МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

БАБЕНКО Олександр Валерійович

Метод багатокритеріального аналізу для вибору SQL або NoSQL баз даних залежно від умов проєкту / A Multi-Criteria Analysis Method for Choosing SQL or NoSQL Databases Based on Project Conditions

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КНм-21
О.В. Бабенко
_____

Науковий керівник:
к.т.н., доцент Д.І. Загородня
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
В.о. завідувача кафедри
_____ Н.М. Васильків

ТЕРНОПІЛЬ - 2024

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «магістр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ М.П. Комар
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Бабенко Олександр Валерійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Метод багатокритеріального аналізу для вибору SQL або NoSQL баз даних залежно від умов проєкту / A Multi-Criteria Analysis Method for Choosing SQL or NoSQL Databases Based on Project Conditions.
керівник роботи к.т.н., доцент Д.І. Загородня,
затверджені наказом по університету від 12 грудня 2023 року № 753.

2. Строк подання студентом закінченої кваліфікаційної роботи 4 грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити
- огляд предметної області баз даних;
- аналіз методів багатокритеріального аналізу для вибору бази даних;
- огляд існуючих рішень для вибору бази даних;
- постановка завдання дослідження;
- розробка методології для проведення аналізу;
- опис використаних даних та методів їх обробки;
- реалізація порівняльного аналізу методів баз даних;
- аналіз результатів експериментальних досліджень.

5. Перелік графічного матеріалу у роботі
- схема архітектури NoSQL баз даних;
- діаграма аналізу часу виконання операцій MS SQL та MongoDB;
- діаграма аналізу пропускної здатності MS SQL та MongoDB;

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи.	до 01.01. 2024 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.03. 2024 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 20.05.2024 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 28.10. 2024 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 11.11.2024 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів.	до 25.11.2024 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту.	до 30.11.2024 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 04.12.2024 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії.	до 14.12. 2024 р.	

Студент _____ О.В. Бабенко
підпис

Керівник роботи _____ к.т.н., доцент Д.І. Загородня
підпис

РЕЗЮМЕ

Кваліфікаційна робота на тему «Метод багатокритеріального аналізу для вибору SQL або NoSQL баз даних залежно від умов проєкту» виконана для здобуття освітнього ступеня «Магістр» за спеціальністю 122 «Комп'ютерні науки» в межах освітньо-професійної програми «Комп'ютерні науки» написана обсягом 71 сторінка і містить 5 ілюстрацій, 5 таблиць, 3 додатки та 40 використаних джерел.

Метою роботи є розробка методу багатокритеріального аналізу для вибору бази даних залежно від специфіки проєкту. Дослідження зосереджене на аналізі реляційних (SQL) та нереляційних (NoSQL) баз даних, їхніх переваг, недоліків та особливостей застосування.

Методи дослідження: методи оцінки продуктивності баз даних; розробка системи критеріїв оптимальності та їхнє практичне тестування з використанням багатокритеріального підходу.

Результати дослідження: розроблено методологію для порівняння баз даних із урахуванням критеріїв продуктивності, масштабованості, консистентності, типу даних та вартості; проведено порівняльний аналіз SQL і NoSQL баз даних у різних проєктних умовах.

Практичне значення роботи полягає у створенні методу для вибору баз даних, який може бути застосований у реальних умовах. Це дозволяє оптимізувати ресурси та підвищити ефективність вибору відносно вимог проекту.

Рекомендації для подальших досліджень включають удосконалення методу шляхом включення більшого спектру баз даних і додаткових критеріїв.

Ключові слова: SQL, NOSQL, БАГАТОКРИТЕРІАЛЬНИЙ АНАЛІЗ, ПРОДУКТИВНІСТЬ, МАСШТАБОВАНІСТЬ, КОНСИСТЕНТНІСТЬ, ВИБІР БАЗИ ДАНИХ.

ABSTRACT

The qualification work on the topic «A Multi-Criteria Analysis Method for Choosing SQL or NoSQL Databases Based on Project Conditions» for Master's degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 71 pages and it contains 5 figures, 5 tables, 3 annexes and 40 sources.

The aim of the qualification work is to develop a multi-criteria analysis method for selecting a database based on the specific requirements of a project. The research focuses on analyzing relational (SQL) and non-relational (NoSQL) databases, exploring their advantages, disadvantages, and application features.

Research methods used in the study uses methods for evaluating database performance, developing a system of criteria optimization, and practical testing through a multi-criteria approach.

Research results: a methodology was developed for comparing databases based on criteria such as performance, scalability, consistency, data type, and cost; a comparative analysis of SQL and NoSQL databases was conducted under various project conditions.

Practical significance: the work offers a method for database selection applicable in real-world conditions, it enables resource optimization and enhances the efficiency of database selection in alignment with project requirements.

Recommendations for further research include improving the method by incorporating a broader spectrum of databases and additional criteria.

Keywords: SQL, NOSQL, MULTI-CRITERIA ANALYSIS, PERFORMANCE, SCALABILITY, CONSISTENCY, DATABASE SELECTION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACID - Atomicity, Consistency, Isolation, Durability

API - Application Programming Interface

ARIMA - Auto Regression Integrated Moving Average

AS - Autonomous System

BGP - Border Gateway Protocol

BSON - Binary JSON

CRUD - Create, Read, Update, Delete

JDBC - Java Database Connectivity

JSON - JavaScript Object Notation

LSTM - Long Short-Term Memory

NoSQL - Not Only SQL

ODBC - Open Database Connectivity

SQL - Structured Query Language

XML - Extensible Markup Language

YCSB - Yahoo! Cloud Serving Benchmark

ВСТУП

Зростання обсягів даних і складності сучасних інформаційних систем вимагає ефективних підходів до вибору технологій для їх зберігання та обробки. Сучасні компанії та організації збирають і зберігають величезні обсяги даних, які постійно зростають не лише за обсягом, а й за складністю та варіативністю. Вибір бази даних, яка найкраще відповідає потребам конкретного проєкту, є одним із ключових завдань при проєктуванні архітектури інформаційних систем. Залежно від типу даних та умов роботи, розробники можуть обирати між реляційними базами даних (SQL) та нереляційними (NoSQL), кожна з яких має свої переваги та недоліки.

Актуальність теми дослідження зумовлена потребою у підвищенні ефективності прийняття рішень при виборі баз даних, зокрема у проєктах із високими вимогами до масштабованості, продуктивності та надійності. В умовах зростаючих обсягів даних і вимог до ефективності систем, необхідність оптимального вибору між SQL та NoSQL базами стає критично важливою. Універсальних рішень для всіх випадків не існує, оскільки кожен тип бази даних краще підходить для певних сценаріїв: SQL для структурованих даних і складних запитів, а NoSQL для великомасштабних, розподілених і нереляційних даних.

Метою даного дослідження є розробка та впровадження методу багатокритеріального аналізу вибору архітектури баз даних, який дозволяє врахувати специфіку проєкту та зробити обґрунтований вибір між реляційними і нереляційними базами даних.

Завданнями дослідження є:

- аналіз особливостей та застосувань SQL і NoSQL баз даних;
- розробка методології для багатокритеріального аналізу вибору архітектури баз даних;
- впровадження алгоритмів оцінювання, що дозволяють враховувати числові та якісні показники;
- тестування методу на прикладі реальних проєктів із великими обсягами даних.

Об’єктом дослідження є процес вибору архітектури баз даних для зберігання та обробки даних у сучасних інформаційних системах. Предметом дослідження є методи багатокритеріального аналізу, які дозволяють оцінити переваги та недоліки SQL і NoSQL баз даних у конкретних умовах.

Для досягнення поставленої мети були використані такі методи дослідження, як аналіз існуючих рішень для вибору баз даних, методи багатокритеріального аналізу, порівняння ефективності SQL і NoSQL баз даних, а також експериментальні тестування.

Наукова новизна дослідження полягає в розробці нового підходу до об’єктивного вибору бази даних за допомогою методу багатокритеріального аналізу, який дозволяє враховувати як числові показники (швидкість виконання запитів, затримки, продуктивність), так і якісні (масштабованість, гнучкість архітектури).

Практичне значення одержаних результатів полягає у створенні алгоритмів і методик, які можна використовувати для прийняття рішень у реальних умовах розробки програмного забезпечення, що потребує обґрунтованого вибору між SQL і NoSQL базами даних. Це допоможе знизити витрати на розробку, підвищити ефективність роботи систем і уникнути помилок при масштабуванні.

Апробація результатів дослідження здійснювалася шляхом представлення основних теоретичних положень і практичних результатів на Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі», а також на Міжнародній науково-практичній інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення»

1 ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ ВИБОРУ АРХІТЕКТУРИ БАЗИ ДАНИХ

1.1 Аналіз прикладної області

Традиційні системи баз даних, які використовуються для зберігання інформації, ґрунтуються на реляційній моделі [1]. Їх часто називають SQL-базами даних, оскільки вони використовують мову структурованих запитів (SQL — Structured Query Language) для взаємодії з даними. Ці бази організовані у вигляді таблиць, які мають чітко визначену структуру — схему. Схема включає визначення типів даних, які зберігаються, а також зв'язків між таблицями.

SQL-бази даних добре підходять для роботи з організованою, структурованою інформацією. Вони забезпечують високу цілісність даних, дозволяючи дотримуватися суворих правил зв'язків між таблицями, таких як "один до багатьох" чи "багато до багатьох". Крім того, мова SQL дозволяє виконувати складні запити, об'єднувати дані з різних таблиць та отримувати аналітичну інформацію. Особливою перевагою реляційних баз даних є підтримка транзакцій. Завдяки цьому вони можуть гарантувати, що всі операції з даними будуть виконані правильно, навіть у випадку збоїв системи. Ця властивість відома як дотримання принципів ACID: атомарність, консистентність, ізольованість та довговічність.

Однак із розвитком технологій та значним збільшенням обсягів даних SQL-бази даних почали стикатися з певними обмеженнями. Наприклад, їм важко працювати з неструктурованими або напівструктурованими даними, такими як мультимедійні файли чи журнали подій. Також вони мають обмежені можливості масштабування, що стало особливо актуальним у часи великих даних (Big Data). Ці виклики стали стимулом для появи альтернативних підходів, які пропонують більше гнучкості для роботи зі складними та розподіленими системами даних. Ці бази даних широко відомі як бази даних NoSQL, що чітко відрізняє їх від традиційних баз даних SQL. Більшість із них базуються на зберіганні простих пар ключ-значення на передумові, що простота веде до швидкості.

Зі збільшенням доступності інтернету та доступністю дешевих сховищ величезні обсяги структурованих, напів структурованих і неструктурованих даних збираються та зберігаються для різноманітних програм. Такі дані зазвичай називають Big Data [2]. Обробка такого величезного обсягу даних вимагає швидкості, гнучких схем і розподілених (тобто нецентралізованих) баз даних. Бази даних NoSQL стали переважною валютою для роботи з великими даними, які, як вони стверджують, відповідають цим вимогам. Це також призвело до різкого збільшення кількості пропозицій баз даних NoSQL. Існує кілька комерційних і відкритих реалізацій баз даних NoSQL (таких як BigTable [3] і HBase [4]).

Велика кількість пропозицій NoSQL призводить до питань про відмінності між цими пропозиціями та їхню придатність для конкретних програм. Тому було опубліковано низку оглядових статей (таких як Tudorica та Bucur [5] або Han та ін. [6]), щоб відповісти на деякі з цих питань. Крім того, існує кілька онлайн-ресурсів і блогів, які також присвячені цим аспектам.

Поширена думка, що GoogleBigTable [6] була першою базою даних NoSQL. BigTable базується на трьох ключах: перший називається ключем рядка, другий називається ключем стовпця, а третій є міткою часу. Фактично, це багатовимірна карта. Ключі стовпців можна класифікувати на групи. Доступ до групи здійснюється як єдине ціле.

Успіх нереляційних баз даних, таких як BigTable і AmazonDynamo [7] ініціював ряд інших розробок нереляційних баз даних із відкритим і закритим кодом. Популярність цих баз даних NoSQL зросла завдяки простоті доступу, швидкості та масштабованості.

Більшість баз даних NoSQL засновані на зберіганні пар ключ-значення. Цілком можливо, що значення можуть бути набором вторинних ключів, які, у свою чергу, містять значення. Спеціальним типом бази даних пар ключ-значення є база даних сімейства стовпців. Він складається зі стовпців і суперстовпців, адресованих за допомогою ключів. Суперстовпець об’єднує кілька пов’язаних стовпців, доступ до якого здійснюється як єдиний блок. Іншим особливим типом бази даних пар ключ-значення є документно-орієнтована база даних. Бази даних, орієнтовані на

документи, виходять за рамки простих значень і мають можливість зберігати об’єкти. Ці об’єкти знаходяться в певній серіалізованій формі, наприклад XML, JSON і BSON (JSON у двійковому кодуванні).

MongoDB — документно-орієнтована база даних, написана мовою C++ [8]. Об’єкти зберігаються серіалізованими як BSON. Об’єкти не обов’язково мають однакову структуру чи поля, а загальні поля не обов’язково мають однаковий тип, що забезпечує гнучке зберігання схем. MongoDB підтримує автоматичний шардинг, коли він розділяє колекції даних і зберігає розділи на доступних серверах. Це призводить до динамічно збалансованого навантаження.

Hypertable — це база даних NoSQL з відкритим кодом, заснована на Google's BigTable [3]. Це база даних сімейства стовпців, написана мовою C++. Як і MongoDB, Hypertable також підтримує автоматичний шардинг.

Apache CouchDB — документно-орієнтована база даних, написана мовою Erlang [9]. Об’єкти зберігаються в серіалізованому форматі JSON. Доступ до бази даних здійснюється через RESTful HTTP API. Таким чином, CouchDB вважається «база даних, яка повністю охоплює Інтернет».

Apache Cassandra — це база даних сімейства стовпців, спочатку розроблена Facebook [10]. Деякі з його ключових функцій включають децентралізацію для зменшення збоїв і реплікацію даних для підвищення відмовостійкості.

RavenDB і Couchbase — це ще дві документоорієнтовані бази даних, які ми розглядаємо [11]. Подібно до MongoDB і CouchDB, вони також пропонують гнучке зберігання схем і використовують JSON як формат для серіалізованих об’єктів. Вони також пропонують реплікацію та шардинг даних.

У той час як бази даних NoSQL мають переваги у швидкості та масштабованості, вони мають ряд недоліків порівняно з традиційними реляційними базами даних. Лівітт перераховує ці проблеми [12]. Він зазначає, що бази даних NoSQL, незважаючи на те, що вони швидкі для простих завдань, вимагають багато часу для складних операцій. Крім того, може бути складно формувати запити для складних операцій. Іншим недоліком є відсутність вбудованої підтримки узгодженості. Лівітт також зазначає, що NoSQL — це

технологія, яку багатьом організаціям ще належить освоїти, а також не вистачає підтримки та інструментів управління, які могли б допомогти.

Bartholomew дає підручник із вступом до історії та відмінностей між базами даних SQL та NoSQL [13]. Сакр та ін. обговорити рішення для управління даними, включаючи NoSQL, для хмарних платформ [14]. Вони обговорюють проблеми, з якими стикаються рішення для управління даними в світлі хмари.

Tiwari надає вичерпний трактат про бази даних NoSQL [15]. Він розповідає про історію, обґрунтування, можливість програмування, архітектуру зберігання та налаштування продуктивності деяких реалізацій. Він також зазначає наступне як основу для порівняння реалізацій: масштабованість, послідовність, підтримка моделей даних, підтримка запитів і інструменти управління.

Подібним чином Індраван-Сантьяго зазначає наступне як базове порівняння: модель даних, модель транзакцій, підтримка ad-hoc запитів, індексування, шредування та тип ліцензії [16]. На цій основі вона порівнює десять реалізацій NoSQL, включаючи Cassandra, HBase, Dynamo, MongoDB і CouchDB. Вона також якісно порівнює реляційні бази даних з базами даних NoSQL і робить висновок, що NoSQL, ймовірно, доповнить реляційні бази даних і покращить можливості керування базами даних.

Хехт і Яблонскі надають огляд баз даних NoSQL, орієнтований на випадки використання [17]. Вони визначають труднощі у виборі бази даних NoSQL відповідно до конкретного випадку використання. Вони використовують як основу для свого порівняння модель даних, підтримку запитів, розділення, реплікацію та керування паралелізмом. У цьому світлі вони порівнюють чотирнадцять баз даних NoSQL, включаючи MongoDB, CouchDB, Cassandra та HBase.

