Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження СКБД для обробки даних у оперативній пам'яті
(тема)

Виконав:

Студент 2 курсу, групи ІПЗм-22-1

Ващенко М. С.
(прізвище, ініціали)

Спеціальності 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

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

Керівник доц. кафедри ПІ Кравець Н.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф._____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо- наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____»_____20_____ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Ващенко Микиті Сергійовичу*

(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження СКБД для обробки даних у оперативній пам'яті* затверджена наказом університету від *29 березня 2024 р. № 250 Ст*

2. Термін подання студентом роботи до екзаменаційної комісії *18 червня 2024 р.*

3. Вихідні дані до роботи *СКБД для обробки даних у оперативній пам'яті, критерії їх оцінки, структури рішень, графи, Б-дерева, Геш-таблиці, Скіп-листи.*

4. Перелік питань, що потрібно опрацювати у роботі *вступ, аналіз предметної галузі, аналіз існуючих інструментів, постановка задачі, дослідження IMDb, аналіз результатів проведених експериментів.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.03.2024	Виконано
2	Постановка задачі	10.04.2024	Виконано
3	Проведення дослідження	01.05.2024	Виконано
4	Підготовка пояснювальної записки	15.05.2024	Виконано
6	Підготовка презентації та доповіді	18.05.2024	Виконано
7	Перевірка на академічний плагіат	08.06.2024	Виконано
8	Нормоконтроль	08.06.2024	Виконано
9	Рецензування	13.06.2024	Виконано
10	Занесення диплома в електронний архів	15.06.2024	Виконано
11	Допуск до захисту у зав. кафедри	16.06.2024	Виконано

Дата видачі завдання 25.03.2024

Студент _____ Ващенко М. С.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Кравець Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Робота містить: 80 с., 11 рис., 13 табл., 21 джер.

Б-ДЕРЕВО, ГРАФИ, ОПЕРАТИВНА ПАМ’ЯТЬ, СКБД, СКІП-ЛИСТИ, ГЕШ-ТАБЛИЦІ, APACHE IGNITE, CASSANDRA, IMDb, MEMSQL, REDIS, VOLTDB.

Об’єктом дослідження є системи керування базами даних (СКБД), що призначені для обробки даних у оперативній пам'яті.

Метою роботи є проведення дослідження СКБД для обробки даних у оперативній пам’яті та експерименту в виявленні найбільш влучного підходу для зберігання і доступу до даних, що зберігаються в оперативній пам’яті.

Методи розробки базуються на таких технологіях, як Java.

В результаті роботи було проведено ретельний аналіз існуючих СКБД для обробки даних у оперативній пам’яті та виділені їх сильні та слабкі сторони. Проведено експеримент, в рамкам якого було реалізовано та досліджено структури даних, що використовуються в системах СКБД в оперативній пам’яті.

B-TREE, GRAPH, RAM, DBMS, SKIP LISTS, HASHTABLE, APACHE IGNITE, CASSANDRA, IMDb, MEMSQL, REDIS, VOLTDB.

The object of investigation is the database management system (DBMS), which is used for processing data from RAM.

The method of work is to conduct research on the DBMS for processing data in RAM and experiment to determine the best approach for saving and accessing data that is stored in RAM

Development methods are based on technologies such as Java.

As a result of the work, a detailed analysis of the existing DBMS for processing data in RAM was carried out and their strengths and weaknesses were seen. RAM.

Я, Ващенко Микита Сергійович студент групи ІПЗм-22-1 здобувач вищої освіти на другому (магістерському) рівні кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження СКБД для обробки даних у оперативній пам’яті», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ACID – atomicity,consistency,isolation, durability

DAG – directed acyclic graph

IMDb – in-memory database

PGA – program global area

RAM – Random Access Memory

RDD – resilient distributed dataset

Redis – Remote Dictionary Server

SAP HANA – High Performance Analytic Appliance

SGA – Shared Global Area

SSD – Solid State Drive

ВСТУП

На початку 21 століття було помітно збільшення обсягів даних, які оброблялися та зберігалися компаніями та організаціями.

Традиційні системи керування базами даних (СКБД) здебільшого працювали на основі дискової пам'яті, що призводило до значних затримок при доступі до даних. З цією проблемою зіштовхувалися індустрії, де потрібна була швидка та ефективна обробка великого обсягу інформації, такі як фінанси, телекомунікації, торгівля та інші.

Однією з ключових подій, яка визначила початок активного розвитку in-memory database (IMDb), було представлення SAP HANA в 2010 році. SAP HANA стала однією з перших широко використовуваних in-memory баз даних і надала можливість зберігати та обробляти дані безпосередньо в оперативній пам'яті, що суттєво прискорило швидкодію системи [1]. Цей крок відкрив нові перспективи для бізнес-додатків, які вимагали миттєвого доступу до актуальних даних.

З появою технологій зберігання даних в оперативній пам'яті в індустрії баз даних відзначається широке впровадження цих технологій, що призвело до значних покращень та інновацій. Розвиток технологій відкрив нові можливості в різних сферах, таких як фінансовий сектор і телекомунікації.

СКБД вже займають ключову роль у забезпеченні швидкого та надійного доступу до інформації. Проте зі зростанням обсягів даних та появою вимог до обробки їх в режимі реального часу, використання IMDb стає ключовою складовою.

Актуальність цього дослідження визначається необхідністю оптимізації процесів управління даними у реальному часі, щоб відповідати високим вимогам продуктивності та швидкості доступу до інформації

У цьому дослідженні розглядаються СКБД, що призначені для обробки даних у оперативній пам'яті. Під час аналізу виводяться основні критерії оцінювання та формується модель порівняння. Також розглядаються їх підходи до збереження даних у оперативній пам'яті.

В цій кваліфікаційній роботі поставлено за мету ретельно оцінити переваги,

недоліки та перспективи використання СКБД для обробки даних у реальному часі, та виявлення найбільш влучного підходу для зберігання і доступу до даних, що зберігаються в оперативній пам'яті [2].

Для досягнення потрібного результату необхідно розглянути та вирішити такі питання:

- проаналізувати існуючі IMDb;
- визначитись з методами та критеріями для порівняння IMDb;
- змоделювати умови для проведення експериментів для оцінки IMDb;
- отримані дані використати для виміру обраних метрик для порівняння IMDb;
- порівняти продуктивності структур даних, які використовуються IMDb;
- проаналізувати отримані результати;
- сформулювати рекомендації по використанню IMDb.

Об’єктом дослідження є ефективність та доцільність використання СКБД, що призначені для обробки даних у оперативній пам'яті.

Предметом дослідження є СКБД для обробки даних у оперативній пам’яті.

Методи дослідження включають вимірювання ефективності IMDb за встановленими критеріями та обчислення кожного показника за допомогою запропонованих математичних формул. А також проведення експерименту і порівняння продуктивності структур даних, що використовуються в IMDb.

Результати роботи можна успішно використовувати для максимально оптимального вибору IMDb в залежності від конкретних вимог та пріоритетів користувача чи організації, а також для подальшого дослідження IMDb.

Результати кваліфікаційної роботи опубліковані у матеріалах і тезах конференцій [3], [4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

IMDb переважно використовуються в управлінні великими обсягами даних та бізнес-аналітиці, де є потреба у сховищі даних та швидкому доступі до них. Ці бази даних працюють швидше, порівняно з традиційними базами даних, оскільки вони зберігають дані безпосередньо в оперативній пам'яті комп'ютера та потребують менше інструкцій для доступу до даних для запису та читання. Ще однією перевагою системи in-memory баз даних є її застосування в реальних вбудованих системах, які мають обмежені ресурси, низькі обчислювальні можливості процесора та вимагають невеликої кількості пам'яті. Крім того, велика кількість даних, що генерується різними галузями, падаючі ціни на оперативну пам'ять та швидка обробка даних – це лише кілька з основних факторів, які очікується, що прискорять розвиток ринку IMDb по всьому світу.

Однак відсутність стандартів та обмеження пам'яті, а також проблеми з безпекою даних та нестабільність пам'яті, є деякими обмежуючими факторами, які можуть стримувати ріст ринку IMDb та ускладнювати впровадження рішень [5].

1.1 Використання IMDb в різних галузях

Розвиток in-memory технологій відкрив багато нових можливостей у різних галузях. У фінансовому секторі, наприклад, банки можуть швидше обробляти транзакції та аналізувати ризики в реальному часі. Це забезпечує більш ефективне управління фінансовими операціями, підвищуючи точність прогнозів та мінімізуючи ризики шахрайства. Банки також можуть проводити миттєвий аналіз великих обсягів фінансових даних, що дозволяє їм приймати обґрунтовані рішення швидше.

У телекомунікаціях інтерактивні системи можуть швидше реагувати на великий потік даних від користувачів. Наприклад, оператори можуть в реальному часі відстежувати та оптимізувати трафік мережі, підвищуючи якість обслуговування клієнтів. Це також дозволяє оперативно реагувати на аварійні ситуації та збої в мережі, забезпечуючи безперервність надання послуг.

Важливим використанням є також сфера наукових досліджень, де швидкий доступ до великих обсягів даних дозволяє проводити складні аналізи та моделювання. Наприклад, у біоінформатиці IMDb допомагають дослідникам швидше аналізувати геномні дані, що сприяє прискоренню відкриттів у галузі медицини та біотехнологій. У фізиці та хімії IMDb використовуються для моделювання складних процесів та реакцій, дозволяючи вченим швидше отримувати результати експериментів [6].

1.2 Технічні переваги IMDb

IMDb мають кілька ключових технічних переваг, що робить їх привабливими для різних застосувань.

По-перше, завдяки зберіганню даних в оперативній пам'яті, час доступу до даних значно зменшується, що дозволяє обробляти запити у реальному часі.

По-друге, відсутність необхідності звертатися до дискової пам'яті значно знижує затримки під час читання та запису даних.

По-третє, IMDb ефективно використовують апаратні ресурси, забезпечуючи високу продуктивність навіть на обмежених апаратних платформах. Нарешті, IMDb можуть бути інтегровані з різними програмними системами та платформами, що робить їх універсальним рішенням для різних галузей.

1.3 Недоліки та обмеження IMDb

Незважаючи на численні переваги, IMDb мають також кілька недоліків та обмежень.

По-перше, висока вартість оперативної пам'яті може зробити впровадження IMDb дорогим рішенням, особливо для великих обсягів даних.

По-друге, фізичні обмеження оперативної пам'яті можуть стати проблемою для обробки надзвичайно великих обсягів даних.

По-третє, оскільки дані зберігаються в оперативній пам'яті, існує ризик їх втрати у випадку збою живлення або інших технічних проблем.

Нарешті, недостатня стандартизація IMDb може ускладнити їх інтеграцію з

існуючими системами та платформами.

1.4 Порівняння звичайних СКБД та IMDb

СКБД представляють собою різноманітні рішення, які були розроблені та оптимізовані для вирішення різноманітних завдань. Вони взаємодіють із різними типами пам'яті з метою оптимізації своєї функціональності та забезпечення ефективного управління даними. Наприклад, Oracle використовує різні компоненти пам'яті, такі як shared pool, buffer cache, redo log buffer, large pool, Java pool, SGA (Shared Global Area) та PGA (Program Global Area), кожен з яких відповідає за конкретні аспекти обробки даних та забезпечує оптимальні умови для виконання запитів і операцій [3].

Однак, коли розглядається питання зберігання даних, загалом існують дві ключові стратегії, які визначаються використанням RAM (Random Access Memory) або SSD (Solid State Drive). Ці стратегії можна розглядати як дві відповідні групи підходів до забезпечення доступу до даних в СКБД.

При порівнянні технологій зберігання даних в RAM та SSD застосовуються різноманітні критерії. Результати порівняння цих критеріїв подано у таблиці 1.1, яка надає комплексний огляд технологій та їх ефективності в контексті конкретного використання.

Таблиця 1.1 – Порівняльна характеристика RAM та SSD

Характеристики	RAM	SSD
1	2	3
Швидкість доступу	Дуже висока, вимірюється в наносекундах	Низька відносно RAM, вимірюється в мікросекундах.
Швидкість передачі даних	Велика, виражається в гігабайтах за секунду.	Середня, виражається в мегабайтах за секунду.
Тип доступу	Прямий доступ, безперервний, використовується для зберігання тимчасових даних під час роботи програм	Зазвичай послідовний або випадковий доступ до даних, використовується для зберігання даних на постійному носії.

Кінець таблиці 1.1

1	2	3
Втрати даних	Втрати при вимкненні живлення, тимчасове сховище	Зберігає дані після вимкнення живлення, постійне сховище
Використання	Використовується для тимчасового зберігання активних даних та запущених програм.	Використовується для постійного зберігання даних та програм.
Ціна	Висока	Відносно низька, на 2 порядки нижче за RAM

1.5 Комбінація звичайних СКБД та IMDb

Комбінація In-memory баз даних з традиційними базами даних в сучасних системах управління даними стає все більш популярною стратегією для оптимізації продуктивності та використання ресурсів. Ця гібридна модель дозволяє максимально використовувати переваги обох типів баз даних залежно від конкретних завдань та потреб.

