(sum("revenue")) aggr.orderBy(desc("sum(revenue)"),asc("o_orderdate")).show(10) aggr.persist(StorageLevel.MEMORY_AND_DISK)
13	val time2 = Calendar.getInstance().getTime()

Рисунок 3.2 – Текст програми-драйвера

Нижче наведено короткий опис операторів програми, наведеної на рисунку 3.2:

- 1 – підключення зовнішніх бібліотек;
- 2 – установка числа завдань reduce, використовуваних при з'єднанні таблиць, рівним 14 (7 робочих вузлів х 2 ядра);
- 3 – підключення бази даних TPC-H с необхідним коефіцієнтом наповнення SF;
- 4 – збереження часу початку виконання запиту Q3;
- 5 – виконання підзапиту C1 (метод persist дозволяє в подальшому

посилатися на побудований набір RDD customer без повторного його створення);

- 6 – побудова фільтра Блума BF1 (змінна blooml) і створення широкомовної змінної broadblooml для автоматичної розсилки фільтра Блума BF1 виконавчим процесам (executor);

- 7 – виконання підзапиту O1 і фільтрація записів за допомогою фільтра Блума BF1;

- 8 – з'єднання таблиць підзапитів C1 і O1 після фільтрації, щоб виключити записи, пов'язані з хибнопозитивним спрацьовуванням фільтра Блума [0];

- 9 – побудова фільтра Блума BF2 (змінна bloom2) і створення широкомовної змінної broadbloom2 для розсилки фільтра Блума;

- 10 – виконання підзапиту L1 і фільтрація записів за допомогою фільтра Блума BF2;

- 11 – з'єднання таблиць C1O1 і L1 по атрибуту a_id після фільтрації;

- 12 – реалізація операцій group by і order by;

- 13 – збереження часу закінчення виконання запиту Q3.

Розроблений методу доступ до сховища даних з каскадним використанням фільтра Блума (КІФБ) порівнювався з методом реалізації запитів в Spark SQL без використання фільтра Блума (БІФБ). Доступ до сховища даних без використання фільтра Блума також проілюструємо на прикладі запиту Q3 з тіста TPC-H.

Далі представлена програма на мові Scala, що реалізує запит Q3.

- 1 – підключення зовнішніх бібліотек;

- 2 – установка числа завдань reduce, використовуваних при з'єднанні таблиць, рівним 14;

- 3 – підключення бази даних TPC-H с необхідним коефіцієнтом наповнення SF;

- 4 – збереження часу початку виконання запиту Q3;

- 5 – виконання початкового запиту Q3 (вихідний запит Select цілком кодується як рядок при виклику методу hiveContext.sql);

- 6 – збереження часу закінчення виконання запиту Q3. Розглянемо результати виконання запиту Q3 з використанням методу БІФБ.

На рисунку 3.4 наведені стадії (stage) БІФБ і час їх реалізації в Spark для SF = 500 (це відповідає приблизно 400 Гб оброблюваних даних). Знаком позначена операція з'єднання таблиць.

1	import java.util.Calendar import breeze.util.BloomFilter import org.apache.spark.storage.StorageLevel import org.apache.spark.sql.functions
2	val sqlContext = new org.apache.spark.sql.SQLContext(sc) sqlContext.setConf("spark.sql.shuffle.partitions", "14")
3	val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc) hiveContext.setConf("spark.sql.orc.filterPushdown", "true") hiveContext.sql("use tpch_flat_orc_100")
4	val time = Calendar.getInstance().getTime()
5	val customer = hiveContext.sql("SELECT l.l_orderkey,sum(l.l_extendedprice*(1-l.l_discount)) as revenue,o.o_orderdate,o.o_shippriority FROM customer c, orders o, lineitem l where o.o_custkey=c.c_custkey and l.l_orderkey=o.o_orderkey and c_mktsegment = 'BUILDING' and o_orderdate< '1995-03-15' and l_shipdate> '1995-03-15' group by l.l_orderkey,o.o_orderdate,o.o_shippriority order by revenue desc, o.o_orderdate").persist(StorageLevel.MEMORY_AND_DISK) customer.show(10)
6	val time2 = Calendar.getInstance().getTime()

