Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод та засоби бенчмаркінга мультимодельної бази даних

(тема)

Виконав:

студент II курсу, групи СПм-20-2

Хомич В.М.
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Можаєв О.О.
(посада, прізвище, ініціали)

Допускається до захисту

В.о. зав. кафедри ЕОМ _____ Волк М.О.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Хомичу Віктору Михайловичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод та засоби бенчмаркінга мультимодельної бази даних

затверджена наказом по університету від “ 24 ” березня 2022 р. № 413 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи 1) системи керування базами даних; 2) моделі даних; 3) генерація даних.

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз предметної області та постановка завдання;
2) модель даних і генерація даних в бенчмарку;
3) курування параметрів та оцінка бенчмарка;
4) системне вивчення та оцінка ефективності.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати – 14 арк. ф. А4

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	Аналіз проблеми та огляд існуючих рішень	28.03.22-03.04.22	
2	Вибір та обґрунтування методики дослідження	04.04.22-10.04.22	
3	Розробка генератора даних	11.04.22-20.04.22	
4	Розробка набору робочих навантажень	21.04.22-28.04.22	
5	Проведення експериментів	29.04.22-05.05.22	
6	Оформлення матеріалів кваліфікаційної роботи	06.05.22-13.05.22	
7	Подання кваліфікаційної роботи керівникові та	14.05.22-15.05.22	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	16.05.22-17.05.22	
	вання		

Дата видачі завдання 28 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Можаєв О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 86 с., 14 рис., 3 табл., 2 дод., 14 джерел.

БЕНЧМАРКІНГ, МУЛЬТИМОДЕЛЬНА БАЗА ДАНИХ, ЗАПИТ, ПАРАМЕТР, РОБОЧЕ НАВАНТАЖЕННЯ.

Метою кваліфікаційної роботи є розробка методу та засобів бенчмаркінга мультимодельної бази даних.

У ході виконання кваліфікаційної роботи розроблено новий генератор даних, а також реалізовано генератор поверх Spark SQL. Розроблено набір робочих навантажень із кількома моделями. Визначено нову проблему під назвою «мультимодельне курування параметрів», спрямовану на введення різноманітності запитів із підпорядкованими параметрами до робочих навантажень. Запропоновано алгоритм на основі зразка для розумного вибору параметрів для запиту бенчмаркінга. Проведено комплексну оцінку чотирьох мультимодельних баз даних і аналітично прозвітовано про порівняння ефективності та отримані дані. Результатом виконання поставлених завдань став бенчмарк мультимодельної бази даних, який складається зі змішаної моделі даних, масштабованого мультимодельного генератора даних і набору робочих навантажень, включаючи мультимодельну агрегацію, об’єднання та транзакцію.

ABSTRACT

Master’s thesis: 86 pages, 14 figures, 3 tables, 2 appendices, 14 sources.

BENCHMARKING, MULTIMODEL DATABASE, QUERY, PARAMETER, WORKLOAD.

The major goal of this thesis is to develop a method and tools for multimodel database benchmarking.

During the qualification work, a new data generator was developed, and a generator on top of Spark SQL was implemented. A set of multimodel workloads has been developed. A new problem called «multimodel parameter control» is identified, aimed at introducing the query diversity with curated parameters to the workloads. A sample-based algorithm for a reasonable choice of parameters for a benchmarking query is proposed. A comprehensive evaluation of four multimodel databases was performed and comparisons of efficiency and obtained data were analytically reported. The task resulted in a multimodel database benchmark consisting of a mixed data model, a scalable multimodel data generator, and a set of workloads, including the multimodel aggregation, join and transaction.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ММБД – мультимодельна база даних

CLV – пожиттєва цінність клієнта (англ., Customer Lifetime Value)

LHS – латинська вибірка гіперкуба (англ., Latin Hypercube Sampling)

RDD – стійкий розподілений набір даних (англ., Resilient Distributed Dataset)

LPG – мічений граф властивостей (англ., Labeled Property Graph)

ВСТУП

Різноманітність даних є однією з найскладніших проблем для досліджень і практики в системах управління даними. Дані природно організовані в різних форматах і моделях, включаючи структуровані дані, напівструктуровані дані та неструктуровані дані. Оскільки все більше компаній усвідомлюють, що дані в усіх формах і розмірах мають вирішальне значення для прийняття найкращих можливих рішень, спостерігається постійне зростання систем, які підтримують величезний обсяг реляційних або нереляційних форм даних. На відміну від традиційних систем керування базами даних, які організовані навколо єдиної моделі даних, яка визначає, як дані можуть бути організовані, збережені й маніпульовані, ММБД розроблена для підтримки кількох моделей даних на одному інтегрованому сервері. Наприклад, документ-орієнтовані, графові, реляційні та ключ-значення є прикладами моделей даних, які можуть підтримуватися мультимодельною базою даних. Ніщо не показує картину так яскраво, як квадрант Gartner Magic для операційних систем керування базами даних, який передбачає, що всі провідні операційні СКБД пропонуватимуть кілька моделей даних, реляційних і NoSQL, в одній платформі СКБД. Наявність єдиної платформи даних для керування як добре структурованими даними, так і даними NoSQL є вигідним для користувачів, позаяк такий підхід значно зменшує проблеми інтеграції, міграції, розробки, обслуговування та експлуатації.

Бенчмаркінг є загальноприйнятою практикою для оцінки систем баз даних, позаяк все більше і більше платформ пропонуються для роботи з мультимодельними даними. Тому стає важливим мати бенчмарки, які можна використовувати для оцінки продуктивності та зручності використання наступного покоління мультимодельних систем баз даних. Існує ряд бенчмарків, які можна використовувати для оцінки систем великих даних,

наприклад, YCSB, BigBench, TPCx-BB, Bigframe. На жаль, ці загального призначення бенчмарки великих даних не призначені для оцінки мультимодельних баз даних. Ретельна оцінка мультимодельних систем баз даних ставить перед собою кілька нових проблем, які необхідно подолати. По-перше, ввід і вивід наявних мультимодельних баз даних досить різноманітний. Оскільки стандартної мультимодельної мови запитів зараз немає, загальнодоступні реалізації даних бенчмаркінга та запитів для різних систем слід розробляти, спільно використовувати, уніфікувати та оптимізувати. По-друге, на відміну від реляційного світу, системи NoSQL дотримуються парадигми «спочатку дані, схема пізніше або ніколи». Для ретельної оцінки має бути можливість контролювати (і систематично змінювати) вхідну схему та складність еволюції схеми для мультимодельних даних. Бенчмарк повинен підвищувати продуктивність, дозволяючи створювати багато мультимодельних даних із різноманітною схемою, використовуючи невеликі ручні зусилля. Нарешті, мультимодельні бази даних повинні підтримувати міжмодельну транзакцію та узгодженість. Тому нові метрики узгодженості, які описують поведінку узгодженості для різних моделей даних, повинні бути запропоновані точним чином.

Саме тому метою даної кваліфікаційної роботи є розробка бенчмарка мультимодельної бази даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Характеристика предметної області

Позаяк дані різних типів і форматів мають вирішальне значення для прийняття оптимальних бізнес-рішень, зараз можна спостерігати значне зростання вимог до аналізу та маніпулювання мультимодельними даними, включаючи структуровані, напівструктуровані та неструктуровані дані. Зокрема, структуровані дані включають реляційні дані, дані ключ/значення та графи. Напівструктуровані дані зазвичай посилаються на документи XML і JSON. Неструктуровані дані, як правило, є текстовими файлами, що містять дати, цифри та факти.

ММДБ – це нова тенденція для системи керування базами даних, яка використовує єдину платформу для керування даними, що зберігаються в різних моделях, таких як документ-орієнтованих, графових, реляційних та ключ-значення. У порівнянні з технологією багатоваріантної персистентності (Polyglot Persistence) [1], яка використовує окремі сховища даних для задоволення різноманітних випадків використання, ММБД вважається наступним поколінням системи керування даними, що включає гнучкість, масштабованість та узгодженість. Вона корисна для сучасних додатків, які вимагають роботи з неоднорідними джерелами даних, одночасно охоплюючи гнучку розробку. Наприклад, підприємства часто отримують бізнес-ідею, інтегруючи графи з соціальних мереж, документи з історії покупок і таблиці з інформації про клієнтів. Дослідники даних зазвичай пишуть сценарії для кожної моделі даних окремо, а потім об’єднують їх у єдину форму, щоб продовжити аналіз у реальному часі та OLAP. Однак зі збільшенням масштабу та складності даних такий метод стає виснажливим та неефективним. Використовуючи потужність мультимодельної бази даних, можна легко приймати й аналізувати різнорідні дані в режимі реального часу,

а отже, швидко коригувати операційну стратегію.

Бенчмаркінг баз даних стає важливим інструментом для оцінки та порівняння СКБД з моменту появи бенчмарка Wisconsin на початку 1980-х років. Відтоді науковці та промисловість запропонували багато бенчмарків баз даних для різних оцінок цілі, наприклад TPC-C для РСКБД, TPC-DI [2] для інтеграції даних, OO7 [3] для об’єктноорієнтованих СКБД і бенчмарки XML для XML СКБД. Пізніше рух NoSQL і великих даних наприкінці 2000-х приніс наступне покоління бенчмарків, таких як YCSB [4] для хмарних систем обслуговування, LDBC [5] для RDF-графових СКБД, BigBench [6] для систем великих даних. Однак ці бенчмарки загального призначення або мікробенчмарки не призначені для мультимодельної бази даних. Позаяк все більше і більше платформ пропонуються для роботи з кількома моделями даних, стає важливим мати еталон для оцінки продуктивності мультимодельної бази даних та порівняння різних мультимодельних підходів.

1.1.1 Історія розвитку мультимодельної бази даних

Для розуміння самої ідеї мультимодельної бази даних потрібна невелика історична довідка про те, як усе починалося та розвивалося. Перша зміна – це та зміна, яка здійснила перехід від централізованих розгортань баз даних до децентралізованих. Сьогодні ця зміна продовжує розгортання, яка радикально розподілена. Остання зміна відбулася як прямий результат обов'язкових вимог для хмарних програм (наприклад, писати, читати, розповсюджувати, залишатися активними скрізь і завжди). Наступні дві зміни стосуються концепції збереження декількох моделей, які стосуються практично використання кількох технологій зберігання даних для підтримки однієї або декількох програм.

Спочатку ідея збереження багатогранності виникла через необхідність розділити різні робочі навантаження на керування даними, щоб не було

конкуренції за дані або обчислювальні ресурси між системами, призначеними для транзакцій, аналітики, пошуку та інших робочих навантажень. Це почалося з того, що різні програми були призначені для окремих сховищ даних, що було добре для старих додатків. Однак хмарні програми вимагали, щоб база даних розвивалася для консолідації робочих навантажень на єдиній платформі для підтримки ідей аналітичної обробки гібридних транзакцій (HTAP), також згадуваних іншими іменами, такими як translytics.

Третя зміна нагадує другу, але включає різні моделі даних і робочі навантаження. Прогрес розпочався переважно з моделі даних RDBMS (Relational DataBase Management System), що переміщується в суміш різних технологій та моделей даних, щоб тепер створити єдину платформу, здатну підтримувати декілька моделей даних проти одного інтегрованого бекенду.

Завдяки останнім двом змінам ціль для хмарних додатків стала у забезпеченні як змішаного робочого навантаження, так і можливостей декількох моделей на одній платформі.

Модель реляційних даних стала популярною після її публікації Едгаром Коддом у 1970 році. Через наростальні вимоги до горизонтальної масштабованості та відмовостійкості після 2009 року стали популярні бази даних NoSQL. Бази даних NoSQL використовують безліч моделей даних із документами, графами та іншими популярними моделями.

ММДБ – це база даних, яка може зберігати, індексувати та запитувати дані у кількох моделях. Протягом деякого часу бази даних переважно підтримували лише одну модель, таку як реляційна база даних, документно-орієнтована база даних, база даних графів або потрійне сховище. База даних, яка об'єднує кілька їх, є мультимодельною.

Протягом деякого часу вважалося, що, крім реляційної, не існують інші моделі баз даних. Реляційна модель та поняття третьої нормальної форми були стандартом для всіх сховищ даних. Однак до початку домінування моделювання реляційних даних з 1980 по 2005 рік зазвичай

використовувалася ієрархічна модель бази даних, і з 2000 по 2010 були популярні багато NoSQL моделей, які не були реляційними (документно-орієнтовані моделі, потрійні сховища, сховища ключів і бази даних графів). Можливо, геопросторові дані, часові дані та текстові дані також є окремими моделями, хоча індексовані текстові дані із запитом зазвичай називаються «пошуковою системою», а не базою даних.

Вперше термін «мультимодель» був пов'язаний із базами даних 30 травня 2012 року в Кельні, Німеччина, під час ключової замітки Луки Гаруллі «Прийняття NoSQL – що далі?». Він передбачав еволюцію продуктів NoSQL 1-го покоління в нові продукти з великою кількістю функцій, які можна використовувати у кількох варіантах.

Ідея мультимодельних баз даних може бути простежена до систем керування об'єктно-реляційними даними (ORDBMS) на початку 1990-х років та у ширшому контексті навіть для інтегрованих СКБД на початку 1980-х років. Система ORDBMS керує різними типами даних, такими як реляційні, об'єктні, текстові та просторові шляхом підключення типів даних, функцій і реалізацій конкретних доменів в ядрі СКБД. База даних з кількома моделями є найпрямішою відповіддю на підхід зв'язування декількох продуктів баз даних, кожен з яких передає іншу модель, щоб досягти можливостей декількох моделей, як описано Мартіном Фаулером. Ця стратегія має два основних недоліки: це призводить до значного збільшення оперативної складності, і немає підтримки узгодженості даних в окремих сховищах даних, тому мультимодельні бази даних почали заповнювати цю прогалину.

Деякі історії найскладніших систем від непотрібної інтеграції «frankenbeans» знаходяться в Інтернеті.

Мультимодельні бази даних створені, щоб запропонувати переваги моделювання багатогранності на основі моделювання, без його недоліків. Складність роботи скорочується з допомогою використання одного сховища даних.

1.1.2 Архітектура мультимодельних баз даних

Основна різниця між доступними мультимодельними базами даних пов'язана з їх архітектурою. Мультимодельні бази даних можуть підтримувати різні моделі або в рушії, або через різні шари поверх рушія. Деякі продукти можуть надати рушій, який підтримує документи та графи, в той час як інші надають шари поверх сховища ключів. З багаторівневою архітектурою, кожна модель даних надається через свій власний компонент.