Boicea та ін. порівняти базу даних NoSQL із базою даних SQL. Вони обирають Oracle для реалізації SQL і MongoDB для реалізації NoSQL [18]. Вони повідомляють, що з великою кількістю записів час вставки є більш важливим фактором в Oracle, а час оновлення та видалення є на кілька факторів більшим у Oracle.

1.2 Методи та інструменти оцінки продуктивності баз даних

Навантажувальне тестування баз даних — це процес оцінки продуктивності системи при високих обсягах операцій читання, запису, оновлення та інших типів запитів. Воно допомагає визначити, як база даних реагує на різні рівні навантаження, і виявляє потенційні вузькі місця або обмеження конфігурації. Існує кілька підходів та інструментів для такого тестування, які варіюються за складністю, типами навантаження та спеціалізацією:

1. Симуляція реального навантаження — використання інструментів, які відтворюють сценарії роботи реальних користувачів, наприклад, транзакції, аналітичні запити чи великі вставки даних.
2. Тестування пропускної здатності — визначення максимальної кількості запитів, які система може обробити за певний час.
3. Тестування затримки (Latency) — оцінка часу виконання окремих запитів або транзакцій.
4. Тестування масштабованості — перевірка, як продуктивність змінюється зі збільшенням обсягу даних, кількості підключень чи серверів.

Серед популярних інструментів — HammerDB для реляційних баз, YCSB для NoSQL, Sysbench для гнучкого тестування реляційних баз і системних ресурсів, а також спеціалізовані засоби на основі JMeter чи власних сценаріїв. Кожен з методів має свої переваги, залежно від специфіки бази даних і цілей тестування.

Yahoo! Cloud Serving Benchmark (YCSB) – це робота з відкритим кодом інструмент генератора навантаження для порівняння сховищ ключ-значення [19]. Його було розроблено для вирішення потреби в стандартизованому підході до тестування масштабованості та ефективності хмарних баз даних у різних умовах навантаження.

YCSB дозволяє створювати та налаштовувати різноманітні сценарії роботи баз даних, такі як операції читання, запису, сканування або змішані робочі навантаження. Це забезпечує розробникам і дослідникам можливість точно імітувати реальні умови експлуатації баз даних. Інструмент аналізує важливі

метрики, включаючи пропускну здатність (кількість операцій на секунду) і затримки виконання (latency), які є критичними для оцінки продуктивності.

Однією з переваг YCSB є його гнучкість і масштабованість. Він підтримує різні моделі розподілу ключів, зокрема рівномірний розподіл, що дозволяє тестувати системи в умовах різних типів навантаження. Крім того, YCSB є модульним, що дає змогу розширювати його функціональність шляхом додавання драйверів для нових систем баз даних.

HammerDB — це потужний інструмент для тестування продуктивності баз даних, який дозволяє проводити навантажувальні тести та оцінювати продуктивність реляційних баз даних [20]. HammerDB працює за допомогою стандартних тестових навантажень, що дають змогу симулювати реальні бізнес-операції. Інструмент дає змогу створювати та налаштовувати власні сценарії для тестування і може використовуватися як на локальних машинах, так і на кластерних рішеннях.

Основні можливості HammerDB включають можливість налаштування кількості користувачів для тестування навантаження, моніторинг результатів у реальному часі та автоматичний аналіз виконання запитів. Крім того, інструмент має зручний інтерфейс, що дозволяє зберігати та порівнювати результати тестів, надаючи цінну інформацію для оптимізації налаштувань бази даних.

Хоча HammerDB найбільше орієнтований на SQL, він не підтримує тестування NoSQL баз даних безпосередньо. Проте для SQL-систем HammerDB є одним із найпоширеніших і найефективніших інструментів для аналізу продуктивності та оптимізації налаштувань.

Sysbench — це інструмент для тестування продуктивності, який використовується для оцінки реляційних (SQL) баз даних, таких як MS SQL, а також для тестування файлових систем та інших компонентів системи [21]. Основні тести включають оцінку швидкості додавання нових даних, вимірювання швидкості читання даних, перевірку ефективності оновлення записів та оцінку продуктивності кількох операцій в рамках однієї транзакції. Sysbench дозволяє налаштовувати кількість потоків і підключень для створення різних рівнів

навантаження. Інструмент простий у використанні, має командний рядок і підходить для швидкого тестування продуктивності баз даних і файлових систем.

1.3 Аналіз існуючих рішень

Бізнеси використовують NoSQL бази даних для підвищення своєї конкурентоспроможності та збільшення загальної рентабельності інвестицій (ROI). Однією з головних переваг найпопулярніших NoSQL баз даних, таких як MongoDB, Couchbase, CouchDB і DynamoDB, є їхня доступність завдяки наявності хмарних рішень для обробки великих даних. Це робить їх чудовим вибором для менших організацій із обмеженим бюджетом.

NoSQL бази даних розроблені так, щоб бути гнучкими та здатними відповідати вимогам управління даними, які висувають різні установи. Вони особливо ефективно працюють із напівструктурованими та неструктурованими даними, з якими реляційні бази даних справляються значно гірше. Їхня нетаблична структура дозволяє зберігати та отримувати інформацію з широким спектром джерел даних, що надає значну гнучкість у роботі з різними типами інформації.

Особливістю NoSQL баз є здатність працювати без строгої табличної схеми, що ідеально підходить для напівструктурованих і неструктурованих даних. Завдяки цьому ці бази даних дозволяють ефективно зберігати, швидко обробляти та зручно отримувати інформацію з різноманітних джерел.

На рисунку 1.1 представлена архітектура баз даних MongoDB та Cassandra, яка демонструє їхню структуру та особливості роботи. У MongoDB дані розподіляються між окремими сегменти даних (shard), кожен із яких являє собою набір реплік, що гарантує стійкість і надійність даних. Для координації запитів використовується компонент під назвою Mongos. Mongos отримує запити від клієнтів і визначає, до якого сегменту їх направити. Конфігураційний сервер MongoDB зберігає всю інформацію про розподіл даних і виступає центральним елементом для управління кластером. У цьому підході важливу роль відіграє

централізація управління, яка забезпечує ефективну маршрутизацію, але може стати потенційною точкою відмови.

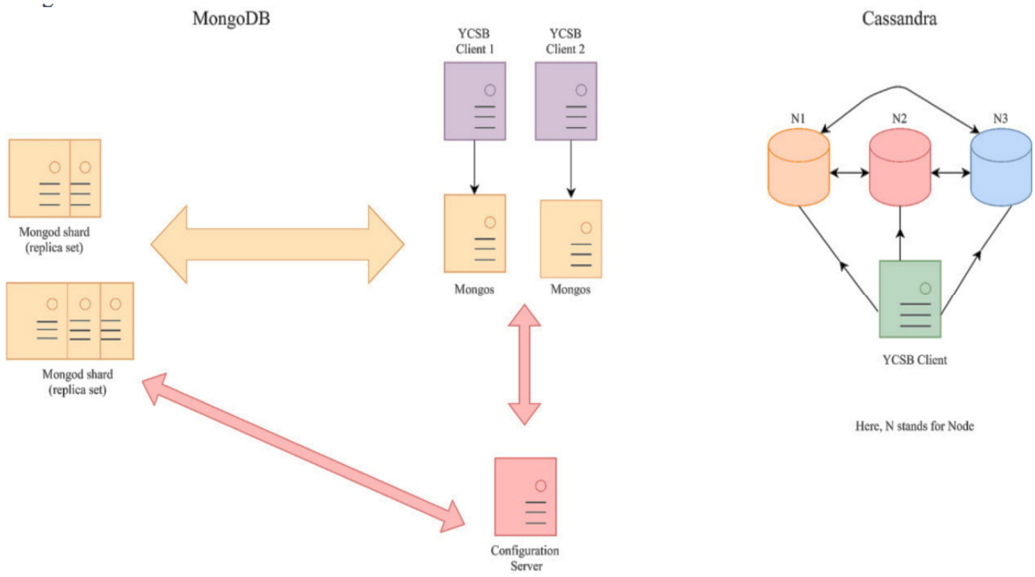


Рисунок 1.1 – Архітектура NoSQL баз даних

У Cassandra архітектура побудована інакше. Всі вузли в кластері рівноправні, немає централізованого координатора або конфігураційного сервера. Це означає, що будь-який вузол може приймати запити, реплікувати дані та обробляти навантаження. Дані автоматично розподіляються між вузлами, забезпечуючи високу доступність і відмовостійкість.

1.4 Постановка задачі дослідження

Мета дослідження — розробити метод багатокритеріального аналізу для вибору архітектури бази даних в залежності від умов проєкту. Метод має дозволити об’єктивну оцінку переваг і недоліків реляційних (SQL) та нереляційних (NoSQL) підходів залежно від особливостей даних і специфіки задачі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Здійснити аналіз особливостей SQL та NoSQL баз даних. Дослідити ключові характеристики SQL та NoSQL баз даних, їхні сильні та слабкі сторони, а

також сфери застосування для різних типів даних та вимог до продуктивності. А також визначити параметри (наприклад, обсяги даних, типи операцій, вимоги до часу відгуку), які є вирішальними для вибору між SQL і NoSQL в залежності від умов у.

2. Вивчення існуючих методів оцінки продуктивності баз даних. Оцінити сучасні методи й підходи, що використовуються для аналізу продуктивності баз даних, а також для вимірювання метрик, таких як швидкість виконання запитів, обсяг споживаної пам'яті, масштабованість, енергоефективність тощо. Виявити наявні методи оцінки ефективності при різних робочих навантаженнях.

3. Розробка метрик для оцінки ефективності між SQL і NoSQL. Визначити ключові метрики, які дозволяють оцінити ефективність між SQL і NoSQL з урахуванням вимог до проєкту. Сформувати систему ваг або алгоритм для автоматичної обробки та порівняння цих метрик для об’єктивної оцінки переваг обраного підходу.

4. Експериментальна перевірка моделі на реальних та симульованих даних. Провести тестування методу на реальних датасетах та симульованих даних, що охоплюють різні сценарії використання SQL та NoSQL. Використати різні реальні датасети, що представляють різноманітні типи даних, такі як транзакційні дані, дані для аналітики, неструктуровані дані з веб-сайтів або медіа-файлів. Провести симуляцію даних для різних сценаріїв, включаючи розподілені системи, гібридні бази даних, а також різні навантаження, що можуть зустрічатися у реальних проєктах (наприклад, великий обсяг одночасних запитів, високі вимоги до часу відгуку, великі обсяги даних).

Проаналізувати результати тестування, перевірити точність рекомендацій моделі, і провести порівняння з результатами, отриманими традиційними методами. Оцінити швидкість виконання запитів, час відповіді при різних обсягах даних та навантаженнях. Перевірити ефективність масштабованості баз даних при зростанні обсягів даних або кількості одночасних користувачів. Визначити, наскільки енергоефективними є різні бази даних при виконанні типових запитів або операцій.

5. На основі отриманих результатів сформулювати практичні рекомендації щодо доцільності та економічної ефективності переходу між SQL і NoSQL для різних сценаріїв використання.

Така постановка задачі забезпечує цілісну та послідовну структуру для дослідження, яка дозволить отримати об'єктивну і комплексну оцінку та доцільності переходу між SQL та NoSQL базами даних.

Висновки до розділу 1

1. Проведено аналіз прикладної області. Встановлено, що сучасні інформаційні системи потребують баз даних, здатних обробляти великі обсяги даних із забезпеченням продуктивності та масштабованості. Для різних проєктів застосовуються як реляційні (SQL), так і нереляційні (NoSQL) бази даних, залежно від типу даних, потреб у масштабуванні та складності запитів.

2. Досліджено методи оцінки продуктивності баз даних, зокрема, такі інструменти як HammerDB, YCSB та Sysbench. Було відзначено їхню здатність виконувати навантажувальні тести, оцінювати затримки, пропускну здатність та інші критичні параметри, важливі для ефективного вибору баз даних.

3. Розглянуто особливості реляційних і нереляційних баз даних, включаючи MongoDB, Cassandra, MS SQL та PostgreSQL. Виявлено ключові переваги та недоліки цих рішень залежно від вимог проєктів. SQL-бази підходять для проєктів із високими вимогами до консистентності та транзакційності, тоді як NoSQL забезпечують масштабованість та гнучкість.

4. Виявлено, що вибір між SQL та NoSQL базами залежить від конкретних умов проєкту: обсягів даних, вимог до швидкості, типу операцій і потреб у масштабуванні. Запропоновано базові критерії для оцінки ефективності баз даних у різних сценаріях.

2 РОЗРОБКА МЕТОДУ БАГАТОКРИТЕРІАЛЬНОГО АНАЛІЗУ БАЗ ДАНИХ

2.1 Опис використаних даних та інструментів

Для проведення дослідження використано такі інструменти як SQL Express для реляційної бази даних та MongoDB як приклад NoSQL; методологію порівняння швидкості виконання CRUD-операцій.

SQL Server Express — це безкоштовна, легка версія Microsoft SQL Server, яка призначена для розробників, малих бізнесів і навчальних цілей. Вона підтримує реляційну модель бази даних і забезпечує більшість функцій, які є у повних версіях SQL Server. Далі наведено основні характеристики, переваги та обмеження SQL Server Express для реляційних баз даних.

До основних характеристик SQL Server Express відносять:

1. SQL Server Express доступний безкоштовно, що робить його привабливим вибором для малих підприємств і розробників.

2. SQL Server Express забезпечує реляційну модель даних, що дозволяє створювати таблиці, визначати відносини між ними та виконувати SQL-запити для отримання та маніпуляції даними.

3. Обмеження на розмір бази даних. Максимальний розмір бази даних у SQL Server Express обмежений 10 ГБ. Це може бути достатнім для малих і середніх проєктів, але може стати обмеженням для великих додатків.

4. Обмеження на апаратні ресурси. SQL Server Express підтримує використання не більше ніж 1 ГБ оперативної пам'яті на процес для бази даних, а також обмежує кількість оброблювальних ядер до 4.

5. SQL Server Express може використовуватися з додатками на .NET Framework, що робить його зручним для розробки веб- і настільних додатків.

6. SQL Server Express забезпечує різноманітні механізми безпеки, включаючи аутентифікацію на основі Windows та SQL Server, а також можливість шифрування даних.

7. SQL Server Management Studio (SSMS) надає зручний інтерфейс для управління базами даних, написання SQL-запитів і адміністрування серверів.

8. Підтримка тригерів, збережених процедур та функцій: SQL Server Express підтримує тригери, збережені процедури та функції, що дозволяє створювати складну логіку роботи з даними.

Перевагами використання SQL Server Express є:

- Вартість, оскільки SQL Server Express безкоштовний і він ідеально підходить для стартапів і невеликих бізнесів з обмеженим бюджетом.
- Простий у використанні. SQL Server Express має інтуїтивно зрозумілий інтерфейс та добре документовану документацію, що полегшує навчання нових користувачів.
- SQL Server Express легко інтегрується з іншими продуктами Microsoft, такими як Visual Studio та Azure.
- SQL Server має добру репутацію за надійність та стабільність, що важливо для будь-якого програмного забезпечення, яке обробляє критично важливі дані.

MongoDB – це нереляційна, документно-орієнтована база даних типу NoSQL, що забезпечує високу гнучкість і масштабованість для зберігання та керування великими обсягами даних. На відміну від реляційних баз даних, таких як SQL Server, MongoDB зберігає дані у вигляді документів у форматі BSON (Binary JSON), що робить її ідеальною для сучасних динамічних веб-додатків, масштабованих систем, інтернету речей (IoT) та інших додатків, де структура даних може змінюватися. Основні характеристики MongoDB:

1. Документно-орієнтована модель. MongoDB зберігає дані у вигляді документів, що мають структуру JSON-подібного формату (BSON). Документи можуть містити вкладені структури, масиви та об'єкти, що робить MongoDB більш гнучкою у порівнянні з традиційними реляційними базами даних.

2. Масштабованість. MongoDB підтримує горизонтальне масштабування через шардінг (розподіл даних по різних серверах). Це означає, що базу даних можна легко розподілити на кілька серверів, забезпечуючи ефективну обробку великих обсягів даних і підтримку високого навантаження.