Рисунок 3.3 – Текст програми драйвера, що реалізує запит Q3 в Spark SQL без використання фільтра Блума

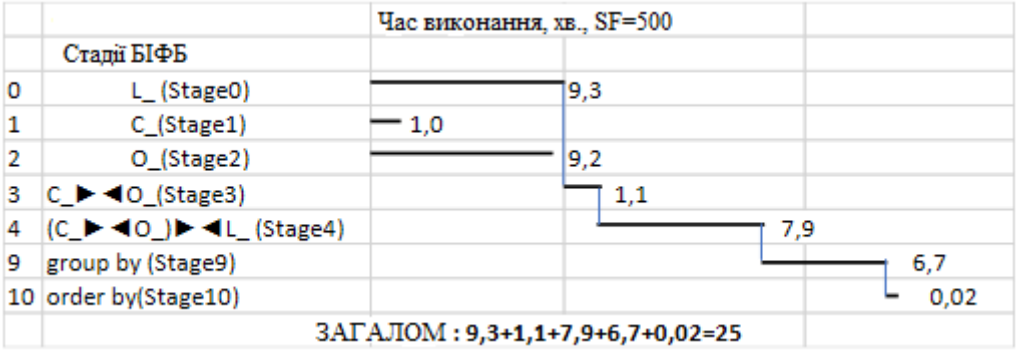

Рисунок 3.4 – Стадії БІФБ без використання фільтра Блума

Можна помітити, що читання вихідних таблиць L_, C_, O_ виконується паралельно. Час виконання запиту склало 25 хв. На рисунку 3.5 наведені характеристики виконання завдань (task) БІФБ за стадіями виконання (SF = 500). Виконавець (executor) – це віртуальна машина Java (JVM), на якій виконуються завдання стадії. За замовчуванням число ядер CPU на один екземпляр виконавця дорівнює 1 (параметр spark.executor.cores).

На рисунку 3.5 стрілками показано, що читання вихідних таблиць (стадії 0 – 2) виконується паралельно на 13 виконавців (13 ядер CPU на 7 вузлах). Середнє число завдань, що виконуються одним ядром, в цьому випадку дорівнює (455 + 24 + 100) / 13 = 44,5. З аналізу квантилів часу виконання одного завдання слід, що завдання стадій 0 – 2 мають довгий правий хвіст функції розподілу, а завдання стадій 4 і 9 – довгий лівий хвіст.

Можна також зауважити, що «збір сміття» і запис на диск перед «перетасуванням» (Shuffle Write) не роблять істотного впливу на час виконання запиту.

0	455	13	44,5	0,2	8	12	16	49	0,028	0,1
1	24	2		0,027	2	3	6	10	0,011	0,055
2	100	13		3	7	10	15	28	0,037	0,2
3	14	13	1,1	31	36	40	42	45	1	0,5
4	14	13	1,1	210	300	318	324	342	4	
9	14	13	1,1	144	264	276	282	288	6	0,028
10	1	1	1	1	1	1	1	1	0,026	

Рисунок 3.5 – Обсяги даних і число записів БІФБ (SF = 500)

На рисунку 3.6 наведені обсяги даних і число записів, 1) прочитаних з файлової системи HDFS (Input), 2) лічених з локальних дисків і переданих по віртуальній мережі на етапі «перетасовки» при виконанні завдань reduce (Shuffle Read), 3) записаних на локальні диски перед «перетасуванням» (Shuffle Write) усіма завданнями відповідних стадій. Стрілки, що показують складові відповідних обсягів Shuffle Read.

	Input	Shuffle Read	Shuffle Write
0	26,2/3 000 028 242		23,7/1 617 261 771
1	0,177/75 000 000		0,070/14 997 607
2	5,70/750 000 000		4,40/364 396 568
3		4,50/379 394 175	1,00/72 858 348
4		24,7/1 690 120 119	
9		24,7/1 690 120 119	0,109/5 658 551
10		0,006/337 532	

Рисунок 3.6 – Обсяги даних і число записів БІФБ (SF = 500)

Розглянемо результати виконання того ж запиту, але з використанням розробленого методу КІФБ. На рисунку 3.5 наведені стадії (stage) КІФБ і час їх реалізації в Spark для SF = 500, число елементів в фільтрі Блума N = 40 млн (ймовірність ложноположительного спрацьовування фільтра дорівнює P = 0,01).