В екосистемі РСКБД представлено кілька ключових характеристик, таких як: агностична стандартна мова постачальника для розробників (SQL); чіткий поділ між логічними (розробниками) та фізичними (DBA) аспектами СКБД; та згуртований набір механізмів на різних мовах (драйверах), за допомогою яких програми можуть взаємодіяти з базами даних. Це дозволяє підприємствам впроваджувати технології інфраструктури даних, не переймаючись блокуванням постачальника.

Однак, хоч наведені вище характеристики добре служать для централізованих додатків, існують перешкоди для його застосування в хмарних додатках:

- архітектура master-slave вимагає поступки щодо часу безвідмовної роботи та відмовостійкості;
- масштабування, запис та розподіл обмежень для робочих хмарних додатків;
- суворе дотримання конструкцій логічного рівня даних, таких як третя нормальна форма (3NF), які оцінюють ефективність зберігання більше, ніж гнучкість програми;
- жорстка модель даних, яка робить використання напів- та неструктурованих даних надзвичайно дорогою у масштабуванні;
- облицювальна архітектурна Band-Aid, яка збільшує операційні витрати експоненційно.

Хоч деякі технології NoSQL, такі як Apache Cassandra, торкаються

раніше згаданих проблем екосистеми РСКБД, рух NoSQL створив фрагментовану технологічну пропозицію для корпоративних клієнтів через дві проблеми, розглянуті нижче.

По-перше, сталість мультимоделі передбачало, що клієнти використовують обмежений набір однієї з моделей (Key Value, Tabular, JSON, Graph), або виконують операції екстракції, перетворення, навантаження в сховищах даних.

По-друге, кожен механізм постачальника NoSQL для взаємодії зі сховищем даних був іншим, як щодо його діалекту, так і там, де вони лежали на логічному/фізичному рівні. Це змусило розробників додатків писати рівні абстракції, якщо їм потрібно більше однієї моделі в їхньому додатку. Крім того, ці рівні абстракції повинні були працювати на різному рівні за фізичним/логічним спектром, щоб підтримувати розробку додатків.

Бази даних з декількома моделями є наступним етапом розвитку для індустрії NoSQL, метою якого є подолання перешкод для прийняття, особливо в тому, що стосується основних розробників та адміністраторів на підприємстві. Це досягається шляхом підтримки кількох моделей даних проти одного інтегрованого бекенду. Така платформа з безліччю моделей має демонструвати:

- підтримку більш ніж однієї постреляційної моделі даних (наприклад, табличного, JSON, графа) на логічному рівні для спрощення розробки для розробників додатків;
- забезпечення того, щоб усі моделі були відкриті через когезійні механізми, тим самим уникаючи когнітивного контекстного перемикання для розробників;
- єдиний шар збереження, який забезпечує геонадлишкові, постійно доступні характеристики та загальну структуру для операційних аспектів, таких як безпека, надання ресурсів, аварійне відновлення тощо;
- розширення можливостей використання різних випадків використання OLTP та OLAP-робочих навантажень для підприємств бізнесу

в рамках підприємства для інновацій з маневровністю.

Найпростішою моделлю є ключові та табличні значення. Все в базі даних може бути досягнуто ключем, де значення можуть бути простими та складними типами. Підтримуються документи та елементи графа як значення, що дозволяють використовувати багатшу модель, ніж те, що зазвичай входить у класичну модель ключ/значення. Класична модель ключ/значення надає «відра» для групування пар ключ/значення у різних контейнерах.

Хоча її все ще можна використовувати як просте сховище даних з ключовими значеннями, вона пройшла довгий шлях від свого початкового коріння. У пізніших версіях підтримуються матеріалізовані уявлення та інші потужні функціональні можливості, які поруч із широким поширенням діалекту CQL представляють великі табличні сховища розробника додатків.

Конкретні приклади використання включають майже всі часові ряди даних (наприклад, IoT, керування користувачами, фінансовий потоковий розклад) та інші подібні важкі ситуації із записом та читанням. У порівнянні з реляційним дизайном таблиці зазвичай моделюються ненормалізованим способом навколо запитів, необхідних для задоволення запитів від додатку.

Дані моделі JSON зберігаються всередині документів. Документ являє собою набір пар ключ/значення (також називається полями або властивостями), де ключ дозволяє отримати доступ до його значення. Значення можуть мати примітивні типи даних, вбудовані документи або масиви інших значень. Документи зазвичай не змушують мати схему, яка може бути вигідною, оскільки вони залишаються гнучкими та легко змінюються. Документи зберігаються в колекціях, що дозволяє розробникам групувати дані в міру їхнього прийняття. Підтримка моделі JSON дозволяє гнучко зберігати дані зі складними вкладеними схемами та може легко переміщувати дані в рівні додатків і з них - обидва є основними прикладами використання популярних баз даних, орієнтованих на документи.

У деяких документах, що не містять схеми, один запис може значно

відрізнятися від наступного, не очікуючи, що база даних забезпечуватиме дотримання будь-якої структури записів. Однак ММБД зберігає вимогу про те, щоб таблиця CQL, що містить дані JSON, була визначена за допомогою схеми, яка реалізує стовпці та типи стовпців у рядку. Слід зазначити, що деякі постачальники баз даних документів тепер визнають необхідність застосування схеми даних JSON і впроваджують механізми для перевірки даних, що завантажуються в базу даних.

CQL підтримує типи колекцій (карти, набори та списки) і типи користувача, які всі зіставляються зі списками JSON і типами карт. Це дозволяє зберігати записи, які мають подібну структурну складність із документами, без повної форми. Розподілені сховища у базовому модулі зберігання також означають, що значення стовпців можуть бути опущені без штрафу, що додатково додає гнучку структуру запису.

Можна визначити стовпці, які можуть використовуватися лише з кількома рядками в таблиці, що відповідає можливості документа JSON залишати значення із запису. Також можна додавати стовпці в таблицю будь-коли без штрафу за продуктивність.

Якщо вимога полягає в тому, щоб мати справу з вхідною стрічкою даних, яка може змінюватися з часом, це може бути оброблено без змін витрат на схему таблиці. У цьому відношенні є багато можливостей зберігання документів у чистих документах, крім можливості обробляти дані, що дійсно не мають схеми, без попереднього знання структури вхідних записів.

Підтримка моделі даних графів реалізована через базу даних графів, створених для хмарних програм, яким необхідно керувати дуже складними та пов'язаними даними. Граф забезпечує безперервний час безперебійної роботи поряд з передбачуваною продуктивністю та масштабованістю для сучасних систем, що працюють зі складними та постійно змінними даними, залишаючись при цьому оперативно простим в керуванні.

Модель графа повинна розглядатись для будь-якого випадку

використання, що включає складні сценарії даних, які складаються з інтенсивних та численних взаємозв'язків між елементами даних.

Об’єктна модель була успадкована об'єктноорієнтованим програмуванням і підтримує успадкування між типами (підтипи розширює супертипи), поліморфізм, коли є намір посилання на базовий клас та пряме зв'язування з/на об'єкти, що використовуються в мовах програмування.

Оскільки дані з усіх моделей зберігаються в одному бекенді, кожна модель даних несе свої переваги:

- безперервна доступність;
- просте географічне розподілення даних;
- оперативна низька латентність;
- лінійна масштабованість;
- операційна завершеність;
- спрощена технологія.

Крім надання декількох моделей даних в одному сховищі даних, деякі бази даних дозволяють розробникам легко визначати моделі даних користувача. Ця можливість активується транзакціями ACID з високою продуктивністю та масштабованістю. Щоб користувацька модель даних підтримувала паралельні оновлення, база даних повинна мати можливість синхронізувати оновлення за кількома ключами. ACID транзакції, якщо вони є досить ефективними, дозволяють таку синхронізацію. JSON-документи, графи та реляційні таблиці можуть бути реалізовані так, щоб вони успадковували горизонтальну масштабованість та відмовостійкість базового сховища даних.

1.1.3 Наявні мультимодельні бази даних і їх порівняння

Порівняння мультимодельних баз даних наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння мультимодельних баз даних

Система	Мова запитів	Первинна модель	Вторинна модель	Стратегія зберігання
AgensGraph	OpenCypher, SQL	Реляційна	Граф, JSON	Один двигун
ArangoDB	AQL	JSON	Граф, ключ-значення	Один двигун
OrientDB	Подібна до SQL	Граф	JSON, ключ-значення	Один двигун
PostgreSQL	Розширена SQL	Реляційна	Всі, крім графа	Один двигун
Marklogic	Xpath	XML	JSON, RDF	Один двигун
MongoDB	API	JSON	Граф	Один двигун
Redis	API	Ключ-значення	Граф, JSON	Один двигун
Spark SQL	Подібна до SQL	DataFrame	Граф, JSON, ключ-значення	Один двигун
Datastax	CQL	Стовпчик	JSON, граф	Кілька двигунів
DynamoDB	API, SQL	–	JSON, граф, ключ-значення	Кілька двигунів
CosmosDB	API, SQL	–	Всі, крім XML	Кілька двигунів
Oracle 12c	Розширена SQL	Реляційна	Всі	Обидва

1.1.4 Поняття бенчмаркінга бази даних

Бенчмаркінг є загальноприйнятою практикою для оцінки та порівняння різних систем баз даних, якими керують постачальники баз даних і наукові

дослідники. Тим не менш, бенчмарки бази даних мають відмінні характеристики від специфікації, формату даних і робочого навантаження. Транзакційні бенчмарки, такі як TPC-C, зазвичай базуються на робочих навантаженнях обробки бізнес-транзакцій. Аналітичні бенчмарки, такі як TPC-H, TPC-DS, зосереджені на складних запитах, які передбачають сканування великих таблиць, об'єднання та агрегацію. TPC-DI має мультимодельні вхідні дані, включаючи XML, CSV та текстові частини, які використовуються для оцінки вартості перетворення в процесі інтеграції даних. Bigbench включає напівструктуровані (журнали) та неструктуровані дані (огляди) у структуровані дані TPC-DS і визначає складні запити, які спрямовані на аналітику великих даних з точки зору бізнесу та техніки. Що стосується бенчмарків NoSQL, бенчмарк YCSB призначений для вимірювання продуктивності різних розподілених «хмарних» систем баз даних із змішаними робочими навантаженнями, такими як операції читання-запису. XMark [7] пропонується для порівняння баз даних XML в обробці запитів XML, починаючи від відповідності рядків і закінчуючи запитами, що містять вирази ієрархічного шляху. Нещодавно було запропоновано бенчмарки LDBC і RBench [8], щоб підкреслити обхід графів і аналіз складних графів, зокрема, на графах властивостей і даних RDF. Однак жоден з них не підходить для бенчмаркінга мультимодельних баз даних через відсутність конкретних робочих навантажень і даних. Це викликає потребу в наскрізному бенчмаркові для обробки мультимодельних запитів.

1.2 Постановка задачі

Загалом, під час оцінки продуктивності мультимодельних баз даних необхідно вирішити три проблеми. Перше завдання – створити синтетичні мультимодельні дані. Існуючі генератори даних не можуть бути безпосередньо використані для оцінки мультимодельних баз даних, позаяк вони включають лише одну модель і моделюють конкретний сценарій. Друга

проблема полягає в розробці мультимодельних робочих навантажень. Такі робочі навантаження є основними операціями в багатьох складних і сучасних програмах. Проте їх вивченню приділено мало уваги. Третя проблема – оцінити вибірковість під час генерації запиту. Причина полягає в тому, що різні значення параметрів для одного шаблону запиту призведуть до різного часу виконання. Більше того, коли йдеться про мультимодельний контекст, це вимагає відповіді на два цікаві запитання: як охарактеризувати робочі навантаження щодо моделі даних і як ефективно вибирати параметри для цілісної оцінки.

Щоб розв’язати першу проблему, розробляємо новий генератор даних для надання корельованих даних у різноманітних моделях даних. Він моделює сценарій соціальної комерції, який поєднує соціальну мережу з контекстом електронної комерції. Крім того, пропонуємо трифазну систему для моделювання поведінки клієнтів у соціальній комерції. Ця структура складається з закупівлі, поширення-купівлі та повторної покупки, яка враховує різноманітні фактори для створення даних розподілу за силовим законом, які широко зустрічаються в реальному житті. Зокрема, пропонуємо нову імовірнісну модель під назвою CLVSC, щоб зробити дрібні прогнози на третій фазі. Для другого завдання спочатку моделюємо значущі бізнес-кейси в соціальній комерції, розділяючи їх на чотири шари: індивідуальний, розмовний, громадський і комерційний. Потім визначаємо набір мультимодельних запитів і транзакцій на основі техніки точки заглушення (англ., choke point technique), яка перевіряє слабкі місця баз даних, щоб зробити еталон складним і цікавим. Точки заглушення робочих навантажень створюваного бенчмарку включають продуктивність мультимодельної агрегації, приєднання та транзакції, що вимагає від бази даних визначати оптимальний порядок мультимодельного з’єднання, обробляти складні агрегації та одночасно гарантувати паралельність та ефективність. Для розв’язання третьої проблеми пропонуємо векторний підхід для представлення проміжних результатів з точки зору мультимоделей. Тоді

формалізуємо проблему як мультимодельне курування параметрів. Оскільки обчислення всіх проміжних результатів області параметрів є дорогим, розробляємо стратифікований метод вибірки для ефективного вибору параметрів для запиту.

Отже, для реалізації мети кваліфікаційної роботи необхідно вирішити наступні завдання:

- розробити новий генератор даних, який надає корельовані дані в різних моделях даних. Запропонувати нову структуру для генерування даних для моделювання поведінки клієнтів у соціальній комерції. Реалізувати генератор поверх Spark SQL, щоб забезпечити ефективність та масштабованість;

- розробити набір робочих навантажень із кількома моделями, включаючи десять запитів і дві транзакції з технічної та бізнес-точок зору;

- визначити нову проблему під назвою «мультимодельне курування параметрів», спрямовану на введення різноманітності запитів із підпорядкованими параметрами до робочих навантажень. Запропонувати алгоритм на основі зразка для розумного вибору параметрів для запиту бенчмаркінга;

- провести комплексну оцінку чотирьох мультимодельних баз даних: ArangoDB, OrientDB, AgensGraph і Spark SQL. Аналітично прозвітувати про порівняння ефективності та отримані дані.

Результатом виконання поставлених завдань повинен стати бенчмарк мультимодельної бази даних.

