Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Исследование производительности постреляционнах баз данных с применением технологий тестирования

Студент ИУ5-44М (Группа)			**Е.А. Кучин** (И.О.Фамилия)
Руководитель ВКР		(Подпись, дата)	**М.В. Виноградова** (И.О.Фамилия)
Консультант		(Подпись, дата)	(И.О.Фамилия)
Нормоконтролер		(Подпись, дата)	**Ю.Н. Кротов** (И.О.Фамилия)

2025 г.

АННОТАЦИЯ

Пояснительная записка к дипломной работе содержит 105 страниц. Работа включает в себя 18 таблиц и 27 иллюстраций. В процессе выполнения было использовано 30 источников.

Цель работы заключается в проведении комплексного сравнительного исследования производительности и масштабируемости постреляционных СУБД (MongoDB, Cassandra) и традиционной реляционной СУБД (PostgreSQL) с применением стандартизированных технологий тестирования для выявления их сильных и слабых сторон.

В работе представлено три раздела: теоретические основы и обзор СУБД, методология и подготовка исследования, проведение тестирования и анализ результатов.

В первой главе проводится теоретический анализ архитектурных особенностей и методов обработки данных в исследуемых СУБД. Выполняется сравнительный анализ технологий и инструментов бенчмаркинга, на основе которого обосновывается выбор универсального бенчмарка Yahoo! Cloud Serving Benchmark (YCSB) для проведения исследования. Также приводится анализ глобального рынка СУБД.

Во второй главе детально описывается тестовое окружение, включая аппаратную и программную конфигурацию. Рассматриваются подходы к загрузке и подготовке большого набора данных (12 ГБ) для каждой из трёх СУБД, учитывая их специфику. Формулируется подробная методология проведения тестирования, включая описание рабочих нагрузок и измеряемых метрик.

В заключительной, третьей главе, проводится серия экспериментов по тестированию производительности MongoDB, Cassandra и PostgreSQL. Для каждой СУБД анализируются показатели пропускной способности и задержек при различных рабочих нагрузках и уровнях параллелизма. Проводится

итоговый сравнительный анализ результатов, на основе которого выявляются ключевые закономерности и особенности поведения каждой системы.

Результатом работы являются сформулированные практические рекомендации по выбору оптимальной СУБД для различных сценариев использования. Продемонстрировано, что Cassandra лидирует в стандартных CRUD-операциях, PostgreSQL показывает исключительную производительность при чтении "горячих" данных, а MongoDB эффективна для операций сканирования. Работа доказывает отсутствие универсального решения и подчеркивает важность выбора СУБД на основе конкретных требований приложения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время информация имеет огромную ценность, поэтому важно уметь грамотно её структурировать, обобщать и представлять для последующего анализа. С ростом объёмов данных, генерируемых при использовании сетевых технологий, традиционные реляционные базы данных (*SQL*) сталкиваются с проблемами масштабирования. В ответ появились базы данных *NoSQL*, которые были разработаны для горизонтального масштабирования и высокой доступности, но за счет компромиссов в функциональности [1].

Поиск – это процесс, при котором проверяется каждый объект в наборе, чтобы найти совпадение с заданным критерием. Он позволяет пользователю находить конкретные данные в наборе данных путем последовательного сравнения заданных критериев с каждым элементом в массиве. Это помогает извлекать необходимую информацию, знания или сведения для пользователя.

Структуризация информации может потребоваться для поиска и сравнения похожих наборов данных, поиска дубликатов, составления списка данных смежной тематики или просто списка рекомендуемых данных [2].

В этой работе рассматриваются кластерные базы данных, сравнивается их производительность. Цель исследования – повышение эффективности полнотекстового поиска в кластерных нереляционных СУБД. Задачи: разработка рекомендаций по эффективному применению механизмов поиска в кластерных нереляционных СУБД на примере *MongoDB*, *Cassandra* и *Neo4j*, оценка зависимости времени выполнения запросов на выборку данных от их селективности, а также наличия кластера.

Введение

Цель работы

Провести комплексное сравнительное исследование производительности и масштабируемости постреляционных СУБД (MongoDB, Cassandra) и реляционной СУБД PostgreSQL при обработке больших объёмов данных с использованием стандартизированных методов тестирования для выявления их сильных и слабых сторон в различных сценариях использования.

Задачи исследования

1. Проанализировать архитектурные особенности исследуемых СУБД (PostgreSQL, MongoDB, Cassandra) и их теоретический потенциал для обработки больших объемов данных
2. Разработать методологию тестирования, обеспечивающую объективное сравнение СУБД с различными моделями данных
3. Реализовать единое тестовое окружение для проведения стандартизированных бенчмарков
4. Подготовить большой набор тестовых данных (>12 ГБ) и разработать методы его эффективной загрузки в каждую из исследуемых СУБД
5. Провести серию тестов производительности с использованием инструмента YCSB для различных типов рабочих нагрузок и уровней параллелизма
6. Проанализировать и визуализировать результаты тестирования с помощью инструмента Apache Superset
7. Сформулировать рекомендации по выбору оптимальной СУБД для различных сценариев использования на основе полученных результатов

Актуальность исследования

Современный мир характеризуется беспрецедентным ростом объёмов данных, генерируемых как в корпоративной среде, так и конечными

пользователями. По оценкам экспертов, глобальный объём цифровых данных увеличивается со скоростью около 2.5 квинтиллиона байт ежедневно, а к 2025 году совокупный объём мировых данных достигнет 175 зеттабайт.

Выбор подходящей системы управления базами данных становится критически важным фактором, определяющим производительность, масштабируемость и общую эффективность приложений. В основе большинства систем обработки данных лежат различные системы управления базами данных (СУБД), которые претерпели значительную эволюцию.

Наряду с традиционными реляционными СУБД, адаптирующимися к новым требованиям, широкое распространение получили NoSQL системы (документоориентированные, колоночные, графовые, ключ-значение), предлагающие преимущества для определённых сценариев обработки данных. Выбор правильной технологии становится критически важным, так как ошибка может привести к проблемам масштабирования, производительности и высоким эксплуатационным расходам.

Рынок СУБД демонстрирует устойчивый рост: общий объём рынка СУБД, оцениваемый в 100,79 млрд долларов США в 2023 году, достигнет 292,22 млрд к 2030 году при среднегодовом темпе роста 14,21%. Особенно заметен рост сегмента NoSQL СУБД, который увеличится с 7,55 млрд долларов в 2023 году до 47,41 млрд к 2030 году. Это обуславливает необходимость в объективных исследованиях производительности различных типов СУБД.

Технологии бенчмаркинга предоставляют объективный инструментарий для оценки производительности различных СУБД, позволяя моделировать сценарии использования и измерять ключевые показатели (пропускную способность, время отклика, масштабируемость). Данное исследование посвящено сравнительному анализу производительности PostgreSQL, MongoDB и Cassandra в контексте обработки большого набора данных с использованием бенчмарка Yahoo! Cloud Serving Benchmark (YCSB).

Объект и предмет исследования

Объект исследования: Системы управления базами данных PostgreSQL, MongoDB и Cassandra как технологические платформы для обработки данных, развернутые в едином тестовом окружении.

Предмет исследования: Показатели производительности и масштабируемости (пропускная способность, время отклика на операции чтения и записи, эффективность параллельной обработки) СУБД PostgreSQL, MongoDB и Cassandra при обработке больших объемов данных с использованием стандартизированных рабочих нагрузок, генерируемых инструментом YCSB с различными параметрами.

Глава 1. Теоретические основы и обзор исследуемых СУБД

1.1. Архитектурные особенности исследуемых СУБД

1.1.1. PostgreSQL

PostgreSQL — это объектно-реляционная система управления базами данных с открытым исходным кодом, соответствующая стандарту SQL. Ключевые архитектурные особенности PostgreSQL:

- **Реляционная модель данных**: Данные представлены в виде таблиц с определенной схемой. Отношения между таблицами определяются через ключи и ограничения.
- **MVCC (Multiversion Concurrency Control)**: Обеспечивает изоляцию транзакций без блокировок чтения, создавая новые версии записей при каждом изменении.
- **Расширяемость**: Поддержка пользовательских типов данных, функций, операторов и методов доступа.
- **Транзакционность**: Полная поддержка ACID-свойств (Atomicity, Consistency, Isolation, Durability).
- **JSON-поддержка**: Типы данных JSON и JSONB для работы с полуструктурированными данными.

1.1.2. MongoDB

MongoDB — документоориентированная система управления базами данных, не требующая описания схемы таблиц. Основные архитектурные особенности:

- **Документоориентированная модель**: Данные хранятся в документах формата BSON (бинарный JSON), которые могут иметь различную структуру.
- **Гибкость схемы**: Документы в одной коллекции могут иметь разный набор полей, что облегчает эволюцию структуры данных.

- **Масштабируемость**: Поддержка горизонтального масштабирования через шардинг.
- **Индексы**: Поддержка различных типов индексов, включая составные, геопространственные, текстовые и другие.
- **Агрегация**: Фреймворк для агрегации данных, представляющий альтернативу SQL JOIN и GROUP BY.