Традиційні БД, які зберігають дані на дисках, можуть бути ефективними для роботи з великими обсягами даних, але іноді повільніші при доступі до них через диск. Сполучення їх IMDb [7], де дані зберігаються безпосередньо в оперативній пам'яті, дозволяє отримати значний приріст у швидкодії та реактивності. Наприклад, традиційні БД можуть використовуватися для зберігання довгострокових архівних даних, тоді як In-memory бази даних використовуються для обробки та аналізу великих обсягів даних у режимі реального часу. Такий підхід дозволяє оптимізувати роботу з інформацією, забезпечуючи швидкий доступ до неї за допомогою In-memory технологій, а також забезпечує довгострокове зберігання за допомогою традиційних методів.

При цьому важливо враховувати, що комбінування цих двох типів баз даних може вимагати додаткового управління та налагодження для забезпечення сумісності та взаємодії між ними. Така гібридна стратегія може бути особливо корисною для підприємств, які потребують оптимального використання якісно різних видів даних.

1.6 Актуальність теми дослідження

Перспективи розвитку IMDb обіцяють подальше вдосконалення у швидкості, ефективності та масштабованості. Інтеграція in-memory технологій з іншими новаторськими напрямками, такими як штучний інтелект, машинне навчання та аналіз даних, може призвести до створення ще більш потужних інструментів для прийняття рішень в режимі реального часу.

Декілька ключових факторів, таких як значний обсяг генерованих даних у різних галузях, зниження вартості оперативної пам'яті та швидка обробка інформації, визначають стратегічну роль у розвитку світового ринку IMDb. Крім того, варто звернутися до сутності кешування в контексті in-memory технологій. Кешування в даному випадку дозволяє ефективно зберігати та швидко отримувати доступ до найбільш актуальних даних, які часто використовуються. Це підвищує швидкість доступу до інформації та реакцію системи на запити користувачів, що робить процес обробки даних у режимі реального часу ще ефективнішим.

Також важливо відзначити великий потенціал комбінації in-memory технологій з традиційними базами даних. Таке поєднання дозволяє оптимально використовувати переваги обох підходів: швидкість та надійність in-memory, а також гнучкість та комплексність традиційних систем. Це створює ефективну симбіоз між збереженням та обробкою даних, сприяючи забезпеченню оптимальної продуктивності та високої якості обробки інформації.

В сфері застосування очікується, що сегмент транзакцій буде мати найбільший обсяг ринку, особливо в умовах росту електронної комерції та використання безготівкових платіжних систем по всьому світу. З точки зору моделі обробки передбачається, що модель розгортання "он-преміс" утримає більший відсоток ринку завдяки конфіденційності та захисту даних організацій. Модель обробки на вимогу передбачається мати найвищий темп зростання завдяки гнучкості, ефективності та доступності для кінцевого споживача.

2 АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТІВ

2.1. Аналіз існуючих IMDb

На даний момент існують сотні різних баз даних IMDb, але всі їх можна поділити на категорії за організацією даних. Далі наведемо ці категорії та приклади найпопулярніших IMDb, що використовують кожен з підходів.

2.1.1 In-memory key-value stores

In-memory key-value stores використовують Геш-таблиці (Hashtable) для зберігання пар "ключ-значення" в пам'яті, забезпечуючи швидкий час доступу. Вони застосовуються для кешування, зберігання сесійних даних та реалізації черг повідомлень завдяки своїй простоті і швидкості доступу. Одним із найпопулярніших інструментів у цій категорії є Redis.

Redis (Remote Dictionary Server) – це надзвичайно швидка in-memory база даних з відкритим вихідним кодом, що базується на структурі даних типу ключ-значення. Redis підтримує ряд складних типів даних, включаючи строки, Геші, списки, множини та сортовані множини. Основна структура даних у Redis зберігається у вигляді Геш-таблиці, яка складається з масиву "відер" (buckets) [8], кожне з яких містить список зв'язаних елементів для уникнення колізій. Ця Геш-таблиця зберігається в оперативній пам'яті, забезпечуючи миттєвий доступ до даних.

Redis працює в однопотоковому режимі, що означає, що всі операції виконуються послідовно. Можливість підтримки складних структур, таких як списки та множини, в Redis реалізована за допомогою комбінації Скіп-листів та Геш-таблиць. Це дозволяє Redis виконувати операції з низькою затримкою та високою пропускною здатністю. Він також надає потужні можливості для кешування та зберігання тимчасових даних, що значно підвищує швидкодію додатків.

Крім того, Redis підтримує різні механізми реплікації, що забезпечують високу доступність та стійкість. Redis дозволяє виконувати скрипти на стороні сервера за допомогою вбудованої підтримки мови Lua, що дає можливість

реалізовувати складні операції над даними без необхідності передачі їх на клієнта.

2.1.2 In-memory columnar databases

In-memory columnar databases є потужним інструментом для обробки великих обсягів даних з високою швидкістю, особливо у контексті аналітичних задач. Ці бази даних зберігають дані у колонковому форматі, що забезпечує швидкий доступ до великих наборів даних, це дозволяє обробляти лише необхідні колонки замість всього рядка.. Вони використовуються для бізнес-аналітики та обробки великих даних. Одним із провідних представників цього типу баз даних є SAP HANA.

SAP HANA (High Performance Analytic Appliance) – це платформа для обробки даних в реальному часі, яка використовує колонкове зберігання для досягнення високої швидкості обробки аналітичних операцій. SAP HANA зберігає дані у колонках, що дозволяє значно прискорити операції агрегації та аналізу, оскільки всі значення певного атрибуту зберігаються разом [9].

Вона підтримує потужні аналітичні інструменти для виконання складних бізнес-запитів, створення звітів та візуалізації даних. Це дозволяє виконувати складні обчислення безпосередньо на рівні бази даних, зменшуючи потребу у передачі даних між базою даних і аналітичними додатками. SAP HANA може обробляти великі обсяги даних та підтримує вертикальне і горизонтальне масштабування. Усі дані зберігаються поруч у колонках, що дозволяє ефективно стискати дані.

Швидкодія досягається за допомогою індексів та розпаралелювання процесів, а також багаторівневого кешу для найуживаніших даних. Платформа підтримує різні механізми реплікації та резервування даних, а також легко інтегрується з іншими системами, що робить її ефективним інструментом для управління даними та аналітики.

Ці архітектурні рішення дозволяють SAP HANA забезпечувати високошвидкісну обробку даних та виконувати складні аналітичні операції в реальному часі, що робить її ідеальним вибором для бізнесів, які потребують

швидкої та надійної аналітики.

2.1.3 In-memory document stores

In-memory document stores, зберігають дані у вигляді документів (JSON або BSON) і забезпечують швидкий доступ та маніпуляцію напівструктурованими даними, що робить їх підходящими для додатків, які потребують гнучкості у роботі з різними типами даних, або з форматом даних що часто змінюється, та швидкого доступу до великих обсягів інформації. Одним із популярних інструментів у цій категорії є Couchbase.

Couchbase є NoSQL базою даних, яка використовується для зберігання даних у форматі документів, основаних на JSON. Вона забезпечує гнучкість у роботі з різними типами даних та швидкий доступ до документів у пам'яті. Використовується для різних завдань, включаючи розробку веб-додатків та аналіз даних.

Couchbase забезпечує високу доступність даних та стійкість до відмов завдяки підтримці механізмів реплікації та автоматичного перемикання у разі відмови одного з серверів. Платформа може використовуватися для аналізу даних завдяки інтеграції з аналітичними інструментами та підтримці запитів на основі SQL (N1QL).

Внутрішньо Couchbase використовує мапу ключів для швидкого доступу до документів у пам'яті. Ключі пов'язані з відповідними документами, що зберігаються у форматі JSON. Для забезпечення постійності даних Couchbase використовує дискову підсистему, де записи можуть бути автоматично збережені на диск.

Розподілена архітектура Couchbase дозволяє розподіляти дані та запити між кількома вузлами кластера, забезпечуючи високу продуктивність та надійність. Couchbase підтримує складні індексаційні механізми для швидкого виконання запитів, а також автоматичне управління кластером, включаючи ребалансування даних та управління реплікацією.

Ці архітектурні рішення роблять Couchbase потужним інструментом для роботи з документами в реальному часі, забезпечуючи високу швидкість доступу до даних, гнучкість та надійність.

2.1.4 In-memory analytical databases

In-memory analytical databases, оптимізовані для швидкої обробки аналітичних запитів за допомогою індексації, колонкового зберігання та векторизації, і застосовуються для бізнес-аналітики та фінансового аналізу. Одним із провідних представників цього типу баз даних є Apache Spark.

Apache Spark – це платформа для обробки даних, яка дозволяє виконувати розподілені обчислення й аналіз даних у реальному часі. Spark включає бібліотеку MLlib для машинного навчання, що дає змогу вирішувати складні аналітичні завдання та будувати моделі машинного навчання безпосередньо на платформі. Вона також підтримує інтеграцію з різними джерелами даних, такими як Hadoop, Cassandra, HBase та інші [10].

Apache Spark побудована на основі кількох ключових концепцій та технологій, які забезпечують її високу продуктивність і гнучкість.

Основна структура даних у Spark – це RDD, що дозволяє виконувати паралельні обчислення над розподіленими даними. RDD забезпечує стійкість до відмов та можливість відновлення даних у разі збоїв. Spark використовує DAG (directed acyclic graph) для оптимізації виконання запитів. DAG дозволяє ефективно розподіляти завдання між вузлами кластера та мінімізувати обчислювальні витрати. Також він підтримує кешування даних у пам'яті, що дозволяє повторно використовувати результати попередніх обчислень і значно знижує час виконання повторних запитів. Spark має модульну архітектуру, що дозволяє легко додавати нові компоненти та інтегруватися з іншими системами. Вона включає модулі для SQL запитів (Spark SQL), потокової обробки (Spark Streaming), машинного навчання (MLlib) та роботи з графами (GraphX). Ці архітектурні рішення дозволяють Apache Spark забезпечувати високошвидкісну обробку даних і виконувати складні аналітичні операції в реальному часі, що

робить її ідеальним вибором для завдань з великими даними та машинним навчанням.

2.1.5 In-memory NewSQL databases

In-memory NewSQL databases поєднують переваги традиційних реляційних баз даних з високою швидкодією та масштабованістю NoSQL баз даних. Комбінуючи це зі зберіганням данних у оперативній пам'яті, це дозволяє досягати високої швидкості обробки транзакцій та забезпечувати високу транзакційну стійкість. Одним із представників цього типу баз даних є VoltDB.

VoltDB – це високопродуктивна in-memory база даних, розроблена для виконання транзакцій у реальному часі. VoltDB поєднує переваги традиційних реляційних баз даних із сучасними архітектурними рішеннями, що дозволяють обробляти великі обсяги даних із низькою затримкою [11].

VoltDB зберігає всі дані в оперативній пам'яті, що забезпечує швидкий доступ до даних і високу продуктивність. База даних підтримує повний набір SQL-операцій, включаючи складні транзакції, агрегування та аналітичні запити. Однією з ключових особливостей VoltDB є те, що вона працює в багатопоточному режимі, дозволяючи виконувати кілька транзакцій одночасно без втрати продуктивності.

VoltDB використовує кілька ключових структур даних для забезпечення високої продуктивності та ефективності обробки транзакцій у реальному часі, зокрема Б-дерева для індексації таблиць, Геш-таблиці для швидкого доступу до даних за ключами, журнали транзакцій для забезпечення цілісності даних і відновлення системи після збоїв, зв'язані списки для організації списків елементів, колонкове зберігання для деяких аналітичних операцій та розподілені структури даних для масштабування і забезпечення високої доступності

База даних підтримує асинхронні операції та використовує різні механізми оптимізації, такі як компіляція запитів у машинний код, що дозволяє досягати високої швидкодії. VoltDB також надає інструменти для моніторингу та

керування продуктивністю, що дозволяє адмініструвати базу даних і налаштовувати її відповідно до потреб бізнесу.

Крім того, VoltDB забезпечує збереження даних у випадку збою за допомогою механізмів персистенції та відновлення даних. Це робить її надійним рішенням для використання в критично важливих додатках, таких як фінансові системи, телекомунікації, онлайн-ігри та інші сфери, де важлива висока швидкодія і надійність транзакцій.

2.1.6 In-memory graph databases

In-memory graph databases є потужними інструментами для зберігання та обробки графових структур даних, що дозволяють ефективно управляти зв'язками між об'єктами. Вони використовують алгоритми графової теорії для ефективного управління і обробки сильно пов'язаних даних, ідеальні для соціальних мереж, рекомендаційних систем та аналізу шахрайства. Одним із провідних представників цього типу баз даних є Neo4j.