2 МОДЕЛЬ ДАНИХ І ГЕНЕРАЦІЯ ДАНИХ В БЕНЧМАРКУ

2.1 Модель даних

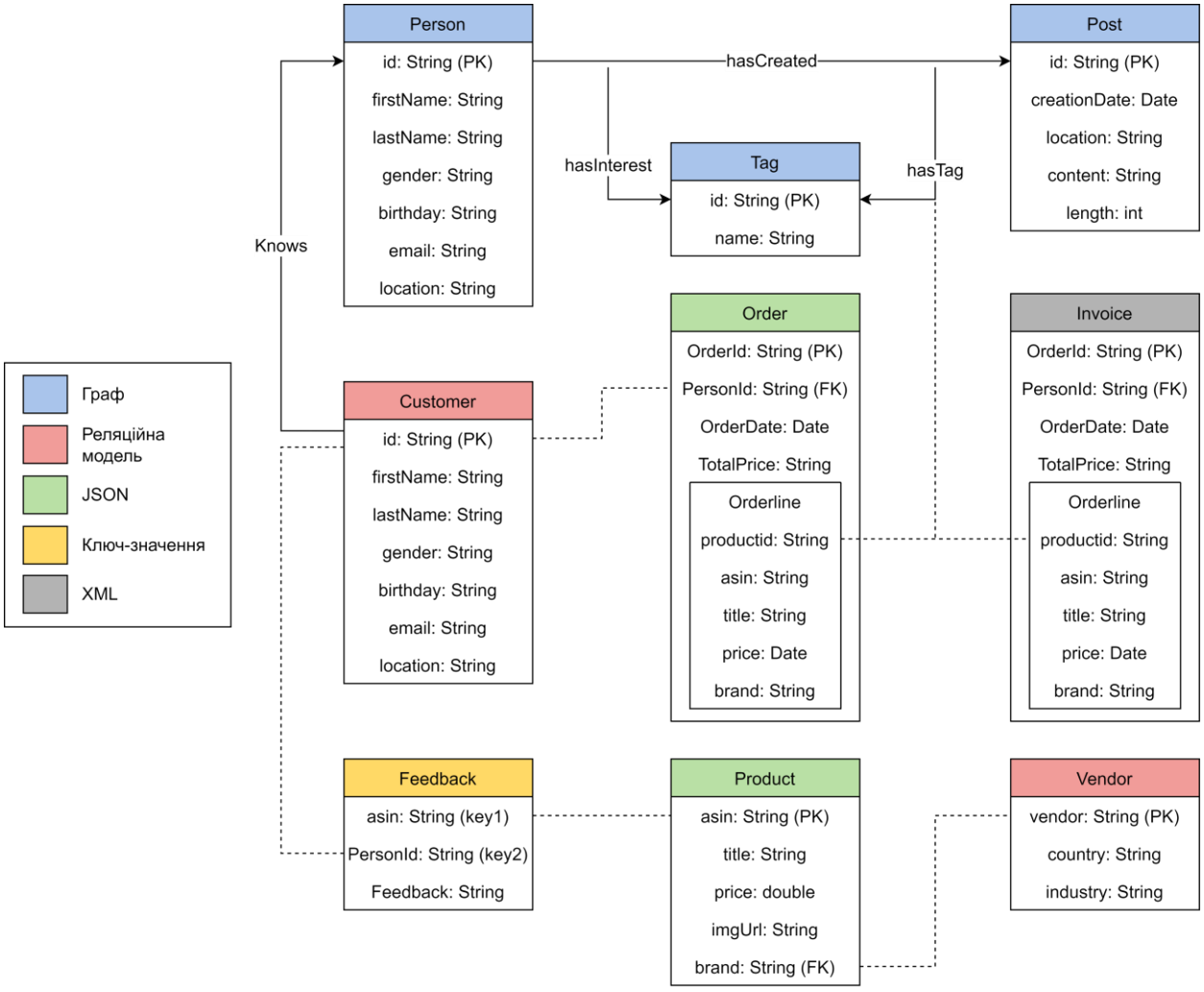


Рисунок 2.1 – Діаграма класів

На рисунку 2.1 показана детальна схема мультимодельних даних, включаючи п'ять моделей даних. Зокрема, реляційна модель включає структурованих клієнтів і постачальників, модель JSON містить напівструктуровані замовлення та продукти. Соціальна мережа моделюється у вигляді графа, який включає три сутності та три відношення, тобто «Person», «Post», «Tag», «person_hasInterest_tag», «person_hasCreated_post»,

«post_hasTag_tag». «Feedback» та «Invoice» моделюються як ключ-значення та XML відповідно. Вони також мають кореляції між моделями даних. Наприклад, клієнт знає друзів (реляційна кореляція з моделлю графа), клієнт здійснює транзакції (JSON корелює з реляційною моделлю).

2.2 Генерація даних

На рисунку 2.2 показано трифазну структуру генерації даних. Зокрема, на етапі покупки LDBC отримує метадані зі сховища, потім генерує дані графа та початкові інтереси осіб. Ці дані передаються генератору для отримання даних про транзакції. На етапі поширення-купівлі інтереси клієнтів формуються на основі інформації, отриманої на попередній фазі. На етапі повторної покупки інтереси всіх клієнтів будуть формуватися на основі моделі CLVSC, яку детальніше буде описано пізніше. На кожному етапі генеруємо дані про транзакції відповідно до інтересів клієнтів і об'єднуємо всі три частини як невіддільну частину мультимодельного набору даних.

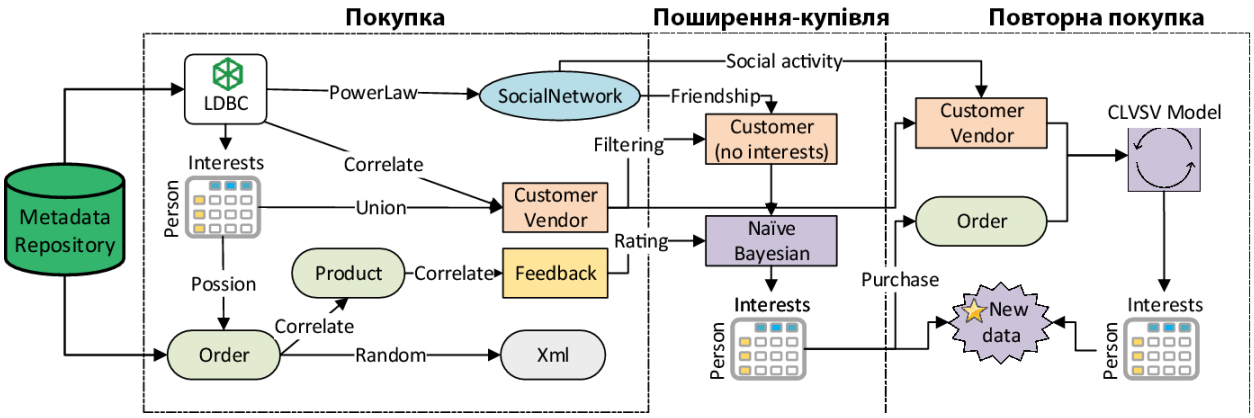


Рисунок 2.2 – Потік робіт генерації даних

2.2.1 Купівля

На цьому етапі враховуємо два фактори під час генерування даних. По-

перше, люди зазвичай купують товари, виходячи зі своїх інтересів. По-друге, люди з більшими інтересами частіше купують продукти, ніж інші. Інтереси особи до продуктів формуються з допомогою LDBC. Ця фаза реалізована на вершині Spark SQL за допомогою Scala, яка використовує велику кількість API та UDF для генерування мультимодельних даних. Зокрема, спочатку визначаємо кількість транзакцій для кожної особи, поділяючи кількість її інтересів на константу c, потім вибираємо розмір для кожної транзакції з розподілу Пуассона з параметром λ, нарешті призначаємо елементи кожній транзакції, випадковим чином вибираючи елементи з їхнього набору інтересів. Замовлення будуть виведені у форматі JSON із вбудованим масивом елементів рядка замовлення. Тим часом рахунки-фактури створюватимуться з тією ж інформацією, але у форматі XML. Крім того, випадковим чином вибираємо реальний огляд продукту та відповідну оцінку з набору даних Amazon як відгук. Отже, дані складаються з п’яти моделей: «Social Network» (граф), «Vendor and Customer» (відношення), «Order and Product» (JSON), «Invoice» (XML), «Feedback» (ключ-значення).

2.2.2 Поширення-купівля

На цьому етапі включаємо дві складові з попереднього генерування даних: основні демографічні дані людини, наприклад, стать, вік, місце розташування, відгуки друзів. Це мотивується спостереженням, що люди з тими ж атрибутами, швидше за все, мають таку ж поведінку, і люди також довіряють рекомендаціям продуктів від друзів. Функція оцінки визначається наступним чином:

$$S_{ui} = \sum_{k} k \cdot \Pr(R_{ui} = k \mid A = a_u) + E(R_{ui} : \forall v \in N(u)), \qquad (2.1)$$

де $\sum_{k} k \cdot \Pr(R_{ui} = k \mid A = a_u)$ – очікування розподілу ймовірності оцінки

цільового користувача u на цільовий елемент i;

$A = \{a_1, a_2, ..., a_m\}$ – обчислений набір атрибутів користувача заснований на наївному байєсіанському методі;

$E(R_{ui} : \forall v \in N(u))$ – це очікування розподілу рейтингу друзів u на цільовий елемент;

$N(u)$ – набір друзів користувача u.

Щоб навчити модель Байєса, впровадили методи, використовуючи програму Python scikit-learn, яка бере профілі користувачів та історію оцінок з попередньої фази як навчальний набір. Для кожної особи без інтересів беремо предмети, оцінені їхніми друзями, як набір кандидатів, а потім ранжуємо їх за допомогою формули (2.1). Нарешті, беремо частину рейтингового списку з однаковим розміром відсотків на етапі покупки, а потім генеруємо нові транзакції. Якщо розмір рейтингового списку менший, ніж на попередньому етапі, випадковим чином згенеруємо решту транзакцій.

2.2.3 Повторна покупка

Модель CLV пропонується для усунення обмежень RFM у прогнозуванні позаконтрактної поведінки клієнтів. Пропонуємо нову імовірнісну модель CLVSC, щоб робити точні прогнози, включаючи соціальну діяльність клієнта щодо бренду. Загалом CLVSC складається з трьох компонентів: очікуваної кількості поведінки, очікуваної грошової вартості та очікуваного позитивного соціального залучення клієнта. Функція оцінки для CLVSC визначається наступним чином:

$$S_{ib}(CLVSC) = E(X^* \mid n^*, x', n, m, \alpha, \beta, \gamma, \delta) \cdot (E(M \mid p, q, v, m_x, x) + E(S \mid \bar{s}, \theta, \tau)), \quad (2.2)$$

де i, b – індекс клієнта та бренду відповідно.

Нехай $E(X^* \mid n^*, x', n, m, \alpha, \beta, \gamma, \delta)$ позначає очікувану кількість поведінки

протягом наступних n^* періодів клієнта з історією спостережуваної поведінки (x', n, m), де x' – кількість поведінки, що відбулася за n період, з останньою поведінкою $m \le n$; (α, β) і (γ, δ) – це параметри бета-розподілу для активної ймовірності та ймовірності неактивної відповідно, поведінка – це покупка або публікація. Використовуючи бета-геометричну/бета-біноміальну (BG/BB) модель, маємо:

$$E(X^* \mid n^*, x', n, m, \alpha, \beta, \gamma, \delta) = \frac{B(\alpha + x + 1, \beta + n - x)}{B(\alpha, \beta)} \times$$
$$\times \frac{B(\gamma - 1, \delta + n + 1) - B(\gamma - 1, \delta + n + n^* + 1)}{B(\gamma, \delta)} : L(\alpha, \beta, \gamma, \delta \mid x, n, m), \qquad (2.3)$$

де $L(\alpha, \beta, \gamma, \delta \mid x, n, m)$ – функція ймовірності.

Цей результат отримано з використанням очікування для спільного апостеріорного розподілу активної і неактивної ймовірності.

Дотримуючись підходу Фейдера, Гарді та Бергера [9] щодо додавання грошової вартості, $E(M \mid p, q, v, m_x, x)$ позначає очікувану грошову вартість. Тоді маємо:

$$E(M \mid p, q, v, m_x, x) = \left(\frac{q - 1}{px + q - 1} \right) \frac{vp}{q - 1} + \left(\frac{px}{px + q - 1} \right) m_x. \qquad (2.4)$$

$E(S \mid \bar{s}, \theta, \tau)$ позначає очікувану соціальну залученість клієнта, припускаємо, що кількість соціальної залученості клієнта відповідає процесу Пуассона зі швидкістю λ, а неоднорідність у λ слідує за гамма-розподілом з параметром форми θ і параметром ставки τ серед клієнтів. Відповідно до сполучення моделі Пуассона-Гамма, точкова оцінка $E(S \mid \bar{s}, \theta, \tau)$ може бути розкладена на множники наступним чином:

$$E(S \mid \bar{s}, \theta, \tau) = \theta'\tau' = \frac{\tau}{1+\tau}\bar{s} + \frac{\tau}{1+\tau}\theta\tau. \quad (2.5)$$

Таким чином, отримана точкова оцінка є середньозваженим середнім значенням $\bar{s}$ по вибірці та попереднім середнім $\theta\tau$.

Реалізовуємо модель CLVSC, використовуючи пакет BTYD, який бере невелику частину вибірок з попередніх фаз як навчальний набір. Для всіх людей оцінюємо їх інтереси до брендів, потім беремо частину інтересів з однаковим розміром інтересів на етапі покупки, нарешті генеруємо нові транзакції.

Позначимо PDF для розподілу обмеженого степеневого закону (BPL) з мінімальним значенням x_{min} наступним чином:

$$p(x) \propto \frac{\alpha - 1}{x_{min}} \left(\frac{x}{x_{min}} \right)^{-\alpha}, \quad (2.6)$$

де α – параметр масштабування для розподілу BPL, який зазвичай знаходиться в діапазоні (2, 3).

Потім, після трифазного генерування даних, слідують розмір покупки особи, покупки друзів та покупок особи враховуючи, що бренд наслідує розподіл BPL з мінімальним значенням x_p, x_{pp} і x_{rp} відповідно.

2.3 Робоче навантаження

Робоче навантаження бенчмарку складається з набору складних запитів лише для читання та транзакцій читання-запису, які включають щонайменше дві моделі даних, спрямовані на охоплення різних бізнес-кейсів та технічних перспектив. Точніше, що стосується бізнес-кейсів, вони поділяються на чотири основні важелі [10]: індивідуальний, розмовний, громадський і

комерційний. У цих чотирьох важелях відбиваються загальноприйняті бізнес-кейси різної деталізації. Що стосується технічної точки зору, то вони розроблені на основі техніки точки заглушення [37], яка поєднує загальні технічні проблеми з новими важкорозв'язними проблемами для обробки мультимодельних запитів, починаючи від кон'юнктивних запитів (OLTP) і закінчуючи аналізом (OLAP). Через обмеженість простору лише підсумуємо їх характеристики.