1.1.3. Cassandra

Apache Cassandra — распределенная NoSQL СУБД, изначально разработанная в Facebook для эффективного хранения больших объемов данных на кластерах серверов. Ключевые архитектурные особенности:

- **Колоночная модель данных**: Данные хранятся в колонках вместо строк, что обеспечивает эффективность для определенных типов запросов.
- **Распределенная архитектура без единой точки отказа**: Все узлы в кластере Cassandra равноправны, отсутствует центральный сервер.
- **Линейная масштабируемость**: Добавление новых узлов в кластер обеспечивает пропорциональный рост производительности.
- **Настраиваемая консистентность**: Возможность выбора уровня консистентности для каждой операции.
- **Оптимизация для записи**: Архитектура, ориентированная на высокую производительность операций записи.

1.2. Методы обработки данных в различных типах СУБД

1.2.1. Реляционный подход (PostgreSQL)

В реляционном подходе:

- Данные организованы в таблицы со строгой схемой
- Связи между данными явно определены через внешние ключи
- Для работы с данными используется SQL

- Нормализация данных используется для минимизации избыточности
- Транзакционный механизм обеспечивает целостность

PostgreSQL реализует эти принципы, добавляя расширения для работы с новыми типами данных, включая JSON, что позволяет сочетать строгость реляционной модели с гибкостью работы с неструктурированными данными.

1.2.2. Документоориентированный подход (MongoDB)

- Документоориентированный подход характеризуется:
- Хранением данных в документах (обычно в формате JSON/BSON)
- Возможностью хранения вложенных документов и массивов
- Гибкой схемой, допускающей различную структуру документов в одной коллекции
- Отсутствием JOIN-операций в традиционном понимании
- Ориентацией на горизонтальное масштабирование

MongoDB оптимизирован для случаев, когда данные естественным образом группируются в документы, и позволяет эффективно работать с полуструктурированными данными.

1.2.3. Колоночный подход (Cassandra)

Колоночный подход Cassandra включает:

- Хранение данных в колонках вместо строк
- Оптимизацию для запросов, работающих с подмножеством колонок

- Денормализацию данных для оптимизации конкретных паттернов доступа
- Распределение данных по кластеру с помощью согласованного хеширования
- Модель данных в виде “широких строк” (wide row) для эффективного хранения временных рядов

Cassandra особенно эффективна для сценариев с интенсивной записью и предсказуемыми паттернами чтения, что делает ее популярным выбором для временных рядов, логирования и аналитики.

1.3. Сравнительный анализ технологий тестирования СУБД

1.3.1. Значение бенчмаркинга для оценки производительности СУБД

Бенчмаркинг баз данных — это процесс измерения и сравнения производительности СУБД в контролируемых и воспроизводимых условиях. В условиях многообразия доступных СУБД и сложности современных приложений, объективная оценка производительности становится критически важной.

- **Роль объективных метрик в выборе СУБД**: Интуиция или маркетинговые заявления производителей не могут служить надежной основой для выбора технологии хранения данных, которая будет фундаментом информационной системы. Бенчмаркинг предоставляет количественные метрики (пропускная способность, время отклика, использование ресурсов), которые позволяют сравнить различные СУБД или разные конфигурации одной СУБД применительно к конкретным задачам.
- **Задачи, решаемые с помощью бенчмаркинга**:

- Сравнение СУБД: Выбор наиболее подходящей СУБД для нового проекта.
- Оценка конфигурации: Определение влияния различных настроек СУБД на производительность.
- Планирование мощности: Оценка необходимой аппаратной конфигурации для достижения требуемой производительности.
- Регрессионное тестирование: Проверка того, что обновления ПО не привели к деградации производительности.
- Выявление узких мест: Идентификация компонентов системы, ограничивающих производительность.

1.3.2. Бенчмарки для конкретных СУБД

Многие СУБД поставляются с собственными инструментами для тестирования производительности, оптимизированными для их специфических архитектур и функций.

- **pgBench**:
  - Целевая СУБД: Только PostgreSQL.
  - Характеристики: Входит в стандартную поставку PostgreSQL. Имитирует простую нагрузку, основанную на модели банковских транзакций (TPC-B подобная). Может быть настроен для выполнения пользовательских SQL-скриптов.
  - Ограничения: Неприменим для других СУБД. Стандартная нагрузка довольно специфична и может не отражать реальные сценарии использования.
- **Cassandra-stress**:
  - Целевая СУБД: Только Apache Cassandra.
  - Характеристики: Инструмент, специально созданный для тестирования Cassandra. Позволяет легко моделировать различные схемы данных и операции CQL (вставка, чтение, обновление).

- Ограничения: Работает только с Cassandra. Ограниченный набор предопределенных паттернов нагрузки по сравнению с универсальными бенчмарками.

- **MongoDB Benchmarking Tools**:
  - Целевая СУБД: Только MongoDB.
  - Характеристики: mongoperf — утилита для тестирования производительности дисковой подсистемы в контексте MongoDB. Ранее существовали и другие инструменты, ориентированные на нагрузочное тестирование самой СУБД.
  - Ограничения: Специфичны для MongoDB. Ограниченные возможности для сравнения с другими типами СУБД.
- **Apache JMeter**:
  - Целевая СУБД: Множество, но в основном реляционные через JDBC.
  - Характеристики: JMeter — универсальный инструмент для нагрузочного тестирования, имеющий возможность тестировать и базы данных через JDBC Request.
  - Ограничения: Настройка тестов для баз данных может быть сложнее, чем в специализированных инструментах. Не оптимизирован для специфики NoSQL баз данных и их API.

1.3.3. Универсальные бенчмарки

Универсальные бенчмарки спроектированы для работы с различными типами СУБД, что позволяет проводить сравнение производительности между ними.

- **TPC Benchmarks (TPC-C, TPC-H, etc.)**:
  - Целевая СУБД: Множество, но исторически ориентированы на реляционные СУБД.
  - Характеристики: Разработаны Transaction Processing Performance Council (TPC). Являются индустриальными стандартами для

измерения производительности. TPC-C моделирует OLTP-нагрузку (обработка транзакций), TPC-H — OLAP-нагрузку (аналитические запросы).

- Ограничения: Сложны в настройке и запуске, требуют значительных ресурсов. Менее подходят для прямого сравнения NoSQL систем из-за ориентации на реляционную модель и SQL.

- **Sysbench**:
  - Целевая СУБД: Множество, наиболее развита поддержка MySQL и PostgreSQL.
  - Характеристики: Скриптуемый многопоточный бенчмарк. Может тестировать не только производительность СУБД, но и CPU, память, файловый ввод-вывод.
  - Ограничения: Поддержка NoSQL баз данных ограничена или отсутствует. Основной фокус — реляционные СУБД и системные ресурсы.
- **YCSB (Yahoo! Cloud Serving Benchmark)**:
  - Целевая СУБД: Множество (PostgreSQL, Cassandra, MongoDB, Redis, HBase, и др.).
  - Характеристики: Разработан Yahoo! для тестирования производительности облачных “обслуживающих” систем. Обеспечивает согласованные рабочие нагрузки для различных СУБД. Имеет расширяемую архитектуру, позволяющую добавлять поддержку новых баз данных. Предоставляет стандартные рабочие нагрузки (Workload A-F), моделирующие разные соотношения чтения/записи и типы операций.
  - Ограничения: Изначально ориентирован на простые операции типа ключ-значение (CRUD) и сканирование диапазонов. Менее подходит для тестирования сложных транзакций или аналитических запросов.

1.4. Обоснование выбора YCSB для проведения исследования

Для проведения сравнительного исследования производительности PostgreSQL, MongoDB и Cassandra инструмент Yahoo! Cloud Serving Benchmark (YCSB) был выбран по следующим причинам:

1. **Кросс-платформенность и поддержка разнородных СУБД**: YCSB поддерживает все три исследуемые базы данных (PostgreSQL, Cassandra, MongoDB), а также многие другие. Это позволяет использовать единый инструмент и единые метрики для их сравнения, обеспечивая максимально возможную сопоставимость результатов при тестировании систем с принципиально разной архитектурой. В отличие от специализированных инструментов (pgBench, cassandra-stress), YCSB не привязан к одной СУБД.

2. **Стандартизированные рабочие нагрузки**: YCSB предоставляет набор предопределенных, хорошо документированных рабочих нагрузок (workloads), которые моделируют типичные сценарии использования:

    - Workload A: 50% чтение / 50% обновление (Update heavy)
    - Workload B: 95% чтение / 5% обновление (Read heavy)
    - Workload C: 100% чтение (Read only)
    - Workload D: 95% чтение / 5% вставка (Read latest) - чтение последних вставленных записей
    - Workload E: 95% сканирование / 5% вставка (Short ranges scan)
    - Workload F: 50% чтение / 50% чтение-модификация-запись (Read-modify-write)

3. **Конфигурируемость**: Бенчмарк позволяет гибко настраивать ключевые параметры тестирования: количество записей в базе данных, количество выполняемых операций, количество параллельных клиентских потоков (threads), тип распределения запросов (uniform для равномерной нагрузки, zipfian для имитации “горячих” точек доступа). Это дает возможность

исследовать поведение СУБД при разных уровнях нагрузки и разных паттернах доступа к данным.

