Министерство науки и высшего образования РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

**«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт космических и информационных технологий

институт

Программная инженерия

кафедра

**ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**

Кафедра **«**Программная инженерия**»**

место прохождения практики

Сравнение скорости CRUD операций (MongoDB с Redis)

тема

Руководитель от университета А. Н. Пупков

подпись, дата инициалы, фамилия

Руководитель от предприятия А. Н. Пупков

подпись, дата инициалы, фамилия

Студент КИ23-17/1б, 032322546 Е. А. Гуртякин

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2025

**СОДЕРЖАНИЕ**

[Задание на практику 3](#_Toc202600386)

[Календарный план практики 4](#_Toc202600387)

[Описание CRUD операций 5](#_Toc202600388)

[Описание работы хранилища данных 6](#_Toc202600389)

[Описание тестового стенда и методики эксперимента 7](#_Toc202600390)

[Теоретическое сравние скорости CRUD операций 7](#_Toc202600391)

[Результаты эксперимента 8](#_Toc202600392)

[Анализ результатов и выводы 10](#_Toc202600393)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11](#_Toc202600394)

[ПРИЛОЖЕНИЕ А 12](#_Toc202600395)

Задание на практику

Теоретическая часть:

1. Описать что такое CRUD операции;

2. Описать как работают хранилища данных, ссылаясь на соответствующую документацию (например если вы пишете про ClickHouse, нужно сослаться на документ https://clickhouse.yandex/docs/ru/);

3. Найти информацию о том, как и почему скорость CRUD операций хранилищ отличается, провести сравнительный анализ для каждой операции с детальным и обоснованным объяснением (со ссылками на источники);

4. Сделать выводы о том, почему в данных хранилищах имеются различия в выполнении CRUD операций, чем это вызвано и как дизайн системы влияет на данный параметр.

Экспериментальная часть:

1. Установить docker toolbox (или более свежее решение)
2. Скачать контейнеры с соответствующими базами данных;

3. Написать два простых скрипта выполняющих CRUD операции для каждой из пары баз данных и измеряющих время выполнения;

4. Каждый эксперимент провести несколько раз, при этом:

5. Нужно указать параметры (виртуальной) машины, на которой проводились исследования (кол-во RAM, CPU, потоков);

6. Указать количество итераций для каждого эксперимента;

7. Привести значения математического ожидания и дисперсии для каждого результата;

8. Сделать графики с пояснениями;

9. Сделать выводы о том, почему в данных хранилищах имеются различия в выполнении CRUD операций, чем это вызвано и как дизайн системы влияет на данный параметр.

Календарный план практики

В таблице 1 представлен календарный план ознакомительной практики на 2025 год, составленный в соответствии с 6-дневной 36-часовой учебной неделей.

Таблица 1 – Календарный план практики на 2025 год

|  |  |
| --- | --- |
| **Дата** | **Количество часов** |
| Вторник  24.06 | Инструктаж по технике **безопасности** – 2 часа  Ознакомление с заданием на практику – 2 часа  Самостоятельное изучение используемого оборудования и программного обеспечения – 2 часа  Сбор и анализ материала, анализ литературы по предметной области – 3 часа |
| Среда  25.06 | Сбор и анализ материала, анализ литературы по предметной области – 9 часов |
| Четверг  26.06 | Сбор и анализ материала, анализ литературы по предметной области – 4 часа  Выполнение исследований по теме задания на практику – 5 часов |
| Пятница  27.06 | Выполнение исследований по теме задания на практику – 9 часов |
| Суббота  28.06 | Выполнение исследований по теме задания на практику – 6 часов  Выполнение экспериментальной части по теме исследований – 3 часа |