2.3.1 Бізнес-кейси

Визначили дві транзакції та чотири рівні запитів, які включають десять мультимодельних запитів для моделювання реалістичних бізнес-кейсів у соціальній комерції. Зокрема, дві транзакції, а саме транзакції «New Order» та «Payment», імітують дві типові мультимодельні транзакції для сценарію онлайн-покупок. Що стосується мультимодельних запитів, то індивідуальний рівень імітує випадок, коли компанії створюють 360-градусний огляд клієнта, збираючи дані з багатьох джерел клієнта. Для цього рівня є один запит. Рівень розмови зосереджується на аналізі напівструктурованих і неструктурованих даних клієнта, включаючи запити № 2 і 3. Ці два запити зазвичай використовуються для компанії, щоб виловити полярність настроїв клієнта з відгуків, а потім налаштувати онлайн-рекламу або операційну стратегію. Запити 4, 5, 6 на громадському рівні спрямовані на дві сфери: вивчення загальних моделей покупок у спільноті та аналіз впливу спільноти на поведінку індивіда щодо покупки. Нарешті, рівень комерції спрямований на оптимізацію асортименту та прозорість продуктивності. Зокрема, запити № 7, 8, 9 визначають продукти або постачальників зі зниженням чи підвищенням продуктивності, а потім знаходять причину покращень. Запит № 10 полягає в обчисленні актуальності, частоти, грошової вартості клієнтів щодо постачальника, а потім знаходження загальних тегів у публікаціях.

2.3.2 Технічні величини

Розробка робочого навантаження заснована на техніці точки заглушення, яка перевіряє багато аспектів бази даних під час обробки запиту. Як правило, ці аспекти можуть стосуватися різних компонентів баз даних, таких як оптимізатор запитів, механізм виконання та система зберігання. Що більше, ці точки заглушення в робочому навантаженні не тільки пов’язані з загальними проблемами обробки запитів для традиційних систем баз даних, а й з деякими новими проблемами мультимодельної обробки запитів. Тут перерахуємо три ключові моменти.

По-перше, це вибір правильного типу та порядку з’єднання. Визначення належного типу та порядку об’єднання для мультимодельних запитів є новою і нетривіальною проблемою. Це пояснюється тим, що оптимізатор запитів вимагає оцінки потужності щодо залучених моделей. На додаток, йому потрібен оптимізатор запитів, щоб розумно визначити оптимальний порядок об’єднання для мультимодельного запиту. Час виконання різних порядків і типів об'єднання може відрізнятися на порядки величини через домінування різних моделей даних. Таким чином, ця точка заглушення перевіряє здатність оптимізатора запитів знаходити оптимальний тип з’єднання та порядок для мультимодельного запиту. У запропонованому робочому навантаженні всі запити включають кілька об’єднань у різних моделях даних.

По-друге, це виконання складної агрегації. Ця зупинка включає два типи запитів, що стосуються складної агрегації. Перший тип – це агрегація до складної структури даних, яка вимагає, щоб ММБД працювала з даними, що не залежать від схеми, під час агрегації. Другий – це запит із наступними агрегаціями, де результати агрегації служать вхідними даними для іншої агрегації. Крім того, ці агрегації передбачають об’єднання результатів кількох моделей. Наприклад, запит № 10 вимагає від мультимодельної бази даних доступу до масиву продуктів у замовленнях JSON під час обробки

першого об’єднання. Тоді результати будуть вхідними для другої агрегації на графі.

І останнє, це забезпечення узгодженості транзакцій читання-запису. Транзакція бази даних повинна мати властивості ACID. Таким чином, ця контрольна точка перевіряє здатність механізму виконання та системи зберігання знайти відповідну техніку контролю паралельності, щоб гарантувати узгодженість та ефективність транзакції читання-запису. Транзакції не тільки передбачають операції читання-запису для кількох сутностей, але також вимагають, щоб ММДБ гарантувала узгодженість моделей даних. Зокрема, транзакція «New Order» містить запит, що в основному читається за трьома об’єктами: «order», «product» і «invoice». Транзакція «New Payment» містить переважно запит на запис за трьома об’єктами: «order», «customer» і «invoice».

3 КУРУВАННЯ ПАРАМЕТРІВ ТА ОЦІНКА БЕНЧМАРКА

3.1 Мотив для курування параметрів

Мультимодельний запит вимагає налаштування параметрів заміни, і різні параметри заміни призведуть до різної ефективності бенчмаркінга. Як показано на рисунку 3.1.а, «PersonId/56» і «BrandName/"Nike"» є двома параметрами заміни, які можна замінити іншими значеннями для шаблону запиту. Результат експерименту на рисунку 3.1.б ілюструє, що при подвійному запуску запиту з двома різними парами параметрів підставляння у двох мультимодельних базах даних: ArangoDB і OrientDB, їх продуктивність є абсолютно протилежною. Цікаво, що це відбувається тому, що ті самі запити з різними параметрами відрізняються за розміром проміжних результатів. Наприклад, запит із парою А включає відносно більші проміжні результати JSON, тоді як запит із парою Б приймає більший розмір графа. Тому виникає відкрите запитання: чи можна розробити ефективний та прозорий метод контролю вибірковості запиту, вибираючи різні значення параметрів для всебічної оцінки? Зокрема, прагнемо представити різноманітність запитів у процесі бенчмаркінга шляхом розумного вибору значень параметрів. Завдяки різноманітності запитів можемо отримати більше інформації від оцінки запиту, як показано на рисунку 3.1. Однак це ставить три нові проблеми для налаштування параметрів у мультимодельному запиті бенчмаркінга: як представити розмір проміжних результатів з точки зору моделі даних, як вибрати параметри ефективно та всебічно, і як охопити різні робочі навантаження, що стосуються моделі даних, щоб представити різноманітність запитів у процесі бенчмаркінга. У світлі цього представляємо представлення залучених розмірів проміжних результатів для запиту з різними параметрами. Потім формалізуємо нову проблему, яка називається мультимодельним куруванням

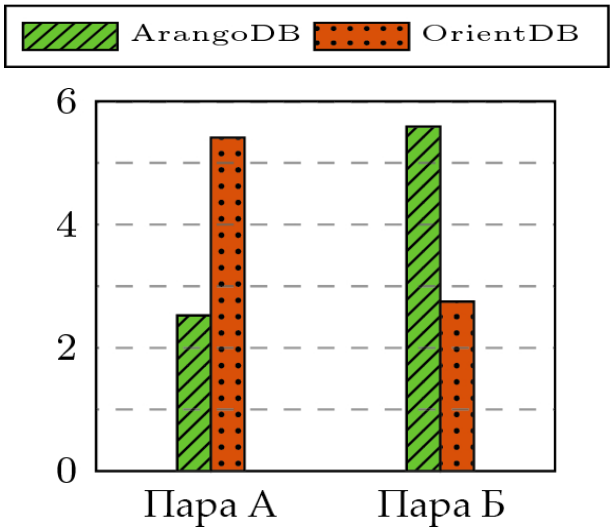
параметрів, яка спрямована на диверсифікацію вибірковості запитів під контролем курування параметрів. У зв'язку з тим, що ця проблема є обчислювально інтенсивною, пропонуємо алгоритм вибірки для ефективного та ефективного розв'язання цієї проблеми.

Приклад запиту з двома парами параметрами підстановки:

Пара А: *@PersonId*=33, *@BrandName*="Adidas"

Пара Б: *@PersonId*=56, *@BrandName*="Nike"

а)



б)

Рисунок 3.1 – Мотив для курування параметрів: а) приклад запиту; б) час виконання запитів у секундах

3.2 Визначення проблеми та її вирішення

Пропонуємо векторний підхід для представлення розміру проміжних результатів, пов'язаних зі значеннями параметрів. Зокрема, обчислюємо розміри всіх проміжних результатів, які відповідають значенню параметра на основі перестановки моделей даних. Наприклад, за допомогою мультимодельного запиту: «Для людини p і бренду продукту b знайдіть друзів p, які купили товари з брендом b». Область параметрів складається з двох базових вимірів параметрів: «person p» і «brand b», тому обчислюємо ненульовий вектор (|G|, |J|, |GJ|) на основі запиту та області параметрів, де |G| означає розмір проміжних результатів із графа, |J| є розміром JSON, |GJ| належить до розмірів об'єднання між моделями граф і JSON, тобто людей,

які є друзями p і купували продукти бренду b. Цей метод не тільки дозволяє представляти розмір результатів незалежно від баз даних, але також допоможе обґрунтувати результати продуктивності шляхом прозорого аналізу проміжних результатів. Отже, визначення вектора розміру виглядає наступним чином: для мультимодельного запиту кожен вектор розміру визначається як $\omega\{c_1,.,c_k,.,c_n\}$, де c_k – це k-й проміжний розмір результату щодо залучених моделей даних або їх комбінацій, c_n – кінцевий розмір результату . Довжина ω знаходиться в межах $[3,2^m-1]$, де m – номер моделі даних.

Інтуїтивно прагнемо виконати той самий запит із векторами різного розміру, щоб перевірити надійність системи; чим більше різнорідні вектори розміру, тим різноманітнішими є запити. Вибираємо суму мінімальних попарних відстаней векторів розміру як міру різноманітності, а для вимірювання невідповідності використовуємо евклідову відстань. Тут міра різноманітності потребує певного пояснення: концепція різноманітності дещо відрізняється від концепції пошуку інформації. Хоч вони спрямовані на диверсифікацію результатів пошуку [11], мета – знайти вектори розміру з різними значеннями з метою оцінки продуктивності. Отже, проблема контролю параметрів зводиться до пошуку найбільш віддалених векторів розміру.

Тепер різноманіття параметрів і проблема курування мультимодельних параметрів визначаються таким чином: для d-вимірної області параметрів P^d кожна точка $p \in P^d$ відповідає вектору розміру ω. Відстань між двома ω – нормована евклідова відстань. Дано набір точок $S_k \subset P^d, k \geq 2$. Різноманітність параметрів визначається як сума мінімальних парних відстаней вектора розміру k.

Враховуючи запит і відповідну область параметрів (два крайні ліві стовпці таблиці 3.1), нам потрібно обчислити вектори розміру. Припустимо, що необхідний розмір параметрів дорівнює 3, мета – повернути три групи

параметрів, у яких розміри векторів найбільш віддалені. Обчислюючи нормовані евклідові відстані серед усіх попарних векторів розміру, тут три вектори мають максимальну мінімальну відстань пари: 0,9 + 0,3 + 0,3 = 1,5, тому вибираємо три групи параметрів для оцінки запиту: (5137 , Adidas), (9001, Nike), (4145, Puma).

Таблиця 3.1 – Курування параметрів

Область параметрів		Розмір вектора		
PersonId	Brand	|G|	|J|	|GJ|…
…	…	…	…	…
5137	Adidas	2	100	5
9001	Adidas	50	100	20
9001	Nike	50	200	301
2995	Nike	100	200	405
4145	Puma	100	300	1001
…	…	…	…	…

3.2.1 Проблема курування мультимодельних параметрів

Дано мультимодельний запит Q з d-вимірною областю параметрів P^d, яка є декартовим добутком базової області, а також розміром k. Мета полягає в тому, щоб вибрати підмножину $S_k \subset P^d$ так, щоб розмаїття параметрів S_k було максимальним.

Існують дві труднощі, щоб розв’язати проблему курування мультимодельних параметрів. По-перше, область параметрів потенційно величезна, що робить неможливим обчислення всіх векторів розміру. По-друге, знаходження k найвіддаленіших точок від набору точок S потребує розв’язання складної задачі, зворотної до проблеми K-центру, яка, як було доведено, є NP-складною. Тому пропонуємо новий алгоритм, заснований на

LHS [12] для розв’язання цієї проблеми. LHS – це багатовимірний стратифікований метод вибірки, який може покращити охоплення вхідного простору шляхом незалежного відбору значень із сегментів, що не перекриваються. Головна ідея полягає в тому, що, хоча важко обчислити всі вектори розмірів, проміжні розміри результату базових розмірів, наприклад, |G| та |J| є незалежними й можуть бути заздалегідь обчислені окремо, потім використовуємо метод LHS з частковими знаннями, щоб гарантувати, що діапазони розмірів базових розмірів повністю представлені.

Доведемо першу лему про те, що різноманітність параметрів між двома векторами розмірів збільшується зі збільшенням відстаней їх базових розмірів. Відповідно до формули (2.6), розміри мультимодельного з'єднання монотонно збільшуються зі збільшенням розміру базового розміру. Оскільки вектор розміру складається лише з розмірів базових розмірів та розмірів їх з’єднання, то виконується лема 1.

Весь процес курування представлений в алгоритмі під назвою MJFast, який зображений на рисунку 3.2. Зокрема, цей алгоритм працює в три етапи: рядки 2-3 обчислюють розмір результату щодо кожного базового виміру, рядки 4-9 використовують метод LHS для вибору векторів розміру, який розділяє діапазон розмірів кожного базового виміру до k сегментів, а потім вибірку векторів розміру k з d-вимірних сегментів. Без втрати загальності розподіляємо k кошиків рівномірно в межах розміру діапазону. Рядки 10-13 показують вектори розмірів у відповідні групи параметрів. Оскільки все одно генеруємо дані, можемо матеріалізувати відповідні підрахунки, наприклад, кількість друзів на людину, кількість продуктів на бренд, як побічний продукт генерації даних. Ця стратегія значно пришвидшить процес визначення параметрів.

Input: Parameter size k, Parameter domain $\mathbb{P}^d$, Curated query Q
Output: S_k as a subset of $\mathbb{P}$, with size k

$V_k, \mathbb{B}^d, S_k \leftarrow \emptyset$; // Initial sets of size vectors, buckets and the result set
foreach $P \in \mathbb{P}$ **do** // Enumerate the base dimension in parameter domain
  foreach $p \in P$ **do**
    $\{s_1^p, \cdots, s_n^p\} \leftarrow \text{Compute}(p, Q)$; // Compute the size vector
    foreach $i \in [1, k]$ **do** // Divide the size ranges into k buckets
      $B_i^p \leftarrow \{s_1^p, \cdots, s_n^p\}$
    end
  end
end
$\mathbb{B}^d \leftarrow B_1 \times \cdots \times B_d$; // Cartesian product of all buckets
foreach $\overrightarrow{b} \in \mathbb{B}^d$ **do**
  $V_k \leftarrow V_k \cup \overrightarrow{b}$; // Sample k vectors without replacement
end
foreach $v' \in V_k$ **do**
  $S_k \leftarrow S_k \cup Lookup(D, v')$; // Search in Dictionary by v' for S_k
end
return S_k

Рисунок 3.2 – Алгоритм MJFast

Знову розглянемо приклад курування параметрів, зображений в таблиці 3.1. Враховуючи, що розмір параметра k є 3, спочатку обчислюємо базовий розмір |G| та |J| окремо, потім сортуємо та групуємо їх у три групи відповідно: |G|: (група 1: 2, група 2: 50, група 3: 100), |J|: (група 1: 100, група 2: 200, група 3: 300). Далі використовуємо метод LHS, щоб гарантувати, що кожна група буде обрана один раз, потім зіставляємо вибрані вектори розміру з групами параметрів, які будуть рекомендованими параметрами.