3. Гнучкість у зміні структури даних. На відміну від реляційних баз даних, де структура даних визначається наперед (застосовується схема), MongoDB не

вимагає суворої схеми для кожного документа. Це дозволяє зберігати різні типи даних в одному колекції та легко змінювати структуру даних, що особливо корисно для швидкозмінних або складних структур.

4. Висока продуктивність. MongoDB має оптимізовану роботу з великими обсягами даних та може ефективно обробляти складні запити завдяки індексуванню, кешуванню та підтримці складних запитів з агрегацією даних.

5. Можливості агрегації. MongoDB надає потужні інструменти для агрегації, такі як Aggregation Framework, що дозволяє обробляти та обчислювати дані на серверному рівні. Це дозволяє виконувати складні операції над даними, як-от групування, сортування, фільтрування, обчислення середніх значень, що робить MongoDB зручним для аналітичних завдань.

6. Підтримка транзакцій. У новіших версіях MongoDB є підтримка ACID-транзакцій на рівні декількох документів і колекцій, що робить її більш придатною для завдань, де потрібна цілісність даних, як в реляційних базах даних.

MongoDB підходить для широкого спектру завдань, таких як:

- Веб-додатки. MongoDB є ідеальним вибором для динамічних веб-додатків, де структура даних може часто змінюватися.
- Big Data та аналітика. MongoDB ефективно працює з великими обсягами даних, завдяки можливостям шардінгу та агрегаційним функціям.
- Інтернет речей (IoT). MongoDB підходить для IoT-додатків, де збираються величезні обсяги даних з численних сенсорів.
- Системи управління контентом (CMS). MongoDB дозволяє зберігати різні типи контенту в одному колекції, що робить її корисною для CMS-додатків.
- Соціальні мережі та месенджери. MongoDB використовується в додатках, де потрібно зберігати складні, гнучкі структури даних, як-от профілі користувачів, повідомлення, відстеження активності.

До переваг MongoDB відносять:

- Гнучкість у структурі даних – можливість зберігати дані без суворої схеми, дозволяє швидко адаптуватися до змін у вимогах.

- Масштабованість. Легка інтеграція горизонтального масштабування через шардінг дозволяє MongoDB обробляти великі обсяги даних.
- Висока продуктивність для певних типів операцій. MongoDB особливо швидко працює з операціями запису та обробки нереляційних даних.
- Потужна система агрегацій. Інструменти агрегації даних дозволяють MongoDB використовувати як аналітичну базу даних.

Але MongoDB має і ряд обмежень такі як те, що вона може використовувати більше пам’яті, ніж реляційні бази даних, через свій підхід до зберігання даних (BSON) і відсутність нормалізації та підтримка ACID обмежена**.** В таблиці 2.1 представлено порівняння MongoDB з реляційними базами даних.

Таблиця 2.1 – Порівняння MongoDB з реляційними базами даних

Критерій	MongoDB	Реляційні бази даних
Модель даних	Документно-орієнтована (NoSQL)	Реляційна (таблиці)
Масштабування	Горизонтальне через шардінг	Переважно вертикальне
Схема	Схема не є обов'язковою	Чітка, заздалегідь визначена
Типи даних	JSON/BSON документи	Таблична структура
Продуктивність	Швидка для операцій запису, нереляційних запитів	Ефективна для транзакцій
Підтримка ACID	Підтримується, але з обмеженнями	Підтримка повного ACID

2.2 Аналіз методів оцінки ефективності SQL та NoSQL баз даних

Основними підходами до оцінки баз даних є порівняльний аналіз, аналіз продуктивності та розрахунок загальної вартості володіння. Для оцінки ефективності SQL та NoSQL баз даних у контексті конкретного проєкту

запропоновано розробити набір метрик, які дозволяють зрозуміти, яка система бази даних найкраще відповідає вимогам проєкту.

Продуктивність системи визначає, наскільки ефективно вона здатна виконувати операції в умовах навантаження. Вона охоплює різні аспекти, включаючи час виконання запитів, пропускну здатність та затримку [22], що є ключовими показниками для вибору оптимальної архітектури бази даних:

1. Час виконання запитів. Це час, необхідний для виконання запиту (наприклад, вибірка, вставка чи оновлення даних). У SQL базах даних запити можуть бути складними, включаючи об'єднання таблиць (JOIN), що може вплинути на час виконання. NoSQL, в свою чергу, часто підтримує простіші запити, але може бути швидким за рахунок специфічної архітектури.

2. Пропускна здатність. Це кількість операцій, яку система здатна виконати за певний час. Наприклад, скільки запитів може обробити база даних на секунду. Це критично важливо для проєктів з великими навантаженнями.

3. Затримка. Це час, що проходить від надходження запиту до отримання результату. Важлива для додатків у реальному часі (наприклад, чат-аплікації чи системи для фінансових транзакцій), де час відповіді має бути мінімальним.

Масштабованість є ключовим фактором для забезпечення ефективної роботи системи в умовах зростаючого навантаження, і вона може бути досягнута різними способами, в залежності від архітектури бази даних [23].

Горизонтальна масштабованість – це здатність системи обробляти зростаюче навантаження шляхом додавання нових серверів. NoSQL бази, як правило, краще масштабуються горизонтально, оскільки підтримують розподілене зберігання даних по кількох серверах, що дає можливість швидко розширювати систему без обмежень на потужність одного сервера.

Вертикальна масштабованість – це можливість збільшення потужності окремого сервера шляхом додавання більше ресурсів, таких як оперативна пам'ять або процесори. SQL бази даних можуть масштабуватися вертикально, але існують обмеження на потужності одного сервера, що може стати бар'єром при великому навантаженні.

Автоматичне масштабування – це здатність системи самостійно адаптуватися до змінюваного навантаження, наприклад, збільшення кількості запитів, автоматично додаючи нові ресурси або перерозподіляючи дані між серверами, забезпечуючи безперебійну роботу навіть при змінних умовах.

Надійність і доступність системи є критично важливими аспектами для забезпечення її безперебійної роботи та здатності справлятися з можливими збоїв чи несправностей [24]. Відмовостійкість це здатність системи продовжувати працювати навіть при відмові одного або кількох її компонентів. NoSQL бази часто мають високу відмовостійкість завдяки своїй розподіленій природі. Здатність системи швидко відновлювати роботу після серйозного збою. SQL і NoSQL бази можуть мати різні механізми для збереження даних та відновлення після збоїв. Доступність важлива для систем, які мають бути доступні 24/7, як-от банківські додатки або інтернет-магазини. Висока доступність означає, що база даних повинна бути доступною навіть під високим навантаженням або після збоїв.

Гнучкість і сумісність є важливими факторами, які визначають, наскільки ефективно система може працювати з різними типами даних і інтегруватися з іншими технологіями [25]. SQL бази використовують таблиці для зберігання даних, що може бути обмеженням для складних, неструктурованих або змінних даних. NoSQL підтримує різні моделі даних: документи, графи, ключ-значення, що дозволяє краще обробляти різноманітні типи даних (наприклад, JSON, графи користувачів). SQL бази мають сувору схему для таблиць, що робить їх більш обмеженими для швидких змін структури даних. NoSQL бази дозволяють більш гнучко змінювати схему або навіть використовувати схеми без певної структури.

SQL бази зазвичай мають добре розвинену підтримку інтеграції з іншими технологіями та стандартами ODBC, JDBC [1]. ODBC (Open Database Connectivity) це стандарт що дозволяє програмам підключатися до різних баз даних через єдиний інтерфейс. Він використовує драйвери для кожної бази даних, які виконують специфічні функції підключення та обробки запитів. Це дозволяє використовувати один код для роботи з різними базами даних (Oracle, SQL Server тощо). ODBC підтримується на різних платформах і в багатьох мовах програмування, що робить

його універсальним. JDBC (Java Database Connectivity) це програмний інтерфейс для роботи з базами даних в середовищі Java. Він дозволяє Java-додаткам взаємодіяти з різними реляційними базами даних за допомогою SQL-запитів. JDBC забезпечує більшу інтеграцію та ефективність для Java-додатків, оскільки це спеціалізоване рішення для роботи з базами даних через Java. Як і ODBC, JDBC також використовує драйвери для підключення до конкретних баз даних, але цей стандарт специфічний для Java.

NoSQL системи можуть мати різні інтерфейси, і сумісність з іншими системами може залежати від конкретної бази [24]. Такі системи, такі як MongoDB, Cassandra, Redis, чи DynamoDB, використовують власні інтерфейси для взаємодії з іншими системами. Ці інтерфейси (API) залежать від конкретної бази і можуть суттєво відрізнятися за функціональністю та способом інтеграції.

Управління та адміністрування баз даних охоплюють аспекти, пов’язані з налаштуванням, моніторингом і підтримкою систем, що є ключовими для забезпечення їх ефективної роботи [26].

1. Легкість налаштування та конфігурації. SQL бази зазвичай потребують більш складного налаштування, особливо при високих навантаженнях або віртуалізації проте NoSQL бази можуть бути простішими в налаштуванні.

2. Інструменти для моніторингу та адміністрування. Для SQL баз даних існує багато інструментів для моніторингу і аналізу продуктивності. NoSQL системи можуть мати обмежену кількість інструментів, хоча для популярних NoSQL систем (наприклад, MongoDB або Cassandra) також існують потужні інтерфейси для адміністрування.

3. Оновлення та підтримка: Оновлення SQL баз даних можуть потребувати великих зусиль для забезпечення сумісності з даними. Для NoSQL баз це може бути простіше, хоча іноді можуть виникати проблеми з сумісністю нових версій.

Безпека є критично важливим аспектом для баз даних, оскільки вона забезпечує захист даних від несанкціонованого доступу, втрати або компрометації [25]. Реляційні SQL бази часто мають більш зрозумілі механізми доступу (наприклад, ролі і привілеї на рівні таблиць і колонок). У NoSQL системах може

бути більш обмежений контроль доступу, хоча в останніх версіях можуть бути додані складні механізми. Обидва типи баз даних можуть підтримувати шифрування даних на диску та під час передачі. Однак, рівень підтримки шифрування може варіюватися в залежності від конкретної системи. SQL бази забезпечують високий рівень безпеки через добре продуману модель прав доступу. NoSQL може не мати таких механізмів, що потребує додаткових засобів для захисту даних.

Вартість є важливим фактором при виборі між SQL і NoSQL базами даних, оскільки вона визначає загальні витрати на впровадження, підтримку і масштабування системи. [26]. Вартість ліцензії та підтримки: SQL бази, особливо комерційні (наприклад, Oracle, Microsoft SQL Server), можуть бути дорогими через ліцензійні витрати. NoSQL рішення часто є безкоштовними з відкритим кодом, що може суттєво знизити витрати на ліцензії. Витрати на підтримку інфраструктури, адміністрування і масштабування бази даних можуть бути різними для SQL та NoSQL, залежно від типу та складності проєкту. Економія при масштабуванні NoSQL баз даних, оскільки вони дозволяють додавати нові сервери без великих витрат, на відміну від SQL, які зазвичай потребують більш потужних серверів.

Користувацький досвід охоплює аспекти, пов'язані з навчанням, впровадженням і інтеграцією баз даних у реальні додатки, що є важливими для ефективного використання систем. Час на навчання та впровадження SQL бази можуть вимагати більше часу на навчання та налаштування через їх складнішу модель даних. NoSQL часто більш доступні для новачків завдяки простішій структурі. Простота інтеграції з додатками: SQL бази часто мають зручні і стандартні інтерфейси для інтеграції з іншими додатками через використання загальних стандартів.

Підтримка транзакцій є важливим аспектом при виборі бази даних, оскільки вона визначає, наскільки система здатна обробляти операції з високим рівнем надійності та консистентності. ACID властивості: SQL бази підтримують ACID (Atomicity Consistency Isolation Durability) транзакції, що гарантує високий рівень консистентності та атомарності операцій. Це означає, що кожна транзакція

виконується повністю або не виконується взагалі, забезпечуючи цілісність даних навіть у разі помилок або відмови системи. Це робить SQL бази ідеальними для застосувань, де важлива строгість і точність обробки даних (наприклад, банківські операції). Eventual consistency: NoSQL бази можуть жертвувати повною консистентністю даних в обмін на доступність і масштабованість. Вони часто використовують принцип eventual consistency, що означає, що дані можуть бути тимчасово неузгодженими між репліками, але зрештою досягають стабільного стану. Це підходить для додатків, де доступність і швидкість виконання запитів важливіші за повну консистентність (наприклад, соціальні мережі або великі системи для зберігання даних, що працюють з великими обсягами).

Для оцінки якості баз даних використовують ієрархічну модель, де кожен вищий рівень містить показники нижчих рівнів, з можливістю розширення за рахунок додаткових критеріїв. На першому рівні визначаються основні фактори якості: продуктивність, масштабованість, надійність, гнучкість. Кожному фактору відповідають конкретні критерії якості (2-й рівень), які оцінюються через метрики (3-й рівень). Метрики складаються з оціночних елементів (4-й рівень), що визначають властивості баз даних відповідно до потреб проєкту [27, 28].

Методи оцінки якості баз даних дозволяють кількісно визначити переваги та недоліки різних систем, спираючись на набір критеріїв. Розглянемо три основні методи оцінки: метод сум, метод відстаней та метод геометричних середніх.

Цей метод полягає у простому додаванні всіх значень показників якості. Метод дозволяє отримати загальну оцінку, що показує, наскільки система відповідає очікуваним критеріям. Однак цей підхід не враховує вагу окремих показників, що може бути критичним у практичних застосуваннях [29, 30]. Формула для розрахунку:

$$S_i = \sum_j \frac{Q_{ij}}{Q_j^{\text{base}}} \quad , \tag{2.1}$$

де Q_{ij} — фактичне значення j-го показника для бази даних;

Q_j^{base} — базисне значення.

Метод відстаней базується на розрахунку близькості до еталонної бази даних за допомогою евклідової відстані. Метод дозволяє зрозуміти, наскільки обрана база даних відхиляється від ідеального варіанту. Він надає більш об'єктивну оцінку, ніж метод сум, адже враховує різницю між фактичними та максимальними значеннями показників [31, 32]. Формула обчислення:

$$D_i = \sqrt{\sum_j (Q_j^{max} - Q_{ij})^2} \quad , \tag{2.2}$$

де Q_j^{max} – максимальне значення показника.

Метод геометричних середніх застосовується, коли всі показники нормалізовані в діапазоні від 0 до 1 Метод геометричних середніх корисний для зваженого обчислення загальної оцінки, особливо коли важливо зберегти пропорційність між показниками. Однак він вимагає попередньої нормалізації даних [33]. Формула розрахунку:

$$G_i = \left(\prod_j Q_{ij} \right)^{\frac{1}{n}} \quad , \tag{2.3}$$

де n — кількість показників.

Слід зазначити що методи мають обмеження. Прості методи, як-метод сум чи відстаней, припускають рівноправність усіх показників і порівняння з універсальним еталоном. Однак для баз даних різні показники мають неоднакову вагу залежно від контексту (наприклад, для транзакційних систем критична консистентність, а для великих даних — масштабованість).

Для порівняння баз даних вводиться поняття критеріїв оптимальності. Кожному показнику присвоюється вага, яка відображає його важливість у контексті конкретного проєкту [34]. Відносні значення розраховуються за формулою:

$$K_{ij} = \frac{Q_{ij}}{Q_j^{max}}, \tag{2.4}$$

де K_{ij} — критерій впливу *j*-го показника для бази даних *i*.

Запропонована методика дозволяє ефективно оцінювати доцільність використання баз даних SQL або NoSQL залежно від особливостей проєкту. Вона є адаптивною та дозволяє враховувати специфічні вимоги, як-от продуктивність, масштабованість чи гнучкість, що робить її універсальним інструментом для вибору баз даних.

2.3 Метод багатокритеріального аналізу

Метод багатокритеріального аналізу є потужним інструментом для вибору архітектури бази даних в залежності від вимог проекту. Оскільки в процесі проектування бази даних необхідно врахувати багато різних факторів, таких як продуктивність, масштабованість, надійність, безпека та вартість, то даний метод дозволить здійснити оцінку різних варіантів і допоможе з вибором оптимального рішення.

Метод складається з чотирьох основних етапів, які забезпечують структурований підхід до вибору найбільш підходящої системи баз даних залежно від вимог проекту [35].