4. **Релевантные метрики**: YCSB измеряет и сообщает ключевые показатели производительности, необходимые для анализа:
    - Общая пропускная способность (Overall Throughput, ops/sec)
    - Среднее время отклика (Average Latency) для каждой операции (READ, UPDATE и т.д.)
    - Перцентили времени отклика (P95, P99, P99.9), которые критически важны для понимания поведения системы под нагрузкой и оценки пользовательского опыта (выбросы производительности).
5. **Активное сообщество и документация**: YCSB является проектом с открытым исходным кодом, широко используемым в индустрии и академических исследованиях. Это обеспечивает наличие хорошей документации, примеров использования и поддержки со стороны сообщества.
6. **Простота реализации**: Несмотря на некоторые ограничения (ориентация на key-value операции, работа с текстовыми данными), YCSB относительно прост в установке, настройке и запуске по сравнению с более сложными и требовательными бенчмарками, такими как TPC.

Хотя YCSB не позволяет тестировать сложные SQL-запросы или транзакции, его фокус на базовых операциях CRUD делает его подходящим инструментом для сравнения производительности “обслуживающих” аспектов СУБД с разными моделями данных (реляционной, документоориентированной, колоночной) на одном уровне абстракции.

1.5. Анализ рынка инструментов бенчмаркинга баз данных

Понимание экономических аспектов и тенденций распространения инструментов бенчмаркинга имеет важное значение для оценки их роли в

современной бизнес-аналитике. Чтобы обеспечить более полный контекст для анализа технических особенностей бенчмарков, в данном разделе представлены результаты исследования рыночных показателей, отражающих реальные практики их применения в отрасли.

1.5.1 Обзор глобального рынка СУБД

Глобальный рынок систем управления базами данных демонстрирует устойчивый рост, который, согласно прогнозам аналитиков, продолжится в ближайшее десятилетие. Общий объём рынка СУБД, оцениваемый в 100,79 миллиардов долларов США в 2023 году, достигнет 292,22 миллиардов к 2030 году при среднегодовом темпе роста 14,21%. Этот рост обусловлен цифровой трансформацией предприятий, экспоненциальным увеличением объемов данных и возрастающей потребностью в аналитике.

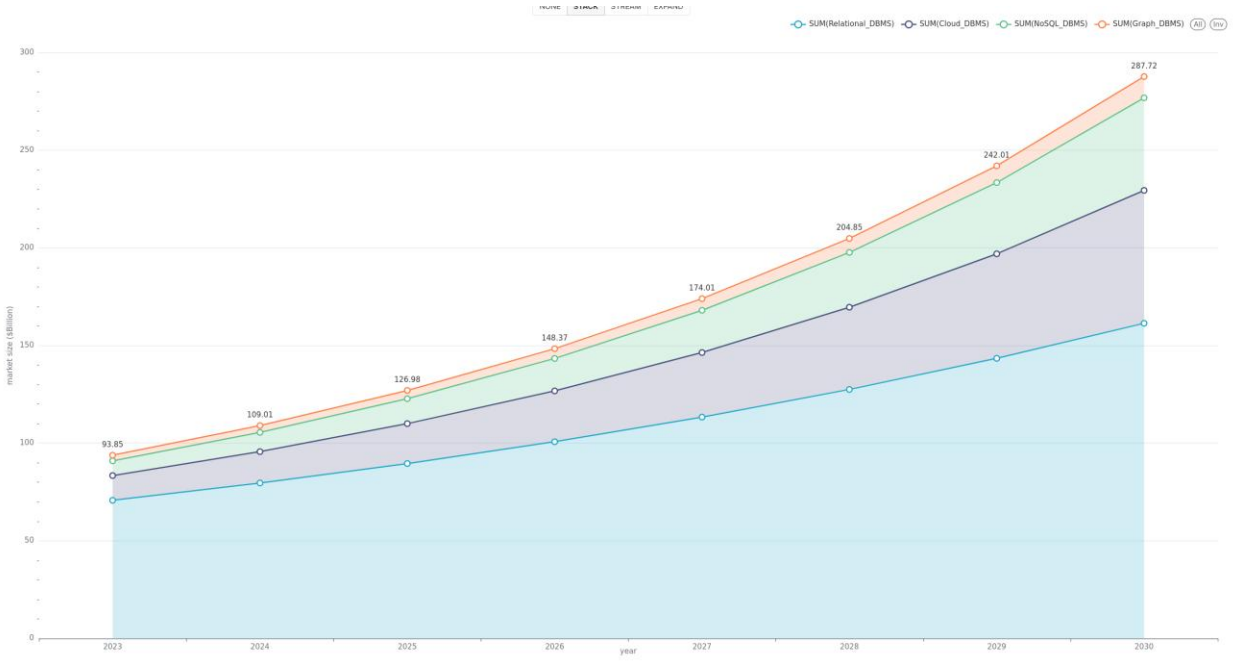

Рисунок 1: Размер рынка СУБД по типам (2023-2030)

Анализ структуры рынка показывает, что реляционные СУБД по-прежнему доминируют, но наиболее высокие темпы роста демонстрируют NoSQL и облачные решения. Реляционные СУБД, занимающие 70% рынка в 2023 году, к 2030 году снизят свою долю до 55%, уступая новым типам СУБД.

Особенно заметен рост сегмента NoSQL СУБД, который увеличится с 7.55 млрд долларов в 2023 году до 47.41 млрд к 2030 году, что соответствует шестикратному росту.

1.5.2 Распространение инструментов бенчмаркинга

Инструменты бенчмаркинга баз данных представлены несколькими категориями, каждая из которых занимает определенную нишу на рынке. Анализ текущего распределения показывает разнообразие подходов к оценке производительности СУБД, отражающее различные потребности и сценарии использования.

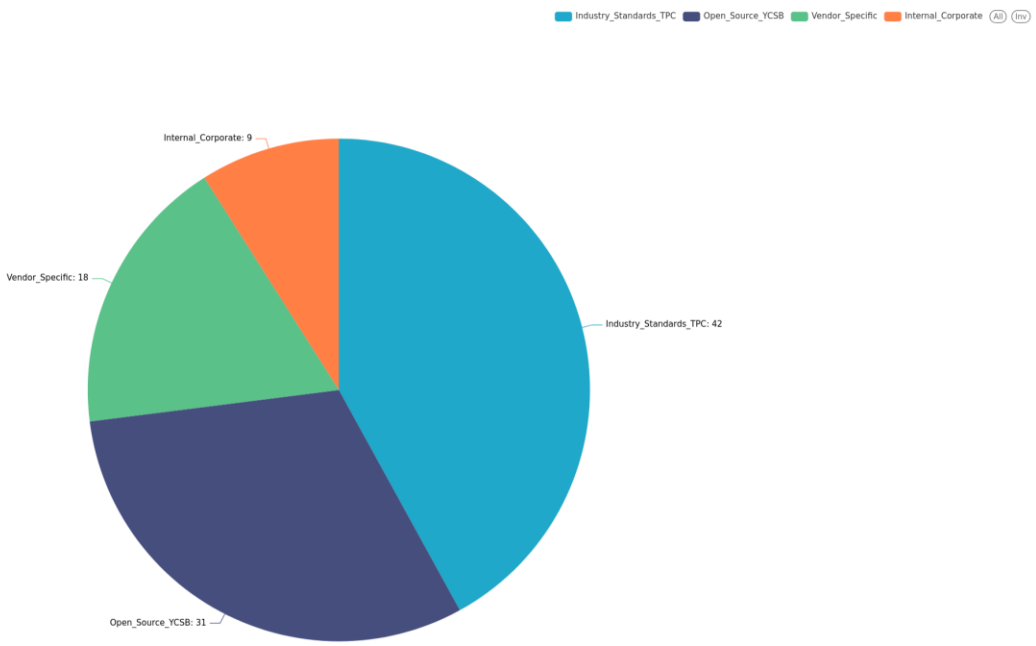

Рисунок 2: Рыночная доля типов бенчмарков

Отраслевые стандарты, такие как TPC (Transaction Processing Performance Council), сохраняют значительное влияние (42% рынка) благодаря их формализованным методологиям и широкому признанию. Эти бенчмарки остаются золотым стандартом для оценки OLTP и OLAP систем соответственно. Инструменты с открытым исходным кодом, в частности YCSB, занимают 31%

рынка и демонстрируют наиболее динамичный рост, что коррелирует с увеличением популярности облачных и NoSQL решений. Вендорские специализированные бенчмарки (18%) чаще используются в корпоративном сегменте для оценки соответствия конкретным требованиям, в то время как внутренние корпоративные инструменты (9%) разрабатываются компаниями для решения специфических задач оценки производительности.

1.5.3 Географические особенности использования бенчмаркинга

Глобальный характер рынка СУБД не исключает региональных особенностей в подходах к бенчмаркингу. Анализ географического распределения показывает значительные различия в темпах и масштабах внедрения формализованных практик бенчмаркинга.