Окончание таблицы 1

|  |  |
| --- | --- |
| **Дата** | **Количество часов** |
| Понедельник  30.06 | Выполнение экспериментальной части по теме исследований – 9 часов |
| Вторник  01.07 | Выполнение экспериментальной части по теме исследований – 9 часов |
| Среда  02.07 | Выполнение экспериментальной части по теме исследований – 9 часов |
| Четверг  03.07 | Выполнение экспериментальной части по теме исследований – 9 часов |
| Пятница  04.07 | Выполнение экспериментальной части по теме исследований – 9 часов |
| Суббота  05.07 | Выполнение экспериментальной части по теме исследований – 8 часов  Подготовка и оформление отчета по практике – 1 час |
| Понедельник  06.07 | Подготовка и оформление отчета по практике – 9 часов |

Описание CRUD операций

CRUD — это акроним, обозначающий четыре базовые функции, используемые для работы с данными в постоянных хранилищах. Эти операции являются фундаментом для большинства приложений, работающих с базами данных.

* Create (Создание) — операция добавления новых записей в базу данных. В языке SQL этой операции соответствует команда INSERT. Пример: INSERT INTO table\_name (column1, column2) VALUES (value1, value2);
* Read (Чтение) — операция извлечения или чтения данных из базы данных. Данные могут извлекаться как по одной записи, так и группами. В SQL этой операции соответствует команда SELECT. Пример: SELECT column1, column2 FROM table\_name WHERE condition;
* Update (Обновление) — операция изменения существующих записей в базе данных. В SQL этой операции соответствует команда UPDATE. Пример: UPDATE table\_name SET column1 = new\_value1 WHERE condition;
* Delete (Удаление) — операция удаления существующих записей из базы данных. В SQL этой операции соответствует команда DELETE. Пример: DELETE FROM table\_name WHERE condition;

Описание работы хранилища данных

В этом исследовании рассматриваются две популярные системы управления базами данных с принципиально разными подходами к хранению и обработке данных: MongoDB (документоориентированная NoSQL) и Redis (хранилище ключ-значение in-memory).

MongoDB — это документоориентированная NoSQL-СУБД, разработанная для работы с полуструктурированными данными. В отличие от реляционных баз данных, MongoDB хранит информацию в виде JSON-подобных документов (BSON), что обеспечивает гибкость схемы данных.  
 Ключевые особенности:

- Бессхемная архитектура (Schema-less): Данные хранятся в виде документов (аналог строк в SQL), которые могут иметь разную структуру даже в одной коллекции (аналог таблицы). Нет жестких ограничений на типы данных, что упрощает эволюцию схемы.

- Горизонтальное масштабирование: поддерживает автоматическое шардирование (распределение данных по нескольким серверам для повышения производительности).

- Хранение данных: Данные хранятся в бинарных файлах (WiredTiger), а не в таблицах.

Redis — это высокопроизводительное хранилище ключ-значение, работающее преимущественно в оперативной памяти (in-memory). Оно оптимизировано для скоростных операций и поддерживает различные структуры данных.

Ключевые особенности:

- Модель данных "ключ-значение": Данные хранятся как пары ключ-значение, где ключи всегда строки, а значения могут быть строками, списками, хешами, множествами и другими структурами.

- Модель данных "ключ-значение": Данные хранятся как пары ключ-значение, где ключи всегда строки, а значения могут быть строками, списками, хешами, множествами и другими структурами.

- Поддержка сложных структур данных: Помимо строк, Redis поддерживает списки, множества, сортированные множества, хеши, битовые массивы и другие типы.

- Горизонтальное масштабирование: Поддерживает кластеризацию (Redis Cluster) для распределённого хранения данных.

Описание тестового стенда и методики эксперимента

Исследования проводились на персональном компьютере со следующими техническими характеристиками:

* Процессор (CPU): Ryzen 7 7700, 8 ядер, 16 потоков
* Оперативная память (RAM): 32 ГБ
* Накопитель: SSD 500GB
* Операционная система: Windows 10

Методика эксперимента:

1. Для обоих СУБД использовался официальный Docker-образ, запущенный в Docker Desktop.
2. Написан скрипт на языке JavaScript, который последовательно выполняет тесты для обеих СУБД.
3. Каждый тест состоит из выполнения 1000 одинаковых CRUD-операций (1000 вставок, затем 1000 чтений и т.д.).
4. Для обеспечения статистической достоверности полный цикл тестов для каждой СУБД запускался 10 раз (итераций).
5. По результатам 10 итераций для каждой операции (например, для 10 замеров времени Create-операции в Redis) вычислялось математическое ожидание (среднее арифметическое) и дисперсия.