Доведемо другу лему про те, що складність алгоритму MJFast для задачі курування мультимодельних параметрів дорівнює O(nlogn+n*c). Оскільки як необхідний розмір параметра k, так і залучений базовий розмір d є дуже малими цілими числами, у загальній часовій складності алгоритму домінують витрати на обчислення та сортування розміру базового виміру, який дорівнює O(nlogn+n*c), де n – розмір загальних значень параметрів, c – константа, яка вимірює середню вартість обчислення розміру для кожного значення.

Тепер доведемо третю лему про те, що враховуючи розмір параметра $k \geq 2$, розмаїття параметрів k вибірок за допомогою латинської вибірки гіперкуба не менше, ніж різноманітність параметрів k вибірок шляхом випадкової вибірки з імовірністю $1-\frac{(k-1)^{k-1}}{k^3}$. Нехай P – домен параметрів, що включає B базових розмірів, для кожного базового виміру b із розміром n у B, b розбивається на k кошиків із рівним розміром n/k. Таким чином, кожне значення потрапляє в кошик з імовірністю 1/k. Для $k \geq 2$ ймовірність того, що кошик містить лише одну випадкову вибірку, дорівнює (k1)1k(1−1k)k−1. Відповідно до LHS відбору, «відро» завжди містить лише один зразок. Об'єднуємо першу лему з нерівністю: $\sqrt{(s_2^{B_2} - s_1^{B_1})^2} \geq \sqrt{(s_2^{B_1} - s_1^{B_1})^2}$, де s_1 і s_2 – дві вибірки , а B_1 і B_2 – це два кошики. Таким чином, можна вивести, що виконується третя лема.

3.3 Оцінка бенчмарку

3.3.1 Генерація даних

Для експериментального налаштування генеруємо синтетичні дані на кластері з трьох машин, кожна з подвійним 4-ядерним процесором Xeon-E5540, 32 ГБ оперативної пам'яті та 500 ГБ HDD. Генератор даних реалізований на Spark за допомогою Scala.

У таблиці 3.2 представлені характеристики трьох згенерованих наборів даних, кожен з яких складається з п'яти моделей даних. Бенчмарк визначає набір масштабних коефіцієнтів (МК), орієнтованих на системи різного розміру. На розмір результівного набору даних в основному впливає кількість осіб (реляційні записи). Для бенчмаркінга баз даних використовуємо генератор даних для створення трьох наборів даних приблизно розміром 1 ГБ, 10 ГБ та 30 ГБ за допомогою масштабних

коефіцієнтів 1, 10 та 30 відповідно. Що стосується ефективності, результати експерименту свідчать, що генератор даних створив мультимодельні набори даних 1 ГБ та 10 ГБ за 10 та 40 хвилин на одній 8-ядерній машині. З точки зору масштабованості, успішно генеруємо мультимодельні дані 30G протягом 60 хвилин у кластері з трьома вузлами. Формування даних є детермінованим, що означає, що розмір створених наборів даних із певним коефіцієнтом масштабування завжди однаковий. Крім того, як показано на рисунку 3.3, відсоток розміру даних кожної моделі має стабільний розподіл щодо різного масштабного коефіцієнта.

Таблиця 3.2 – Характеристики наборів даних

Масштабний коефіцієнт	Час генерації, хв	Число ($\times 10^4$) і розмір, МБ				
		Реляційні записи	Пари ключ-значення	Об’єкти JSON	Об'єкти XML	Вузли та ребра графа
1	10	1,2 і 1,1	25,2 і 233,7	25,2 і 219,2	25,2 і 326,5	(123,1, 338,9) і 236,6
10	40	7,4 і 6,5	234,2 і 2313,1	234,2 і 2189,8	234,2 і 3568,6	(969,3, 3208,3) і 2095,8
30	60 (3 вузли)	18,3 і 15,8	636,8 і 6367,8	636,8 і 6184,9	636,8 і 11771,3	(2674,3, 10951,5) і 6191,5

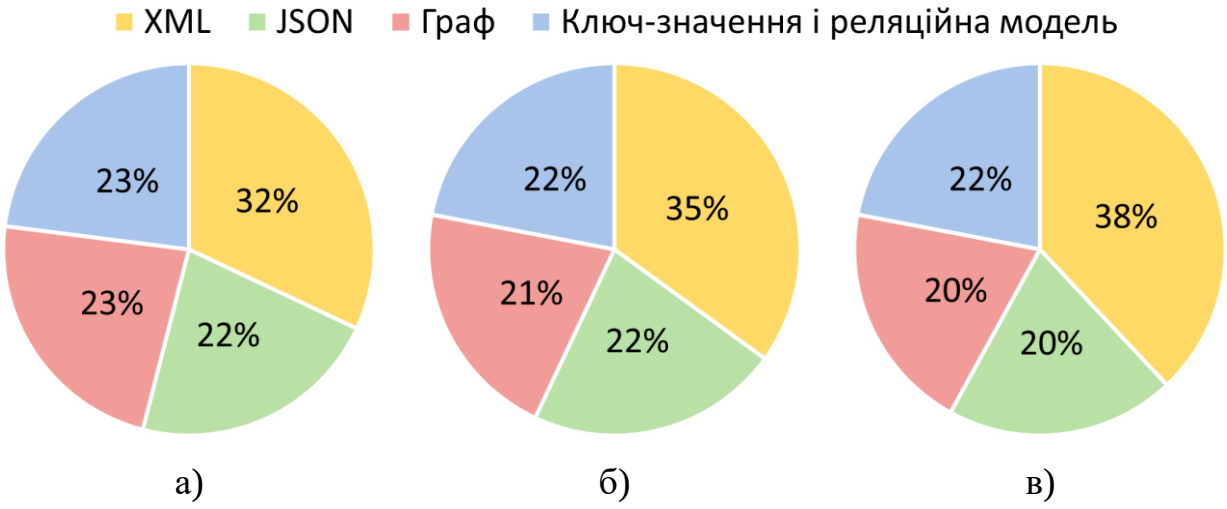


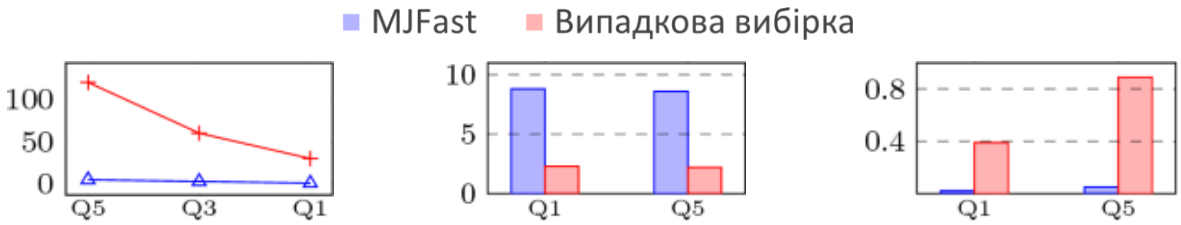
Рисунок 3.3 – Мультимодельний розподіл створеного набору даних: а) при масштабному коефіцієнті 1; б) при масштабному коефіцієнті 10; в) при масштабному коефіцієнті 30

3.3.2 Курування параметрів

У цій частині порівнюємо запропонований алгоритм MJFast із випадковим методом у трьох аспектах: час генерації, сума парних відстаней та KL-розбіжність D_{KL}. Експерименти проводяться на машині з 4-ядерним процесором i5-4590 і 16 ГБ оперативної пам'яті. Зокрема, час генерації вимірює, наскільки швидко параметри можуть бути згенеровані. Сума парних відстаней вимірює різноманітність параметрів генерованих параметрів. KL-дивергенція D_{KL} оцінює стабільність підходів до курування параметрів.

Як показано на рисунку 3.4.а, метод значно скоротив час генерування параметрів для трьох запитів. Генеруємо параметри для Q5, Q3, Q1 за 5, 3, 1 хвилину відповідно. Генерація всіх векторів розміру займе приблизно 2 години, 1 год і пів години відповідно. Причина того, що цей метод набагато ефективніший, полягає в тому, що обчислюються лише розміри результату для базових розмірів. Для останніх двох аспектів розглянемо два випадки: запити Q1 (однопараметричний) і Q5 (подвійний параметр) відповідно.

Обидва випадки мають значення k рівним 10. На рисунку 3.4.б показано, що параметри, вибрані даним підходом, мають у 3 рази більшу різноманітність у порівнянні з випадковою вибіркою, що підтверджує теоретичний аналіз у третій лемі. Нарешті, рисунок 3.4.в показує, що цей підхід дає у 2 рази менший D_{KL}, що вказує на те, що даний підхід є більш надійним, ніж випадковий під час генерування параметрів кілька разів.



а) б) в)

Рисунок 3.4 – Курування параметрів ефективності, різноманітності та стабільності: а) час генерації у хвилинах; б) значення сум парної відстані; в) значення KL-розбіжності

4 СИСТЕМНЕ ВИВЧЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1 Тестовані системи

Досліджуємо та оцінюємо чотирьох представників: OrientDB, ArangoDB, AgensGraph і Spark SQL. На рисунках 4.1, 4.2, 4.3 показані розподілені архітектури OrientDB, ArangoDB і Spark відповідно, тоді як AgensGraph опущено, оскільки на цю мить він не підтримує розподілений режим. Крім того, представляємо реалізацію запиту для кожної системи, щоб допомогти користувачеві отримати уявлення про те, як реалізувати мультимодельний запит у них за допомогою бенчмарка.