Neo4j – це система управління базами даних, спеціалізована на зберіганні, обробці та аналізі графів. Вона використовує модель Property Graph, де вузли та зв'язки можуть мати властивості, що забезпечує гнучкість моделювання складних структур. Для взаємодії з базою даних використовується мова запитів Cypher, яка дозволяє легко виконувати складні запити [12].

Neo4j має високу продуктивність завдяки індексам, оптимізованим для графів, і можливості зберігання даних у пам'яті, а також підтримує масштабованість, як горизонтальну, так і вертикальну. Інтеграція з іншими технологіями, такими як Apache Spark, Hadoop, Kafka, та підтримка ACID транзакцій забезпечують надійність та цілісність даних. Neo4j активно використовується у сферах, де важливі взаємозв'язки між даними, таких як соціальні мережі, рекомендаційні системи, управління мережевою інфраструктурою, кібербезпека, управління даними в організаціях, та виявлення шахрайства.

Ці властивості роблять Neo4j потужним інструментом для роботи з графовими даними, забезпечуючи гнучкість, високу продуктивність та легкість використання для розробників та аналітиків даних у різних галузях.

2.2 Аналіз існуючих методів та підходів до зберігання даних

У сучасних базах даних для зберігання даних використовується багато різних структур даних, кожна з яких має свої переваги та недоліки в залежності від конкретних вимог і характеристик додатків.

2.2.1 Геш-таблиці

Геш-таблиця (Hashtable) – структура даних, що забезпечує ефективне зберігання та швидкий пошук елементів за ключем. Завдяки своїй здатності забезпечувати доступ до даних за амортизованим O(1) часом, Геш-таблиці широко використовуються в комп'ютерних науках і різних програмних додатках.

Основний принцип роботи Геш-таблиці полягає у збереженні пари "ключ-значення" Для зберігання елемента Геш-таблиця використовує Геш-функцію.

Геш-функція – це математична функція, яка перетворює вхідні дані (ключ) довільної довжини на вихідне значення фіксованої довжини, щоб визначити індекс, за яким елемент буде зберігатися [13]. Пошук елемента відбувається за тим самим процесом: Геш-функція застосовується до ключа для отримання індексу, після чого значення витягується з масиву за цим індексом (див. рисунок 2.1).

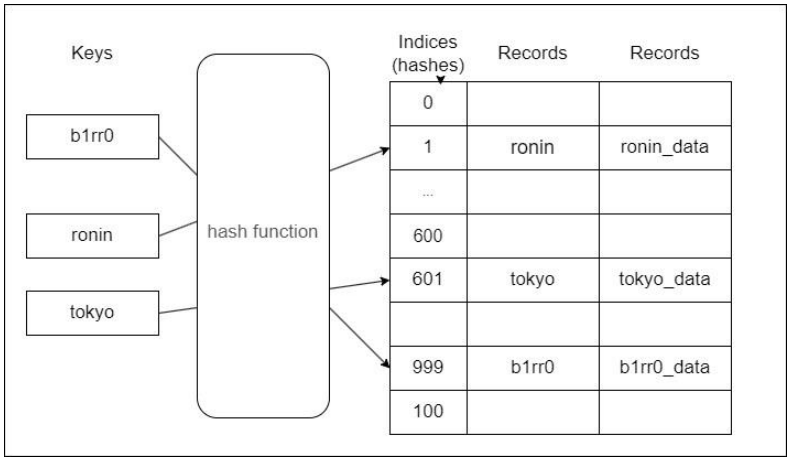

Рисунок 2.1 – Збереження даних у Геш-таблиці

Це дозволяє швидко знаходити потрібні елементи та робить Геш-таблиці незамінними в багатьох системах.

Оскільки кількість можливих ключів набагато більша ніж розмір масиву, можливі колізії, коли два різні ключі отримують однаковий індекс.

Для вирішення колізій використовуються різні методи, а саме:

– відкрита адресація**;**

– подвійні Геш-таблиці**;**

– ланцюжки.

У методі відкрита адресація, коли виникає колізія, новий елемент розміщується в іншій вільній позиції масиву. Можливий вибір наступної вільної позицій з якимось кроком, чи використання додаткової Геш-функції для обчислення відступу.

У методі подвійні Геш-таблиці використовується дві таблиці і дві Геш-функції, якщо місце у першій таблиці зайнято, то за допомогою іншої Геш-функції ми знайдемо ій місце у другій таблиці.

У методі ланцюжки кожен елемент масиву є списком (ланцюжком) всіх елементів, які мають однаковий Геш-індекс. Коли виникає колізія, новий елемент додається до цього списку. Це дозволяє зберігати всі елементи з однаковим Геш-значенням у одній позиції масиву, але з додатковою пам'яттю для зберігання ланцюжків. І і якщо велика кількість елементів будуть знаходитися у одній комірці, то швидкість доступу впаде до швидкості доступу всередині цієї комірки(див. рисунок 2.2).

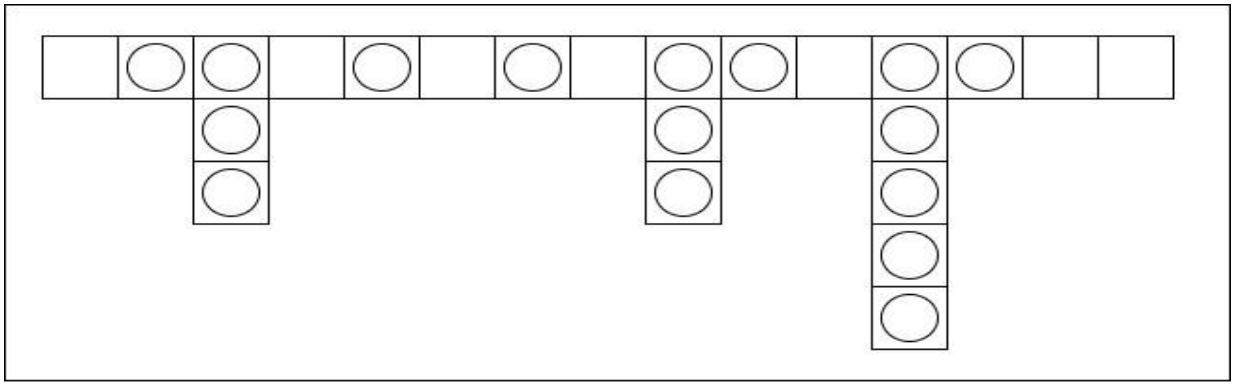

Рисунок 2.2 – Ланцюжки в середини Геш-таблиці

В цьому випадку можна збільшувати розмір Геш-таблиці, при цьому доводиться перераховувати Геш-коди для всіх елементів та вставляти їх у розширену версію, що є дуже ресурсоємною операцією, тому цей підхід використовують, коли інші підходи для вирішення вже не діють.

Переваги:

– швидкий доступ. Середня складність операцій вставки, пошуку і видалення становить O(1);
– простота реалізації. Базова реалізація Геш-таблиці є відносно простою і не вимагає складних алгоритмів.

Недоліки:

– колізії. Обробка колізій може ускладнювати реалізацію і знижувати ефективність у випадку великої кількості колізій;
– пам'ять. Геш-таблиці можуть споживати значну кількість пам'яті, особливо у випадку використання ланцюжків або великого запасу для уникнення колізій;
– залежність від Геш-функції. Якість роботи Геш-таблиці залежить від вибору Геш-функції. Погана Геш-функція може призвести до великої кількості колізій і знизити ефективність структури даних.

2.2.2 Б-дерево

Б-дерево (B-tree) – це структура даних, яка використовується для зберігання і пошуку великих обсягів даних. Воно є дуже популярним вибором для файлових систем і баз даних (частіше дискових), тому що забезпечує ефективні операції вставки, видалення і пошуку по елементах та по діапазонах. Також ж на відміну від інших дерев дозволяє швидко (бо увесь вузол лежить поруч), в випадках з дисковими базами даних, вивантажувати в оперативну пам’ять лише один вузол за раз. Це може бути корисно якщо більше ніж один в оперативну пам’ять не влазить. І під час пошуку робити усього логарифм (від висоти дерева) операцій зчитування з диску [14].

Основні властивості Б-дерева.

Самобалансуюче дерево пошуку, яке зберігає відсортовані дані (див. рисунок 2.3).

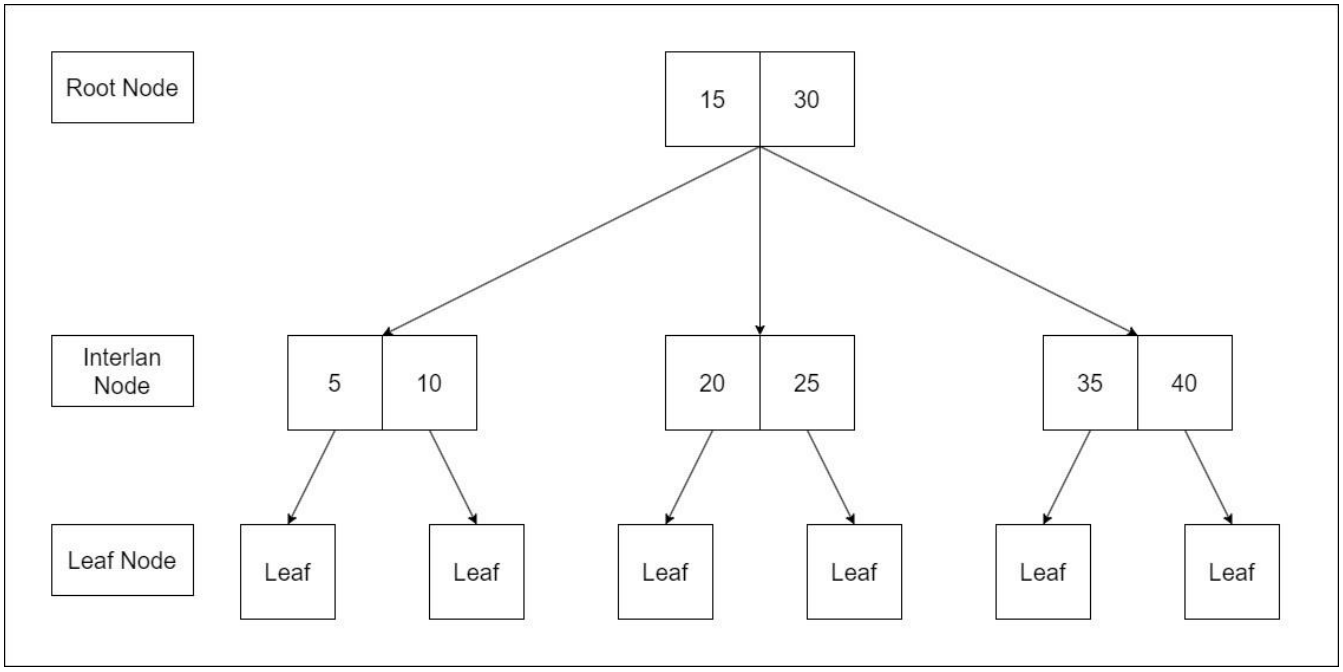

Рисунок 2.3 – Візуалізація Б-дерева

Мінімальний ступінь t- параметр, що визначає нижню і верхню межі кількості нащадків у вузлі.

Кожен вузол може містити максимум 2t - 1 ключів.

Кожен внутрішній вузол (не лист) має мінімум t і максимум 2t дочірніх вузлів.

Корінь має щонайменше два дочірніх вузли, якщо він не є листом.

Усі листи знаходяться на одному рівні. (це дозволяє нам мати логарифмічну складність операцій).

Основні операції.

Пошук починається з кореня і триває до листових вузлів. У кожному вузлі виконується бінарний пошук для визначення піддерева, в яке слід рухатися.

Під час вставки новий ключ додається до відповідного листового вузла. Якщо вузол переповнюється (кількість ключів перевищує 2t-1), він розділяється, і середній ключ піднімається до батьківського вузла. Процес може повторюватися до кореня.

Якщо ключ видаляється з вузла і кількість ключів стає менше t, вузол зливається з сусіднім вузлом або відбувається переміщення ключів з батьківського вузла. Це забезпечує збалансованість дерева.

Переваги:

– ефективне зберігання і пошук. Б-дерева забезпечують логарифмічну складність операцій пошуку, вставки та видалення;
– підтримка великих обсягів даних. Завдяки великій кількості дітей у кожного вузла, Б-дерева можуть ефективно обробляти великі обсяги даних;
– балансування. Самобалансування дерева забезпечує стабільну продуктивність для всіх операцій.

Недоліки:

– складність реалізації. Відносно складні алгоритми вставки і видалення можуть бути важкими для реалізації та налагодження;
– великий обсяг пам'яті. Потреба у зберіганні додаткових структур для балансування може збільшити обсяг використаної пам'яті.