На першому етапі визначаються ключові показники (критерії), за якими будуть оцінюватися бази даних:

- продуктивність (швидкість обробки запитів);
- масштабованість (здатність обробляти зростаючі обсяги даних);
- гнучкість (легкість адаптації до змін);
- надійність (відсутність збоїв);
- консистентність даних.

Для кожного з показників визначається вага, яка вказує на його важливість у контексті конкретного проекту. Наприклад, для систем, що потребують високу продуктивність, цей критерій може мати вагу 0.4, тоді як масштабованість — 0.2.

На наступному кроці проводиться визначення еталонних (максимальних) значень для кожного показника. Це може бути зроблено шляхом:

- використання загальноприйнятих стандартів;
- встановлення цільових показників, які бажані для проекту;
- використання показників найбільш якісної бази даних у певній категорії.

Цей етап дозволяє створити еталонну базу даних, з якою будуть порівнюватися реальні системи [36].

Критерії впливу для кожної бази даних розраховуються на основі фактичних значень показників та їх ваг. Щоб критерії були порівнюваними, їхні значення потрібно нормалізувати до діапазону [0, 1]. Формула нормалізації для критеріїв, де більше значення краще [37]:

$$Q_{ij} = \frac{X_{ij} - X_{\min}}{X_{\max} - X_{\min}}, \tag{2.5}$$

де Q_{ij} – нормалізоване значення критерію *jj* для варіанту *ii*;

X_{ij}– фактичне значення;

X_{min} та X_{max}— мінімальне та максимальне значення критерію відповідно.

Формула розрахунку критерія:

$$W_{ij} = Q_{ij} \cdot \text{вага}_j, \tag{2.6}$$

де W_{ij} – критерій впливу *j*-го показника для бази даних *i*;

Q*ij* – фактичне значення *j*-го показника для бази даних *i*;

вага$_j$ – вага *j*-го показника.

У підсумку для кожної бази даних обчислюється підсумкова зважена оцінка критеріїв впливу:

$$S_i = \sum W_{ij}, \tag{2.7}$$

де S_i — загальна оцінка бази даних *i*.

Чим вища оцінка, тим більш підходящою є база даних для конкретного проекту.

На завершальному етапі результати всіх розрахунків порівнюються між собою. Найбільш оптимальною вважається база даних, яка отримала найвищу загальну оцінку S_i.

Запропонований метод є універсальним і може бути адаптований до різних типів баз даних, включаючи SQL (наприклад, MS SQL, PostgreSQL) та NoSQL (наприклад, MongoDB, Cassandra). Він дозволяє враховувати специфічні вимоги проекту [38-40], такі як:

- продуктивність для транзакційних систем (OLTP);
- масштабованість для обробки великих обсягів даних (Big Data);
- надійність і консистентність для систем, критично важливих для бізнесу.

Перевагами такого алгоритму є:

- адаптивність, можна змінювати ваги та критерії залежно від потреб проекту.
- універсальність, підходить як для SQL, так і для NoSQL баз даних.
- об’єктивність, базується на кількісних оцінках, що зменшує вплив суб’єктивних факторів.

Проте в даного алгоритму є і обмеження:

- методика вимагає наявності точних даних про показники баз даних.
- необхідно заздалегідь визначити ваги показників, що може бути складно без глибокого розуміння проекту.

Таким чином, даний метод є ефективним інструментом для вибору бази даних, орієнтованої на специфіку конкретного проекту. Він дозволяє оптимізувати вибір, знижуючи ризики та забезпечуючи максимальну відповідність вимогам.

2.4 Результати дослідження методу

Результати проведеного дослідження дозволили визначити сильні та слабкі сторони застосування SQL та NoSQL баз даних у різних умовах. Аналіз показав, що кожен підхід має свої переваги залежно від вимог проєкту та типу даних.

При оцінці продуктивності було виявлено, що SQL бази даних, такі як MS SQL, забезпечують високу швидкість виконання транзакцій, особливо для структурованих даних у стабільних середовищах. Однак, коли мова йде про роботу з великими обсягами або неструктурованими даними, NoSQL бази, наприклад MongoDB та Cassandra, показують значно кращі результати завдяки своїй архітектурі, орієнтованій на горизонтальне масштабування.

Питання масштабованості також стало важливим аспектом дослідження. NoSQL бази довели свою перевагу у горизонтальному масштабуванні завдяки шардінгу, що дозволяє легко розподіляти дані по серверам. Натомість SQL бази забезпечують ефективну вертикальну масштабованість, але їхня продуктивність зменшується із зростанням обсягів даних.

Гнучкість роботи з даними також була важливою метрикою. NoSQL бази виявилися більш адаптивними до змін у структурі даних, що робить їх ідеальним вибором для динамічних додатків. Водночас, SQL бази з їх жорсткою схемою даних краще підходять для додатків, де необхідно забезпечити строгий контроль за структурою.

Різні методи оцінки також дали цікаві результати. Метод суми значень був простим у реалізації, але не завжди враховував важливість окремих параметрів. Метод відстаней до еталону дозволив визначити оптимальні бази даних, але вимагав точного визначення еталонних значень. Метод геометричних середніх виявився найбільш збалансованим, особливо для метрик із суттєво різними значеннями.

Реальне тестування показало, що SQL бази є ідеальними для фінансових систем та додатків, де критично важлива консистентність даних. У свою чергу, NoSQL бази продемонстрували перевагу в обробці великих обсягів мультимедійних даних та роботи у хмарних середовищах.

У підсумку, дослідження підтвердило, що вибір між SQL та NoSQL базами даних залежить від специфіки проєкту. Наприклад, для платформ, що працюють із великими обсягами динамічних даних, таких як стрімінгові сервіси або соціальні мережі, NoSQL є оптимальним рішенням. Водночас для транзакційних систем,

таких як банківські додатки, краще підходять SQL бази. Ці результати дозволяють зробити обґрунтований вибір баз даних та забезпечити ефективність їх використання в умовах конкретних завдань.

Висновки до розділу 2

1. Проведено опис використаних даних та інструментів. Визначено набір тестових даних та інструментів для оцінювання баз даних, які відповідають реальним сценаріям роботи інформаційних систем. Зокрема, бази даних різного типу SQL: PostgreSQL, MS SQL та типу NoSQL: MongoDB, Cassandra, та інструменти для моделювання робочого навантаження, такі як YCSB і Sysbench.
2. Проведено порівняльний аналіз методів оцінки продуктивності оцінки ефективності SQL та NoSQL баз даних, зокрема, за такими критеріями, як час виконання запитів, масштабованість, консистентність і вартість. Виявлено, що SQL-бази даних демонструють кращі результати для структурованих даних із високими вимогами до консистентності, тоді як NoSQL забезпечують значно вищу масштабованість у розподілених системах.
3. Розроблено метод багатокритеріального аналізу для вибору оптимальної бази даних залежно від потреб проєкту. Метод базується на системі критеріїв (продуктивність, масштабованість, консистентність, тип даних, вартість) із застосуванням нормалізації даних і вагових коефіцієнтів для визначення інтегральної оцінки.
4. Протестовано метод на конкретних прикладах, що продемонструвало його ефективність у виборі бази даних для різних типів проєктів. Результати дослідження показали, що NoSQL-бази, такі як MongoDB, є оптимальними для великомасштабних розподілених систем, а SQL-бази, як MS SQL, найкраще підходять для транзакційних систем із високими вимогами до консистентності.

3 РЕАЛІЗАЦІЯ МЕТОДУ БАГАТОКРИТЕРІАЛЬНОГО АНАЛІЗУ ДЛЯ ВИБОРУ АРХІТЕКТУРИ БАЗИ ДАНИХ

3.1 Опис архітектури та можливостей методу

У цьому підрозділі детально описується методологія багатокритеріального аналізу, застосованого для вибору оптимальної архітектури бази даних, а також основні критерії оцінки, що враховувалися в процесі прийняття рішення

На основі отриманих результатів пропонується метод, який включає такі критерії вибору:

- продуктивність (оцінка швидкості виконання операцій);
- гнучкість (здатність бази даних адаптуватися до змін у структурі даних);
- масштабованість (можливість збільшення обсягу даних без втрати продуктивності).

Продуктивність бази даних тісно пов’язана з особливостями та вимогами конкретного проєкту. Умови, за яких працює система, включають розмір даних, характер запитів, кількість одночасних користувачів, частоту оновлення даних та інші аспекти. Врахування цих умов дозволяє об'єктивно оцінити базу даних і вибрати архітектуру, яка найкраще відповідає вимогам проєкту.

У невеликих базах продуктивність операцій (CRUD) зазвичай стабільна і висока, оскільки дані можуть легко кешуватися в пам’яті. SQL бази даних, такі як MS SQL або PostgreSQL, зазвичай демонструють перевагу завдяки оптимізації запитів і наявності індексів. Продуктивність може знижуватися через обмеження пам’яті або ресурсів сервера. У таких випадках NoSQL бази даних, як-от MongoDB чи Cassandra, часто показують кращі результати завдяки горизонтальному масштабуванню та децентралізованій архітектурі.

Для систем, які переважно виконують операції зчитування (аналітичні системи, звіти), SQL бази даних із сильними механізмами індексації та оптимізованими запитами підходять краще.

Якщо більшість запитів пов’язані з записом або оновленням (наприклад, логування чи обробка транзакцій у реальному часі), NoSQL бази, як-от DynamoDB

або Redis, забезпечують низькі затримки й високу продуктивність. Для складних транзакцій або запитів із великою кількістю зв’язків SQL бази забезпечують більшу функціональність через мову запитів SQL. Для простих операцій доступу до документів чи ключів NoSQL бази часто є ефективнішими.

У багатокористувацьких системах продуктивність залежить від здатності бази обробляти конкурентні запити:

- SQL бази використовують блокування рядків і транзакцій для забезпечення цілісності даних, що може знизити продуктивність при високих навантаженнях.

- NoSQL бази, як-от Cassandra, реалізують принципи BASE (Basic Availability, Soft State, Eventual Consistency), що дозволяє обробляти великий обсяг запитів одночасно.

У системах, де дані часто оновлюються (наприклад, соціальні мережі чи біржові платформи), NoSQL бази з асинхронною реплікацією можуть бути ефективними завдяки низьким затримкам.

У проєктах зі значними піковими навантаженнями (наприклад, під час розпродажів в e-commerce) продуктивність бази може залежати від механізмів кешування, реплікації та балансування навантаження. Redis або Memcached у поєднанні з основною базою даних допомагають мінімізувати вплив пікової активності.

Гнучкість бази даних є важливим критерієм, який визначає її здатність адаптуватися до змін у структурі даних та умов проєкту. Цей аспект особливо критичний для сучасних систем, де вимоги до даних можуть динамічно змінюватися через розвиток бізнесу, технологій або ринкових умов.

Зміна структури даних у SQL базах вимагає модифікації схеми (таблиць, зв’язків), що може бути складним і потребує часу, особливо якщо обсяг даних великий. Для нереляційніх (NoSQL) баз даних завдяки схемо-незалежній моделі NoSQL бази, як-от MongoDB або Couchbase, дозволяють зберігати дані з різними структурами в одній колекції. Це значно спрощує процес оновлення структури.

SQL бази даних зазвичай оптимізовані для структурованих даних, але можуть працювати із напівструктурованими даними через використання JSON, XML чи подібних форматів. NoSQL бази, як-от MongoDB або Elasticsearch, розроблені для роботи як зі структурованими, так і з неструктурованими чи напівструктурованими даними, що забезпечує вищу гнучкість.

Реляційні бази даних частіше вимагають змін у схемі при додаванні нових функцій або зміні вимог. Наприклад, для додавання нового поля необхідно модифікувати таблиці, що може спричинити тимчасове зниження продуктивності. Нереляційні бази дозволяють додавати нові поля до документів або змінювати структуру даних без зміни всієї бази, що особливо важливо для динамічних систем.

Одним із ключових аспектів вибору бази даних є її здатність обробляти динамічні запити, особливо в умовах змінних структур даних. Реляційні бази даних (SQL) пропонують потужні можливості для виконання складних запитів завдяки використанню стандартної мови SQL. Ця мова дозволяє створювати багаторівневі запити, виконувати обчислення, об’єднувати дані з кількох таблиць, а також аналізувати великі набори даних. Однак SQL бази вимагають чітко визначеної структури даних, тому будь-які зміни в структурі можуть ускладнити створення запитів або вимагати їхнього переписування.

Натомість нереляційні бази даних (NoSQL) забезпечують більшу гнучкість у роботі з динамічними структурами даних. Вони дозволяють виконувати запити до даних, які не відповідають фіксованій схемі, що є перевагою для проєктів, де дані постійно змінюються. Наприклад, у MongoDB можна зберігати документи з різними наборами полів у межах однієї колекції, що значно полегшує адаптацію бази до змін у вимогах.

Таким чином, SQL бази є оптимальним вибором для систем зі стабільною структурою даних і необхідністю виконання складних запитів, тоді як NoSQL бази краще підходять для гнучких і динамічних сценаріїв.

Якщо важлива стабільна структура SQL бази даних краще підходять для систем, де структура даних рідко змінюється або має строгі правила, наприклад, у банківських або фінансових системах.

Якщо потрібна динаміка NoSQL бази даних є ідеальним вибором для проєктів, що розвиваються швидко, або для роботи з неструктурованими даними, наприклад, у соціальних мережах, IoT-проєктах чи аналітиці.

3.2 Реалізація аналізу відповідно до вибраних критеріїв методу

Основні кроки для створення методу:

1. Визначення метрик для кожного критерію.

2. Візуалізація результатів.

3. Оцінка і рекомендації.

Для обчислення продуктивності використовуємо:

- час виконання запитів (зчитування/запис). Середній час, необхідний для виконання різних типів запитів (записи, зчитування, оновлення, видалення).
- пропускна здатність (Throughput). Кількість запитів, які можна виконати за одиницю часу.
- затримка (Latency). Час затримки на обробку запитів (особливо важливо для NoSQL баз, що підтримують низьку затримку при великому навантаженні).

Визначається середній час, необхідний для виконання різних CRUD-операцій (вставка, зчитування, оновлення, видалення). Цей показник дозволяє оцінити швидкість бази даних у виконанні окремих запитів**.**

У багатокористувацьких системах продуктивність залежить від здатності бази обробляти конкурентні запити:

- швидке зчитування важливе для систем звітності чи аналітики;
- швидкий запис важливий для систем, які обробляють великі обсяги даних у реальному часі, наприклад, логування чи IoT.

Пропускна здатність визначає кількість запитів, які можна виконати за одиницю часу (зазвичай за секунду). Висока пропускна здатність особливо важлива для систем із великим числом одночасних користувачів або запитів:

- для високонавантажених веб-додатків важливо, щоб база обробляла тисячі запитів на секунду без зниження продуктивності;
- NoSQL бази, такі як Cassandra або Redis, часто демонструють вищу пропускну здатність завдяки своїй архітектурі.

Вимірюється час затримки на обробку одного запиту від моменту надсилання до отримання результату. Ця метрика особливо критична для реальних додатків, що потребують мінімальних затримок, наприклад, систем торгівлі, ігор чи обробки даних у реальному часі:

- реляційні бази можуть демонструвати більшу затримку через обробку складних транзакцій;
- нереляційні бази (наприклад, DynamoDB) оптимізовані для зниження затримки навіть за умови великого навантаження.

Поєднання цих показників дозволяє об'єктивно оцінити продуктивність бази даних у конкретному проєкті:

- якщо головною метою є обробка великого обсягу запитів із мінімальною затримкою, варто обирати NoSQL рішення;
- для складних аналітичних запитів і забезпечення транзакційної цілісності перевагу слід віддавати SQL базам.

Оцінка цих показників дає змогу вибрати базу даних, яка найкраще відповідає технічним і бізнес-вимогам проєкту.

3.2.1 Аналіз продуктивності

Для аналізу продуктивності використовувалося обладнання з процесором AMD Ryzen 7 6850, 32 ГБ оперативної пам'яті, твердотільним диском об'ємом 1 ТБ і операційною системою Linux Ubuntu 24.04 LTS. Це забезпечило ефективне виконання тестів продуктивності SQL та NoSQL баз даних, зокрема часу виконання запитів, пропускної здатності та затримки.