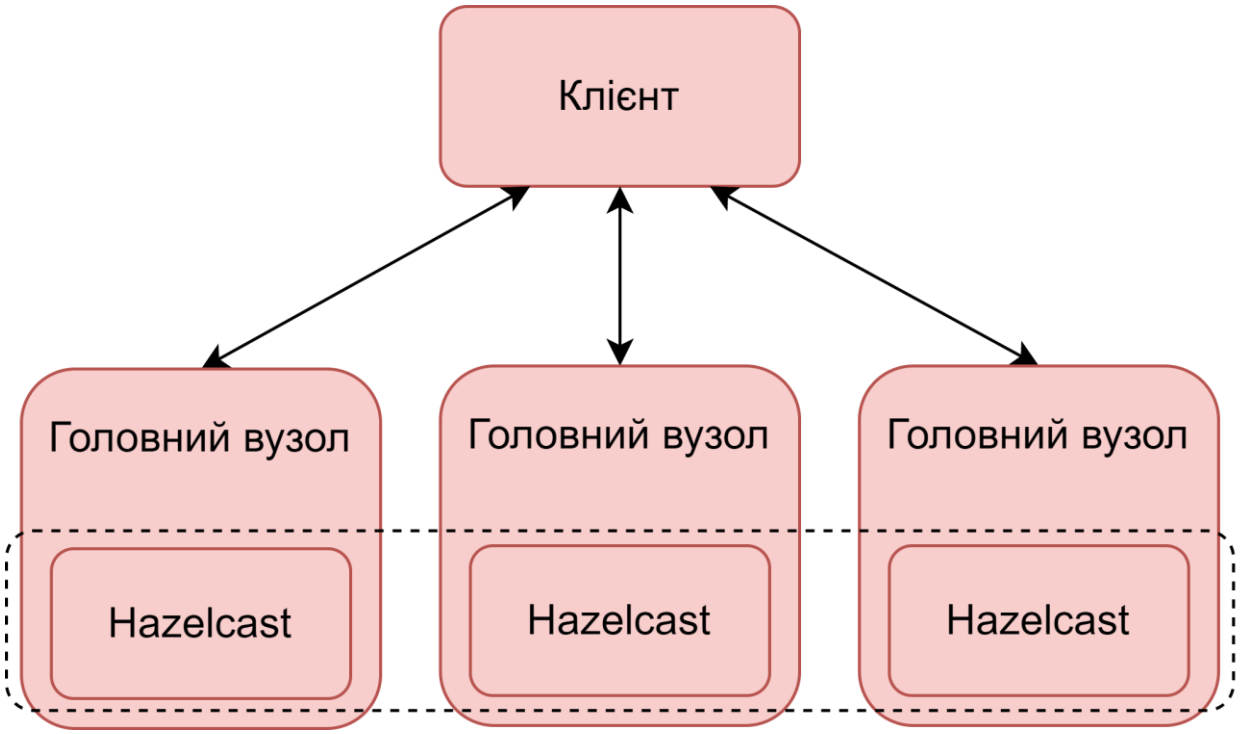


Рисунок 4.1 – Розподілена архітектура OrientDB

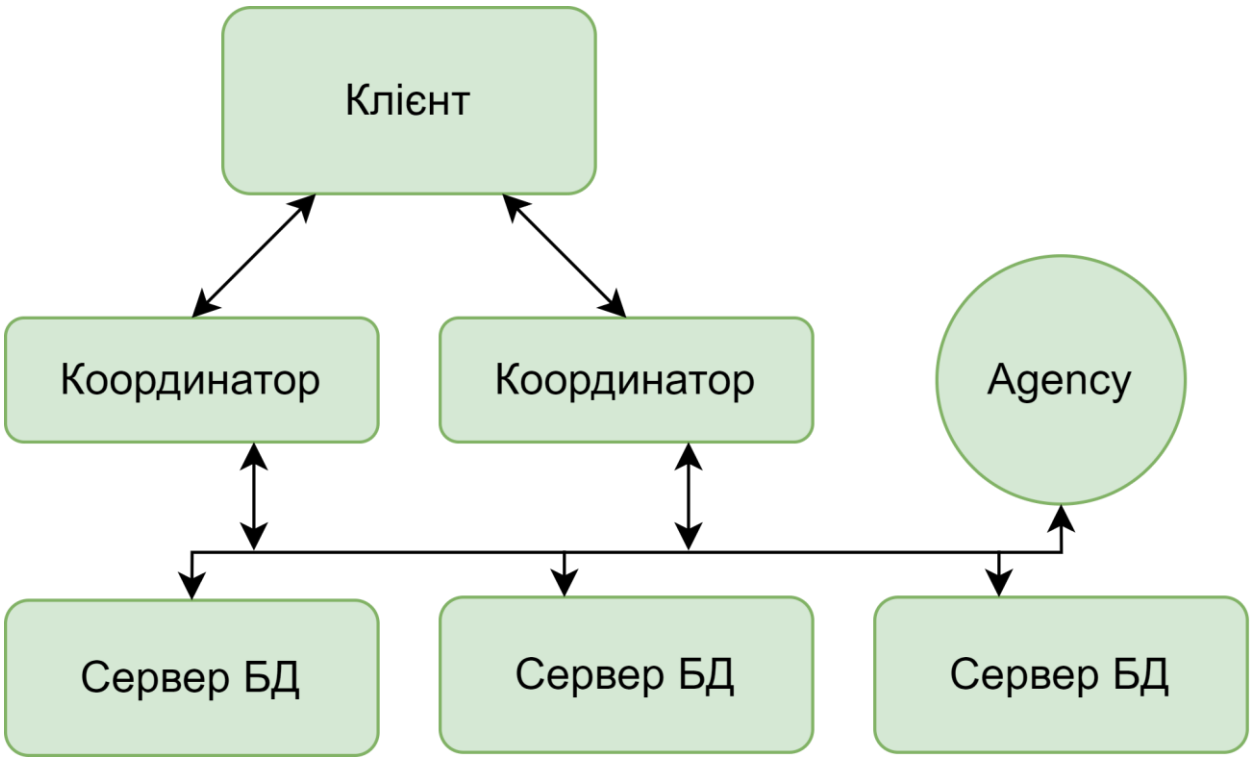


Рисунок 4.2 – Розподілена архітектура ArangoDB

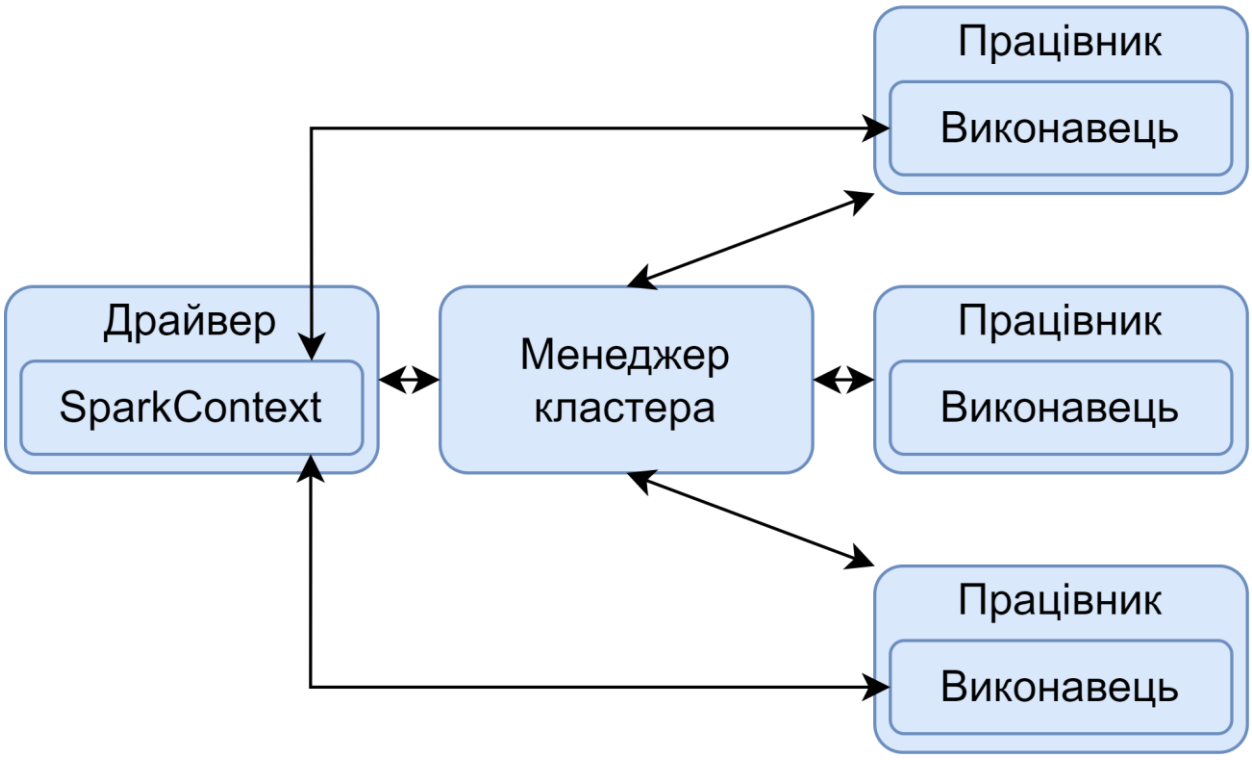


Рисунок 4.3 – Розподілена архітектура Spark

4.1.1 OrientDB

OrientDB – це ММБД з відкритим вихідним кодом, яка підтримує мультимодельні запити, транзакції, шардинг і реплікацію. Основною моделлю є графічна модель. У OrientDB запис – це найменша одиниця, яка може бути завантажена і збережена в базі даних. Це може бути документ, запис байтів (BLOB) і вершина, а також ребро. Записи визначаються класами, які можуть бути без схем, з повною схемою або змішаними. Щобільше, OrientDB реалізує безіндексну структуру суміжності для проходження графів без будь-якого пошуку індексу. Як наслідок, реалізується модель даних, що об’єднує моделі документів, графів, об’єктів і ключ-значень. Коли справа доходить до мови запитів, то забезпечується уніфікована мова запитів під назвою OrientDB SQL для запиту даних кількох моделей. Однак вона не повністю відповідає стандартному синтаксису SQL, такому як SQL ANSI-92, хоч це мова, подібна до SQL.

Як показано в крайній лівій частині рисунку 4.1, OrientDB має розподілену архітектуру з кількома майстрами, де кожен сервер в кластері може стати онлайн-головним вузлом, який підтримує як читання, так і запис. Він реалізований на основі фреймворку Hazelcast, який може автоматично керувати конфігурацією кластера, зв’язком та перемиканням збоїв, а також синхронізацією між серверами. OrientDB підтримує розділення даних за визначеним атрибутом, але це можна зробити лише вручну. Проте OrientDB автоматично вибирає локальні сервери для зберігання сегментів після розподілу даних. Зокрема, користувач може налаштувати параметр writeQuorum, щоб вирішити, скільки головних вузлів повинні виконувати записи паралельно, після чого інші невирівняні сервери виконують відкладені оновлення.

4.1.2 ArangoDB

ArangoDB – це ММБД з відкритим вихідним кодом, яка підтримує мультимодельні запити, транзакції, шардинг і реплікацію. Вона забезпечує нову мову запитів. З точки зору моделі даних, ArangoDB спочатку є документно-орієнтованою базою даних, яка реалізована на механізмі зберігання ключ-значення, RocksDB. Документи в ArangoDB мають формат JSON, вони організовані та згруповані в колекції, кожна колекція може розглядатися як унікальний тип даних у базах даних. Модель даних графа реалізується шляхом зберігання документа JSON для кожної вершини й документа JSON для кожного ребра. Ребра зберігаються в спеціальних колекціях ребер, які гарантують, що кожне ребро має атрибути «from» та «to», які посилаються на початкову та кінцеву вершини ребра. ArangoDB розробляє уніфіковану та декларативну мову запитів ArangoDB (AQL) для маніпулювання мультимодельними даними. Зокрема, основний будівельний блок AQL створений на основі виразів FLWR (For, Let, Where, Return) із XQuery в XML, за винятком того, що «Where» замінено на «Filter». AQL дозволяє виконувати такі операції з документами, як вибір, фільтрація, прогнозування, агрегація та об’єднання. Вона також підтримує обхід графа за допомогою синтаксису обходу AQL, який дозволяє обхід графа та відповідність шаблону в графі властивостей, наприклад сусідів, найкоротшого шляху тощо. Що ще цікавіше, операції для всіх трьох моделей, тобто ключ-значення, документ і граф, можуть бути інтегровані в один запит AQL і виконуватися в базах даних в цілому. Ця функція дозволяє об’єднувати та об’єднувати корельовані дані з різних джерел одночасно.

ArangoDB має багаторольову розподілену архітектуру, яка складається з трьох екземплярів, які відіграють різні ролі в кластері. А саме агент, координатор і сервер бази даних. Зокрема, екземпляр агента є центральним місцем, що використовується для зберігання конфігурації кластера та надання послуги синхронізації на основі алгоритму Raft Consensus.

Координатор – це екземпляр без стану, який з'єднує клієнтів і кластер ArangoDB. Він координує завдання запитів і призначає ці завдання локальним серверам, де зберігаються дані. Сервери баз даних розміщують фрагменти даних і можуть виконувати вхідні запити частково або цілком. Однією з головних особливостей ArangoDB є автоматичне розподілення, яке може автоматично розподіляти дані в сегментах на різні сервери на основі хеш-значення атрибута «_key», але користувач також може призначати атрибути для розподілу даних.

4.1.3 AgensGraph

AgensGraph – це ММБД з відкритим вихідним кодом, яка реалізує модель графа в ядрі PostgreSQL. Отже, вона може підтримувати модель графів, реляцій, ключ-значень та документів в одному ядрі. Основною перевагою є те, що користувачі можуть не тільки повною мірою використовувати реляційну базу даних із SQL, але й одночасно керувати графами та документами. Що стосується мови запитів, то вона використовує гібридний запит для маніпулювання даними, не винаходячи абсолютно нову мову запитів. Запит поєднує синтаксис SQL/JSON і мову запитів графів Cypher, які можуть бути взаємно сумісні. У випадку внутрішнього подання AgensGraph реалізує модель графа, зберігаючи два набори вершин у двох таблицях та використовуючи крайову таблицю, яка містить початковий і кінцевий зовнішні ключі, щоб зв’язати дві таблиці. Ця стратегія зберігання графа подібна до ArangoDB, яка також зберігає вершини та ребра окремо. Крім того, внутрішня вершина представлена двійковим форматом JSON – JSONB, що означає, що вона також може керувати типом документа.

4.1.4 Apache Spark

Apache Spark – це дуже популярна платформа розподілених обчислень

з відкритим вихідним кодом для великомасштабної аналітики даних, яка використовує RDD для кешування проміжних даних у наборі вузлів. Попри те, що Spark був розроблений для роботи зі структурованими даними, він збільшується, щоб забезпечити однорідний доступ до даних для роботи з мультимодельними даними. Зокрема, модуль Spark SQL тепер може обробляти різні типи даних, такі як дані графа, ключ-значення та дані JSON. Spark SQL використовує DataFrame як абстракцію даних, яка запозичує концепцію DataFrame з проєкту pandas. Однак DataFrame в Spark SQL сильно покладається на незмінні, ті, що знаходяться в пам’яті, розподілені та паралельні можливості RDD.

Spark дотримується архітектури майстра/робітника. У програмі драйвера є об’єкт SparkContext, який дозволяє користувачам налаштовувати загальні властивості, наприклад, головну URL-адресу, ядра ЦП та робочу пам’ять. Він також може планувати завдання, які будуть виконуватися в робочих. Режим планування за замовчуванням – FIFO (першим прийшов – першим вийшов). Як показано в крайній правій частині рис. 9, SparkContext підключається до єдиного координатора, який називається менеджером кластерів, який може бути головною схемою Spark, Mesos або YARN. Менеджер кластера розподіляє ресурси між робочими вузлами в кластері. Кожен робочий вузол може мати одного або більше виконавців, які виконують обчислення та зберігають дані для призначених завдань. Кожна програма отримує свої власні процеси, що виконуються і залишаються в актуальному стані протягом усієї роботи додатка та запускають завдання в кількох потоках.

4.2 Результати бенчмаркінга

У цьому розділі повідомляємо про результати експериментів для вибраних систем. Оцінюємо продуктивність впровадженого бенчмарка щодо згенерованих наборів даних. Зокрема, завдання бенчмарка включають

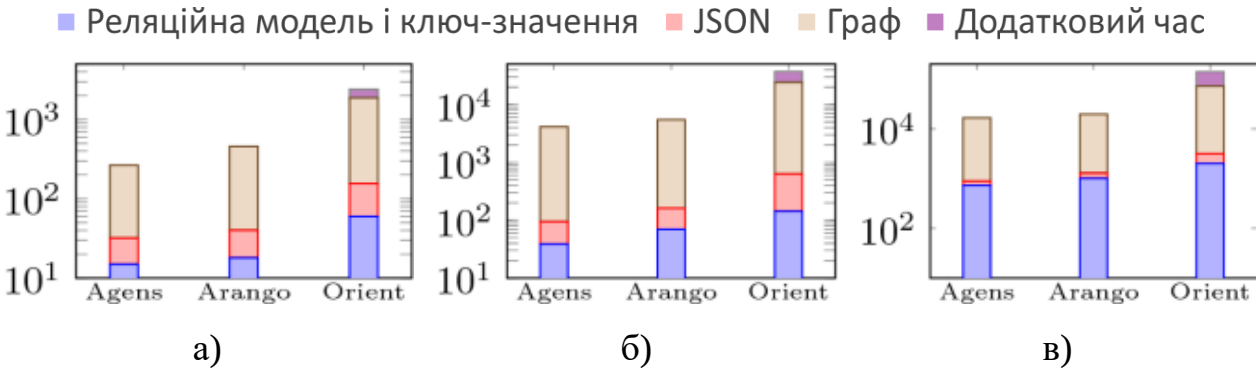
приймання даних, мультимодельну обробку запитів і транзакцій, а також розподілене виконання [13].

4.2.1 Налаштування

Для експериментального налаштування проводимо тестові експерименти в автономному режимі на машині з подвійним 6-ядерним процесором Xeon-E5649, 100 ГБ ОЗУ та 500 ГБ HDD. Клієнтська машина має 4-ядерний процесор i5-4590 з 16 ГБ оперативної пам’яті. Вибираємо три репрезентативні мультимодельні бази даних: OrientDB, AgensGraph і ArangoDB. На стороні клієнта розробляємо клієнтську програму Java, інтегровану з офіційним драйвером кожної БД. Усі тестові навантаження реалізовуємо в програмі. Проводимо розподілені експерименти на кластері з трьох екземплярів Amazon EC2. Вибираємо три репрезентативних мультимодельних кластери баз даних: OrientDB, ArangoDB і Spark. Перевірені сервери БД розгортаємо на екземплярах t3.large з 2 віртуальними процесорами та 8 ГБ оперативної пам’яті.