2.2.3 Скіп-листи

Скіп-листи (Skip Lists) – це структура даних, яка забезпечує ефективний пошук, вставку і видалення елементів з середньою логарифмічною – складністю. Вони були розроблені як альтернатива збалансованим деревам, таким як AVL і червоно-чорні дерева, і забезпечують простішу реалізацію та ефективну продуктивність.

Скіп-лист являє собою багаторівневий зв'язаний список, де кожен рівень представляє підмножину елементів (вузлів) нижнього рівня (див. рисунок 2.4).

Скіп-лист складається з кількох рівнів. Найнижчий рівень є звичайним зв'язаним списком, а кожен наступний рівень містить підмножину елементів нижнього рівня.

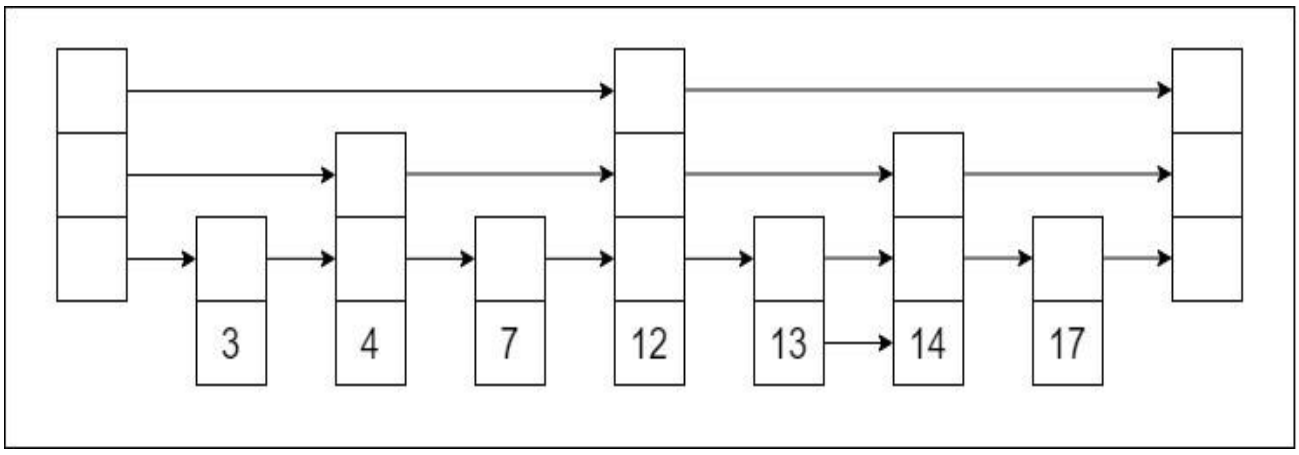

Рисунок 2.4 – Візуалізація Скіп-листа

Кожен вузол у Скіп-листі містить кілька посилань, які вказують на наступні вузли на різних рівнях. Кількість посилань у вузлів визначається випадковим чином під час вставки.

Основні операції.

Пошук починається з найвищого рівня і рухається вниз, поки не буде знайдено потрібний елемент або досягнуто найнижчого рівня. На кожному рівні порівнюємо ключ шуканого елемента з ключами елементів на поточному рівні. Якщо ключ поточного елемента менший, переходимо до наступного елемента на тому ж рівні. Якщо ключ поточного елемента більший або рівний, переходимо на рівень нижче і повторюємо порівняння. Коли доходимо до найнижчого рівня, здійснюємо остаточне порівняння ключів для знаходження шуканого елемента.

Новий елемент вставляється у всі рівні, де він повинен з'явитися, і кількість рівнів для цього елемента визначається випадковим чином.

Елемент видаляється з усіх рівнів, де він присутній.

Переваги:

- простота реалізації. Простіший алгоритм вставки та видалення;
- ефективність. Забезпечує середню складність близьку до логарифмічної для пошуку, вставки та видалення;
- гнучкість. Легко налаштовується шляхом зміни максимальної кількості рівнів.

Недоліки:

– випадковість. Ефективність залежить від випадкового вибору рівня для кожного елемента, що може призвести до непередбачуваних результатів у деяких випадках;
– використання пам'яті. Збільшення кількості рівнів призводить до збільшення використання пам'яті.

2.2.4 Графи

Представлення даних у вигляді графів , що складаються з вузлів (об'єктів) і ребер (зв'язків між об'єктами) подано на рисунку 2.5.

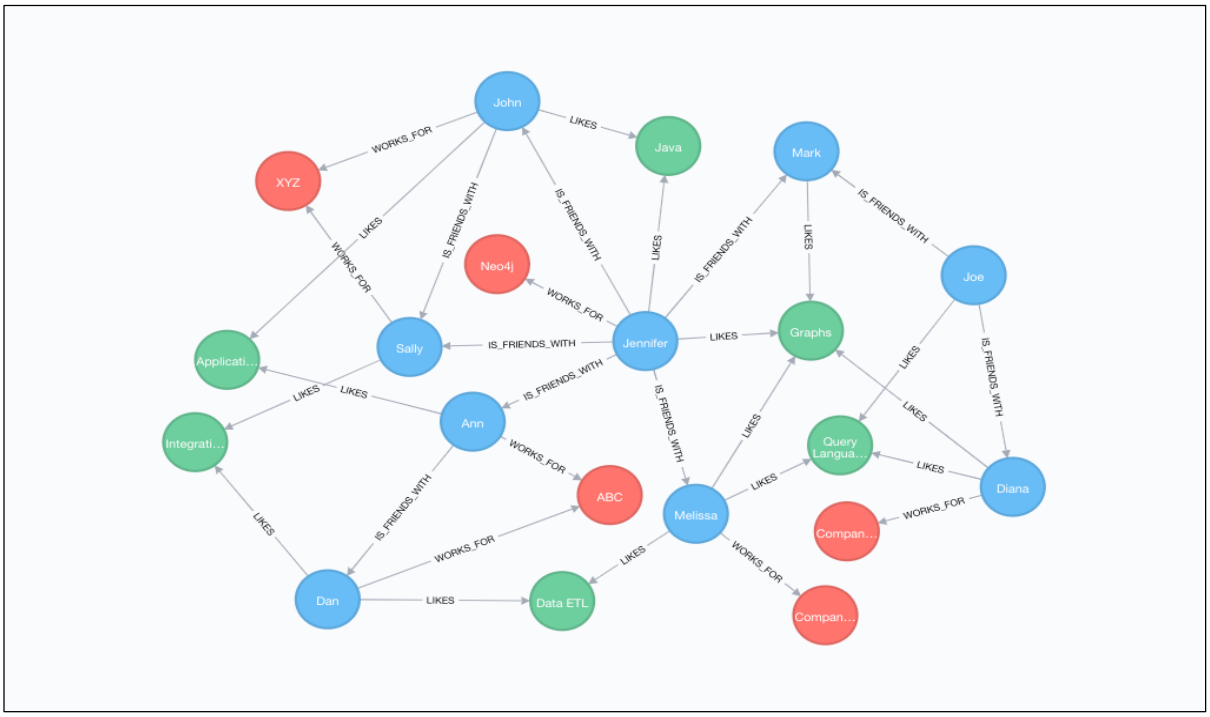

Рисунок 2.5 – Представлення графа (за даними [15])

Цей підхід дозволяє природньо моделювати складні взаємозв'язки між даними і забезпечує ефективний механізм для їхнього зберігання, запиту та маніпуляції [16].

Основні елементи графа.

Вузли – об'єкти, що представляють сутності в графі. Вони мають унікальні ідентифікатори та набір властивостей.

Ребера – об'єкти, що представляють зв'язки між вузлами. кожне ребро має напрямок, тип і властивості.

Властивості – додаткові дані, що зберігаються у вузлах і ребрах у вигляді пар ключ-значення.

Операції над графом.

Пошук, додавання та видалення вузла, ребра, властивостей (для прискорення пошуку використовуються індекси).

Обхід графа. Для обходу використовуються ефективні алгоритми обходу графа, такі як пошук у глибину (DFS) і пошук у ширину (BFS).

Знаходження коротких шляхів між вузлами за допомогою алгоритмів Дейкстри, Беллмана-Форда, Флойда-Уоршалла.

Інші аналітичні операції. Пошук компонентів зв'язності, циклів. Топологічне сортування тощо.

Переваги:

- природне моделювання зв'язків. Графи дозволяють природно моделювати складні взаємозв'язки між даними, що особливо корисно для соціальних мереж, транспортних систем, управління ресурсами та інших застосувань;
- ефективність пошуку та запитів. Графові бази даних оптимізовані для швидкого виконання запитів, які включають складні взаємозв'язки між даними. Запити, які включають кілька рівнів взаємозв'язків, можуть виконуватися набагато швидше, ніж у традиційних реляційних базах даних;
- гнучкість. Структура графа легко адаптується до змін у даних та відносинах між ними, що дозволяє легко оновлювати та розширювати базу даних без значних змін у її схемі.

Недоліки:

- складність реалізації. Графові структури даних і алгоритми для роботи з ними можуть бути складнішими для реалізації та розуміння порівняно з іншими структурами даних;

- використання пам'яті. Графи можуть споживати значну кількість пам'яті, особливо якщо кожен вузол має багато властивостей або якщо в графі дуже багато ребер;
- складність масштабування. Хоча графові бази даних добре справляються зі зв'язками на місцевому рівні, масштабування на великі обсяги даних та складні запити може бути викликом.

3 ПОСТАНОВКА ЗАДАЧІ

В даній кваліфікаційній роботі дослідження буде дослідження існуючих баз даних, що використовують різні підходи до збереження даних, з метою оцінки їх ефективності, масштабованості, безпеки, сумісності та витрат. Також для більшого глибокого розуміння роботи IMDb, будуть порівняні структури даних, на яких вони базуються. За допомогою дослідження будемо краще розуміти чому продуктивність і ефективність різних IMDb відрізняється.

3.1 Основні завдання дослідження

Метою дослідження є порівняння існуючих IMDb по декількох критеріях, а саме:

- оцінювання швидкодії та продуктивності різних IMDb. Аналіз має включати швидкість доступу до даних, час виконання запитів та операцій, обсяг та швидкість завантаження даних;
- визначення можливостей масштабування системи в залежності від об’єму даних, що обробляються, та потреб користувачів;
- оцінювання рівня захищеності даних у системі, механізми резервного копіювання, відновлення та захисту від втрати інформації;
- визначення можливостей взаємодії обраної системи з іншими існуючими системами в організації та їх сумісність з іншими інструментами для обробки даних;
- оцінювання витрат на впровадження, підтримку та обслуговування обраної системи в порівнянні з іншими варіантами, враховуючи витрати на ліцензії, навчання персоналу та технічну підтримку;

Ціль дослідження полягає у глибокому розумінні ефективності та можливостей різних IMDb для оптимального вибору.

3.2 Експериментальна частина

Мета експерименту у виявленні найбільш влучного підходу для зберігання і

доступу до даних, що зберігаються в оперативній пам’яті. Зокрема, дослідження спрямоване на проведення експерименту зі створення та аналізу систем та структур даних мовою програмування Java.

Головна задача експериментальної частини є аналіз існуючих структур даних, що використовуються в системах IMDb, а саме:

- реалізувати Геш-таблиці з відкритою адресацією;
- реалізувати Геш-таблиці з бакетами на основі бінарних дерев;
- реалізувати B-дерева;
- реалізувати Скіп-списки;
- реалізувати графові сховища даних;
- проаналізувати ефективності виконання основних операцій;
- проаналізувати використання пам’яті;
- визначити переваги та недоліки;
- визначити кращий метод для поставленої задачі.

Таким чином, метою експерименту є надання повного порівняльного аналізу, який допоможе розробникам обрати найбільш ефективну структуру даних для конкретних завдань.

4 ТЕОРИТИЧНЕ ДОСЛІДЖЕННЯ IMDb

4.1 Формулювання гіпотез

4.1.1 Гіпотеза про продуктивність

H0 (нульова гіпотеза). Використання різних IMDb не впливає на швидкодію та продуктивність системи для обробки даних в оперативній пам'яті.

H1 (альтернативна гіпотеза). Використання конкретних IMDb суттєво покращує час виконання операцій, швидкість доступу до даних та реакцію системи на завдання в порівнянні з іншими.

4.1.2 Гіпотеза стосовно масштабованості

H0. Немає значущих різниць у можливостях масштабування різних IMDb відносно об'єму даних та кількості одночасних користувачів в оперативній IMDb.

H1. Деякі IMDb виявляють суттєво кращі можливості масштабування при обробці даних в оперативній пам'яті.

4.1.3 Гіпотеза щодо надійності

H0. Рівень безпеки даних та надійності роботи IMDb в оперативній пам'яті є схожим для різних систем.

H1. Деякі IMDb виявляють себе як більш безпечні та надійні при обробці даних в оперативній пам'яті.