Операція читання (SELECT) MS SQL має кращу продуктивність на 6 мс порівняно з MongoDB. Це може свідчити про те, що MS SQL краще оптимізований для читання даних. MongoDB показує кращу швидкість виконання вставок

(INSERT), випереджаючи MS SQL на 10 мс. Це може бути результатом більш ефективної архітектури документо орієнтованих баз даних для великих обсягів даних. Відмінності оновлення (UPDATE) між MS SQL і MongoDB незначні, з MS SQL, що випереджає MongoDB на 10 мс. Це вказує на те, що MongoDB не має значних проблем з оновленням даних, хоча є потенціал для подальшої оптимізації. Проте операція видалення (DELETE) MS SQL також випереджає MongoDB на 5 мс у випадку видалення, що може бути пов'язано з оптимізацією індексів та механізмів управління даними в MS SQL.

Загалом, MS SQL показує кращу продуктивність при більшості операцій, однак MongoDB виявляється ефективнішим при вставках. Таблиця 3.1 відображає продуктивність двох баз даних — MS SQL та MongoDB.

Таблиця 3.1 – Порівняння продуктивності MS SQL та MongoDB

Операція	MS SQL	MongoDB	Різниця
SELECT	2	8	MS SQL швидше на 6 мс
INSERT	2	1	MongoDB швидше на 10 мс
UPDATE	10	20	MongoDB швидше на 10 мс
DELETE	2	7	MS SQL швидше на 5 мс

Для аналізу продуктивності вимірювався час відгуку системи на запит — час, що проходить між початком запиту та отриманням відповіді. Порівнювались два типи показників: середній час відгуку для виконаної операції та деталізований аналіз. Такий підхід забезпечує комплексне уявлення про продуктивність та доцільність вибору тієї чи іншої бази даних залежно від потреб конкретного проєкту.

Код програми, що використовувався для аналізу, наведено у додатку А. Результати роботи даного коду наступні:

```
python3 graph2.py
MS SQL
Duration Insert: 0.91 seconds
Duration Update: 20.19 seconds
Duration Select: 0.80 seconds
```

```
Duration Delete: 1.40 seconds

MongoDb
Duration Insert: 0.20 seconds
Duration Update: 8.43 seconds
Duration Select: 1.66 seconds
Duration Delete: 1.23 seconds
```

Для зручності використання візуалізуються результати у вигляді діаграми на рисунку 3.1, яка показує порівняння продуктивності баз даних MS SQL та MongoDB за різними сценаріями.

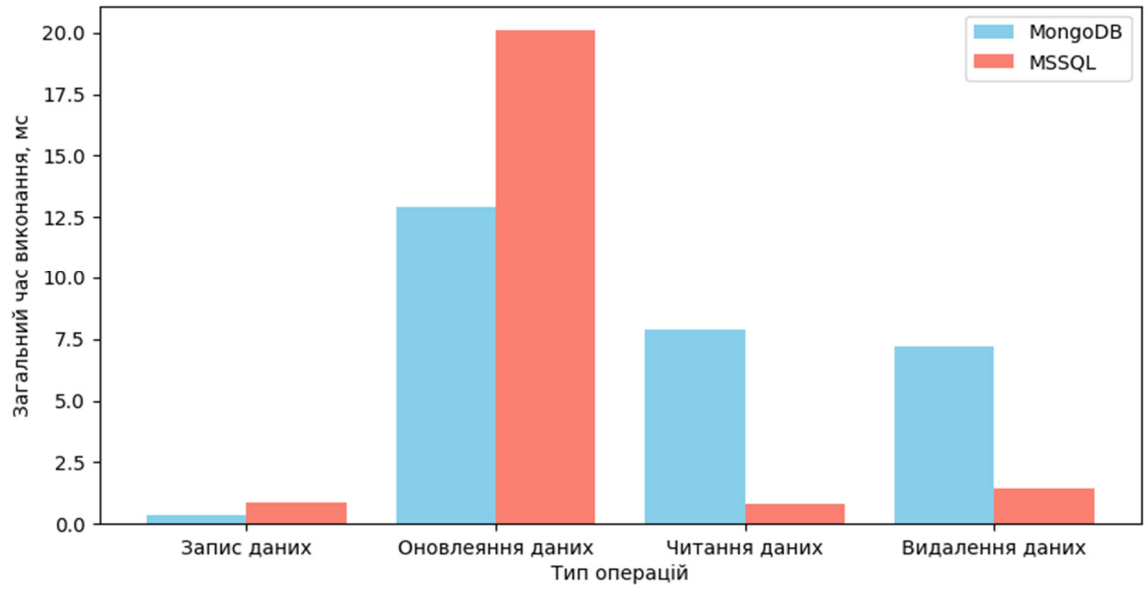


Рисунок 3.1 – Аналіз часу виконання операцій MS SQL та MongoDB

Затримка (latency) є важливим показником продуктивності бази даних, що відображає час, необхідний для виконання операцій читання та запису. За результатами тестування, у MS SQL затримка при операціях читання становить 1 мс, а при записі — 2 мс. Для MongoDB ці показники є дещо кращими: затримка при читанні — 5 мс, при записі — 1 мс.

MongoDB має меншу затримку, як при читанні, так і при записі, що робить її більш ефективною для систем з високим навантаженням. У таких умовах швидка обробка запитів є критично важливою, оскільки навіть незначне зниження затримки може суттєво підвищити загальну продуктивність і знизити час відповіді системи. Це може бути особливо важливим для додатків, де велике значення має

швидка реакція на запити, наприклад, у реальному часі або при роботі з великими обсягами даних.

3.2.2 Аналіз пропускної здатності

Пропускна здатність баз даних показує, скільки операцій або запитів система може обробити за одиницю часу. У порівнянні з реляційною базою даних MS SQL, яка демонструє пропускну здатність у 15 000 транзакцій на секунду за результатами тесту HammerDB TPCC Benchmark, MongoDB показує результат 12 000 запитів на секунду за тестом YCSB.

MS SQL має вищу пропускну здатність, оскільки оптимізована для обробки великих транзакцій у реляційних базах даних, що забезпечує ефективне управління даними з високими вимогами до цілісності та консистентності.

У той же час, MongoDB має трохи меншу пропускну здатність, але її головною перевагою є можливість горизонтального масштабування. Це означає, що за допомогою додавання нових серверів до кластеру, MongoDB здатна значно збільшити пропускну здатність, що може бути критично важливим у реальних умовах, де обсяги даних і навантаження на систему можуть суттєво зростати.

Порівняльний аналіз пропускної здатності MS SQL та MongoDB здійснювався шляхом виконання 10 000 запитів для чотирьох основних операцій: вставки, оновлення, читання та видалення. Код програми, що використовувався для цього аналізу, наведено у додатку Б. Результати виконання даного навантажувального тестування наступні:

```
python3 loadTesting.py
MS SQL
Duration Insert: 16.44 seconds
Duration Update: 8.72 seconds
Duration Select: 1.66 seconds
Duration Delete: 17.94 seconds

MongoDb
Duration Insert: 1.53 seconds
Duration Update: 1.59 seconds
Duration Select: 1.58 seconds
Duration Delete: 1.41 seconds
```

Для зручності використання візуалізуються результати даного навантажувального тестування у вигляді діаграми на рисунку 3.2, яка аналіз пропускної здатності MS SQL та MongoDB для основних операцій.

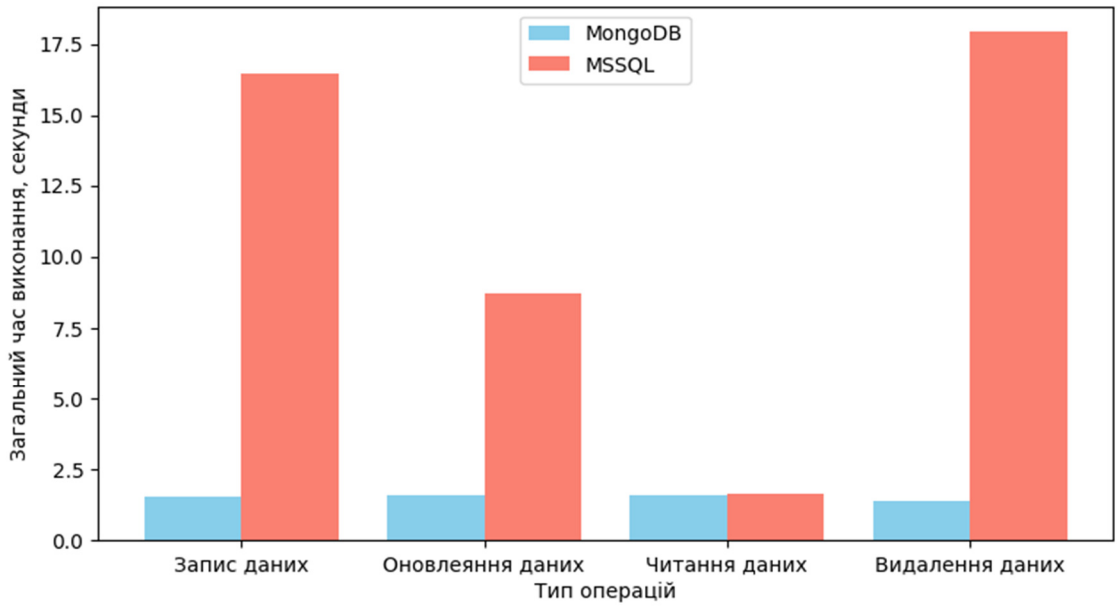


Рисунок 3.2 – Результати аналізу пропускної здатності MS SQL та MongoDB для основних операцій

Результати аналізу показали наступне:

1. Вставка даних MongoDB впоралася із завданням за 1,5 секунди, тоді як MS SQL знадобилося 16 секунд. Це свідчить про значно вищу швидкість MongoDB у виконанні операцій вставки.

2. Оновлення даних в MongoDB також зайняла 1,5 секунди, тоді як MS SQL потребувала 8 секунд. Це демонструє перевагу MongoDB в обробці змін у великих обсягах даних.

3. Читання даних результати виявилися майже однаковими: MongoDB виконала її за 1,6 секунди, а MS SQL — за 1,7 секунди. Це вказує на схожу ефективність обох баз даних у цій категорії.

4. Видалення даних MongoDB виконала видалення за 1,4 секунди, тоді як MS SQL витратила на цю операцію 18 секунд. MongoDB продемонструвала значно кращу продуктивність у цьому аспекті.

Загальний аналіз показує, що MongoDB має значну перевагу в швидкості виконання операцій вставки, оновлення та видалення даних, тоді як у читанні продуктивність обох баз даних є майже однаковою. Це робить MongoDB більш підходящою для застосувань, де важлива швидка обробка великих обсягів даних.

3.2.3 Аналіз гнучкості

Гнучкість обчислюється на основі:

- часу на зміну структури даних. Час, необхідний для зміни структури бази даних (наприклад, додавання нових полів, зміна типів даних);
- підтримки різних типів даних. Можливість бази даних підтримувати різні типи даних (структуровані, напів структуровані, неструктуровані);
- наявності підтримки для несхемних даних. Здатність бази даних адаптуватися до змін у структурі даних (особливо актуально для NoSQL).

Час на зміну структури даних є важливим аспектом гнучкості бази даних, оскільки необхідність у частих змінах структури може впливати на доступність системи. У випадку з MS SQL, процес зміни структури (наприклад, додавання нових колонок або зміна типів даних) може займати до 15 хвилин. Для цього часто потрібно виконувати міграції та створювати нові індекси, що може призвести до тимчасового простою бази даних, оскільки зміни можуть потребувати блокування таблиць і впливати на доступність системи.

Натомість, у MongoDB, час на зміну структури (наприклад, додавання нових полів до документів) значно менший — лише 5 хвилин. MongoDB, як документно-орієнтована база даних, дозволяє змінювати структуру даних без необхідності виконувати важкі міграції або переробляти існуючі індекси. Це дозволяє значно зменшити час простою і зробити процес більш гнучким, що особливо корисно в умовах, де структура даних може часто змінюватися в залежності від розвитку бізнес-логіки.

3.2.4 Аналіз масштабованості

Масштабованість визначається як:

- час реакції при збільшенні обсягу даних, тобто зміни в часі виконання запитів при збільшенні обсягу даних;
- горизонтальне масштабування – можливість додавати нові сервери для обробки додаткових обсягів даних;
- вертикальне масштабування – можливість збільшувати обсяги оброблюваних даних на одному сервері без значного падіння продуктивності.

При збільшенні обсягу даних від 100 ГБ до 1 ТБ, MS SQL демонструє збільшення часу виконання запитів на 20%. Це означає, що з ростом обсягу даних продуктивність знижується, що може бути наслідком обмежень архітектури реляційної бази даних при роботі з великими обсягами даних.

MongoDB, у свою чергу, має менший приріст часу виконання запитів — лише на 10%. Це свідчить про кращу продуктивність при зростанні обсягу даних, що можна пояснити її здатністю до горизонтального масштабування. Завдяки розподіленій архітектурі, MongoDB може ефективно справлятися з великими обсягами даних, додаючи нові сервери для обробки навантаження.

MS SQL підтримує горизонтальне масштабування, але його реалізація потребує складних налаштувань, включаючи налаштування кластерів та балансування навантаження. Це може вимагати значних додаткових ресурсів і часу на налаштування, що ускладнює процес масштабування, особливо в умовах швидко зростаючих даних.

MongoDB, в свою чергу, має вбудовану підтримку горизонтального масштабування рисунок 3.3, що дозволяє з легкістю додавати нові сервери для розширення обсягу даних.

Це значно спрощує процес масштабування і дозволяє ефективно реагувати на збільшення навантаження або обсягів даних без складних налаштувань або додаткових ресурсів.

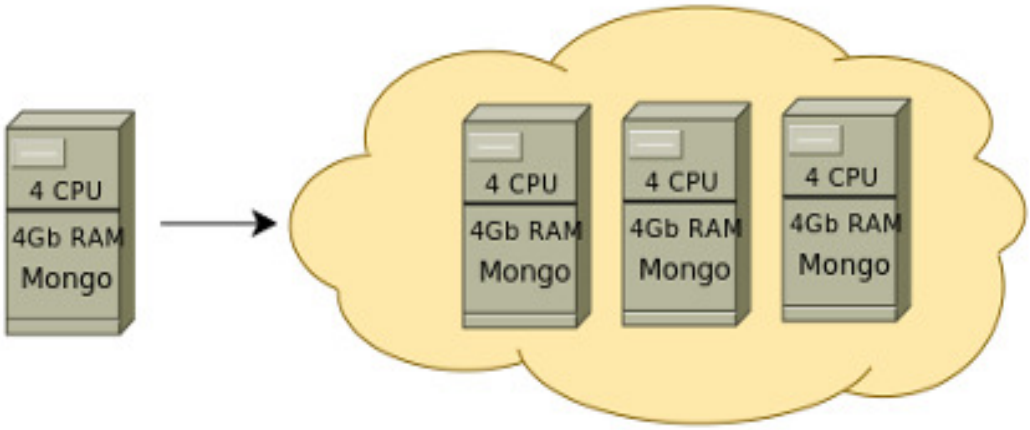


Рисунок 3.3 – Приклад горизонтального масштабування бази даних

Вертикальне масштабування в MS SQL можливе шляхом додавання ресурсів до одного сервера рисунок 3.4, але після певного порогу це не дає значного збільшення продуктивності. Це обмеження часто виникає через апаратні ресурси, які не завжди можуть забезпечити потрібну потужність для обробки великих обсягів даних або високих навантажень.

MongoDB також підтримує вертикальне масштабування, однак для досягнення найкращої продуктивності при великих навантаженнях вона зазвичай рекомендує використовувати горизонтальне масштабування. Горизонтальне масштабування дозволяє значно покращити ефективність і зберегти високу продуктивність навіть при значних обсягах даних.

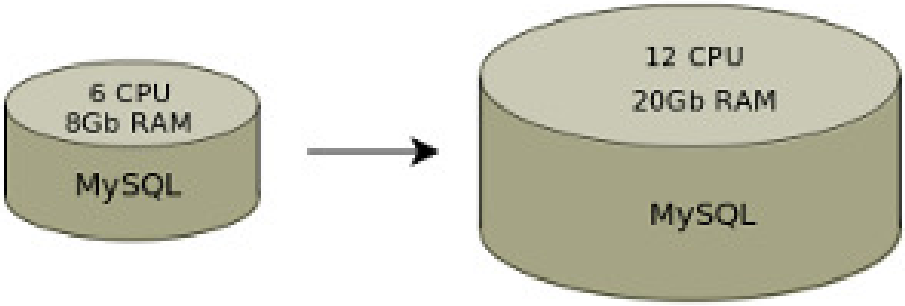


Рисунок 3.4 – Приклад вертикального масштабування бази даних

MongoDB має значні переваги у плані масштабованості. Вона забезпечує кращу продуктивність при зростанні обсягу даних завдяки горизонтальному

масштабуванню, що реалізоване "з коробки" та не вимагає складних налаштувань. MS SQL також може масштабуватися горизонтально, але цей процес значно складніший і потребує більше ресурсів. При вертикальному масштабуванні MS SQL стикається з обмеженнями, тоді як MongoDB, завдяки своїй розподіленій архітектурі, краще справляється з великими навантаженнями.