4.2.2 Поглинання даних

Результати завантаження трьох наборів даних показані на рисунку 4.4 (час обробки в логарифмічній шкалі). Для справедливого порівняння виконуємо всі утиліти імпорту за допомогою одного потоку (було виявлено, що OrientDB і AgensGraph не можуть підтримувати паралельне імпортування). Для кращої ілюстрації розділюємо час завантаження даних на чотири аспекти, тобто реляційний і ключ-значення, JSON, граф і додаткові витрати, які зображаються на графах у вигляді смуги з нагромадженням.



а) б) в)

Рисунок 4.4 – Час обробки для імпорту мультимодельних наборів даних з одним потоком в секундах: а) при масштабному коефіцієнті 1; б) при масштабному коефіцієнті 10; в) при масштабному коефіцієнті 30

Найбільш разючою особливістю результатів для часу завантаження є різниця в продуктивності OrientDB у порівнянні з AgensGraph і ArangoDB. Попри закриття служби та локальне завантаження даних, час завантаження є напрочуд високим (приблизно 20 годин для імпорту набору даних із MK30), а накладні витрати на створення посилань різко збільшуються у міру зростання даних (приблизно 8, 217, 1139 хвилин для трьох набори даних, відповідно). Це тому, що для реляційних даних потрібен час для створення унікального RID для запису фізичної позиції для кожного рядка. Також для даних JSON OrientDB має перетворити кожен напівструктурований об’єкт JSON в об’єкт ODocument, а для даних графа OrientDB використовує списки суміжності для зберігання відносин між усіма вузлами; таким чином, пошук за індексом необхідний під час вилучення кожного ребра [14].

Загалом, AgensGraph перевершує як OrientDB, так і ArangoDB в однопотоковому налаштуванні. В основному це пов’язано з тим, що мультимодельні дані можна безпосередньо імпортувати з файлів даних у реляційні таблиці шляхом масового завантаження (використовуємо мікропакети з 10 000 записів). Крім того, продуктивність ArangoDB порівнянна, оскільки вона також завантажує або таблиці CSV, або файли JSON в документи пакетами без будь-яких додаткових витрат. Крім того,

якщо завантажуються дані в ArangoDB за допомогою кількох потоків (зазвичай більш як чотири потоки), ArangoDB стає переможцем у завданні імпорту даних. Зокрема, використання шістнадцяти потоків для завантаження даних відповідає покращенню продуктивності в 4,8, 4,9, 5,2 для трьох наборів даних відповідно.

4.2.3 Наскрізна продуктивність запитів

У цьому розділі спочатку оцінюємо продуктивність мультимодельних запитів з кінця в кінець, а потім збираємо статистичну інформацію про ресурс, який використовується тестованими системами. Виконуємо запит із тими самими підпорядкованими параметрами щодо систем, щоб забезпечити однаковий результат кожного запуску. Використовуємо індекси за замовчуванням, які створені на основі первинних ключів, а вторинний індекс не створюється. Ставимо три запитання для оцінки: як порівнюється продуктивність у різних системах, наскільки добре масштабуються системи в міру збільшення вхідних даних, яке використання ресурсів системою під час виконання запитів?

На рисунку 4.5 показано час виконання в усіх запитах зі зміною розміру вхідних даних. Реалізуємо всі запити, визначені в бенчмарку, використовуючи їх мови запитів. Цікаво, що результати показують, що тестовані системи досягають різної продуктивності щодо чотирьох бізнес-категорій у робочих навантаженнях. Для двох точкових запитів до мультимодельних даних, а саме, Q1 у випадку використання індивідуального рівня та Q2 у варіанті використання розмовного рівня, тоді як ArangoDB та OrientDB досягають порівнянної продуктивності, AgensGraph працює набагато повільніше. Виявили, що AgensGraph може дуже швидко відповідати на окремі точкові запити, але при обробці мультимодельного точкового запиту стає набагато повільніше. Насамперед це пов’язано з окремим реляційним сховищем і сховищем графів в AgensGraph, що тягне за

собою додаткові витрати на передачу окремих результатів запиту до кінцевих результатів. ArangoDB на порядок швидше, ніж інші системи в Q3, яка об'єднує замовлення JSON з графом публікації разом із фільтром у вбудованому масиві JSON. Завдяки власному сховищу документів і розширеному оператору масиву, ArangoDB може виконувати Q3 оригінально та ефективно. OrientDB працює повільніше, тому що він повинен вирівняти вбудовану структуру, щоб виконати фільтрацію, що призводить до додаткових витрат. Дивовижне спостереження стосується AgensGraph, який використовує перехресне з'єднання для вирівнювання масиву JSON у випадку Q3, що призводить до великих накладних витрат, оскільки він перераховує замовлення з усіма зведеними масивами JSON.

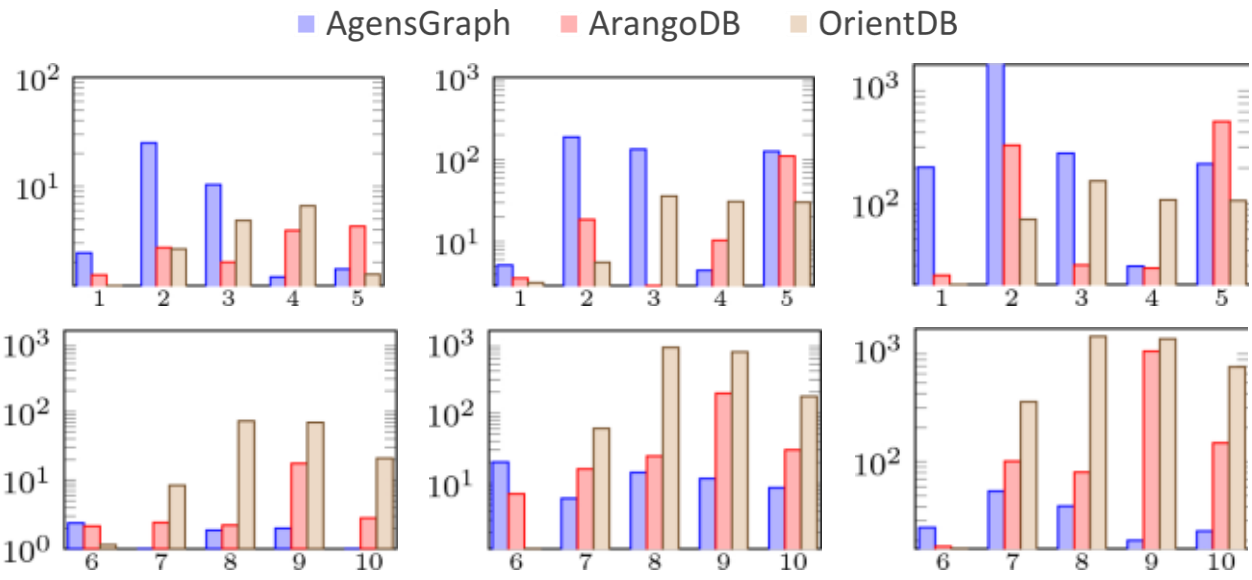


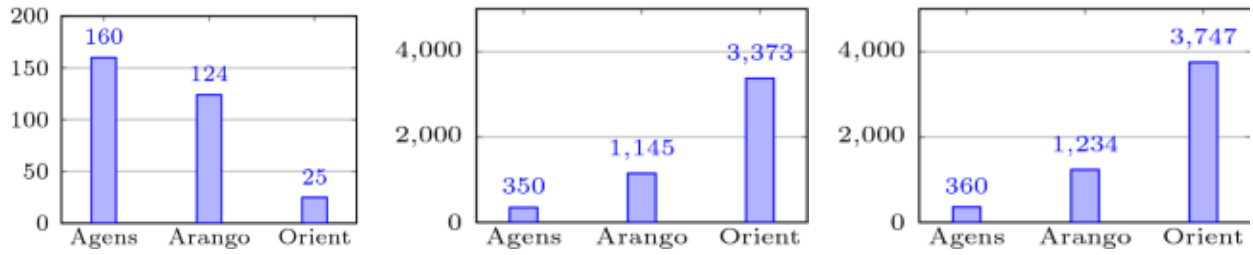
Рисунок 4.5 – Час обробки запитів в секундах: а) при масштабному коефіцієнті 1; б) при масштабному коефіцієнті 10; в) при масштабному коефіцієнті 30

У випадку використання громадського рівня (Q4-Q6), перше спостереження полягає в тому, що порядок операцій у цих запитах значно впливає на продуктивність. У випадку Q4, який починається з агрегації

документів, а потім обходу графа, AgensGraph і ArangoDB працюють краще, ніж OrientDB. У випадку Q5 і Q6, які починаються з обходу графа, а потім виконують агрегацію документів, OrientDB показує перевагу. Це означає, що продуктивність мультимодельного з'єднання залежить від їх основних моделей, тобто, AgensGraph спочатку є реляційною базою даних, ArangoDB спочатку є документно-орієнтованою базою даних, а OrientDB є нативною базою даних графів. Інше спостереження полягає в тому, що рівень обходу графа впливає на продуктивність AgensGraph і ArangoDB, коли рівень більше ніж 3, вони набагато повільніше, ніж OrientDB. Це пов'язано з тим, що обидва вони використовують рекурсивне з'єднання при обході графа, що не підходить для глибокого обходу графа. Результати для випадку використання комерційного рівня чітко демонструють, що AgensGraph перевершує ArangoDB і OrientDB. Для набору даних MK30 він приблизно в 9 і 25 разів швидше, ніж ArangoDB і OrientDB, відповідно. Це пов'язано з його таблично структурованим макетом даних, який є більш ефективним для умови «Group By». OrientDB найповільніше виконує ці аналітичні запити, хоча він також містить умову «Group By», що вказує на те, що сховище графів недостатньо потужне при складних агрегаціях через накладні витрати на повторювані зв'язки.

4.2.4 Мультимодельна продуктивність транзакцій ACID

Виконуємо дві окремі трансакції («New Order» та «New Payment») і змішану трансакцію, об'єднавши їх у тестованих системах. Трьом БД вдається відкотити недійсні транзакції та зафіксувати дійсні, що означає, що властивості ACID для двох мультимодельних транзакцій гарантуються. Рисунок 4.6 ілюструє 99-процентильну затримку продуктивності транзакції. Результати показують, що AgensGraph і ArangoDB краще справляються з великою кількістю транзакцій («New Payment»), а OrientDB ефективнішим у виконанні транзакцій, пов'язаних із читанням («New Order»).



а) б) в)

Рисунок 4.6 – Продуктивність транзакції в мілісекундах: а) «New Order»; б) «New Payment»; в) змішаної

4.2.5 Розподілена мультимодельна обробка запитів

Проводимо розподілені експерименти на кластері з трьох екземплярів Amazon EC2. Використовуємо набір даних із масштабним коефіцієнтом MK10. Перевірені сервери БД розгортаємо на екземплярах t3.large з 2 віртуальними процесорами та 8 ГБ оперативної пам'яті. Додаємо останню версію Spark для порівняння з двома розподіленими мультимодельними базами даних, а саме ArangoDB і OrientDB. Зокрема, кожен робочий у кластері Spark з трьома вузлами за замовчуванням має 1 ГБ робочої оперативної пам'яті та 1 виконавця. Використовуємо HDFS як сховище для Spark, у якому коефіцієнт реплікації встановлюємо на 3, і використовуємо Spark SQL для реалізації перевірених завдань, кількість розділів shuffle spark.sql.shuffle.partitions налаштовуємо зі значенням за замовчуванням 200. Вимикаємо кеш результатів у всіх системах для справедливого порівняння.

Розроблюємо три завдання, пов'язані з мультимодельною розподіленою обробкою запитів, яка включає завдання вибору, об'єднання та агрегації. Ці три завдання походять з оригінального запиту бенчмарка Q1, Q5 та Q8 відповідно.

Завдання вибору полягає в пошуку пов'язаних даних із різних джерел даних за ідентифікатором особи @id. Запит в Orient SQL наведений в прикладі 4.1.

```
SELECT profile, order, feedback FROM Customer where id=@id;
```

Приклад 4.1 – Запит в Orient SQL

Завдання об’єднання складається з об’єднань між трьома мультимодельними об’єктами (реляційного «person», графу «knows», і JSON «order»). Реалізація в Orient SQL наведена в прикладі 4.2.

```
SELECT person, order.items
FROM (TRAVERSE ('Knows') FROM person WHERE id=@id and $depth<3)
WHERE @brand in order.items.brand
```

Приклад 4.2 – Реалізація в Orient SQL

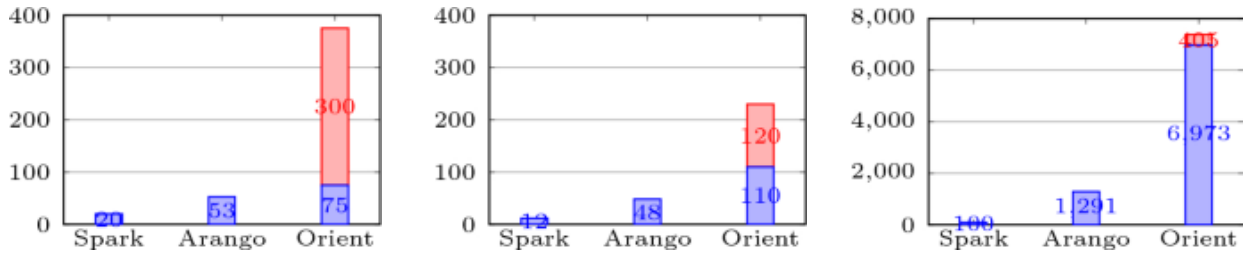
Завдання агрегації складається з двох підзадач, які виконують складну агрегацію трьох сутностей («product», «PostHasTag» і «order»). Перша частина завдання – знайти 10 найпопулярніших куплених товарів, які мають найбільшу загальну кількість. Потім друге завдання – підрахувати їх популярність у соцмережах, підрахувавши пов’язані дописи. Реалізація цього завдання наведена в прикладі 4.3.

```
SELECT items, count(items)
FROM (SELECT * FROM order) UNWIND order.items)
GROUP BY items.id
ORDER BY count DESC limit 10;
SELECT In('PostHasTag').size() as popularity
FROM Product WHERE id in items.id
ORDER BY popularity DESC;
```

Приклад 4.3 – Реалізація завдання агрегації

На рисунку 4.7 показано час завантаження для імпорту табличних, JSON та графових даних відповідно. Результати показують, що Spark приблизно в 10 разів швидше, ніж ArangoDB, і в 50 разів швидше, ніж OrientDB. Це пояснюється тим, що в Spark головний вузол просто копіює

кожен файл даних з локального диска в екземпляр HDFS, а потім розподіляє репліки на інші підпорядковані вузли в кластері. OrientDB має високі накладні витрати на імпорт даних, які зображаються у вигляді червоних верхніх сегментів панелі стека. На додаток до накладних витрат, таких як створення посилань, виявлено, що OrientDB завантажує дані до свого кластера послідовно, тобто вузол за вузлом. Таким чином, зі збільшенням розміру вузла час завантаження буде збільшуватися пропорційно. На відміну від цього, імпортування даних у кластер ArangoDB можна виконувати паралельно між вузлами.