4.1.4 Гіпотеза стосовно сумісності та інтеграції

H0. Усі розглянуті IMDb мають подібні можливості щодо взаємодії та інтеграції з іншими системами в оперативній пам'яті.

H1. Деякі IMDb виявляють більшу сумісність та легкість інтеграції в операційному середовищі оперативної пам'яті.

4.1.5 Гіпотеза щодо вартості та економічності

H0. Витрати на впровадження, підтримку та експлуатацію різних IMDb в оперативній пам'яті подібні.

H1. Деякі IMDb є економічно вигіднішими враховуючи вартість ліцензій, підтримки, навчання персоналу та загальні витрати на обслуговування.

4.2 Визначення критерій оцінювання

У експерименті важливо вимірювати та аналізувати різні параметри для отримання об'єктивних даних і наукових висновків. Нижче наведено перелік критеріїв, які повинні бути включені до експерименту.

Продуктивність. Швидкодія та продуктивність системи є одним із ключових аспектів для обробки даних в оперативній пам'яті. Цей критерій оцінює час виконання операцій, швидкість доступу до даних та реакцію системи на завдання.

Масштабованість. Здатність системи масштабуватися відносно об'єму даних та кількості одночасних користувачів важлива для забезпечення стабільної роботи під зростаючими навантаженнями.

Надійність. Цей критерій включає в себе стійкість системи до відмов, механізми резервного копіювання та відновлення даних для запобігання втратам.

Сумісність та інтеграція. Здатність системи до взаємодії з іншими існуючими технологіями та системами в організації. Це може включати підтримку різних форматів даних та протоколів обміну.

Вартість та економічність. Оцінка витрат на впровадження, підтримку та експлуатацію системи. Це включає вартість ліцензій, підтримки, навчання персоналу та загальні витрати на обслуговування.

4.3 Вибір методології

Для проведення оцінки різних IMDb планується використовувати комплексний підхід, який включатиме методи прийняття рішень в умовах множинних критеріїв.

Цей метод передбачає врахування та зважування різних аспектів кожної IMDb, враховуючи їхні переваги та недоліки.

Аналіз альтернатив передбачає вивчення різних IMDb з урахуванням їхніх технічних характеристик, швидкодії, масштабованості та інших параметрів.

Важливо ретельно врахувати величезний обсяг генерованих даних у різних галузях, а також конкретні потреби та вимоги користувачів.

Для визначення ефективності кожної альтернативи планується застосування лінійної адитивної згортки з нормуючими множниками. Цей підхід дозволяє нормалізувати критерії, враховуючи їхні значення за шкалами MAX та MIN. Такий метод дозволяє об'єктивно порівнювати різні IMDb за певними параметрами, піддаючи їх однаковому стандарту оцінювання.

Зокрема, буде проведено експериментальне дослідження за принципом Парето для визначення, чи існують альтернативи, які можуть одночасно вдосконалити деякі аспекти без втрати інших. Цей підхід дозволяє виокремити оптимальні варіанти, що відповідають найкращим критеріям.

Такий комплексний аналіз дозволить об'єктивно оцінити ефективність різних IMDb та визначити оптимальний вибір відповідно до конкретних потреб та умов використання.

4.4 Опис підходу до експерименту

4.4.1 Вибір альтернатив для експерименту

Для задачі вибору системи IMDb було розглянуто наступні альтернативи:

- Redis – це швидка ключ-значення база даних, яка працює в оперативній пам'яті. Вона відома своєю швидкодією та підтримкою різних типів даних. Redis може бути використаний для кешування, сесійного зберігання, а також для роботи з потоками даних;
- MemSQL – реляційна база даних, яка використовує оперативну пам'ять для прискорення роботи з даними. Вона поєднує у собі силу SQL та швидкодію NoSQL. MemSQL може бути ефективним в роботі з великими об'ємами даних;
- Apache Ignite – це розподілена база даних, яка може працювати в оперативній пам'яті. Вона підтримує SQL та може бути використана для реалізації різних сценаріїв, включаючи аналітику, кешування та обробку транзакцій;

- VoltDB – це реляційна база даних, спеціалізована на використанні в оперативній пам'яті. Вона забезпечує низький латентний час та високу пропускну здатність для трансакційної роботи;
- Cassandra – розподілена база даних, яка підтримує широкі можливості роботи з даними в режимі реального часу. Це може бути корисним для систем, які вимагають високої доступності та масштабованості [17].

4.4.2 Аналіз шкал вимірювань

Зважаючи на обрані критерії для вибору системи керування базами даних, розглянемо шкали та їх аналіз для кожного критерію.

Продуктивність. Тип шкали: абсолютна шкала.

Час виконання запиту (у мілісекундах):

- 25 мс: швидкодія системи, яка може бути оптимальною для багатьох додатків;
- 75 мс: прийнятний час відповіді для середнього навантаження;
- 150 мс: поганий час реакції, що може викликати незручності для користувачів.

Масштабованість. Тип шкали: інтервальна шкала.

Кількість одночасних користувачів:

- до 100: для невеликих організацій або тестових середовищ;
- від 500 до 1000: прийнятний для середніх підприємств;
- від 5000 і більше: велика масштабованість для великих корпорацій з високим навантаженням.

Надійність. Тип шкали: порядкова шкала.

Рівень резервного копіювання:

- резервне копіювання кожні 24 години (низький рівень захисту від втрати даних);
- резервне копіювання щогодини (середній рівень захисту);
- резервне копіювання в режимі реального часу (високий рівень надійності та захисту).

Сумісність та інтеграція. Тип шкали: порядкова шкала.

Рівень сумісності з іншими системами:

– погано сумісна (вимагає значного зусилля для інтеграції);
– достатньо сумісна (є можливості для інтеграції, але з деякими обмеженнями);
– добре сумісна (зручна для інтеграції з різними системами без значних труднощів).

Вартість та економічність. Тип шкали: порядкова шкала.

Витрати на підтримку (у доларах на рік):

– $5,000: низькі витрати на підтримку для дешевих альтернатив;
– $20,000: середні витрати на підтримку для більш розширених функціональних можливостей;
– $50,000+: високі витрати на підтримку для підприємств з великим обсягом даних та високими вимогами до надійності.

4.4.3 Порядок проведення експерименту

Визначення гіпотез. Поставимо перед собою гіпотези щодо впливу вибору IMDb на продуктивність, масштабованість, надійність, сумісність та вартість.

Вибір альтернатив. Оберемо п'ять різних IMDb для подальшого аналізу: Redis, MemSQL, Apache Ignite, VoltDB, Cassandra.

Визначення метрик. Розробимо конкретні метрики для оцінки кожного критерію, такі як час відгуку, масштаб обробки, надійність, рівень сумісності та економічність.

Збір та нормалізація даних:

– зберемо дані для кожної IMDb відповідно до визначених метрик;
– нормалізуємо дані, використовуючи метод лінійної адитивної згортки з нормуючими множниками.

Визначення ваг критеріїв. Присвоїмо ваги кожному критерію в залежності від їхньої важливості.

Застосування методу прийняття рішень. Використаємо метод прийняття

рішень для порівняння та оцінки кожної IMDb.

Вибір оптимальної альтернативи. Обчислимо сумарні оцінки та визначимо найбільш ефективну IMDb.

4.5 Багатокритеріальний аналіз

Векторний опис альтернатив за обраними критеріями з використанням відповідних шкал представлено у вигляді таблиці 4.1.

Таблиця 4.1 – Векторний опис альтернатив

Альтернатива	Продуктивність	Масштабованість	Надійність	Сумісність та інтеграція	Вартість та економічність
Redis	75 мс	До 100	Висока	Добре сумісна	$20,000
MemSQL	50 мс	від 500 до 1000	Середня	Достатньо сумісна	$50,000+
Apache Ignite	100 мс	5000 і більше	Середня	Достатньо сумісна	$50,000+
VoltDB	40 мс	від 500 до 1000	Висока	Добре сумісна	$50,000+
Cassandra	150 мс	5000 і більше	Низька	Погано сумісна	$5,000+

Дані були взяті з хмарного провайдера AWS [18].

Усі критерії рівнозначні, тому задачу векторно можна описати в вигляді рівнозначної суми усіх критеріїв (формула 1):

$$w(x) = f1 + f2 + f3 + f4 + f5 \quad (1)$$

де $f1$ – продуктивність з коефіцієнтом 1;

$f2$ − масштабованість з коефіцієнтом 1;

$f3$ – надійність з коефіцієнтом 1;

$f4$ − сумісність та інтеграція з коефіцієнтом 1;

$f5$− вартість та економічність з коефіцієнтом 1.

Визначимо множину Парето. Усі альтернативи не гірші за інші хоча б за одним критерієм, тобто жодна альтернатива не може бути видалена.

Перейдемо до нормалізації критеріїв з урахуванням мінімуму та максимуму на основі наступних формул. Для нормалізації продуктивності, вартості використаємо формулу мінімізації (формула 2). Для нормалізації масштабованості, надійності, сумісності та інтеграцій використаємо формулу максимізації (формула 3).

$$x_{\,normamized\ max} = \frac{x - max(x)}{min(x) - max(x)} \tag{2}$$

$$x_{\,normamized\ min} = \frac{x - min(x)}{max(x) - min(x)}, \tag{3}$$

де *x* – значення що зараз обробляється;

min(x) – мінімальне значення серед усіх;

max(x) – максимальне значення серед усіх.

Результати розрахунків представлено в таблиці 4.2.

Виконаємо розрахунки за формулою (1) і зазначмо результати в таблиці 4.2.

Таблиця 4.2 – Результати розрахунків

Альтернатива	Продук-тивність	Масштабованість	Надійність	Сумісність та інтеграція	Вартість та економічність	Сума
Redis	0,68	0,00	1,00	1,00	0,66	3,34
MemSQL	0,91	0,20	0,50	1,00	0	2,61
Apache Ignite	0,45	1,00	0,50	0,50	0	2,45
VoltDB	1,00	0,20	1,00	1,00	0	3,20
Cassandra	0,00	1,00	0,00	0,00	1	2,00

В рамках розглянутих гіпотез можна зробити висновок, що обрані IMDb мають різну продуктивність, різні можливості масштабування, різні рівні

надійності, різні рівні сумісності та інтеграцій, а також різну вартість та економічність.

Найкращій результат лінійної адитивної згортки показав Redis з балом 3,40. Однак він не є найкращім у кожному критерію, тому якщо б критерії не були рівноцінні між собою, то результат був би іншим.

Також результати вказали на індивідуальні переваги кожного інструменту в своїй сфері. Redis виділяється високим показником продуктивності, в той час як Apache Ignite володіє вражаючими показниками масштабованості та сумісності. MemSQL виявляється більш збалансованим за різними критеріями, зокрема високими показниками продуктивності та надійності. VoltDB виявляється загальним лідером, з високими балами у всіх аспектах, зокрема у продуктивності та масштабованості. З іншого боку, Cassandra показує себе відмінно у масштабованості та сумісності, але має низькі показники продуктивності. Результати аналізу дозволяють визначити оптимальний вибір в залежності від конкретних вимог та пріоритетів користувача чи організації.

5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ IMDb

5.1 Опис програмної системи

В рамках експериментальної частини буде проведено порівняння продуктивності структур даних, які використовуються даними IMDb. А саме реалізацію Геш-таблиці з відкритою адресацією, Геш-таблиці з бакетами на основі бінарних дерев, B-дерева, Скіп-списків та графового сховища даних.

Для тестування структур даних, до структур даних було висунуто 3 задачі, оптимальність, яких є найкритичнішою для системи, а саме:

- збереження;
- пошук;
- видалення.

Для реалізації виконання задач було обрано середовище з такими характеристиками:

- процесор: AMD Ryzern 5 1600 Six-Core 3.20GHz;
- оперативна пам'ять: 16GB DDR4;
- жорсткий диск: SATA 960GB Kingston A400;
- операційна система: Windows 10 Pro.

Всі експерименти будуть виконані з використанням мови Java 17 з аргументом -Xmx6144m [19]. Також будуть використані стандартних бібліотеки Java для колекцій, Runtime для заміру часу роботи, та використаної оперативної пам’яті. Для більшої точності кожен експеримент буде проведено 5 разів та результат буде усереднений.

Обсяг даних для кожної задачі був обраний середнього розміру:

- 10 000 елементів;
- 1 000 000 елементів;
- 10 000 000 елементів.

5.2 Результати експериментів

5.2.1 Геш-таблиці

Для сховища даних на основі Геш-таблиці (Hashtable) буде реалізовано основні операції зберігання, пошуку та видалення ключ-значення, а також проведемо заміри швидкості виконання цих операцій. Головна проблема Геш-таблиць – це колізії, і для вирішення цих проблем буде створено два варіанти реалізації Геш-таблиці [20]. Спочатку був створений загальний інтерфейс HashTable з основними методами.

```
public interface HashTable<K, V> {
    void put(K key, V value);
    V get(K key);
    V remove(K key);
}
```

Для першої реалізації уникнення колізій ми використаємо метод відкритої адресації з лінійним зондуванням. Це дозволить ефективно розміщувати і знаходити елементи в таблиці.