3.3 Аналіз та оцінка результатів реалізації

Використовуючи проведений аналіз продуктивності, пропускної здатності, гнучкості та масштабованості можна здійснити рекомендацію по вибору бази даних базуючись на оцінках показників відповідних проєктів. Для прикладу використано проєкти бухгалтерії, відеостудії, онлайн магазину, аналітичної платформи, соціальної мережі, представлені в таблиці 3.2. Для кожного проєкту надається нормалізована оцінка від 0 до 1 за кожним із критеріїв (формула 2.5).

Таблиця 3.2 – Нормалізовані оцінки критеріїв відносно заданих проєктів

Критерій	Бухгалтерія	Відеостудія	Онлайн-магазин	Аналітична платформа	Соціальна мережа
Кількість користувачів	0.5	0.1	0.85	0.25	0.95
Складність запитів	0.8	0.3	0.4	0.98	0.2
Обсяг даних	0.0	0.5	0.75	0.5	0.95
Тип даних	0.5	0.9	0.1	0.1	0.95
Масштабованість	0.0	0.5	0.95	0.25	0.95
Консистентність	0.95	0.0	0.6	1.0	0.4
Вартість	0.5	0.2	0.0	0.8	0.2

В таблиці 3.3 представлено ваги критеріїв в залежності від проєкту. В даному випадку ваги критеріїв були обчислені на основі експертного методу. Експерти дали оцінки важливості критеріїв у балах (наприклад, від 1 до 10) які потім були нормалізовані, щоб сума всіх ваг дорівнювала 1.

Таблиця 3.3 – Ваги критеріїв

Критерій	Вага	Степінь важливості	Рекомендації відносно заданих проектів
Кількість користувачів	0.2	Важливий критерій для проєктів із високою взаємодією користувачів	соціальні мережі та онлайн-магазини
Складність запитів	0.2	Визначає продуктивність і відповідність бази даних складним завданням	аналітична платформа, бухгалтерія
Обсяг даних	0.15	Ключовий фактор для платформ, які працюють із великими наборами даних (	аналітика, соціальна мережа).
Тип даних	0.15	Враховується для проєктів із специфічними вимогами до типу даних, таких як мультимедійні файли	Відеостудія, соціальна мережа
Масштабованість	0.25	Найвища вага, ключова вимога для багатьох сучасних проєктів	соціальна мережа, онлайн-магазини
Консистентність	0.15	Важливий для проєктів із високими вимогами до точності даних	бухгалтерія, аналітична платформа
Вартість	0.1	Менш важливий для проєктів із високими бюджетами, але має значення для невеликих проєктів	відеостудія, бухгалтерія

Підсумкова зважена оцінка критеріїв з рекомендацією проаналізованих баз даних MongoDB та MS SQL впливу обчислюється за формулою 2.7 представлена в таблиці 3.4.

Результати реалізації запропонованої методики показали її ефективність у вирішенні завдань вибору баз даних для різних типів проєктів. У ході дослідження було проаналізовано, наскільки отримані результати відповідають очікуванням та як методика впливає на процес прийняття рішень.

Перш за все, методика продемонструвала свою здатність забезпечувати коректний вибір баз даних залежно від специфіки завдань. Наприклад, для фінансових додатків із високими вимогами до консистентності та структурованості даних оптимальним вибором стали SQL бази, такі як MS SQL. У той же час, для динамічних веб-додатків із великими обсягами неструктурованих даних найкраще

підійшли NoSQL бази, зокрема MongoDB. Ці результати підтверджують відповідність запропонованого підходу реальним вимогам.

Таблиця 3.4 – Зважена оцінка критеріїв

Проєкт	Зважена оцінка	Пояснення	Вибір між MongoDB та MS SQL	Найважливіші критерії
Соціальна мережа	0.822	Висока оцінка вказує на її високі вимоги до кількості користувачів, масштабованості та обсягу даних	MongoDB	Висока масштабованість, підтримка неструктурованих даних
Онлайн-магазин	0.698	потреби у масштабованості та складності запитів	MongoDB / MS SQL	Залежно від пріоритету: гнучкість (MongoDB) / транзакційність (MS SQL).
Аналітична платформа	0.635	Висока важливість у складності запитів та консистентності	MS SQL	Підтримка складних аналітичних запитів, висока консистентність.
Бухгалтерія	0.575	високу оцінку через консистентність і вартість, але інші критерії є менш важливими	MS SQL	Висока точність і консистентність, критична для бухгалтерських систем.
Відеостудія	0.360	займає останнє місце, оскільки її потреби в кількості користувачів і консистентності є низькими	MongoDB	Підтримка великих обсягів мультимедійних файлів.

Другою важливою особливістю методики стала її універсальність. Завдяки використанню ієрархічного підходу, що враховує вагові коефіцієнти для кожного

критерію, методика дозволила адаптувати процес оцінки до різних сценаріїв використання. Це було особливо важливо для проєктів, де різні показники якості, такі як продуктивність, масштабованість чи надійність, мають різну вагу залежно від контексту.

Ще один важливий аспект стосувався прозорості та зручності застосування методики. Інструменти для розрахунків, включаючи методи суми значень, відстаней до еталону та геометричних середніх, забезпечили чіткі та зрозумілі результати. Це дозволило розробникам легко інтерпретувати отримані дані та приймати обґрунтовані рішення.

Під час реалізації також було виявлено деякі обмеження. Наприклад, для проєктів із високою невизначеністю щодо вимог до баз даних може бути складно визначити еталонні значення для методів оцінки. У таких випадках додаткового уточнення потребують вагові коефіцієнти, що може вплинути на точність результатів.

Загалом, результати реалізації підтвердили практичну цінність розробленої методики. Вона не лише спрощує процес вибору баз даних, але й мінімізує ризики, пов’язані з неправильним вибором, що у свою чергу позитивно впливає на загальну ефективність і якість розроблених інформаційних систем.

На основі проведених тестів і отриманих результатів можна зробити кілька важливих висновків. По-перше, вибір між SQL та NoSQL залежить від пріоритетів. Якщо важлива гнучкість у роботі з даними, краще підходить NoSQL. Для складних транзакцій і забезпечення цілісності даних оптимальним варіантом буде SQL. Якщо ж головним є масштабованість і продуктивність, вибір варто базувати на конкретних сценаріях використання та результатах тестування.

По-друге, тести дають змогу прогнозувати, як база даних працюватиме з більшими обсягами інформації. Це дозволяє заздалегідь обрати систему, яка впорається із зростаючим навантаженням.

По-третє, порівняння різних баз даних допомагає зрозуміти, яка з них краще адаптується до змін, таких як збільшення кількості запитів або даних. Це спрощує вибір найбільш ефективної бази для довготривалого використання.

У результаті дослідження запропоновано метод багатокритеріального аналізу було проведено аналіз переваг і недоліків SQL та NoSQL баз даних, порівняно реалізацію сховища ключ-значення в базах даних NoSQL і SQL. Проведене тестування продуктивності дозволяє зробити висновок, що реляційні бази даних, такі як MS SQL, виконують прості операції запис та оновлення записів гірше, ніж досліджувана NoSQL MongoDB система. Однак операція читання, видалення записів у реляційних SQL базах даних виконується набагато краще.

Бази даних NoSQL, як правило, оптимізовані для зберігання ключів і значень, бази даних SQL – ні. Проте не всі бази даних NoSQL працюють краще, ніж база даних SQL. Навіть у базах даних NoSQL існує велика варіація продуктивності залежно від типу операції. Спостерігається слабка кореляція між продуктивністю та моделлю даних, яку використовує кожна база даних. Що стосується операцій підрахунку та угруповання записів, то MS SQL виконує ці операції швидше, оскільки при роботі з NoSQL базами основні обчислення часто виконуються на стороні користувача, що знижує ефективність.

Для операцій видалення MS SQL показують найкращий час виконання при середньому обсязі даних. Однак, для великих обсягів даних NoSQL системи демонструють кращу продуктивність.

Загалом, можна зробити висновок, що системи, орієнтовані на Big Data, значно ефективніше працюють з NoSQL базами. Водночас для типових рішень та систем з меншими обсягами даних реляційні бази, такі як MS SQL, можуть забезпечити високу продуктивність. Отже, вибір бази даних має ґрунтуватися на типі вирішуваних завдань, обсягах даних та вимогах до часу відгуку системи.

Висновки до розділу 3

1. Розроблена архітектура методу багатокритеріального аналізу. Вона є модульною та адаптивною, що дозволяє враховувати різні вимоги проєктів. Метод інтегрує можливості нормалізації даних, обчислення вагових коефіцієнтів для

критеріїв та автоматизованого вибору бази даних на основі багатокритеріальної моделі.

2. Забороновано метод для аналізу баз даних відповідно до визначених критеріїв (продуктивність, масштабованість, консистентність, тип даних, вартість). Метод дозволяє автоматично визначати інтегральну оцінку для SQL і NoSQL баз даних. Реалізація методу була протестована на конкретних прикладах із використанням баз даних, таких як MS SQL та MongoDB.

3. Проведений аналіз результатів реалізації показав ефективність методу у виборі оптимальної бази даних залежно від умов проєкту. Зокрема, для соціальних мереж із високими вимогами до масштабованості та обсягу даних оптимальним вибором стала MongoDB. Для аналітичних платформ і бухгалтерських систем було підтверджено доцільність використання SQL-баз, таких MS SQL.

4. Розроблений метод допомагає мінімізувати ризики неправильного вибору бази даних, що може призвести до зниження продуктивності системи або проблем із масштабованістю.

ВИСНОВКИ

1. Проведено аналіз прикладної області. Встановлено, що сучасні інформаційні системи потребують баз даних, здатних обробляти великі обсяги даних із забезпеченням продуктивності та масштабованості. Для різних проєктів застосовуються як реляційні (SQL), так і нереляційні (NoSQL) бази даних, залежно від типу даних, потреб у масштабуванні та складності запитів.

2. Досліджено методи оцінки продуктивності баз даних, зокрема, такі інструменти як HammerDB, YCSB та Sysbench. Було відзначено їхню здатність виконувати навантажувальні тести, оцінювати затримки, пропускну здатність та інші критичні параметри, важливі для ефективного вибору баз даних.

3. Розглянуто особливості реляційних і нереляційних баз даних, включаючи MongoDB, Cassandra, MS SQL та PostgreSQL. Виявлено ключові переваги та недоліки цих рішень залежно від вимог проєктів. SQL-бази підходять для проєктів із високими вимогами до консистентності та транзакційності, тоді як NoSQL забезпечують масштабованість та гнучкість.

4. Виявлено, що вибір між SQL та NoSQL базами залежить від конкретних умов проєкту: обсягів даних, вимог до швидкості, типу операцій і потреб у масштабуванні. Запропоновано базові критерії для оцінки ефективності баз даних у різних сценаріях.

5. Проведено опис використаних даних та інструментів. Визначено набір тестових даних та інструментів для оцінювання баз даних, які відповідають реальним сценаріям роботи інформаційних систем. Зокрема, бази даних різного типу SQL: PostgreSQL, MS SQL та типу NoSQL: MongoDB, Cassandra, та інструменти для моделювання робочого навантаження, такі як YCSB і Sysbench.

6. Проведено порівняльний аналіз методів оцінки продуктивності оцінки ефективності SQL та NoSQL баз даних, зокрема, за такими критеріями, як час виконання запитів, масштабованість, консистентність і вартість. Виявлено, що SQL-бази даних демонструють кращі результати для структурованих даних із

високими вимогами до консистентності, тоді як NoSQL забезпечують значно вищу масштабованість у розподілених системах.

7. Розроблено метод багатокритеріального аналізу для вибору оптимальної бази даних залежно від потреб проєкту. Метод базується на системі критеріїв (продуктивність, масштабованість, консистентність, тип даних, вартість) із застосуванням нормалізації даних і вагових коефіцієнтів для визначення інтегральної оцінки.

8. Протестовано метод на конкретних прикладах, що продемонструвало його ефективність у виборі бази даних для різних типів проєктів. Результати дослідження показали, що NoSQL-бази, такі як MongoDB, є оптимальними для великомасштабних розподілених систем, а SQL-бази, як MS SQL, найкраще підходять для транзакційних систем із високими вимогами до консистентності.

9. Розроблена архітектура методу багатокритеріального аналізу. Вона є модульною та адаптивною, що дозволяє враховувати різні вимоги проєктів. Метод інтегрує можливості нормалізації даних, обчислення вагових коефіцієнтів для критеріїв та автоматизованого вибору бази даних на основі багатокритеріальної моделі.

10. Забороновано метод для аналізу баз даних відповідно до визначених критеріїв (продуктивність, масштабованість, консистентність, тип даних, вартість). Метод дозволяє автоматично визначати інтегральну оцінку для SQL і NoSQL баз даних. Реалізація методу була протестована на конкретних прикладах із використанням баз даних, таких як MS SQL та MongoDB.

11. Проведений аналіз результатів реалізації показав ефективність методу у виборі оптимальної бази даних залежно від умов проєкту. Зокрема, для соціальних мереж із високими вимогами до масштабованості та обсягу даних оптимальним вибором стала MongoDB. Для аналітичних платформ і бухгалтерських систем було підтверджено доцільність використання SQL-баз, таких MS SQL.

12. Розроблений метод допомагає мінімізувати ризики неправильного вибору бази даних, що може призвести до зниження продуктивності системи або проблем із масштабованістю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. K. Kline, SQL in a nutshell, 3rd ed. O'Reilly Media, November 2008.

2. P. Warden, Big Data Glossary. O'Reilly Media, September 2011.

3. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation – Volume 7, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15.

4. L. George, HBase: The Definitive Guide. O'Reilly Media, August 2011.

5. B. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes," in Roedunet International Conference (RoEduNet), 2011 10th, june 2011, pp. 1 –5.

6. J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on, oct. 2011, pp. 363 –366.

7. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," SIGOPS Oper. Syst. Rev., vol. 41, no. 6, pp. 205–220, Oct. 2007.

8. K. Chodorow and M. Dirolf, MongoDB: The Definitive Guide. O'Reilly Media, September 2010.

9. J. C. Anderson, J. Lehnardt, and N. Slater, CouchDB: The Definitive Guide. O'Reilly Media, January 2010.

10. E. Hewitt, Cassandra: The Definitive Guide. O'Reilly Media, November 2010.

11. M. Brown, Getting Started with Couchbase Server. O'Reilly Media, June 2012.

12. N. Leavitt, "Will NoSQL databases live up to their promise?" Computer, vol. 43, no. 2, pp. 12 –14, feb. 2010.

13. D. Bartholomew, "SQL vs. NoSQL," Linux Journal, no. 195, July 2010.

14. S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," Communications Surveys Tutorials, IEEE, vol. 13, no. 3, pp. 311–336, 2011.

15. S. Tiwari, Professional NoSQL. Wiley/Wrox, August 2011.

16. M. Indrawan-Santiago, “Database research: Are we at a crossroad? Reflection on NoSQL,” in Network-Based Information Systems (NBiS), 2012 15th International Conference on, sept. 2012, pp. 45 –51.

17. R. Hecht and S. Jablonski, “NoSQL evaluation: A use case oriented survey,” in Cloud and Service Computing (CSC), 2011 International Conference on, dec. 2011, pp. 336 –341.

18. A. Boicea, F. Radulescu, and L. I. Agapin, “MongoDB vs Oracle – database comparison,” in Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on, sept. 2012, pp. 330 –335.

19. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb,” in Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC ’10. ACM, 2010, pp. 143–154.

20. Chen, Y., Xie, X., Wu, J. (2015). “A Benchmark Evaluation of Enterprise Cloud Infrastructure,” In: Cheng, R., Cui, B., Zhang, Z., Cai, R., Xu, J. (eds) Web Technologies and Applications. APWeb 2015. Lecture Notes in Computer Science(), vol 9313. Springer, Cham.