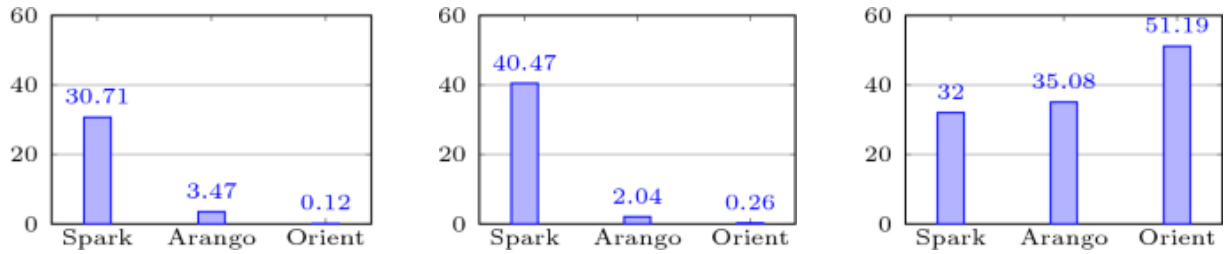


а) б) в)

Рисунок 4.7 – Час обробки імпорту мультимодельних наборів даних у секундах: а) табличних даних; б) даних JSON; в) графових даних

Час обробки для виконання завдань запиту показано на рисунку 4.8. Для завдань вибору та приєднання Spark приблизно в 13 разів повільніше, ніж ArangoDB, і повільніше на три порядки, ніж OrientDB. Причина має дві складки. Спочатку Spark має просканувати всі файли HDFS, щоб створити RDD, а потім виконати запит через відсутність підтримки індексів, хоч обидві мультимодельні бази даних мають спеціальне сховище та індекси за замовчуванням після процесу імпорту. Наприклад, ArangoDB має двійкове сховище JSON з основним індексом для кожного документа. OrientDB має локальне сховище з розбитим на сторінки на диску з фізичними ідентифікаторами вказівників. Ці структури дозволяють обом базам даних значно прискорити виконання запиту за допомогою сканування індексу та

об’єднання без повного сканування наборів даних. По-друге, для виконання складної задачі об’єднання, наприклад, об’єднання між друзями з двома стрибками в графі «knows» і «orders», обидві мультимодельні бази даних можуть виконувати об’єднання локально на кожному вузлі (зокрема для OrientDB, кожен вузол має весь набір даних у режимі репліки; таким чином, об’єднання повністю виконується в одному вузлі). У Spark SQL не можна уникнути з’єднання в випадковому порядку, оскільки не можна вручну керувати розподільником для доступу до локальності даних. Що стосується завдання агрегації, то попри накладні витрати на завантаження даних, Spark дещо перевершує ArangoDB і OrientDB, оскільки Spark SQL може скористатися перевагами кількох розділів RDD, щоб прискорити групу шляхом агрегації.



а) б) в)

Рисунок 4.8 – Розподілений час обробки мультимодельного запиту в секундах: а) завдання вибору; б) завдання приєднання; в) завдання агрегації

4.2.6 Підсумок оцінки діяльності

Загалом, експериментальні результати показують, що для імпорту даних AgensGraph приблизно в 1,5 і 9 разів швидше, ніж ArangoDB та OrientDB в однопотокових налаштуваннях для введення MK30, відповідно. Основним вузьким місцем OrientDB є створення посилань на графи, що витрачає приблизно 20 годин на отримання даних. Крім того, ArangoDB може значно прискорити імпорт даних у 5 разів за 16 потоків. Щодо обробки

запитів, то OrientDB є перевершеним для точкових запитів (Q1 і Q2) і запитів, орієнтованих на шлях (Q5 і Q6). Зокрема, OrientDB в середньому в 3,6 і 36 разів швидше, ніж ArangoDB та AgensGraph для введення МК30 щодо точкових запитів, відповідно. Основним вузьким місцем AgensGraph для точкових запитів є отримання графа та реляційних даних без з'єднання. Що стосується запитів, орієнтованих на шлях, OrientDB у 2 і 4 рази швидше, ніж AgensGraph та ArangoDB, відповідно. ArangoDB найкращий у фільтрації документів та запитах на приєднання (Q3 і Q4), який в середньому в 4,5 і 5 разів швидше, ніж OrientDB та AgensGraph для введення МК30. AgensGraph перевершує інші у виконанні складних запитів агрегації (Q7-Q10), які в середньому в 9 і 25 разів швидше, ніж ArangoDB та OrientDB для введення МК30 відповідно. Для обробки транзакцій AgensGraph приблизно в 3,4 і 10 разів швидше, ніж ArangoDB та OrientDB, коли обидві транзакції виконуються разом.

Що стосується розподіленої мультимодельної обробки запитів, помічаємо, що Spark приблизно в 10 разів швидше, ніж ArangoDB, і в 50 разів швидше, ніж OrientDB при імпортуванні даних. Для завдань вибору та приєднання OrientDB є переможцем, що приблизно в 13 разів швидше, ніж ArangoDB, і в 187 разів швидше, ніж Spark SQL, відповідно. Для завдання агрегації Spark SQL в 1,6 раза швидше, ніж OrientDB, використовуючи кілька розділів; продуктивність ArangoDB порівнянна з Spark SQL.

Завдяки детальним експериментам перевірили доцільність та застосовність створеного бенчмарка. Запропонований бенчмарк також демонструє багато переваг у бенчмаркінгу мультимодельних баз даних. По-перше, багаті формати даних і скошені розподіли, надані бенчмарку, дають можливість змоделювати складний реалістичний сценарій. Навіть якщо база даних може не підтримувати всі типи даних у бенчмарку, вона все одно може використовувати його для часткової оцінки. Це вказує на те, що він є загальним бенчмарком, який може підтримувати широкий спектр баз даних, які варіюються від SQL/NoSQL мультимодельних баз даних до систем

великих даних. По суті, всі бази даних у таблиці 1.1 можуть бути реалізовані в ньому з перетворенням даних або без нього. Крім того, складні робочі навантаження допомагають визначити кілька вузьких місць у перевірених базах даних під час оцінки обробки запитів. Таке розуміння навряд чи можна отримати за допомогою інших тестів, оскільки їм бракує складних і значущих мультимодельних робочих навантажень. І останнє, але не менш важливе, курування параметрів гарантує, що оцінка запиту є цілісною та надійною. Підсумовуючи, детальний аналіз за допомогою створеного бенчмарку буде представляти особливий інтерес для розробників ядер движків, системних адміністраторів та дослідників мультимодельних баз даних.

Також маємо наступні ключові спостереження з порівняльного оцінювання: на додаток до безіндексної структури суміжності, реляційне сховище та сховище документів також можуть ефективно реалізувати LPG. Це підтверджується продуктивністю запитів AgensGraph і ArangoDB. Тестовані мультимодельні бази даних можуть підтримувати мультимодельні об'єднання, такі як graph-JSON, JSON-relational та graph-relational. Однак у них відсутні конкретні алгоритми для оптимізації плану виконання. Тестовані бази даних можуть підтримувати мультиоб’єктні та мультимодельні транзакції ACID в автономному режимі, але вони не можуть підтримувати розподілені транзакції ACID. Доступ до локальних даних ArangoDB і OrientDB є більш гнучким, ніж Spark, оскільки вони можуть використовувати функцію шардингу для розділення даних із підтримкою глобального індексу.

ВИСНОВКИ

Метою даної кваліфікаційної роботи була розробка методу та засобів бенчмаркінга мультимодельної бази даних. Для її реалізації спочатку було розроблено новий генератор даних, який надає корельовані дані в різних моделях даних, запропоновано нову структуру для генерування даних для моделювання поведінки клієнтів у соціальній комерції, а також реалізовано генератор поверх Spark SQL, щоб забезпечити ефективність та масштабованість.

Наступним етапом написання бакалаврської роботи була розробка набору робочих навантажень із кількома моделями, включаючи десять запитів і дві транзакції з технічної та бізнес-точок зору.

Далі було визначено нову проблему під назвою «мультимодельне курування параметрів», спрямовану на введення різноманітності запитів із підпорядкованими параметрами до робочих навантажень. Запропоновано алгоритм на основі зразка для розумного вибору параметрів для запиту бенчмаркінга.

Фінальним етапом написання даної роботи було проведення комплексної оцінки чотирьох мультимодельних баз даних: ArangoDB, OrientDB, AgensGraph і Spark SQL. Аналітично прозвітовано про порівняння ефективності та отримані дані.

Результатом виконання поставлених завдань став бенчмарк мультимодельної бази даних, який складається зі змішаної моделі даних, масштабованого мультимодельного генератора даних і набору робочих навантажень, включаючи мультимодельну агрегацію, об’єднання та транзакцію.

Поточні системи забезпечують хорошу підтримку для управління даними з кількома моделями, але вони також відкривають нові можливості для подальшого вдосконалення та майбутніх досліджень. Подальші

дослідження полягають у впровадженні гнучкості в генерацію даних, оскільки схему даних і модель даних у реальному додатку можна динамічно змінювати, в оцінці продуктивності мультимодельних баз даних щодо різних стратегій розподілу, а також в оцінці продуктивності мультимодельних баз даних щодо оптимізації запитів різних систем, наприклад, оптимізатор плану виконання, індекси, кеш результатів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stonebraker M. The Case for Polystores – ACM SIGMOD Blog [Електронний ресурс] / Michael Stonebraker // ACM SIGMOD Blog. – Режим доступу: http://wp.sigmod.org/?p=1629 (дата звернення: 10.05.2022). – Назва з екрана.

2. TPC-DI [Електронний ресурс] / Meikel Poess [та ін.] // Proceedings of the VLDB Endowment. – 2014. – Т. 7, № 13. – С. 1367–1378. – Режим доступу: https://doi.org/10.14778/2733004.2733009 (дата звернення: 10.05.2022). – Назва з екрана.

3. Carey M. J. The 007 Benchmark [Електронний ресурс] / Michael J. Carey, David J. DeWitt, Jeffrey F. Naughton // ACM SIGMOD Record. – 1993. – Т. 22, № 2. – С. 12–21. – Режим доступу: https://doi.org/10.1145/170036.170041 (дата звернення: 10.05.2022). – Назва з екрана.

4. Benchmarking cloud serving systems with YCSB [Електронний ресурс] / Brian F. Cooper [та ін.] // the 1st ACM symposium, Indianapolis, Indiana, USA, 10–11 черв. 2010 р. – New York, New York, USA, 2010. – Режим доступу: https://doi.org/10.1145/1807128.1807152 (дата звернення: 10.05.2022). – Назва з екрана.

5. The LDBC Social Network Benchmark [Електронний ресурс] / Orri Erling [та ін.] // SIGMOD/PODS'15: International Conference on Management of Data, Melbourne Victoria Australia. – New York, NY, USA, 2015. – Режим доступу: https://doi.org/10.1145/2723372.2742786 (дата звернення: 10.05.2022). – Назва з екрана.

6. BigBench [Електронний ресурс] / Ahmad Ghazal [та ін.] // the 2013 international conference, New York, New York, USA, 22–27 черв. 2013 р. – New York, New York, USA, 2013. – Режим доступу: https://doi.org/10.1145/2463676.2463712 (дата звернення: 10.05.2022). – Назва з екрана.

7. XMark [Електронний ресурс] / Albrecht Schmidt [та ін.] // VLDB '02: Proceedings of the 28th International Conference on Very Large Databases. – [Б. м.], 2002. – С. 974–985. – Режим доступу: https://doi.org/10.1016/b978-155860869-6/50096-2 (дата звернення: 10.05.2022). – Назва з екрана.

8. Qiao S. RBench [Електронний ресурс] / Shi Qiao, Z. Meral Özsoyoğlu // SIGMOD/PODS'15: International Conference on Management of Data, Melbourne Victoria Australia. – New York, NY, USA, 2015. – Режим доступу: https://doi.org/10.1145/2723372.2746479 (дата звернення: 10.05.2022). – Назва з екрана.

9. Fader P. S. RFM and CLV: Using Iso-Value Curves for Customer Base Analysis [Електронний ресурс] / Peter S. Fader, Bruce G. S. Hardie, Ka Lok Lee // Journal of Marketing Research. – 2005. – Т. 42, № 4. – С. 415–430. – Режим доступу: https://doi.org/10.1509/jmkr.2005.42.4.415 (дата звернення: 10.05.2022). – Назва з екрана.

10. Huang Z. From e-commerce to social commerce: A close look at design features [Електронний ресурс] / Zhao Huang, Morad Benyoucef // Electronic Commerce Research and Applications. – 2013. – Т. 12, № 4. – С. 246–259. – Режим доступу: https://doi.org/10.1016/j.elerap.2012.12.003 (дата звернення: 10.05.2022). – Назва з екрана.

11. Search Result Diversity Evaluation Based on Intent Hierarchies [Електронний ресурс] / Xiaojie Wang [та ін.] // IEEE Transactions on Knowledge and Data Engineering. – 2018. – Т. 30, № 1. – С. 156–169. – Режим доступу: https://doi.org/10.1109/tkde.2017.2729559 (дата звернення: 10.05.2022). – Назва з екрана.

12. Mckay M. D. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code [Електронний ресурс] / M. D. Mckay, R. J. Beckman, W. J. Conover // Technometrics. – 2000. – Т. 42, № 1. – С. 55–61. – Режим доступу: https://doi.org/10.1080/00401706.2000.10485979 (дата звернення: 10.05.2022). – Назва з екрана.

13. Хомич В. М. Метод та засоби бенчмаркінга мультимодельної бази даних / Віктор Михайлович Хомич, Олександр Олександрович Можаєв // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : тези доп. дванадцатої міжнар. науково-техн. конф., Баку – Харків – Жиліна, 27–28 квіт. 2022 р. – Харків, 2022. – С. 53.

14. Barkovska O. Research of the text processing methods in organization of electronic storages of information objects [Електронний ресурс] / Olesia Barkovska, Viktor Khomych, Oleksandr Nastenko // Innovative Technologies and Scientific Solutions for Industries. – 2022. – 1 (19). – С. 5–12. – Режим доступу: https://doi.org/10.30837/itssi.2022.19.005 (дата звернення: 10.05.2022). – Назва з екрана.