Програмну реалізацію цієї версії наведено у А.1 Додатку А

Для другого випадку вирішимо колізій за допомогою бакетів, що базуються на бінарних дерев. Код наведено у А.2 додатку А.

Також було написано код для заміру продуктивності коду:

```
public class PerformanceTest implements TestRunner {

    @Override
    public void run(List<Integer> list, RedisHashTable<String,
String> map) {
        int numElements = list.size();
        long startTime, endTime;
        System.out.println(map.getClass().getSimpleName() + "  count:
" + list.size());
        // Insert
        long memo1 = Runtime.getRuntime().totalMemory() -
                Runtime.getRuntime().freeMemory();
        startTime = System.nanoTime();
        for (int i = 0; i < numElements; i++) {
            map.put("key" + list.get(i), "value" + i);
        }
        endTime = System.nanoTime();
```

```
            System.out.println("Insert time: " + (endTime - startTime) /
1e6 + " ms");
            // Search
            startTime = System.nanoTime();
            for (int i = 0; i < numElements; i++) {
                map.get("key" + list.get(i));
            }
            endTime = System.nanoTime();
            System.out.println("Search time: " + (endTime - startTime) /
1e6 + " ms");
            long memo2 = Runtime.getRuntime().totalMemory() -
                    Runtime.getRuntime().freeMemory();
            System.out.println("Memory used with all Items " + (memo2 -
memo1) / 1e+6);

            // Remove
            startTime = System.nanoTime();
            for (int i = 0; i < numElements; i++) {
                map.remove("key" + list.get(i));
            }
            endTime = System.nanoTime();
            System.out.println("Remove time: " + (endTime - startTime) /
1e6 + " ms");
        }
    }
```

Результати вимірювання представлені в таблицях 5.1, 5.2, 5.3.

Таблиця 5.1 – Результати роботи RedisLikeHashMap та RedisLikeHashMapWithTree для 10_000 елементів

Metric	RedisLikeHashMap	RedisLikeHashMapWithTree
Count	10,000	10,000
Insert time (ms)	18.435401	22.5316
Search time (ms)	5.829999	4.2931
Memory used with all Items (mb)	3.73284	5.828008
Remove time (ms)	6.1555	5.1359
Memory used (mb)	5.830064	7.214248

Таблиця 5.2 – Результати роботи RedisLikeHashMap та RedisLikeHashMapWithTree для 1_000_000 елементів

Metric	RedisLikeHashMap	RedisLikeHashMapWithTree
Count	1,000,000	1,000,000
Insert time (ms)	699.0828	922.7486

Продовження таблиці 5.2

Metric	RedisLikeHashMap	RedisLikeHashMapWithTree
Search time (ms)	504.8774	516.5229
Memory used with all Items (mb)	211.14508	451.422744
Remove time (ms)	509.0381	543.7568
Memory used (mb)	320.196984	543.697432

Таблиця 5.3 – Результати роботи RedisLikeHashMap та RedisLikeHashMapWithTree для 10_000_000 елементів

Metric	RedisLikeHashMap	RedisLikeHashMapWithTree
Count	10,000,000	10,000,000
Insert time (ms)	6817.1806	10667.5518
Search time (ms)	5161.090001	6525.682701
Memory used with all Items (mb)	1979.957448	1787.547208
Remove time (ms)	7004.6872	6878.7847
Memory used (mb)	1667.931816	525.323376

Відображення цих даних приведено на графіках (див. рисунок 5.1).

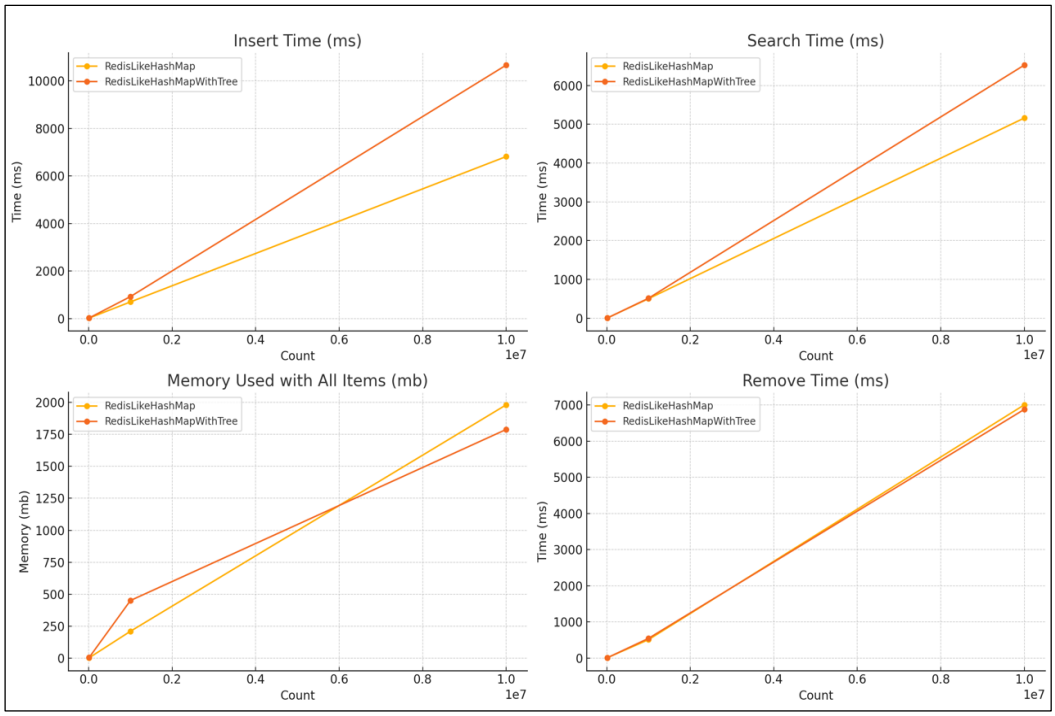

Рисунок 5.1 – Графіки продуктивності RedisLikeHashMap та RedisLikeHashMapWithTree

5.2.2 Б-дерево

В рамках створення сховища даних, базованого на Б-дереві буде реалізовано основні операції зберігання, пошуку та видалення ключ-значення, а також проведено заміри швидкості виконання цих операцій [21].

Код вузла (Node) приведено нижче:

```
private static class Node<T extends Comparable<T>> {
    private List<T> keys;
    private List<Node<T>> children;
    private Node<T> parent;
  private Node(Node<T> parent, int maxKeySize, int maxChildrenSize) {
        this.parent = parent;
        this.keys = new ArrayList<>(maxKeySize);
        this.children = new ArrayList<>(maxChildrenSize);
    }
```

Далі приведемо основну частина коду Б-дерева приведена у А.3 додатку А.

Проведемо аналогічне тестування для різних значень мінімального ступеня t (Min key Size). Наведено код для запуску з різними значеннями t (4, 8, 64). Також були виконані дослідження для більших значень t, але вони дали значно гірші результати. Код представлено нижче:

```
    BTreePerformanceTest beTreePerformanceTest = new
BTreePerformanceTest();
    System.out.println("Min Key Size: 4");
    bTreePerformanceTest.run(dataSet, new BTree(4));
    System.out.println("Min Key Size: 8");
    bTreePerformanceTest.run(dataSet, new BTree(8));
    System.out.println("Min Key Size: 64");
    bTreePerformanceTest.run(dataSet, new BTree(64));
```

Результати вимірювання представлені в таблицях 5.4, 5.5, 5.6.

Таблиця 5.4 – Результати роботи BTree 10_000 елементів

Min Key Size	Count	Insert Time (ms)	Memory Used After Insert (MB)	Search Time (ms)	Remove Time (ms)
4	10,000	15.688	0.710912	7.9234	17.0482

Продовження таблиці 5.4

Min Key Size	Count	Insert Time (ms)	Memory Used After Insert (MB)	Search Time (ms)	Remove Time (ms)
8	10,000	15.1215	0.461456	9.5057	15.4838
64	10,000	33.0103	8.389352	33.1618	47.8612

Таблиця 5.5 – Результати роботи BTree 1_000_000 елементів

Min Key Size	Count	Insert Time (ms)	Memory Used After Insert (MB)	Search Time (ms)	Remove Time (ms)
4	1,000,000	1405.6403	58.361704	688.991	1131.2025
8	1,000,000	1445.1367	37.959368	652.2535	1195.3313
64	1,000,000	3573.9868	149.988648	1549.853	3134.4299

Таблиця 5.6 – Результати роботи BTree 10_000_000 елементів

Min Key Size	Count	Insert Time (ms)	Memory Used After Insert (MB)	Search Time (ms)	Remove Time (ms)
4	10,000,000	18623.4478	417.582896	12109.5885	18206.9994
8	10,000,000	21006.543	209.461128	11847.761	19217.1733
64	10,000,000	52712.754	755.02812	27764.2289	44579.8281

Відображення цих данних приведено на графіках (див. рисунок 5.2).

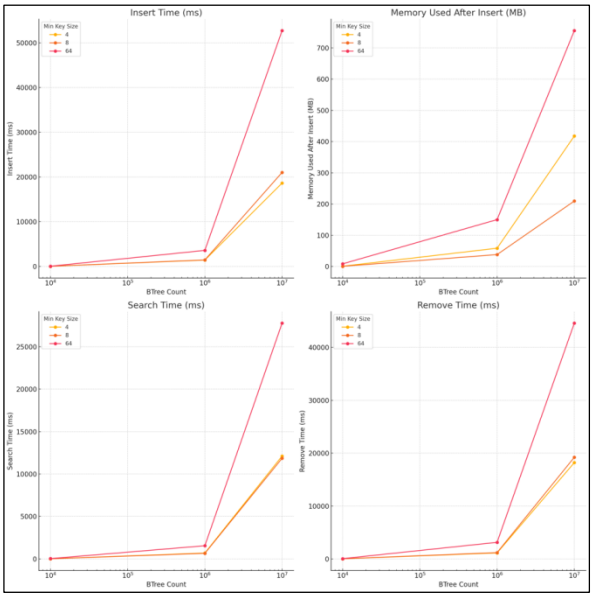

Рисунок 5.2 – Графіки продуктивності BTree

5.2.3 Скіп-лист

Для сховища даних на основі Скіп-листа (Skip Lists) буде реалізовано операції зберігання, пошуку та видалення, а також проведено заміри швидкості виконання цих операцій. Основний код приведено в А.4 додатку А.

Результати вимірювані представлені в таблиці 5.7.

Таблиця 5.7 – Результати роботи SkipList

SkipList Count	Insert Time (ms)	Memory Used After Insert (MB)	Search Time (ms)	Remove Time (ms)
10,000	13.2131	1.632752	7.8202	7.4536
1,000,000	1725.2952	59.972048	1691.4678	1508.2793
10,000,000	31437.7766	605.285896	36916.9861	30872.3417

Відображення цих даних приведено на графіках (див. рисунок 5.3).

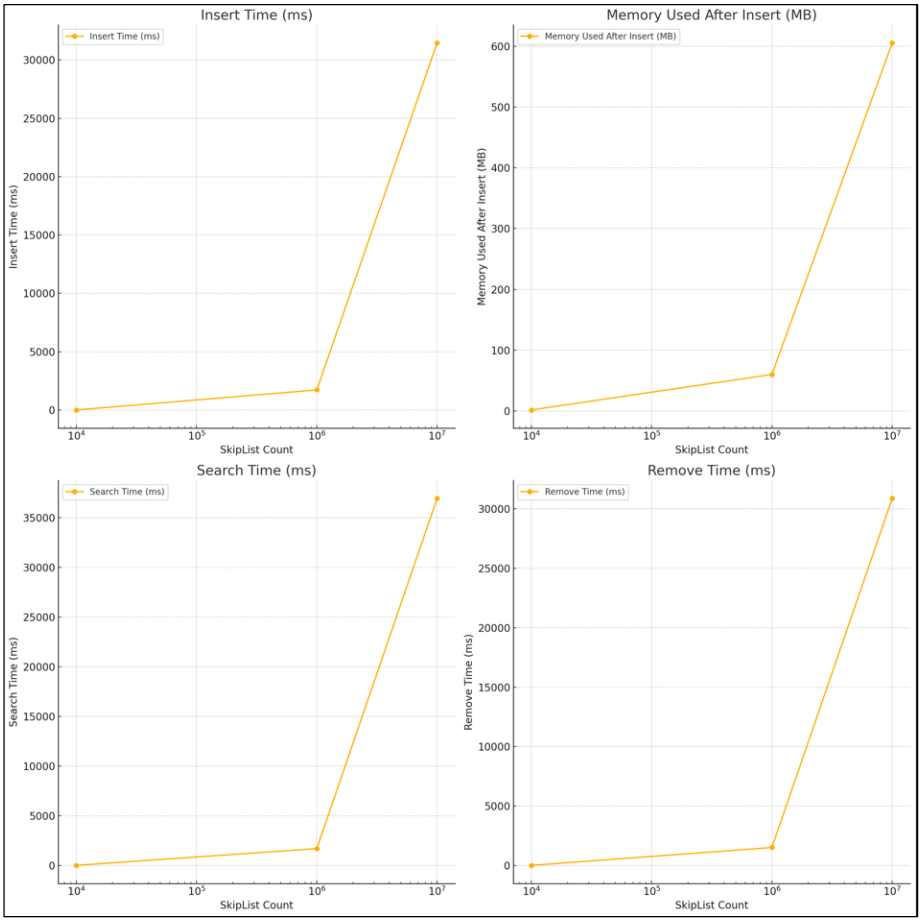

Рисунок 5.3 – Графіки продуктивності Скіп-листа

5.2.4 Графи

В рамках створення сховища даних, базованого на графах (Graph) буде реалізовано основні операції зберігання, пошуку та видалення вузлів (nodes), ребер (relationships), властивостей (properties), а також проведено заміри швидкості виконання цих операцій. Вузли представляють сутності, ребра – зв'язки між вузлами, властивості – додаткові дані. Індекси забезпечують швидкий пошук вузлів та ребер за їх властивостями [22]. Код наведено у А.5 додатку А.