21. Krogh, J.W. (2020). Benchmarking with Sysbench. In: MySQL 8 Query Performance Tuning. Apress, Berkeley, CA: Apress, Berkeley. ISBN 978-1-4842-5583-4.

22. Mahmood, A., & Shaikh, Z. (2022). "Comparative Performance Analysis of SQL and NoSQL Databases in High-Throughput Environments." Journal of Cloud Computing: Advances, Systems, and Applications.

23. Chen, L., Wang, R. (2023). SQL vs. NoSQL: A Comparative Analysis of Database Performance and Scalability. Journal of Data Management, 5(2), 101-118.

24. Tran, P., & Do, N. (2023). "Hybrid Decision Support System for Optimal Database Selection in Dynamic Environments." International Journal of Intelligent Systems.

25. Indrawan-Santiago, M. (2020). "Database Comparison of SQL and NoSQL Databases in the Context of Data Operations for IoT Applications." International Journal of Applied Engineering Research.

26. Arora, A., & Aggarwal, A. (2019). "Comparative Analysis of SQL and NoSQL Databases for Big Data Applications." Journal of Big Data, 6(1).

27. Stonebraker, M., & Çetintemel, U. (2019). "One Size Fits All": An Idea Whose Time Has Come and Gone. Proceedings of the VLDB Endowment, 12(2), 40-50.

28. Han, J., Kamber, M. (2022). Data Mining: Concepts and Techniques. Elsevier.

29. Patel, H., Kumar, S. (2021). Machine Learning Approaches for Database Optimization. International Journal of Computer Science, 10(4), 77-88.

30. Олійник, О. В. (2020). Використання SQL та NoSQL для великих даних. Системи обробки інформації, 4(142), 120-127.

31. Diouri, M. E., & Tahiry, M. (2021). "Performance Comparison of NoSQL Databases: A Case Study for Cloud-Based Big Data Analytics." IEEE Access.

32. Agrawal, R., & Biswas, P. (2020). "Machine Learning Approaches for Database Performance Optimization: A Survey." ACM Computing Surveys.

33. Wu, J., & Shen, Z. (2021). "Applying Machine Learning Techniques to Database Management Systems: Challenges and Opportunities." Proceedings of the VLDB Endowment.

34. Zhou, P., & Xu, Q. (2019). "Performance Evaluation and Analysis of NoSQL Databases in Heterogeneous Environments." Future Generation Computer Systems.

35. Souri, A., Rahmani, A. M., & Zamani, M. (2020). "Case Study: Transitioning from SQL to NoSQL Database for Real-Time Big Data Analytics." IEEE Transactions on Big Data.

36. Lee, C., & Kwak, J. (2019). "Transition from SQL to NoSQL: A Practical Guide to Database Migration." Database Management Journal.

37. Silva, R., & Machado, R. (2022). "Decision-Making Model for Selecting Between SQL and NoSQL Databases Using Fuzzy Logic and Machine Learning." Expert Systems with Applications.

38. Бабенко О.В. Аналіз продуктивності баз даних MSSQL та MongoDB // Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі». 5 листопада 2024 р. – Тернопіль: ЗУНУ, 2024. – С.158-160.

39. Бабенко О.В. Аналіз методів та інструментів оцінки продуктивності SQL та NoSQL баз даних для їх оптимального вибору // Міжнародна науково-практична інтернет-конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» [Електронний ресурс] – Режим доступу: http://www.konferenciaonline.org.ua/ua/article/id-2020/

40. Комар М.П., Саченко А.О., Васильків Н.М., Загородня Д.І. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за другим (магістерським) рівнем вищої освіти. – Тернопіль: ЗУНУ, 2024. – 32 с.

Додаток А

Програмна реалізація порівняльного аналізу продуктивності

```
import numpy as np
import pyodbc
import time
import random
import matplotlib.pyplot as plt
from pymongo import MongoClient
from datetime import datetime

server = 'localhost'
username = 'sa'
password = 'pass123'

connectionstring = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};PORT=1433;UID={username};PWD={password}'
conn = pyodbc.connect(connectionstring)

cursor = conn.cursor()

# Create table
cursor.execute("""
CREATE TABLE BenchmarkTest (
ID INT IDENTITY PRIMARY KEY,
DataValue NVARCHAR(100),
CreatedAt DATETIME DEFAULT GETDATE()
)
""")

for i in range(100000):
cursor.execute("INSERT INTO BenchmarkTest (DataValue) VALUES (?)", f"Test Data {i}")

conn.commit()

# Insert benchmark
start_time = time.time()
cursor.execute("INSERT INTO BenchmarkTest (DataValue) VALUES (?)", "Test Data Last")
conn.commit()

end_time = time.time()
elapsed_time_ms = (end_time - start_time) * 1000 # Convert seconds to milliseconds
print(f"Duration Insert: {elapsed_time_ms:.2f} milliseconds")

# Update a random record
random_id = random.randint(1, 100000)
new_value = f"Updated Data {random_id}"

startUpdateTime = time.time()
```

```
    cursor.execute("UPDATE BenchmarkTest SET DataValue = ? WHERE ID
= ?", new_value, random_id)
    conn.commit()

    endUpdateTime = time.time()
    elapsedUpdateTime = (endUpdateTime - startUpdateTime) * 1000
    print(f"Duration Update: {elapsedUpdateTime:.2f} milliseconds")

    record_id = random.randint(1, 100000)
    startSelectTime = time.time()
    cursor.execute("SELECT * FROM BenchmarkTest WHERE ID = ?",
record_id)
    record = cursor.fetchone()  # Fetch one result
    conn.commit()

    endSelectTime = time.time()
    elapsed_time_ms2 = (endSelectTime - startSelectTime) * 1000
    print(f"Duration Select: {elapsed_time_ms2:.2f} milliseconds")

    # Delete a record by ID
    record_id_to_delete = record_id

    startDeleteTime = time.time()
    cursor.execute("DELETE FROM BenchmarkTest WHERE ID = ?",
record_id_to_delete)
    conn.commit()

    endDeleteTime = time.time()
    elapsed_time_ms3 = (endDeleteTime - startDeleteTime) * 1000
    print(f"Duration Delete: {elapsed_time_ms3:.2f} milliseconds")

    # Close the connection
    cursor.close()
    conn.close()

    # Connect to MongoDB server
    client                                                    =
MongoClient("mongodb://admin:password@localhost:27017/")

    # Access or create a database
    db = client['BenchmarkDB']
    # Access or create a collection
    collection = db['BenchmarkTest']

    # Delete all documents
    result = collection.delete_many({})

    for i in range(100000):
    # Insert a sample document
    document = {
        "DataValue": f"Test Data {i} ",
        "CreatedAt": datetime.utcnow()
    }
```

```
    result = collection.insert_one(document)

    start_time = time.time()
    document = {
    "DataValue": f"Test Data Test1 ",
    "CreatedAt": datetime.utcnow()
    }
    result = collection.insert_one(document)

    end_time = time.time()
    elapsedTimeMongoInsert = (end_time - start_time) * 1000
    print(f"Duration      Insert:      {elapsedTimeMongoInsert:.2f}
milliseconds")

    # Update benchmark
    random_id = 100
    startUpdateTime = time.time()
    collection.update_one(
    {"_id": random_id},  # Filter by the inserted document's ID
    {"$set": {"DataValue": "Updated Data"}})

    endUpdateTime = time.time()
    elapsed_timeUpdateMongoDB = (endUpdateTime - startUpdateTime) *
1000
    print(f"Duration     Update:     {elapsed_timeUpdateMongoDB:.2f}
milliseconds")

    startSelectTime = time.time()
    random_doc = collection.find_one({"_id": random_id})

    endSelectTime = time.time()
    elapsed_timeSelectTime = (endSelectTime - startSelectTime) * 1000
    print(f"Duration      Select:      {elapsed_timeSelectTime:.2f}
milliseconds")

    # Delete one record
    startDeleteTime = time.time()
    random_doc = collection.delete_one({"_id": random_id})

    endDeleteTime = time.time()
    elapsed_time_msDelete = (endDeleteTime - startDeleteTime) * 1000
    print(f"Duration      Delete:      {elapsed_time_msDelete:.2f}
milliseconds")
```

Додаток Б

Програмна реалізація оцінки продуктивності CRUD-операцій

```
import numpy as np
import pyodbc
import time
import random
import matplotlib.pyplot as plt
from pymongo import MongoClient
from datetime import datetime

server = 'localhost'
username = 'sa'
password = 'pass123'

connectionstring = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};PORT=1433;UID={username};PWD={password}'
conn = pyodbc.connect(connectionstring)

cursor = conn.cursor()
cursor.execute("DROP TABLE BenchmarkTest")
conn.commit()

# Create table
cursor.execute("""
CREATE TABLE BenchmarkTest (
ID INT IDENTITY PRIMARY KEY,
DataValue NVARCHAR(100),
CreatedAt DATETIME DEFAULT GETDATE()
)
""")

for i in range(10000):
cursor.execute("INSERT INTO BenchmarkTest (DataValue) VALUES (?)", f"Test Data {i}")

conn.commit()

# Insert benchmark
start_time = time.time()
for i in range(10000):
cursor.execute("INSERT INTO BenchmarkTest (DataValue) VALUES (?)", f"Insert test data {i}")
conn.commit()

end_time = time.time()
elapsed_time_ms = (end_time - start_time)
print(f"Duration Insert: {elapsed_time_ms:.2f} seconds")

# Update a random record
startUpdateTime = time.time()
```

```
    for i in range(10000):
    random_id = random.randint(1, 10000)  # Assuming the ID is from
1 to 10000
    new_value = f"Updated Data {random_id}"
    cursor.execute("UPDATE BenchmarkTest SET DataValue = ? WHERE ID
= ?", new_value, random_id)
    conn.commit()

    endUpdateTime = time.time()
    elapsedUpdateTime = (endUpdateTime - startUpdateTime)
    print(f"Duration Update: {elapsedUpdateTime:.2f} seconds")

    startSelectTime = time.time()
    for i in range(10000):
    record_id = random.randint(1, 10000)
    cursor.execute("SELECT * FROM BenchmarkTest WHERE ID = ?",
record_id)
    record = cursor.fetchone()  # Fetch one result
    conn.commit()

    endSelectTime = time.time()
    elapsed_time_ms2 = (endSelectTime - startSelectTime)
    print(f"Duration Select: {elapsed_time_ms2:.2f} seconds")

    # Execute the DELETE query
    startDeleteTime = time.time()
    for i in range(10000):
    record_id_to_delete = random.randint(1, 10000)
    cursor.execute("DELETE FROM BenchmarkTest WHERE ID = ?",
record_id_to_delete)
    conn.commit()  # Commit the deletion

    endDeleteTime = time.time()
    elapsed_time_ms3 = (endDeleteTime - startDeleteTime)
    print(f"Duration Delete: {elapsed_time_ms3:.2f} seconds")

    # Close the connection
    cursor.close()
    conn.close()

    # Connect to MongoDB server
    client                                                    =
MongoClient("mongodb://admin:password@localhost:27017/")
    # Access or create a database
    db = client['BenchmarkDB']
    # Access or create a collection
    collection = db['BenchmarkTest']

    # Delete all documents
    result = collection.delete_many({})

    for i in range(10000):
    # Insert a sample document
```

```
document = {
     "DataValue": f"Test Data {i} ",
     "CreatedAt": datetime.utcnow()  # UTC timestamp GETDATE()
}
result = collection.insert_one(document)

# Insert benchmark
start_time = time.time()
for i in range(10000):
# Insert a sample document
document = {
     "DataValue": f"UPDATE test data {i} ",
     "CreatedAt": datetime.utcnow()  # UTC timestamp GETDATE()
}
result = collection.insert_one(document)

end_time = time.time()
elapsedTimeMongoInsert = (end_time - start_time)
print(f"Duration Insert: {elapsedTimeMongoInsert:.2f} seconds")

# Update benchmark
startUpdateTime = time.time()
for i in range(10000):
random_id = random.randint(1, 10000)
collection.update_one(
     {"_id": random_id},  # Filter by the inserted document's ID
     {"$set": {"DataValue": "Updated Data"}})

endUpdateTime = time.time()
elapsed_timeUpdateMongoDB = (endUpdateTime - startUpdateTime)
print(f"Duration     Update:     {elapsed_timeUpdateMongoDB:.2f}
seconds")

startSelectTime = time.time()
for i in range(10000):
random_id = random.randint(1, 10000)
random_doc = collection.find_one({"_id": random_id})

endSelectTime = time.time()
elapsed_timeSelectTime = (endSelectTime - startSelectTime)
print(f"Duration Select: {elapsed_timeSelectTime:.2f} seconds")

startDeleteTime = time.time()
for i in range(10000):
random_id = random.randint(1, 10000)
# delete_one the document with the chosen ID
random_doc = collection.delete_one({"_id": random_id})

endDeleteTime = time.time()
elapsed_time_msDelete = (endDeleteTime - startDeleteTime)
print(f"Duration Delete: {elapsed_time_msDelete:.2f} seconds")
```

Додаток В

Копії публікацій

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ, АСПІРАНТІВ ТА МОЛОДИХ ВЧЕНИХ «ІНТЕЛЕКТУАЛЬНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ»

5 ЛИСТОПАДА 2024

KI.WUNU.EDU.UA/CONFERENCE/

ТЕРНОПІЛЬ

2024

Бабенко О.В.
магістрант 2 курсу ФКІТ ЗУНУ
Науковий керівник к.т.н., доцент Загородня Д.І., кафедра ІОСУ ЗУНУ

АНАЛІЗ ПРОДУКТИВНОСТІ БАЗ ДАНИХ MSSQL ТА MONGODB

Вступ. У сучасних інформаційних системах продуктивність баз даних є одним із ключових чинників, що визначають ефективність роботи програмних рішень. Розвиток технологій зберігання та обробки даних призвів до появи різних підходів до архітектури баз даних, серед яких виділяються реляційні (SQL) та нереляційні (NoSQL) бази даних. Кожна з них має свої переваги та недоліки, залежно від потреб конкретного проєкту [1-2]. Дана стаття присвячена аналізу продуктивності бази даних MsSQL (реляційна) та MongoDB (нереляційна). У роботі розглядаються ключові метрики продуктивності, зокрема: час виконання запитів для оцінки швидкості виконання CRUD-операцій; пропускна здатність, яка відображає кількість запитів, що система може обробити за одиницю часу; затримка обробки запитів, яка є критично важливою для систем із високим навантаженням.

Основна частина. Для обчислення продуктивності баз даних використовуються такі ключові показники: час виконання запитів, пропускна здатність і затримка.

Час виконання запитів відображає середній час, необхідний для виконання CRUD-операцій (вставка, зчитування, оновлення, видалення), що дозволяє оцінити швидкість бази даних під час роботи з окремими запитами. У багатокористувацьких системах швидке зчитування є важливим для звітності та аналітики, тоді як швидкий запис критично необхідний для реальних додатків, таких як логування або IoT, де обробляються великі обсяги даних у режимі реального часу.

Пропускна здатність характеризує кількість запитів, які система може виконати за одиницю часу. Цей показник має велике значення для високонавантажених систем, які обслуговують велику кількість одночасних користувачів або запитів. Бази даних із високою пропускною здатністю здатні підтримувати стабільну продуктивність навіть за умови інтенсивного навантаження. Особливою перевагою NoSQL баз, таких як Cassandra або Redis, є архітектурні рішення, які дозволяють досягати високої пропускної здатності.

Затримка є ще однією важливою метрикою, яка визначає час, необхідний для обробки одного запиту, починаючи від моменту надсилання до отримання результату. Ця характеристика є критично важливою для систем, які потребують миттєвого відгуку, таких як торговельні платформи, ігри або сервіси обробки даних у реальному часі. Реляційні бази даних зазвичай мають вищу затримку через обробку складних транзакцій, тоді як нереляційні бази, наприклад DynamoDB, демонструють кращу ефективність у зниженні затримок навіть за високого навантаження.

Для проведення експериментів використовувалося обладнання з процесором AMD Ryzen 7 6850, 32 ГБ оперативної пам'яті, твердотільним диском об'ємом 1 ТБ і операційною системою Linux Ubuntu 24.04 LTS. Це забезпечило ефективне виконання тестів продуктивності SQL та NoSQL баз даних, зокрема часу виконання запитів, пропускної здатності та затримки.