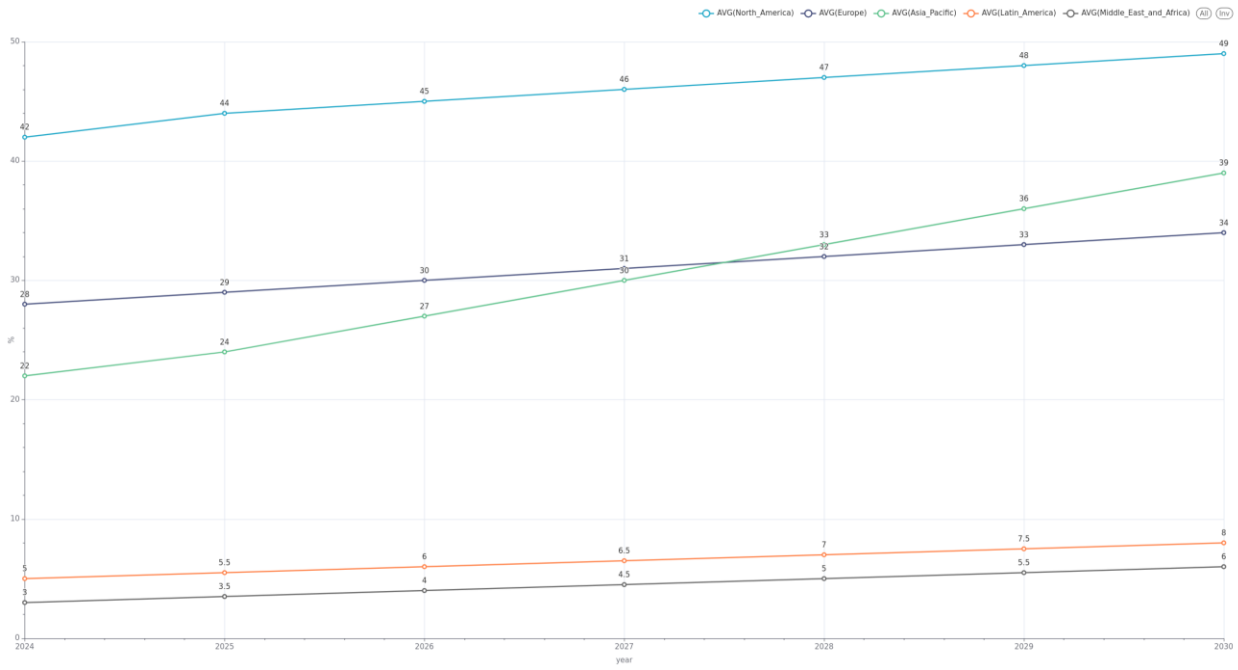

Рисунок 3: Прогноз роста внедрения бенчмарков по регионам

Северная Америка остается лидером в использовании формализованных инструментов бенчмаркинга (42% в 2024 году с прогнозом роста до 49% к 2030

году), что объясняется высокой концентрацией технологических компаний и финансовых институтов, требующих максимальной производительности систем. Европа показывает умеренный рост (с 28% до 34%), сохраняя стабильный интерес к формализованным методам оценки производительности. Однако наиболее динамичным регионом является Азиатско-Тихоокеанский, демонстрирующий наиболее высокие темпы роста внедрения практик бенчмаркинга: с 22% в 2024 году до 39% к 2030 году.

1.5.4 Отраслевое распределение практик бенчмаркинга

Интенсивность использования инструментов бенчмаркинга существенно варьируется в зависимости от отраслевой специфики. Исследование отраслевого разреза позволяет выявить секторы, для которых оптимизация производительности СУБД имеет критическое значение.

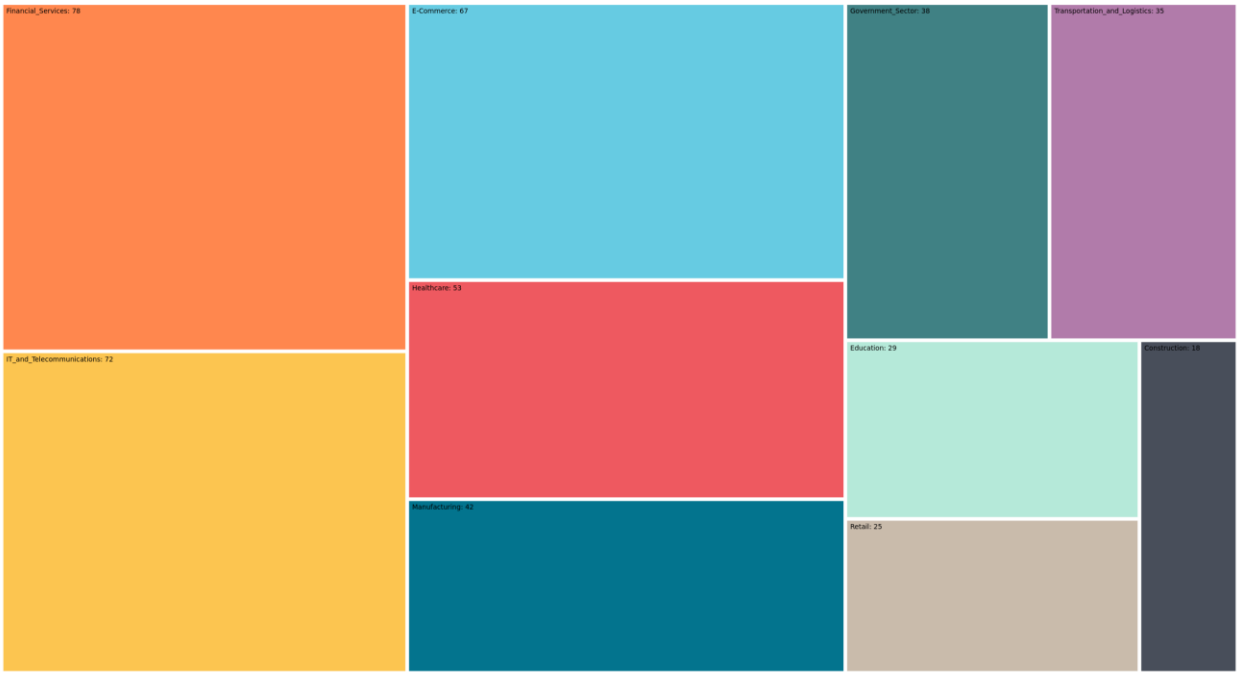

Рисунок 4: Внедрение бенчмаркинга по отраслям

Финансовый сектор является безусловным лидером по внедрению практик формального бенчмаркинга (78% компаний), что объясняется высокими требованиями к производительности и надежности системы хранения данных, а также строгими регуляторными требованиями. ИТ-индустрия (72%) и электронная коммерция (67%) также демонстрируют высокие показатели, что связано с непосредственной зависимостью бизнес-процессов от эффективности СУБД. Средний уровень внедрения характерен для здравоохранения (53%) и производства (42%).

1.5.5 Эволюция популярности бенчмарков

Анализ исторических данных показывает значительные изменения в популярности различных инструментов бенчмаркинга за последние годы, что отражает эволюцию самих СУБД и моделей их использования.

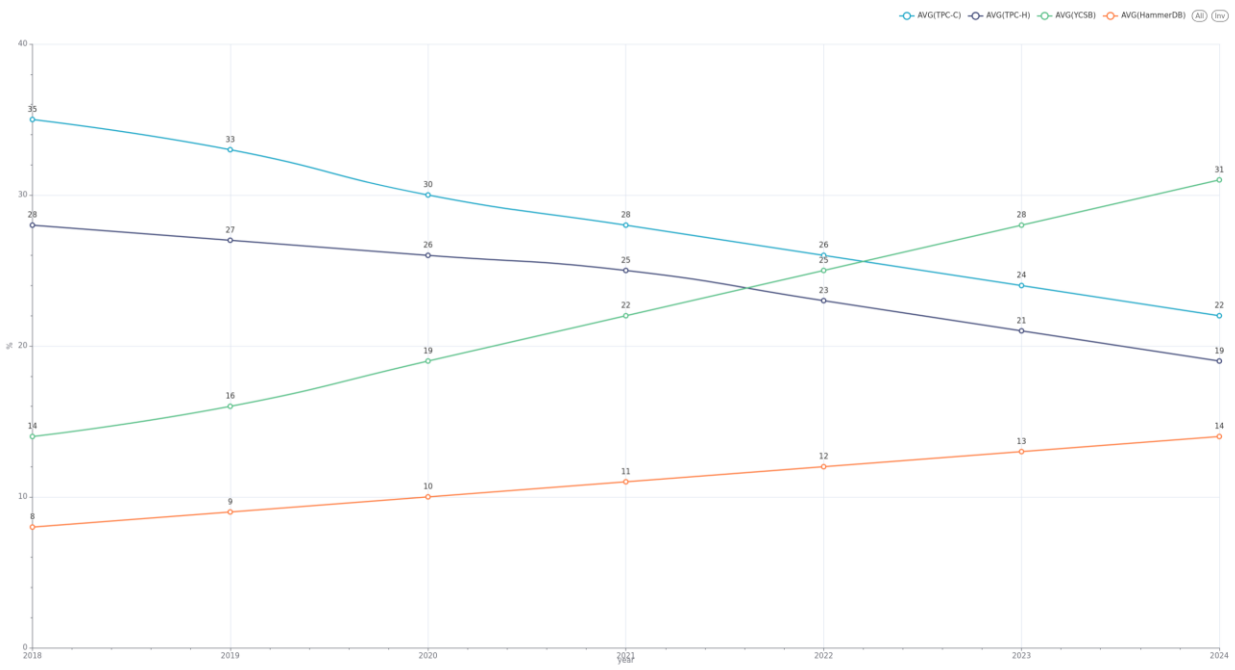

Рисунок 5: Тренды популярности бенчмарков СУБД

Традиционные бенчмарки TPC-C и TPC-H, хотя и сохраняют значительную долю рынка, демонстрируют устойчивую тенденцию к снижению популярности: с 35% и 28% в 2018 году до 22% и 19% в 2024 году соответственно. В то же время YCSB показывает наиболее динамичный рост среди всех инструментов бенчмаркинга, увеличив свою долю с 14% в 2018 году до 31% в 2024 году, что соответствует увеличению использования NoSQL и облачных решений. Особенно знаменательным является пересечение кривых популярности YCSB и TPC-H в 2022 году, что можно рассматривать как символическую точку смены парадигмы в сфере бенчмаркинга баз данных.