Тестуємо за допомогою коду:

```
        List<Integer> dataSet = new ArrayList<>();
        for (int i = 0; i < 10_000_000; i++) {
            dataSet.add(r.nextInt());
        }
        GraphTestRaner graphDatabaseTest = new GraphTestRaner();
        List<Relation> relations = new ArrayList<>();
        int relCount = 1_000_000;
        for (int i = 0; i < relCount; i++) {
            relations.add(new
Relation(RelationType.getRandomRelation().toString(), "" +
dataSet.get(Math.abs(r.nextInt()) % dataSet.size()), "" +
dataSet.get(Math.abs(r.nextInt()) % dataSet.size())));
        }
        System.out.println("Relation count: " + relCount);
        graphDatabaseTest.run(dataSet.stream()
                .map(String::valueOf).toList(), relations, new GraphStore());
```

Результати приведені в таблицях 5.8, 5.9, 5.10 нижче.

Таблиця 5.8 – Результати роботи GraphStore для 10_000 елементів

Relation count	Insert time (ms)	Memory used after insert (MB)	Search time (ms)	Search relation time (ms)	Remove time (ms)
500	18.1343	3.483416	2.3428	0.1519	5.3768
50000	67.8707	0.101392	2.3745	7.8310	70.2705
1000000	453.7964	113.582592	2.4783	31.0455	7156.6570

Таблиця 5.9 – Результати роботи GraphStore для 1_000_000 елементів

Relation count	Insert time (ms)	Memory used after insert (MB)	Search time (ms)	Search relation time (ms)	Remove time (ms)
500	718.9059	347.2594	189.2971	0.1382	318.4349
50000	833.0203	354.90644	158.7607	7.5076	422.8915
1000000	2189.4766	496.415088	145.5704	29.1133	7625.2166

Таблиця 5.10 – Результати роботи GraphStore для 10_000_000 елементів

Relation count	Insert time (ms)	Memory used after insert (MB)	Search time (ms)	Search relation time (ms)	Remove time (ms)
500	7063.4600	3272.0208	1324.2952	0.1478	3026.4014
50000	7025.3193	3082.81344	1252.7484	7.1459	3170.3360
1000000	13698.8646	3133.341752	1079.8584	26.7861	15308.3623

Відображення цих даних приведено на графіках (див. рисунок 5.4 (а), (б)).

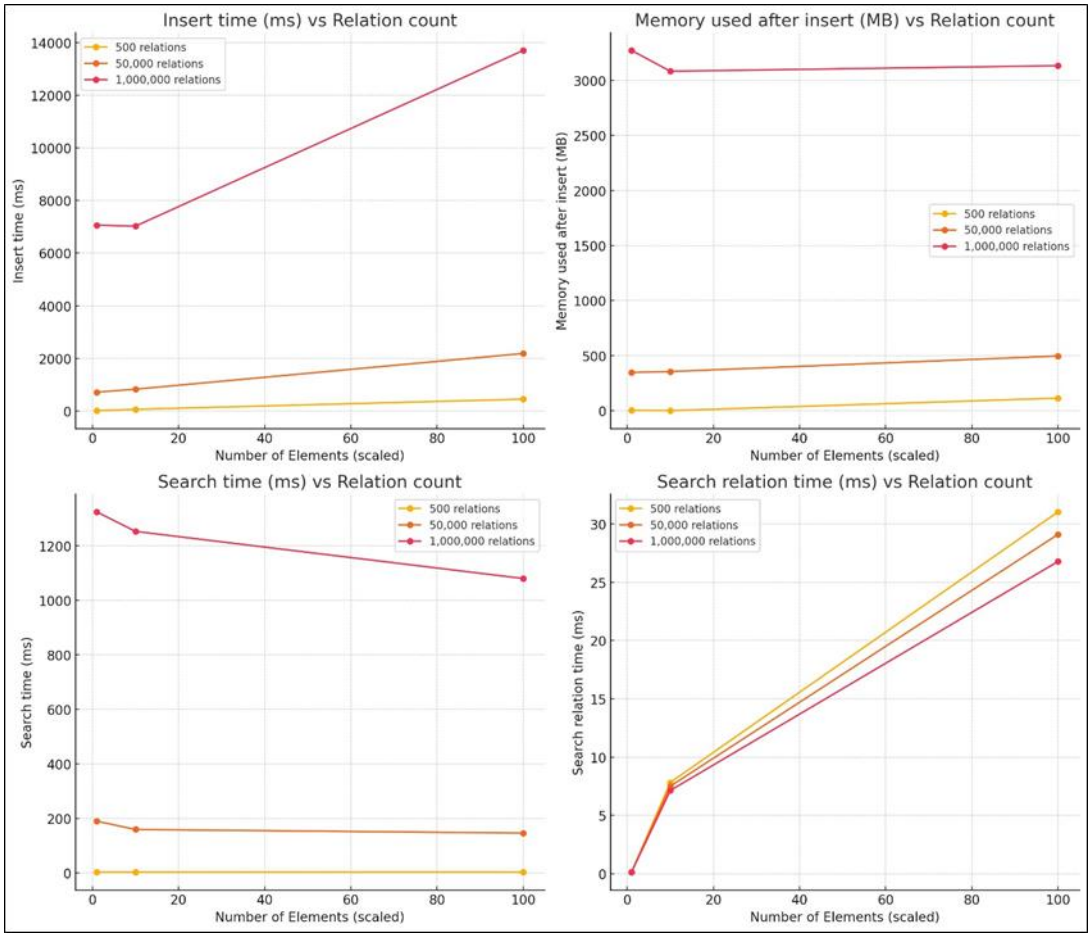

Рисунок 5.4 (а) – Графіки продуктивності GraphStore

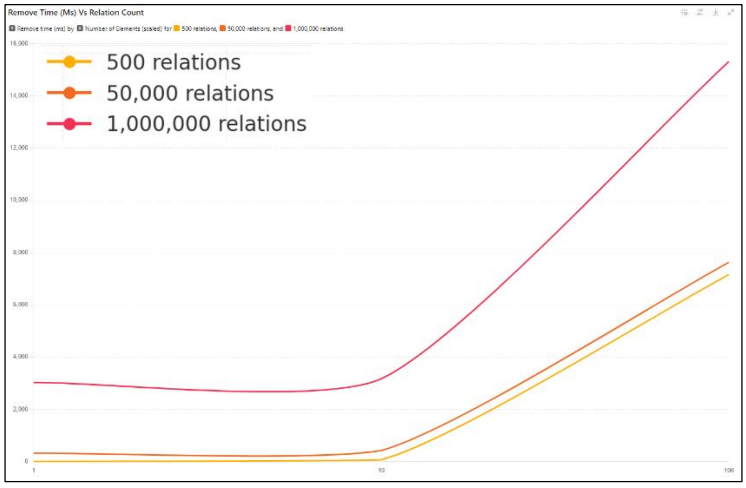

Рисунок 5.4 (б) – Графіки продуктивності GraphStore

5.3 Аналіз результатів проведених експериментів

В рамках експериментальної частини було реалізовано та протестовано кілька варіантів структур даних, що найчастіше використовуються в IMDb, зокрема Геш-таблиці з відкритою адресацією, Геш-таблиці з бакетами на основі бінарних дерев, Б-дерева, Скіп-списки та графові сховища даних. Кожна структура даних мала свої переваги та недоліки в залежності від операцій та обсягів даних. Результати порівняння представлені на рисунках 5.5 та 5.6.