Час виконання. Час відгуку системи на запит - це час, що проходить між початком запиту та отриманням відповіді. Порівнювались два типи показників: середній час відгуку для виконаної операції та деталізований аналіз.

Операція читання (SELECT) у MsSQL демонструє вищу продуктивність, випереджаючи MongoDB на 6 мс. Це може свідчити про кращу оптимізацію MsSQL для обробки операцій читання. Водночас MongoDB показує кращі результати під час виконання вставок (INSERT), перевершуючи MsSQL на 10 мс, що, ймовірно, є результатом ефективнішої архітектури документно-орієнтованих баз даних для роботи з великими обсягами даних.

Відмінності в продуктивності під час оновлення (UPDATE) між MsSQL та MongoDB є незначними, з перевагою MsSQL у 10 мс. Це вказує на відсутність суттєвих проблем у MongoDB із оновленням даних, хоча існує можливість подальшої оптимізації. Операція видалення (DELETE) у MsSQL також демонструє перевагу, перевершуючи MongoDB на 5 мс. Ця різниця може бути пов’язана з кращою оптимізацією індексів та механізмів управління даними в MsSQL.

Загалом, MsSQL демонструє кращу продуктивність під час виконання більшості операцій, тоді як MongoDB виявляється ефективнішим у виконанні вставок. Таблиця 1 подає детальне порівняння продуктивності MsSQL та MongoDB.

Таблиця 1 – Порівняння продуктивності MsSQL та MongoDB

Операція	MsSQL	MongoDB	Різниця
SELECT	2	8	MsSQL швидше на 6 мс
INSERT	2	1	MongoDB швидше на 10 мс
UPDATE	10	20	MongoDB швидше на 10 мс
DELETE	2	7	MsSQL швидше на 5 мс

Для зручності результати роботи моделі подано у вигляді діаграми на рисунку 1. Вона ілюструє рекомендації щодо вибору між SQL та NoSQL базами даних залежно від конкретних критеріїв, оцінює продуктивність баз даних за кожним показником і порівнює MsSQL та MongoDB у різних сценаріях. Такий підхід забезпечує всебічне уявлення про продуктивність баз даних та обґрунтованість їх вибору залежно від потреб конкретного проєкту.

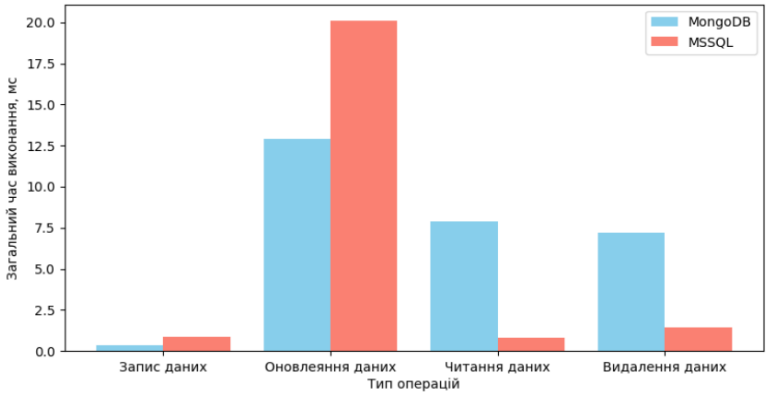


Рисунок 1 – Аналіз часу виконання операцій БД

Затримка. Затримка (latency) є ключовим показником продуктивності бази даних, що визначає час, необхідний для виконання операцій читання та запису. За результатами тестування, у MsSQL затримка під час читання становить 1 мс, а під час запису — 2 мс. У MongoDB ці показники дещо кращі: затримка під час читання складає 5 мс, а під час запису — лише 1 мс.

Менша затримка MongoDB як у читанні, так і в записі робить її більш ефективною для систем із високим навантаженням. У таких умовах швидка обробка запитів є критично важливою, оскільки навіть незначне зменшення затримки може суттєво підвищити загальну продуктивність системи та скоротити час її відповіді. Це має особливе значення для застосунків, де швидкість реакції є критичною, наприклад, у реальному часі або при роботі з великими обсягами даних.

Пропускна здатність. Пропускна здатність баз даних відображає кількість операцій або запитів, які система може обробити за одиницю часу. За результатами тестування, реляційна база даних MsSQL демонструє пропускну здатність у 15 000 транзакцій на секунду (тест HammerDB TPCC Benchmark), тоді як MongoDB досягає 12 000 запитів на секунду (тест YCSB).

MsSQL забезпечує вищу пропускну здатність завдяки оптимізації для обробки великих транзакцій, що є характерним для реляційних баз даних. Це дозволяє ефективно управляти даними з високими вимогами до їх цілісності та консистентності. У свою чергу, MongoDB має дещо меншу пропускну здатність, але її ключовою перевагою є можливість горизонтального масштабування. Завдяки додаванню нових серверів до кластеру, MongoDB може суттєво

збільшити свою пропускну здатність, що є критично важливим для реальних умов, де обсяги даних і навантаження на систему постійно зростають.

Порівняння пропускної здатності MsSQL та MongoDB здійснювалося шляхом виконання 10 000 запитів для чотирьох основних операцій: вставки, оновлення, читання та видалення. Детальні результати тестування представлені на рисунку 2.

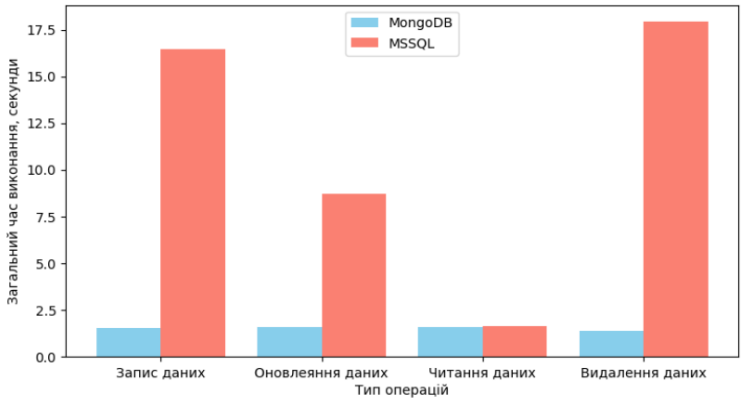


Рисунок 2 – Результати аналізу пропускної здатності MsSQL та MongoDB для основних операцій

Висновки. Результати тестування свідчать про те, що MongoDB значно перевершує MsSQL у швидкості виконання операцій вставки, оновлення та видалення даних. Наприклад, вставка даних у MongoDB тривала лише 1,5 секунди, тоді як MsSQL потребувала 16 секунд. Аналогічно, операція оновлення у MongoDB була швидшою — 1,5 секунди проти 8 секунд у MsSQL. Видалення даних виявило ще більший розрив: MongoDB виконувала цю операцію за 1,4 секунди, тоді як MsSQL витрачала на це 18 секунд. Щодо операцій читання, різниця між системами виявилася незначною: MongoDB виконувала завдання за 1,6 секунди, а MsSQL — за 1,7 секунди. Це вказує на те, що обидві бази демонструють схожу ефективність у цій категорії.

Загальний аналіз підтверджує, що MongoDB є більш ефективною для роботи з великими обсягами даних, особливо де потрібна швидка обробка запитів, наприклад, у системах реального часу або за високого навантаження. У той же час, MsSQL залишається потужним інструментом для виконання задач, що потребують стабільної продуктивності при складних транзакціях.

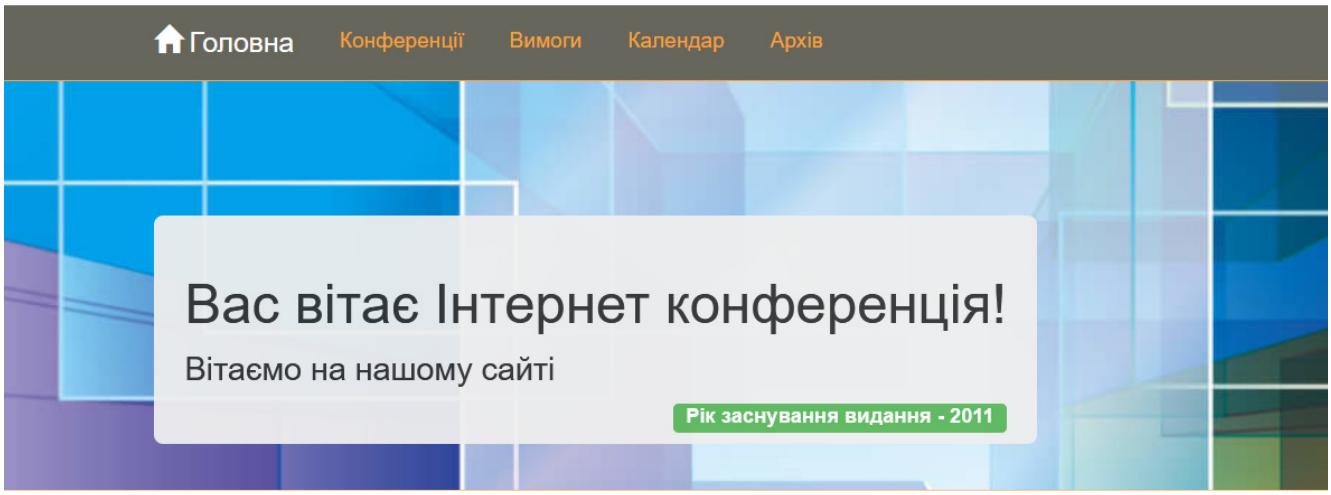
Порівняння SQL та NoSQL баз даних демонструє, що вибір між ними залежить від специфіки проєкту. NoSQL є оптимальними для обробки великого обсягу запитів із мінімальною затримкою, тоді як SQL краще підходять для складних аналітичних запитів та задач, де потрібна транзакційна цілісність. Запропонована оцінка дає змогу вибрати базу даних, яка найкраще відповідає технічним і бізнес-вимогам, забезпечуючи стабільність і масштабованість систем навіть за умов високого навантаження.

Список літератури

1. A. Rao, D. Khankhoje, U. Namdev, C. Bhadane, and D. Dongre, "Insights into NoSQL databases using financial data: A comparative analysis," Procedia Computer Science, vol. 215, pp. 8–23, 2022. DOI: 10.1016/j.procs.2022.12.002.

2. W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review," Big Data and Cognitive Computing, vol. 7, no. 2, art. 97, 2023. DOI: 10.3390/bdcc7020097.

3. Т. Нікітіна і О. Морозова, "Порівняльний аналіз продуктивності баз даних SQL та NoSQL," Системи управління, навігації та зв'язку. Збірник наукових праць, вип. 1, с. 125–128, 2019. DOI: 10.26906/SUNZ.2019.1.125.



АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТІВ ОЦІНКИ ПРОДУКТИВНОСТІ SQL ТА NOSQL БАЗ ДАНИХ ДЛЯ ЇХ ОПТИМАЛЬНОГО ВИБОРУ

12.12.2024 10:27

[1. Інформаційні системи і технології]

Автор: **Бабенко Олександр Валерійович, магістрант, Західноукраїнський національний університет, м. Тернопіль**

Вступ

Протягом тривалого часу спостерігається постійне зростання обсягів даних, а також безперервний розвиток методів їх створення, накопичення, зберігання та використання. Характер даних також еволюціонує, трансформуючись із чітко структурованих до більш неструктурованих форматів. Це зумовлює необхідність ефективного зберігання та управління такими даними, які не можуть бути оброблені традиційними методами реляційних баз даних, таких як MySQL, Oracle, Microsoft SQL Server.

У зв'язку з цим бази даних NoSQL набули широкої популярності та стали ключовим інструментом для управління сучасними даними. Однією з головних переваг найпопулярніших NoSQL баз даних, таких як MongoDB, Couchbase, CouchDB і DynamoDB, є їхня доступність завдяки інтеграції з хмарними рішеннями для обробки великих обсягів даних. Це робить їх чудовим вибором навіть для невеликих організацій із обмеженим бюджетом.

Таким чином, питання вибору між SQL і NoSQL структурами баз даних є надзвичайно актуальним у контексті безперервного розвитку технологій зберігання та обробки даних [1].

Методи оцінки

Для оцінки продуктивності баз даних зазвичай використовують навантажувальне тестування, яке дозволяє визначити продуктивність системи при високих обсягах операцій читання, запису, оновлення та інших типів запитів. Це допомагає оцінити, як база даних реагує на різні рівні навантаження, а також виявити потенційно вузькі місця або обмеження конфігурації. Існує кілька підходів для такого тестування, які варіюються за складністю, типами навантаження та спеціалізацією:

1. Симуляція реального навантаження — використання інструментів, які відтворюють сценарії роботи реальних користувачів, наприклад транзакції, аналітичні запити чи великі вставки даних.

2. Тестування пропускної здатності — визначення максимальної кількості запитів, які система може обробити за певний час.

3. Тестування затримки (Latency) — оцінка часу виконання окремих запитів або транзакцій.

4. Тестування масштабованості — перевірка змін продуктивності зі збільшенням обсягу даних, кількості підключень чи серверів.

Аналіз інструментів

Проведено аналіз, на основі котрого складена порівняльна таблиця популярних інструментів навантажувального тестування.

Інструмент	Спеціалізація	Ключові функції	Переваги	Недоліки
HammerDB	Реляційні бази даних (SQL)	Імітація бізнес-процесів, підтримка кластеризації, моніторинг у реальному часі	Ефективний для SQL, підтримка кластеризації, аналіз у реальному часі	Не підтримує NoSQL бази даних
YCSB	NoSQL бази даних	Кастомізація сценаріїв, підтримка розподілу ключів, модульність для нових драйверів	Гнучкість для NoSQL, модульність, підтримка складних сценаріїв	Сфокусований лише на NoSQL системах
Sysbench	SQL бази даних, файлові системи та ресурси системи	Підтримка багатопотоковості, CLI, гнучке налаштування	Універсальний для різних систем, зручний у використанні через CLI	Переважно для SQL систем, відсутній графічний інтерфейс
Apache JMeter	Веб-додатки, API та SQL/NoSQL бази даних	Складні сценарії навантаження з великою кількістю паралельних запитів	Підтримка складних сценаріїв, висока масштабованість	Складність налаштування для розширених тестів
pgbench	PostgreSQL	Тестування транзакцій, запитів та навантаження на рівні окремих баз даних	Простий у використанні, оптимізований для PostgreSQL	Обмежений виключно PostgreSQL
OLTPBench	Онлайн-обробка транзакцій (OLTP)	Підтримка змішаних робочих сценаріїв для SQL та NoSQL	Широка підтримка OLTP-навантажень, універсальність	Менш поширений, обмежена підтримка спільноти
Locust	API та бази даних (відкритий код)	Генерація навантаження, великомасштабні тести	Масштабованість, відкритий код, кастомізовані сценарії	Більше фокусу на API, ніж на метриках баз даних
Database Benchmark	Операції читання та запису в SQL базах даних	Простий і легкий для тестування основних операцій	Легкий, швидкий для оцінки	Обмежені можливості для розширеного тестування
Percona Toolkit	Оптимізація MySQL, MariaDB, MongoDB	Аналіз повільних запитів, виявлення вузьких місць	Потужна оптимізація, популярний серед адміністраторів MySQL	Обмежений лише MySQL-сумісними системами
Artillery	API та бази даних	Простий у налаштуванні, підтримка великих сценаріїв тестування	Простота конфігурації, підходить для великомасштабного навантаження	Простий для складних сценаріїв тестування баз даних

Висновок

У даній роботі було проведено аналіз методів і інструментів для тестування продуктивності баз даних. Навантажувальне тестування виявилося ключовим підходом для оцінки здатності систем обробляти великі обсяги операцій, а також для ідентифікації вузьких місць і обмежень конфігурації. Даний підхід забезпечує основу для ефективного вибору SQL або NOSQL бази даних залежно від потреб конкретного проєкту.

Література:

1. Rao, Ashish & Khankhoje, Dhruvi & Namdev, Uditi & Bhadane, Chetashri & Dongre, Deepika. (2022). Insights into NoSQL databases using financial data: A comparative analysis. Procedia Computer Science. 215. 8-23. 10.1016/j.procs.2022.12.002.

Науковий керівник: Загородня Діана Іванівна, кандидат технічних наук, доцент, Західноукраїнський національний університет, м. Тернопіль