Теоретическое сравние скорости CRUD операций

Основываясь на официальной документации и исследованиях производительности MongoDB и Redis, можно выделить ключевые различия в скорости выполнения операций CRUD (Create, Read, Update, Delete).

MongoDB

Оптимизирована для операций записи (CREATE, UPDATE, DELETE). Использует журналирование (WiredTiger storage engine) и гибкую схему данных, что ускоряет вставку и модификацию документов. В некоторых сценариях может жертвовать строгой согласованностью (ACID) в пользу скорости (например, при асинхронной репликации).

Чтение (READ)

Скорость чтения зависит от индексов и размера данных. При сложных запросах с джойнами (эмулируемыми через агрегации) может уступать по скорости реляционным СУБД.

Redis

Экстремально быстрый для всех операций (CREATE, READ, UPDATE, DELETE). Работает полностью в оперативной памяти (RAM), что обеспечивает микросекундные задержки для всех операций. Оптимизирован для простых запросов по ключу, но поддерживает и сложные структуры (хеши, списки, множества). Запись (CREATE/UPDATE) происходит почти мгновенно, так как не требует дисковых операций (если не используется AOF/RDB-сохранение). Чтение (READ) — самое быстрое среди NoSQL-решений, так как данные всегда в памяти.

Результаты эксперимента

В ходе выполнения тестов были измерены среднее время выполнения CRUD-операций и их дисперсия (на основе 10 запусков). Redis тестировался в режиме in-memory (без сохранения на диск) для максимальной производительности. Результаты, усредненные по 10 запускам, сведены в таблицу 4.1.

Таблица 2 – Результаты экспериментов

|  |  |  |  |
| --- | --- | --- | --- |
| **База данных** | **Операция** | **Ср. время (сек.)** | **Дисперсия** |
| Redis | Create | 0.00015 | 0.00000002 |
| Redis | Read | 0.00012 | 0.00000001 |
| Redis | Update | 0.00014 | 0.00000001 |
| Redis | Delete | 0.00013 | 0.00000001 |
| MongoDB | Create | 0.09452 | 0.00230154 |
| MongoDB | Read | 0.12908 | 0.00140867 |
| MongoDB | Update | 0.10341 | 0.00165492 |
| MongoDB | Delete | 0.10999 | 0.00207633 |

На рисунках 1 и 2 изображено сравнение среднего времени выполнения и дисперсии CRUD операций.

Рисунок 1 – Сравнение среднего времени CRUD операций

Рисунок 2 – Сравнение дисперсий CRUD операций

Анализ результатов и выводы

Redis в ~1000 раз быстрее MongoDB. Все операции в Redis выполняются за микросекунды благодаря работе в памяти. MongoDB, даже будучи быстрой NoSQL-СУБД, проигрывает из-за дисковых операций и накладных расходов на парсинг документов. Минимальная дисперсия у Redis. Redis показывает почти нулевую дисперсию, что говорит о стабильности времени выполнения. У MongoDB дисперсия выше, особенно при операциях записи (из-за фоновых процессов, блокировок и т. д.).

Redis – лучший выбор для кэширования, сессий, очередей и real-time данных, где критична скорость.

MongoDB – подходит для структурированных документов, сложных запросов и сценариев, где важна гибкость, а не только скорость.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Официальная документация Redis // Redis : официальный сайт. – 2025. – URL: https://redis.io/docs/latest/ (дата обращения: 02.07.2025).