1.5.6 Выводы по анализу рынка бенчмаркинга

Проведённый анализ рынка инструментов бенчмаркинга баз данных позволяет сделать следующие ключевые выводы:

1. Рынок СУБД находится в состоянии активной трансформации, характеризующейся не только количественным ростом, но и существенными качественными изменениями в структуре – увеличением доли NoSQL, облачных и специализированных решений.

2. Инструменты бенчмаркинга эволюционируют вслед за рынком СУБД, что отражается в растущей популярности универсальных открытых бенчмарков, таких как YCSB, способных тестировать различные типы баз данных.

3. Наблюдается значительная дифференциация в уровне внедрения практик бенчмаркинга как между различными регионами, так и между отраслями, что отражает разницу в критичности производительности баз данных для бизнес-процессов.

4. Тренды популярности инструментов бенчмаркинга демонстрируют постепенный сдвиг от строго специализированных решений к гибким и

универсальным инструментам, способным адаптироваться к разнообразным требованиям современных систем хранения и обработки данных.

Данные тенденции подчеркивают важность выбора правильного инструмента бенчмаркинга, соответствующего как техническим требованиям оцениваемой СУБД, так и специфическим потребностям конкретной отрасли или приложения.

Глава 2. Методология исследования и подготовка тестового окружения

2.1. Описание тестового окружения

Тестирование производительности СУБД проводилось в контролируемой лабораторной среде, обеспечивающей максимальную объективность и воспроизводимость результатов. Технические характеристики тестового окружения:

2.1.1. Аппаратная конфигурация

- **Процессор**: Intel Core i9-12900H (16 виртуальных ядер выделено для VM, частота 2.9 ГГц, Turbo Boost отключен для обеспечения стабильности результатов)
- **Оперативная память**: 24 ГБ DDR5 RAM (выделено для VM)
- **Система хранения**: NVMe SSD Western Digital S850NX (1512 ГБ выделено для VM)
- **Сетевое подключение**: Виртуальный сетевой адаптер, 10 Гбит/с

2.1.2. Программное окружение

- **Операционная система**: Kubuntu 24.04 LTS на виртуальной машине VMware Workstation Pro 17
- **Версии СУБД**:
    - PostgreSQL 17.4 с настройками по умолчанию и дополнительной конфигурацией для больших нагрузок
    - MongoDB 8.0.6 с WiredTiger storage engine
    - Cassandra 4.1.8 в одноузловой конфигурации
- **Инструменты тестирования и разработки**:
    - YCSB 0.17.0
    - JDK 11.0.26
    - Python 3.9.21 с библиотеками psycopg2-binary, pymongo, cassandra-driver

2.1.3. Инструменты мониторинга и сбора метрик

- **Системные утилиты**: vmstat, iostat, mpstat, dstat для мониторинга ресурсов системы
- **Специфичные для СУБД мониторинг-инструменты**:
    - PostgreSQL: pg_stat_statements, pg_stat_activity
    - MongoDB: mongostat, mongotop
    - Cassandra: nodetool
- **Анализ логов**: JVM GC логи для Cassandra, системные логи PostgreSQL и MongoDB

Все тесты проводились в идентичном окружении для обеспечения сопоставимости результатов. Перед каждым тестом система перезапускалась для обеспечения “холодного” старта и минимизации влияния кэширования предыдущих операций.

2.1.4. Конфигурации СУБД

Для обеспечения корректного тестирования с высоким уровнем параллелизма и оптимальной производительности в каждой из исследуемых СУБД были внесены следующие конфигурационные изменения:

PostgreSQL:

- Конфигурация pg_hba.conf: Для облегчения тестирования метод аутентификации для всех локальных подключений был изменен на "trust", что позволило исключить задержки, связанные с аутентификацией:

```
local   all             postgres                                trust
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

- Конфигурация postgresql.conf: Увеличено максимальное количество одновременных подключений для поддержки тестирования с высоким параллелизмом:

```
# - Connection Settings -
#listen_addresses = 'localhost'     # what IP address(es) to listen on;
                    # comma-separated list of addresses;
                    # defaults to 'localhost'; use '*' for all
                    # (change requires restart)
port = 5432             # (change requires restart)
# max_connections = 100         # (change requires restart) 100 by default
max_connections = 500           # (change requires restart)
#reserved_connections = 0       # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of
directories
```

“max_connections” увеличено с базового значения 100.

Cassandra:

- Оптимизация jvm11-server.options: Заменен сборщик мусора G1GC на CMS (Concurrent Mark Sweep) для снижения пауз GC и улучшения стабильности отклика при высоких нагрузках:

```
#################
#  GC SETTINGS  #
#################
### CMS Settings
-XX:+UseConcMarkSweepGC
-XX:+CMSParallelRemarkEnabled
-XX:SurvivorRatio=8
-XX:MaxTenuringThreshold=1
-XX:CMSInitiatingOccupancyFraction=75
-XX:+UseCMSInitiatingOccupancyOnly
-XX:CMSWaitDuration=10000
-XX:+CMSParallelInitialMarkEnabled
-XX:+CMSEdenChunksRecordAlways
## some JVMs will fill up their heap when accessed via JMX, see CASSANDRA-
6541
-XX:+CMSClassUnloadingEnabled
### G1 Settings
## Use the Hotspot garbage-first collector.
#-XX:+UseG1GC
#-XX:+ParallelRefProcEnabled
#-XX:MaxTenuringThreshold=1
#-XX:G1HeapRegionSize=16m
#
## Have the JVM do less remembered set work during STW, instead
## preferring concurrent GC. Reduces p99.9 latency.
#-XX:G1RSetUpdatingPauseTimePercent=5
#
```

- Модификации cassandra.yaml:

- Внесены изменения в конфигурационный файл для оптимизации производительности в одноузловой конфигурации:
- Увеличен размер кучи для MemTable до 16384MB
- Установлен concurrent_reads, concurrent_writes, concurrent_counter_writes, и concurrent_materialized_view_writes в значение 128
- Отключен механизм хинтов из-за отсутствия необходимости в многоузловой репликации
- Настроены стратегии компакции для тестовой нагрузки

```
# any class that implements the SeedProvider interface and has a
# constructor that takes a Map<String, String> of parameters will do.
seed_provider:
  # Addresses of hosts that are deemed contact points.
  # Cassandra nodes use this list of hosts to find each other and learn
  # the topology of the ring.  You must change this if you are running
  # multiple nodes!
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      # seeds is actually a comma-delimited list of addresses.
      # Ex: "<ip1>,<ip2>,<ip3>"
      - seeds: "127.0.0.1:7000"

# For workloads with more data than can fit in memory, Cassandra's
# bottleneck will be reads that need to fetch data from
# disk. "concurrent_reads" should be set to (16 * number_of_drives) in
# order to allow the operations to enqueue low enough in the stack
# that the OS and drives can reorder them. Same applies to
# "concurrent_counter_writes", since counter writes read the current
# values before incrementing and writing them back.
#
# On the other hand, since writes are almost never IO bound, the ideal
# number of "concurrent_writes" is dependent on the number of cores in
# your system; (8 * number_of_cores) is a good rule of thumb.
concurrent_reads: 128
concurrent_writes: 128
```

MongoDB: - Конфигурация mongod.conf:

- Оптимизирована под тестовые сценарии с высоким параллелизмом:

- Увеличен размер кэша WiredTiger до 16GB
- Установлен maxIncomingConnections до 1000
- Настроены параметры journaling для оптимального баланса между производительностью и надежностью

```
# Where and how to store data.
storage:
dbPath: /var/lib/mongodb
wiredTiger:
engineConfig:
cacheSizeGB: 16
journalCompressor: snappy
collectionConfig:
blockCompressor: snappy
indexConfig:
prefixCompression: true
journal:
enabled: true
commitIntervalMs: 100