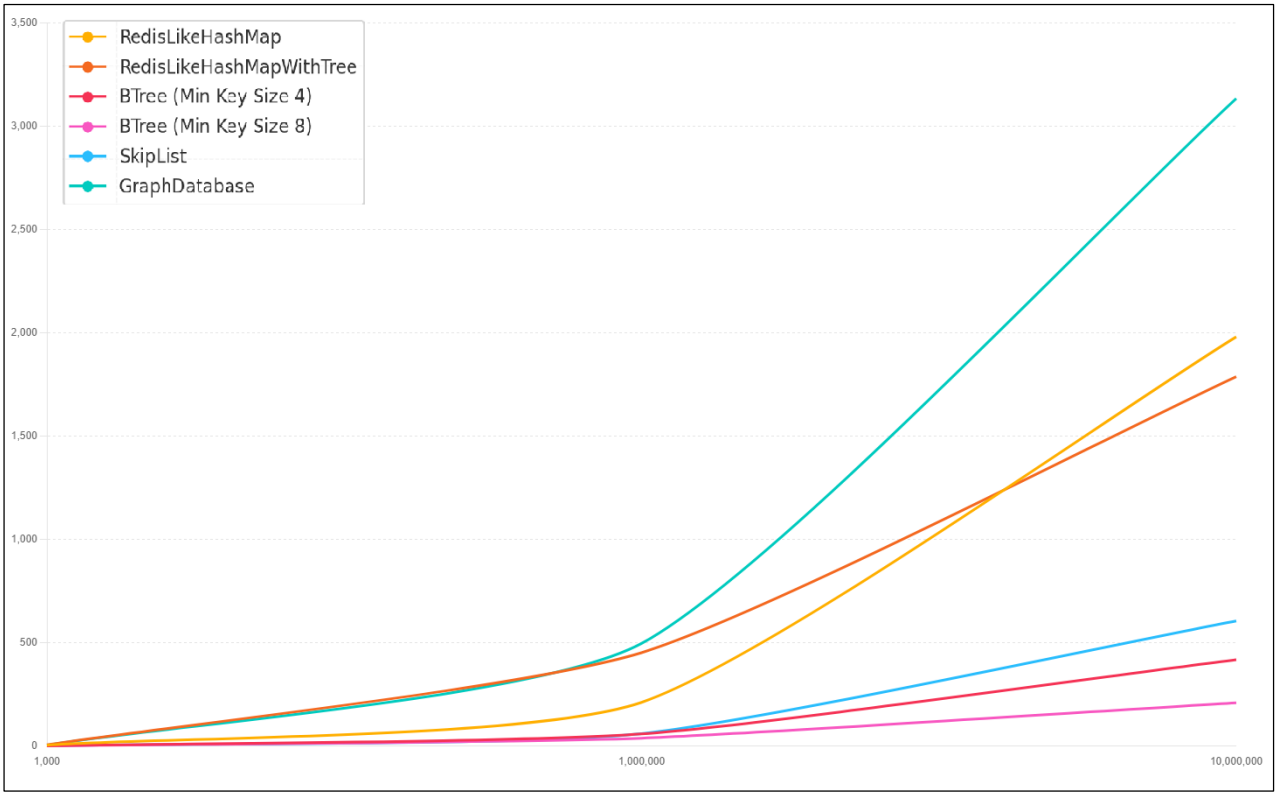

Рисунок 5.5 – Графік зайнятої пам'яті описаних структур

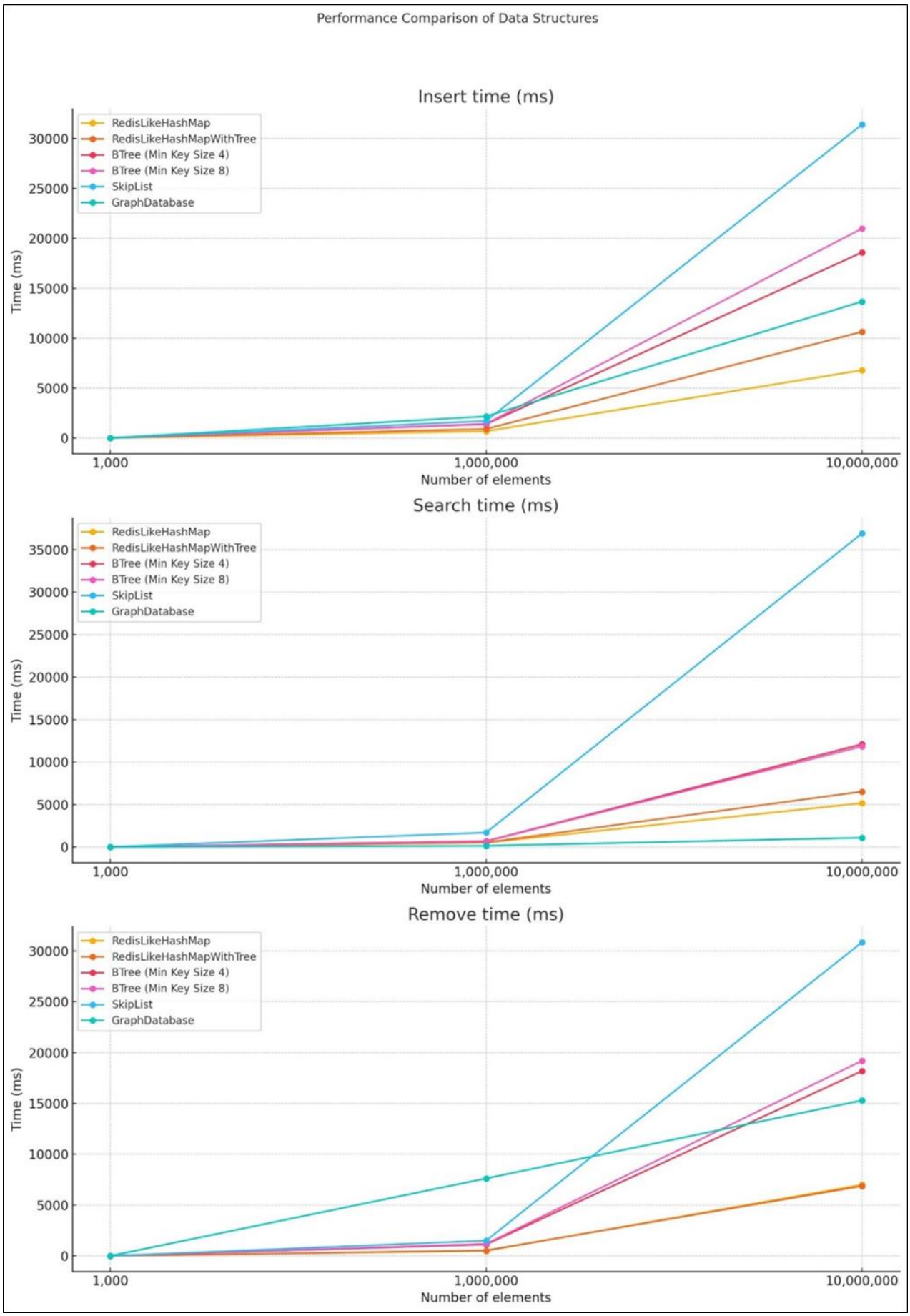

Рисунок 5.6 – Графіки швидкодії описаних структур

Геш-таблиці з відкритою адресацією показали кращі результати по вставці та видаленню ключів для невеликих та середніх обсягів даних, але поступалися іншим структурам при пошуку.

Геш-таблиці з бакетами на основі бінарних дерев продемонстрували більш стабільний час пошуку, але вимагали більше пам'яті.

Б-дерева забезпечили хорошу продуктивність при великих обсягах даних завдяки збалансованій структурі, але їхня реалізація виявилася складнішою.

Скіп-списки показали найкращий час вставки для великих обсягів даних, але потребували більше пам'яті та мали гірший час видалення.

Графові сховища даних були ефективними для складних зв'язків між даними, але їхня продуктивність знижувалася з ростом кількості зв'язків, також там зберігаються зв'язки поміж нодами, тому пам'яті використовується значно більше.

Загалом, вибір структури даних має базуватися на конкретних вимогах до системи, таких як тип операцій (вставка, пошук, видалення), інтерфейси доступу, обсяги даних та обмеження по пам'яті. Кожна з реалізованих структур має свої унікальні особливості, які слід враховувати при проектуванні ефективної системи керування даними.

ВИСНОВКИ

Під час написання кваліфікаційної роботи магістра було виконано дослідження IMDb та визначено їх основні переваги та недоліки. А також для більшого глибокого розуміння роботи IMDb, були порівняні структури даних на яких вони базуються.

Для досягнення мети написання кваліфікаційної роботи було виконано всебічний аналіз IMDb.

Для дослідження були обрані п'ять ключових інструментів: Redis, MemSQL, Apache Ignite, VoltDB, Cassandra. У ході роботи були оцінені їхні можливості, переваги та обмеження. Дослідження включало формулювання основних гіпотез, розробку експериментального плану та проведення детального аналізу. В рамках проекту був створений комплексний план, який визначав цільові інструменти, критерії оцінки та методологію аналізу даних

У рамках дослідження ефективності IMDb було сформульовано ключові гіпотези. Гіпотези стосуються продуктивності, масштабованості, надійності, сумісності та інтеграції, а також вартості та економічності використання різних IMDb. Представлені гіпотези розглядають вплив вибору конкретної IMDb на швидкодію, масштабованість, рівень надійності, сумісність з іншими системами та загальні витрати на впровадження та підтримку.

Було створено експериментальний план для перевірки гіпотез, який включає нормалізацію даних методом лінійної адитивної згортки. Цей підхід дозволяє вирівнювати величини та створювати об'єктивну основу для порівняння. Додатково було застосовано альтернативний принцип Парето через метод прийняття рішень для систематичного порівняння та оцінки кожної системи керування базами даних. Це допомогло виділити ключові аспекти, які впливають на вибір найбільш ефективної IMDb в контексті обробки даних в оперативній пам'яті.

Результати вказали на індивідуальні переваги кожного інструменту в своїй сфері. Redis виділяється високим показником продуктивності, в той час як Apache Ignite володіє вражаючими показниками масштабованості та сумісності. MemSQL

виявляється більш збалансованим за різними критеріями, зокрема високими показниками продуктивності та надійності. VoltDB виявляється загальним лідером, з високими балами у всіх аспектах, зокрема у продуктивності та масштабованості. З іншого боку, Cassandra показує себе відмінно у масштабованості та сумісності, але має низькі показники продуктивності. Результати аналізу дозволяють визначити оптимальний вибір в залежності від конкретних вимог та пріоритетів користувача чи організації.

Щоб краще зрозуміти різницю між цими IMDb було проведено детальний аналіз їх підходів до структуризації даних. Було виявлено, що різні IMDb використовують специфічні алгоритми для оптимізації зберігання та доступу до даних, що впливає на їх продуктивність та ефективність у різних сценаріях використання. Наприклад, Redis базується на Геш-таблицях, Скіп-листах та інших структурах, VoltDB в свою чергу найбільше використовує комбінації різних дерев, а Neo4j використовує графи. В рамках експерименту було розглянуто найпопулярніші структури даних такі, як графи, Геш-таблиці, Скіп-листи та Б-дерева. Кожна з них мала свої переваги та недоліки, якісь займають більше пам’яті, в якихось більший час операцій.

Тому IMDb використовують велику кількість оптимізації, комбінацій із структур даних, більш складні структури для забезпечення більшої гнучкості та масштабованості.

Таким чином, IMDb представляють собою потужний інструмент для підвищення продуктивності та ефективності обробки даних у сучасних IT-системах. Вибір конкретної системи повинен базуватися на аналізі вимог до продуктивності, масштабованості, надійності, сумісності та економічності, а також на розумінні алгоритмів, які використовуються для зберігання та обробки даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. In-Memory Database Market – Global Industry Analysis. URL: https://www.maximizemarketresearch.com/market-report/in-memory-database-market/13339/ (дата звернення 02.04.2024).

2. In-memory databases: use cases and pros-cons. URL: https://arunininfosys.medium.com/in-memory-databases-use-cases-and-pros-cons-f68cbee572c0 (дата звернення 10.04.2024).

3. Vashchenko M. Optimizing data organization: a journey into sorted string tables // 28th International Youth Forum "Radioelectronics and Youth in the 21st Century". Coll. forum materials. T. 6., - Kharkiv: Khnure. 2024. – P. 448 – 449. URL: https://openarchive.nure.ua/entities/publication/1d16e319-fde0-4d2d-ae64-f3ade2a135e1.

4. Ващенко М. С., Кравець Н.С. Дослідження СКБД для обробки даних у оперативній пам’яті // Міжнародний науковий журнал «Грааль науки» за матеріалами: VI Міжнародної науково-практичної конференції: Наука про постіндустріальне суспільство: процеси глобалізації та трансформації. – Вінниця: ГО «Європейська наукова платформа», 2023. – No 34. – С. 182 – 184. URL: https://archive.journal-grail.science/index.php/2710-3056/issue/view/08.12.2023/22.

5. Mark L. Gillenson. Fundamentals of Database Management Systems: Wiley. John Wiley & Sons, LTD, 2024. – P. 104 – 245.

6. Paulraj Ponniah. Database Design and Development: An Essential Guide for IT Professionals: Wiley-IEEE Press; 1st edition, 2003 – P. 554.

7. Andreas Meier, Michael Kaufmann. SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management: Springer Vieweg; 1st ed. 2019 edition – P. 10 – 245.

8. Josiah L. Carlson. Redis in Action: Manning Publications, 2013. – P. 152–245.

9. Devraj Bardhan, Axel Baumgartl, Nga-Sze Choi, Mark Dudgeon, Piotr Górecki, Asidhara Lahiri, Bert Meijerink, Andrew Worsley-Tonks. SAP S/4HANA: An

Introduction: SAP Press; Fourth edition, 2021. – P. 44 – 240.

10. Bill Chambers, Matei Zaharia. The Definitive Guide: Big Data Processing Made Simple: O'Reilly Media; 1st edition, 2018. – P. 22 – 230.

11. Using VoltDB (2022) URL: https://docs.voltdb.com/UsingVoltDB/ (дата звернення 16.04.2024).

12. Building Knowledge Graphs: A Practitioner's Guide: https://neo4j.com/knowledge-graphs-practitioners-guide/ (дата звернення 20.04.2024).

13. Second Edition. The Algorithm Design Manual: Springer-Verlag London Limited, 2008. – P. 739.

14. Aditya Y Bhargava Foreword by Daniel Zingaro. Grokking Algorithms, Second Edition: Manning; 2nd edition, 2024. – P. 320.

15. Neo4j 4.0 graph database platform brings unlimited scaling URL:https://techcrunch.com/2020/02/04/neo4j-4-0-graph-database-platform-brings-unlimited-scaling/(дата звернення 05.05.2024).

16. A Gentle Introduction To Graph Theory (2017) URL: https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8 (дата звернення 07.05.2024).

17. Jeff Carpenter, Eben Hewitt. Cassandra: The Definitive Guide, (Revised) Third Edition: Distributed Data at Web Scale: O'Reilly, 2022. – P. 450.

18. AWS Marketplace URL: https://aws.amazon.com/marketplace (дата звернення 11.05.2024).

19. Allen B. Downey, Chris Mayfield. Think Java: How to Think Like a Computer Scientist: O'Reilly, 2016– P. 252.

20. Roberto Tamassia, Michael H. Goldwasser, Michael T. Goodrich. Data Structures and Algorithms in Java, International Student Version: Wiley; 6th Edition International Student Version, 2014. – P. 720.

21. Мартин Клеппман. Високонавантажених програми. Програмування, масштабування, підтримка: Print2print P. 202 – 255.

22. Narsingh Deo. Graph Theory with Applications to Engineering and Computer Science: Prentice-Hall of India Pvt.Ltd; 1st edition, 2004. - P. 34 – 55.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

3. Vashchenko M, Kravets N. Optimizing data organization: a journey into sorted string tables // 28th International Youth Forum "Radioelectronics and Youth in the 21st Century". Coll. forum materials. T. 6., - Kharkiv: Khnure. 2024. – P. 448 – 449. URL: https://openarchive.nure.ua/entities/publication/1d16e319-fde0-4d2d-ae64-f3ade2a135e1.

4. Ващенко М. С., Кравець Н.С. Дослідження СКБД для обробки даних у оперативній пам’яті // Міжнародний науковий журнал «Грааль науки» за матеріалами: VI Міжнародної науково-практичної конференції: Наука про постіндустріальне суспільство: процеси глобалізації та трансформації. – Вінниця: ГО «Європейська наукова платформа», 2023. – No 34. – С. 182 – 184. URL: https://archive.journal-grail.science/index.php/2710-3056/issue/view/08.12.2023/22.