На Stage2 запускаються завдання map, які сканують набір RDD «customer» (in MemoryTableScan RDD), вже створений на стадії StageO. При цьому читаються деякі дані з HDFS (0,079Гб / 1501). Ці завдання поміщають записи RDD «customer» на локальний диск (Shuffle Write) для подальшої «перетасовки» і з'єднання з таблицею O_ на стадії Stage3. Завдання стадій Stage 1 і Stage2 виконуються паралельно. Аналогічні дії Spark виконує на стадії Stage6 з таблицею C1O1. Завдання стадій Stage6 і Stage7 виконуються паралельно. Час виконання запиту склало 11 хв, що в 25/11 = 2,3 рази менше, ніж без використання фільтра Блума.

На рисунку 3.7 показано вплив параметра N (число елементів, млн) фільтра Блума на показники виконання запиту P3 з КІФБ.

Обсяг Shuffle Read (Shuffle Write) істотно зменшується при збільшенні N до 30 млн для SF = 250 і до 40 млн для SF = 500. Подальше зростання N не приводить до суттєвого зменшення обсягу. Це пов'язано зі скороченням числа хибнопозитивних спрацьовувань фільтра Блума. Час, показаний на

рисунок 3.7, є випадковим (позначається, що кластер розгорнуто у віртуальному середовищі). Але при SF = 500 спостерігається тенденція до зменшення часу виконання запиту до N = 40 млн.

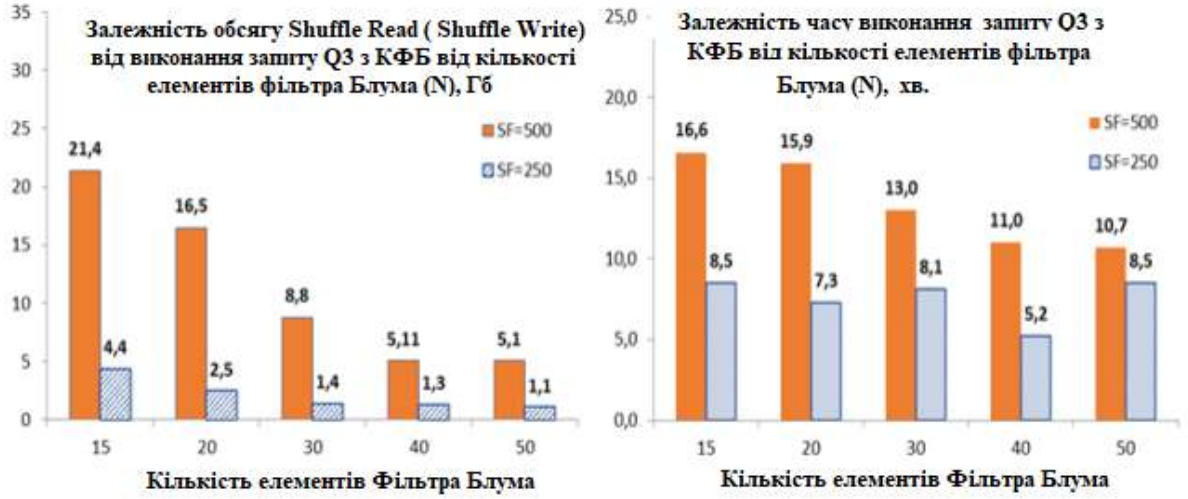

Рисунок 3.7 – Залежність обсягу ShuffleRead (ShuffleWrite)

В даному експерименті вплив N на час виконання запиту нижче, ніж на обсяг даних «перетасовки»: для SF = 500 при збільшенні N с 15 млн до 40 млн обсяг зменшився в 21,4 / 5,11 = 4,2 рази, а час - тільки в 16,6 / 11 = 1,5 рази. Це пов'язано з тим, що мережа між вузлами є віртуальною, тобто «швидкою» (деякі вузли розташовуються на одному фізичному сервері).