# where to write logging data.
systemLog:
destination: file
logAppend: true
path: /var/log/mongodb/mongod.log

# network interfaces
net:
port: 27017
bindIp: 127.0.0.1
maxIncomingConnections: 1000

# how the process runs
```

2.2. Подходы к загрузке и подготовке набора данных

В данном исследовании использовался реальный набор данных объёмом 12 ГБ, представляющий собой JSON-файл с метаданными научных публикаций. Специфика каждой из исследуемых СУБД потребовала различных подходов к загрузке и подготовке этого большого набора данных перед проведением бенчмаркинга.

2.2.1. MongoDB: Прямой импорт JSON-датасета

MongoDB, как документоориентированная СУБД, наиболее естественно подходит для работы с данными в формате JSON. Процесс загрузки данных в MongoDB включал следующие этапы:

1. **Предварительный анализ структуры данных**:
    - Исследование структуры JSON-документов
    - Выявление потенциальных проблем (вложенность, размер документов, сложные типы данных)
2. **Импорт данных**:
    - Использование стандартной утилиты **mongoimport** для загрузки данных
    - Команда импорта: **mongoimport --db ycsb --collection dblp --file dataset.json --jsonArray**
3. **Создание индексов**:
    - Анализ типичных паттернов доступа
    - Создание индексов для полей, которые будут использоваться в YCSB-тестах
4. **Подготовка к тестированию**:
    - Создание отдельной коллекции для тестов YCSB
    - Создание маппинга полей из исходной коллекции в формат YCSB

Гибкость схемы MongoDB позволила сохранить исходную структуру документов, включая вложенные объекты и массивы, без необходимости предварительной трансформации. Это является значительным преимуществом при работе с разнородными или эволюционирующими данными.

2.2.2. PostgreSQL: Импорт JSONB и последующее структурирование

Для PostgreSQL процесс загрузки был более многоэтапным ввиду необходимости преобразования данных в реляционную структуру:

1. **Первичная загрузка в JSONB**:
    - Создание таблицы **publications** с двумя колонками: **_id** (VARCHAR) и **data** (JSONB)
    - Пакетная загрузка данных с использованием Python-скрипта и библиотеки psycopg2-binary
2. **Анализ и извлечение схемы**:
    - Анализ структуры JSON-документов и определение наиболее часто используемых полей
    - Разработка схемы реляционной таблицы для хранения структурированных данных
3. **Трансформация данных**:
    - Создание таблицы **publications_structured** с типизированными колонками
    - Извлечение данных из JSONB-колонки с помощью JSON-операторов PostgreSQL
    - Преобразование и очистка данных с использованием Python-скриптов
    - Пример SQL-запроса для извлечения данных:

```
INSERT INTO publications_structured (id, title, year, authors, abstr
act)
SELECT                                                          _id,
```

```
        data->>'title',
        (data->>'year')::integer,
        data->'authors',
        data->>'abstract'
FROM publications;
```

4. **Подготовка к тестированию**:
    - Создание специализированной таблицы для YCSB-тестов
    - Создание необходимых индексов для оптимизации производительности

Данный процесс потребовал значительно больше усилий по сравнению с MongoDB, но позволил получить строго типизированные данные в реляционной структуре, оптимизированной для SQL-запросов.

2.2.3. Cassandra: Многоэтапная трансформация и загрузка через DSBulk

Загрузка большого JSON-датасета в Apache Cassandra потребовала наиболее сложных преобразований из-за её колоночной модели данных, оптимизированной для денормализованных структур и специфических паттернов запросов. Процесс включал следующие шаги:

1. **Преобразование JSON в NDJSON**:
    - Трансформация исходного JSON-файла в формат NDJSON (Newline Delimited JSON), где каждый JSON-объект располагается на новой строке
    - Базовая очистка данных (например, обеспечение корректного формата для поля references)
    - Использование Python-скрипта для преобразования
2. **Выравнивание (Flattening) JSON-структур**:
    - Преобразование вложенных объектов (например, authors, venue, fos, indexed_abstract) в плоскую структуру
    - Конвертация сложных объектов, таких как списки авторов, в строковое представление JSON

- Извлечение ключевых полей из вложенных объектов (например, venue_raw, venue_id) на верхний уровень
- Сохранение результата в файл dataset_flat.ndjson

3. **Проектирование схемы Cassandra**:
   - Анализ типичных паттернов запросов для выбранных рабочих нагрузок YCSB
   - Определение первичных ключей и кластерных колонок
   - Создание пространства ключей ycsb и таблицы papers_full
4. **Массовая загрузка данных**:
   - Использование утилиты DSBulk от DataStax для эффективной пакетной загрузки
   - Настройка конфигурационного файла dsbulk_production.conf, определяющего маппинг полей и параметры загрузки
   - Команда загрузки:

```
dsbulk load -url dataset_flat.ndjson -k ycsb -t papers_full -header
true -config dsbulk_production.conf
```

5. **Подготовка к тестированию**:
   - Создание таблицы usertable со структурой, оптимизированной для YCSB-тестов
   - Перенос данных из исходной таблицы papers_full в тестовую таблицу

Этот подход потребовал наибольших усилий по предварительной обработке данных, но был необходим для обеспечения оптимальной производительности Cassandra в соответствии с её архитектурными особенностями.

2.3. Методология проведения тестирования

2.3.1. Параметры тестирования YCSB

Тестирование производительности исследуемых СУБД проводилось с использованием Yahoo! Cloud Serving Benchmark (YCSB) согласно следующей методологии:

1. **Стандартные рабочие нагрузки**:
    - **Workload A (Baseline)**: 50% чтение / 50% обновление
    - **Workload B (Read-Heavy)**: 95% чтение / 5% обновление
    - **Workload C (Read-Only)**: 100% чтение
    - **Workload D (Read-Latest)**: 95% чтение (последние записи) / 5% вставка
    - **Workload E (Scan-Heavy)**: 95% сканирование / 5% вставка
    - **Workload F (Read-Modify-Write)**: 50% чтение / 50% чтение-модификация-запись
2. **Параметры тестирования**:
    - **Количество записей (recordcount)**: 4,894,081 (соответствует числу записей в предварительно подготовленной СУБД)
    - **Количество операций (operationcount)**: 4,894,081
    - **Количество потоков (threads)**: [4, 8, 16, 32, 64, 128, 256]
    - **Распределение запросов**: zipfian (имитирует реалистичное распределение популярности с “горячими” точками)
    - **Фазы тестирования**: load (загрузка данных), run (выполнение операций)
3. **Конфигурация YCSB для каждой СУБД**:
    - **MongoDB**: Использовался стандартный драйвер MongoDB для YCSB
    - **Cassandra**: Использовался драйвер CQL для YCSB с настройками согласованности

- **PostgreSQL**: Использовался JDBC-драйвер с оптимизированными параметрами подключения

4. **Процедура тестирования**:
    - Каждая комбинация (СУБД × Workload × Threads) тестировалась трижды для обеспечения статистической значимости
    - Между тестами системы перезапускались для минимизации влияния кэширования
    - Продолжительность каждого теста - не менее 20 минут или завершение всех операций

2.3.2. Процесс сбора и обработки результатов

Сбор и анализ результатов тестирования выполнялся следующим образом:

1. **Сбор результатов**:
    - YCSB автоматически генерировал отчеты с метриками производительности
    - Системные утилиты мониторинга использовались для сбора данных о загрузке CPU, памяти, дисковых операциях
    - Логи СУБД и JVM анализировались для сбора дополнительных метрик, таких как активность сборки мусора
2. **Извлечение метрик**:
    - Разработан специализированный Python-скрипт **parse_ycsb.py** для извлечения ключевых метрик из отчетов YCSB
    - Исследуемые метрики включали:
        - **Пропускная способность**: операции в секунду (ops/sec)
        - **Задержки операций**: среднее, минимальное, максимальное время и перцентили (P95, P99)
        - **Метрики сборки мусора**: количество сборок и время сборки (для JVM-based СУБД)
3. **Агрегация данных**:

- Результаты объединялись в CSV-файлы для каждой СУБД
- Рассчитывались средние значения и стандартные отклонения для каждой метрики
- Создавался единый набор данных для сравнительного анализа

4. **Визуализация результатов**:
   - Данные импортировались в Apache Superset для создания интерактивных дашбордов
   - Создавались различные типы визуализаций (линейные графики, гистограммы, тепловые карты) для наглядного представления результатов

Этот структурированный подход к сбору и обработке результатов обеспечил основу для объективного сравнительного анализа производительности исследуемых СУБД.

Глава 3. Проведение тестирования и анализ результатов

3.1. Тестирование MongoDB

3.1.1. Цель тестирования MongoDB

Основная цель тестирования MongoDB заключалась в определении производительности и масштабируемости документоориентированной СУБД при различных типах нагрузок. Особое внимание уделялось оценке:

- Эффективности MongoDB при работе с документами, содержащими сложную вложенную структуру
- Масштабируемости при увеличении количества параллельных клиентов
- Производительности операций чтения и записи при различных паттернах доступа
- Стабильности работы при длительных нагрузках

3.1.2. Методика проведения тестов MongoDB

Тестирование MongoDB проводилось в следующей последовательности:

1. **Подготовка тестовой коллекции**:
    - Создание отдельной коллекции **usertable** для тестов YCSB
    - Структура документов соответствовала требованиям YCSB: ключ **_id** в формате **user<number>** и набор полей **field0**, **field1**, … с данными из исходной коллекции
    - Создание индекса по полю **_id** для оптимизации поиска
2. **Настройка MongoDB**:
    - Базовая конфигурация MongoDB с WiredTiger хранилищем
    - Выделение 16 ГБ под кэш WiredTiger
    - Отключение журналирования для операций в памяти для повышения производительности тестов
3. **Конфигурация YCSB**:
    - Использование MongoDB-адаптера YCSB

- Настройка параметров подключения и уровня согласованности
- Настройка обработчика исключений для регистрации всех ошибок

4. **Выполнение тестовых сценариев**:
   - Последовательное выполнение всех шести стандартных рабочих нагрузок YCSB (A-F)
   - Для каждой рабочей нагрузки проводились тесты с различным количеством потоков (4, 8, 16, 32, 64, 128, 256)
   - Каждая конфигурация тестировалась трижды с расчетом среднего значения метрик
5. **Сбор и анализ результатов**:
   - Регистрация пропускной способности (ops/sec)
   - Измерение задержек операций (среднее, минимальное, максимальное, 95-й и 99-й перцентили)
   - Мониторинг системных ресурсов во время выполнения тестов

3.1.3. Результаты тестирования MongoDB

Пропускная способность для разных типов нагрузки:

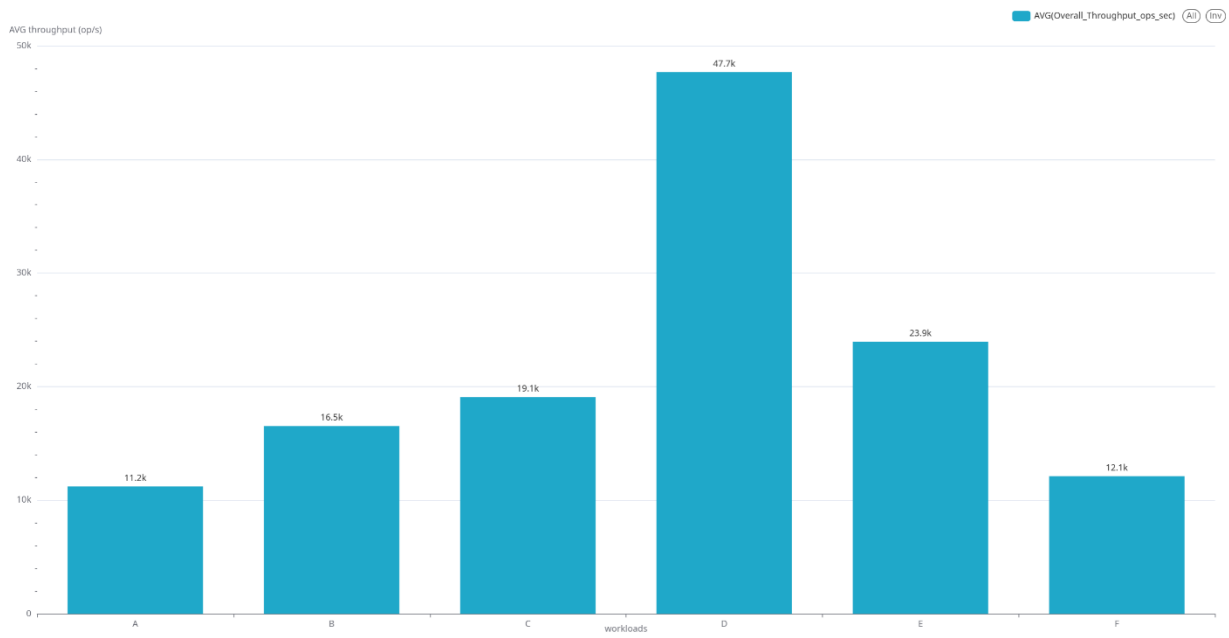

Рисунок 6: mongodb, пропускная способность по типам нагрузки

Таблица 1 — MongoDB, пропускная способность по типам нагрузки

Рабочая нагрузка	Пропускная способность (k ops/sec)