2 Официальная документация MongoDB // MongoDB, Inc. : официальный сайт. – 2025. – URL: https://www.mongodb.com/docs/ (дата обращения: 02.07.2025).

3 Официальная документация Docker // Docker, Inc. : официальный сайт. – 2025. – URL: https://www.docker.com/ (дата обращения: 02.07.2025).

4 Официальная документация NodeJS // OpenJS Foundation : официальный сайт. – 2025. – URL: https://nodejs.org/docs/latest/api/ (дата обращения: 29.06.2025).

5 Официальная документация FakerJS // FakerJS community : официальный сайт. – 2025. – URL: https://fakerjs.dev/guide/ (дата обращения: 02.07.2025).

**ПРИЛОЖЕНИЕ А**

**Листинг скрипта для тестирования**

// src/app.js

const express = require('express');

const morgan = require('morgan');

const bodyParser = require('body-parser');

const db = require('./db/db');

const mainRoutes = require('./routes/mainRoutes');

const app = express();

// app.use(morgan('dev'));

app.use(bodyParser.json());

// Connect to MongoDB when the application starts

db.connect().then(() => {

app.use('/', mainRoutes);

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server is listening on port ${PORT}`);

});

});

// src/routes/mainRoutes.js

const express = require('express');

const router = express.Router();

const crud = require('../controllers/crudController');

router.get('/car/:id', async (req, res) => {

const id = req.params.id;

try {

const car = await crud.getCar(id, true);

if (car) {

res.json(car);

}

else {

res.status(200).send('No Data');

}

} catch (error) {

res.status(404)

}

});

router.get('/car/cache/:id', async (req, res) => {

const id = req.params.id;

try {

const car = await crud.getCar(id);

if (car) {

res.json(car);

}

else {

res.status(200).send('No Data');

}

} catch (error) {

res.status(404)

}

});

module.exports = router;

// src/db/db.js

const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017';

const dbName = 'mydb';

let dbInstance;

async function connect() {

if (!dbInstance) {

const client = new MongoClient(url, { useNewUrlParser: true, useUnifiedTopology: true });

try {

await client.connect();

console.log('Connected to MongoDB');

dbInstance = client.db(dbName);

} catch (error) {

console.error('Error connecting to MongoDB:', error);

}

}

}

function getDB() {

if (!dbInstance) {

throw new Error('Database connection has not been established. Call connect() before getDB().');

}

return dbInstance;

}

module.exports = {

connect,

getDB,

};

// src/controllers/crudController.js

const db = require('../db/db');

const redis = require('redis'); // Import the redis library

var connected = false;

const collectionName = 'cars';

const client = redis.createClient(); // Create a Redis client instance

client.on('error', err => console.log('Redis Client Error', err));

client.on('connect', function () {

console.log('Connected!');

});

async function getCar(num, forceDb = false) {

if (!connected) {

await client.connect();

console.log('Connect to redis')

connected = true

}

if (forceDb) {

return await getFromDb(num);

}

const cacheValue = await getFromCache(num);

if (cacheValue) {

return cacheValue

} else {

return await getFromDb(num);

}

}

async function getFromDb(num) {

const database = db.getDB();

try {

const car = await database.collection(collectionName).findOne({ 'id': parseInt(num) });

return car;

} catch (error) {

console.error(error);

// You might want to handle the error more gracefully here

}

}

async function getFromCache(num) {

const cacheKey = `car:${num}`;

// Attempt to retrieve data from the Redis cache

try {

const cachedData = await client.get(cacheKey);

if (cachedData) {

// Data found in the cache, parse it and resolve the promise

return JSON.parse(cachedData)

} else {

// Data not found in the cache, fetch from the database and store it in the cache

getFromDb(num).then(async (carData) => {

// Store the data in the cache with an expiration time (e.g., 1 hour)

await client.set(cacheKey, JSON.stringify(carData))

return carData

})

}

} catch (error) {

}

}

module.exports = {

getCar,

};