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведено порівняльний аналіз методів обробки та зберігання даних в розподілених системах. Також були розглянуті методи доступу до даних в паралельних розподілених сховищах даних на платформі MapReduce/Spark. Розроблен модифікований метод обробки даних та виконання запитів до паралельного розподіленого сховища даних на базі технології MapReduce/Spark з використанням фільтру Блума.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Курченко С.І., Климова І.М. Методи обробки даних в розподілених системах // Проблеми інформатизації : одинадцята міжнародна науково-технічна конференція. Черкаси – Баку– Харків – Бельсько-Бяла, 2023, т.1, с.84.

2. Варшавский П.Р., Еремеев А.П. Моделирование рассуждений на основе прецедентов в интеллектуальных системах поддержки принятия решений // Искусственный интеллект и принятие решений. 2009. № 2. С. 45-47.

3. Варшавский П.Р., Еремеев А.П. Методы правдоподобных рассуждений на основе аналогий и прецедентов для интеллектуальных систем поддержки принятия решений// Новости искусственного интеллекта. - 2006. - №3. С. 39-62.

4. Финн В.К. Об интеллектуальном анализе данных // Новости искусственного интеллекта, №3, 2004, С. 3-19.

5. W. Frawley, G. Piatetsky-Shapiro, C. Matheus Knowledge Discovery in Databases: An Overview. - AI Magazine. - 1992. pp. 213-228.

6. Kitchin Rob. The Data Revolution. United States: Sage. 2014, p. 6.

7. Piatetsky-Shapiro G, Frawley W J. Knowledge Discovery in Databases. USA: MIT Press, 1991.

8. Agrawal R., Mannila H., Srikant R., Toivonen H. and Verkamo I. Fast Discovery of Association Rules. In Advances in Knowledge Discovery and Data Mining, eds. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Menlo Park, Calif.: AAAI Press, 1996, pp. 307-328.

9. Fayyad U., Piatetsky-Shapiro G., Smyth P., Advances in Knowledge Discovery and Data Mining, (Chapter 1), AAAI/MIT Press, 1996.

10. Барсегян А.А., Куприянов М.С., Степаненко В.В., Холод И.И. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP.

// 2-еизд., - СПб: БХВ-Петербург, 2007.

11. А.А. Барсегян, И.И. Холод, М.Д. Тесс, М.С. Куприянов, С.И. Елизаров. Анализ данных и процессов. 3-е изд. – СПб.: БХВ-Петербург, 2009.

12. Интеллектуальный анализ данных средствами MS SQL Server 2008 - [Электронный ресурс].

13. Data Mining – технология добычи данных – [Электронный ресурс]. URL: http://bourabai.ru/tpoi/datamining.htm: (дата обращения: 05.01.2017).

14. Дюк В.А., Самойленко А.П. Data Mining: учебный курс СПб.: Питер, 2001.

15. Чубукова И.А. Data Mining, БИНОМ. Лаборатория знаний, Интернет- университет информационных технологий - ИНТУИТ.ру, 2006.

16. Филипов В.А. Интеллектуальный анализ данных: методы и средства. М.: Эдиториал УРСС, 2001.

17. Дюран Б., Оделл П. Кластерный анализ. М.: Статистика, 1977.

18. А.Н.Тихонов, В.Я.Арсенин. Методы решения некорректных задач. Наука, Москва, 1974.

19. Judea Pearl, Stuart Russell. Bayesian Networks. UCLA Cognitive Systems Laboratory, Technical Report (R-277), November 2000.

20. Ферстер Э., Ренц Б. Методы корреляционного и регрессионного анализа = Methoden der Korrelation - und Regressiolynsanalyse. – М.: Финансы и статистика, 1981.

21. Управленческий учет: учебник / под ред. А.Д. Шеремета. 4-е изд. – М.: ИНФРА-М, 2009.