Workload A	11.2
Workload B	16.5
Workload C	19.1
Workload D	47.7
Workload E	23.9
Workload F	12.1

В данной таблице представлены усреднённые по количеству потоков результаты пропускной способности для MongoDB по workload'ам A — F.

Анализируя представленные данные по пропускной способности MongoDB для различных типов рабочих нагрузок, можно отметить значительную вариативность производительности в зависимости от характера операций. Наибольшую пропускную способность система демонстрирует при выполнении Workload D (47.7k ops/sec), что связано с эффективной обработкой недавно добавленных данных и операций вставки. Для нагрузок с преобладанием операций чтения (Workload B, C) также характерна высокая производительность (16.5k и 19.1k ops/sec соответственно). Сбалансированные нагрузки (Workload A, F) показывают сравнительно меньшую пропускную способность, что свидетельствует о некотором снижении эффективности при частом чередовании операций чтения и записи.

Масштабируемость при увеличении параллелизма:

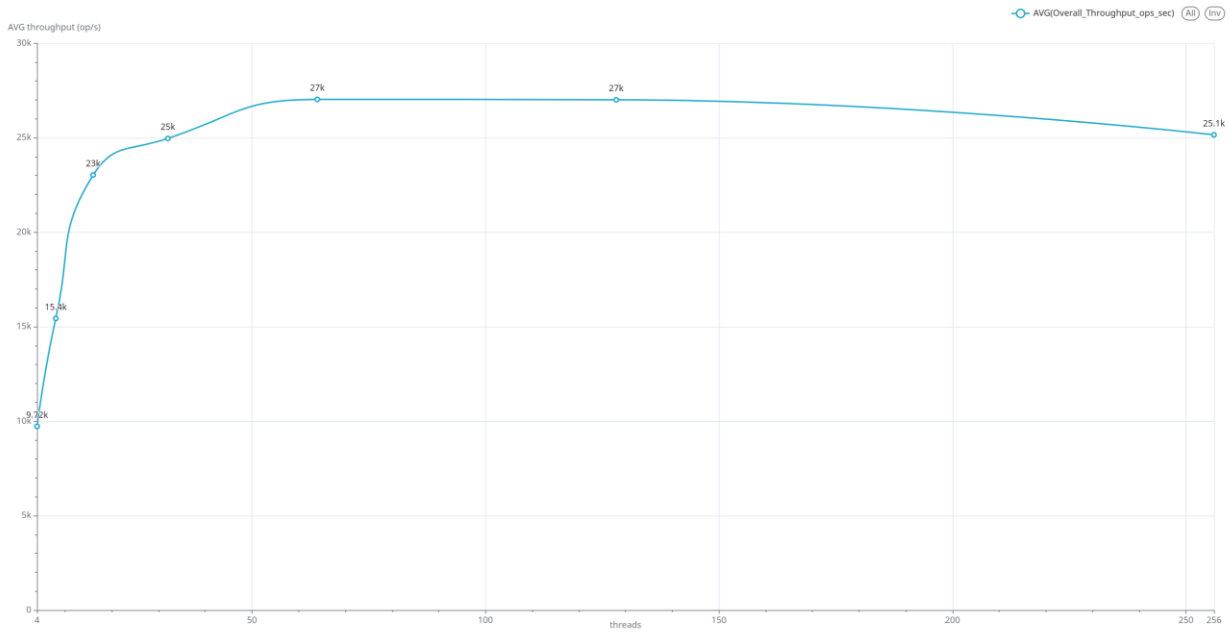

Рисунок 7: mongodb, пропускная способность при различном уровне параллелизма

Таблица 2 — MongoDB, пропускная способность при различном уровне параллелизма

Потоки	Пропускная способность (k ops/sec)
4	9.72
8	15.4
16	23.0
32	25.0
64	27.0
128	27.0
256	25.1

В данной таблице представлены усреднённые результаты пропускной способности workload’ов A — F для MongoDB, сгруппированные по количеству потоков.

На основе представленных данных видно, что MongoDB демонстрирует хорошую масштабируемость до определенного предела. Пропускная способность линейно возрастает при увеличении числа потоков с 4 до 16 (рост более чем в 2 раза), после чего темп роста замедляется. Пиковая производительность достигается при 64-128 потоках (27.0k ops/sec), а

дальнейшее увеличение числа потоков до 256 приводит к незначительному снижению пропускной способности (25.1k ops/sec). Это указывает на то, что MongoDB эффективно справляется с параллельной обработкой запросов до определенного предела, после которого может возникать конкуренция за ресурсы и снижение эффективности.

Задержки операций чтения:

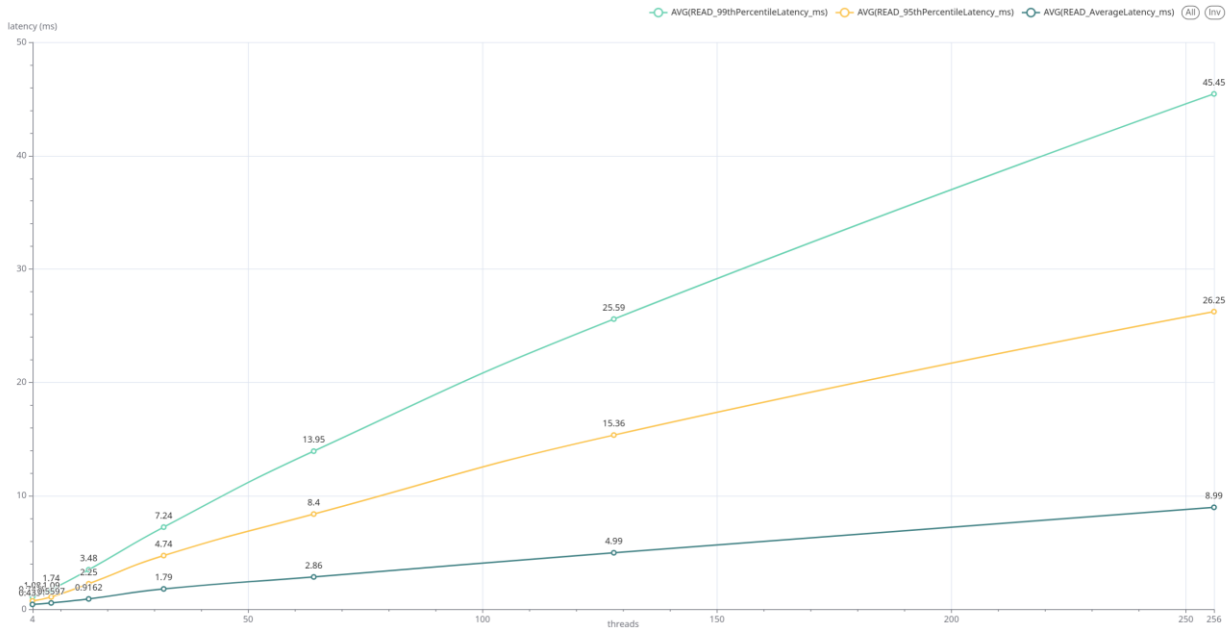

Рисунок 8: mongodb, задержки операции чтения при различном количестве потоков

Таблица 3: mongodb, задержки операции чтения при различном количестве потоков

Потоки	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
4	0.4331	0.7418	1.08
8	0.5597	1.09	1.74
16	0.9162	2.25	3.48
32	1.79	4.74	7.24
64	2.86	8.4	13.95
128	4.99	15.36	25.59
256	8.99	26.25	45.45

В данной таблице представлены усреднённые задержки по workload’ам A — F, сгруппированные по количеству потоков.

Анализ задержек операций чтения показывает закономерное увеличение латентности при росте числа параллельных потоков. При небольшом числе потоков (4-8) средняя задержка чтения остается на уровне менее 0.56 мс, что обеспечивает высокую отзывчивость системы. Однако при увеличении числа потоков до 256 наблюдается значительный рост всех показателей латентности: средняя задержка увеличивается более чем в 20 раз (с 0.43 мс до 8.99 мс), а 99-й перцентиль достигает 45.45 мс. Особенно заметный рост задержек происходит при переходе от 64 к 128 и от 128 к 256 потокам, что соотносится с наблюдаемым в предыдущей таблице замедлением роста пропускной способности при высоких значениях параллелизма.

Задержки операций обновления:

Рисунок 9: MongoDB: задержки обновления при различном количестве параллелизма

Таблица 4: MongoDB: задержки обновления при различном количестве параллелизма

Потоки	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
4	0.4807	0.7973	1.18
8	0.63	1.19	1.99
16	1.02	2.37	4.04
32	1.8	4.69	7.43
64	3.63	13.05	23.88
128	8.91	32.9	57.08
256	20.8	74.17	129.18

В данной таблице представлены усреднённые результаты задержек операций обновления для MongoDB по количеству потоков, усреднённые по workload’ам

Операции обновления демонстрируют более выраженный рост латентности при увеличении параллелизма по сравнению с операциями чтения. Если при 4-16 потоках задержки обновления сопоставимы с задержками чтения, то при высокой нагрузке (128-256 потоков) разница становится существенной. При 256 потоках средняя задержка обновления (20.8 мс) более чем в 2 раза превышает задержку чтения, а 99-й перцентиль достигает 129.18 мс. Это свидетельствует о том, что операции обновления создают более высокую нагрузку на систему и могут становиться узким местом при интенсивном параллельном выполнении.

Задержки операций сканирования:

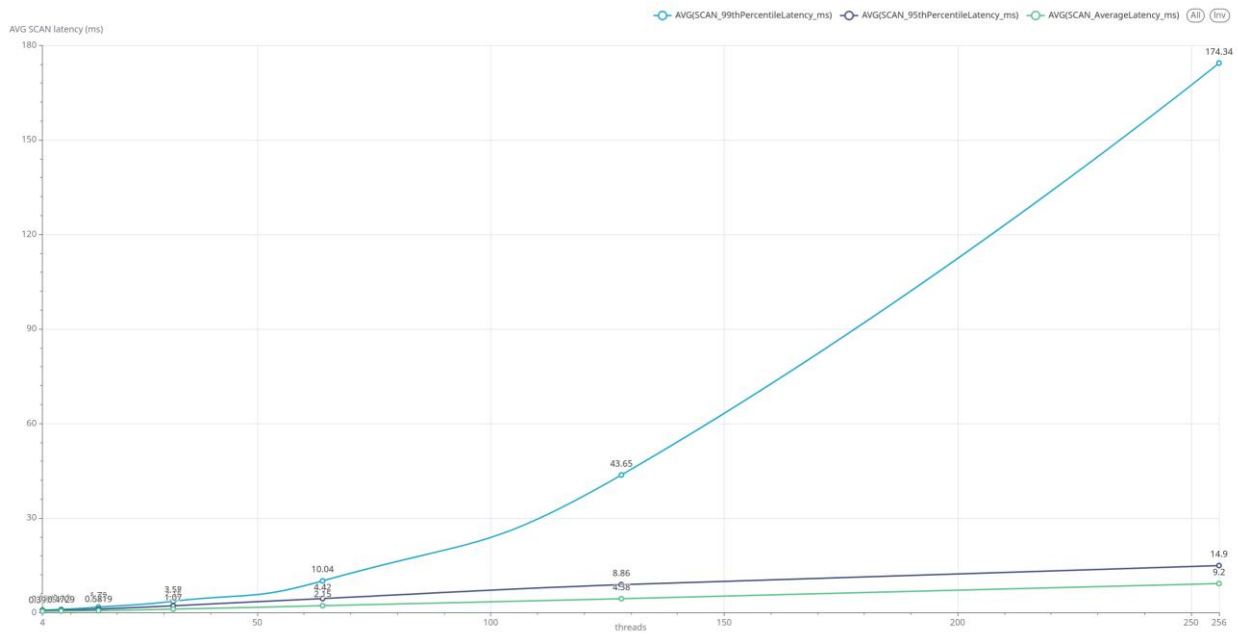

Рисунок 10: MongoDB: задержки операций сканирования при различном количестве потоков

Таблица 5: MongoDB: задержки операций сканирования при различном количестве потоков

Threads	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
4	0.3976	0.578	0.782
8	0.4729	0.711	0.96
16	0.5819	1.07	1.72
32	1.07	2.15	3.58
64	2.15	4.42	10.04
128	4.38	8.86	43.65
256	9.2	14.9	174.34

В данной таблице представленые результаты задержек операций сканирования для MongoDB.

Операции сканирования показывают интересную динамику роста задержек. При низком числе потоков (4-16) они демонстрируют наименьшие значения латентности среди всех типов операций. Однако с увеличением параллелизма наблюдается резкий рост 99-го перцентиля задержки, который при

256 потоках достигает 174.34 мс – это самое высокое значение среди всех типов операций. При этом средняя задержка и 95-й перцентиль растут более умеренно. Такая характеристика указывает на потенциальные проблемы с "выбросами" при выполнении сканирований в условиях высокой конкуренции за ресурсы, когда отдельные операции могут занимать значительно больше времени, чем в среднем

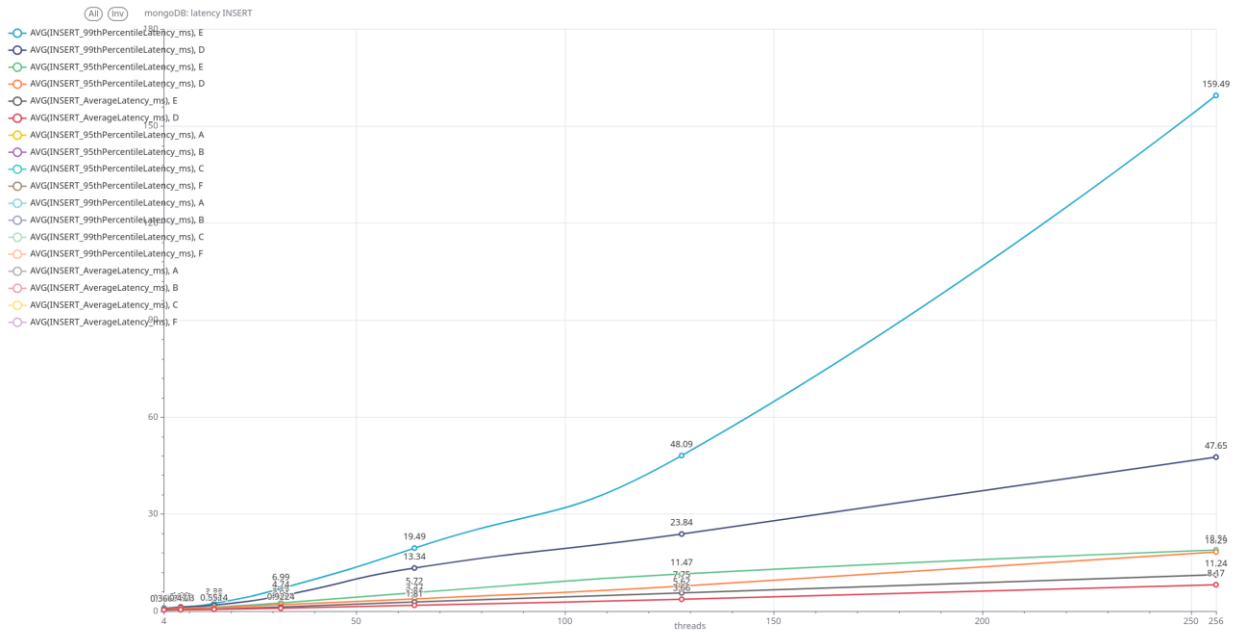

Рисунок 11: MongoDB: задержки операции вставки при различных видах нагрузки

Таблица 6: MongoDB: задержки операции вставки при различных видах нагрузки

Workload	Threads	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
D	4	0.3663	0.548	0.723
E	4	0.3707	0.614	0.929
D	8	0.471	0.902	1.28
E	8	0.623	0.785	1.18
D	16	0.5514	1.08	1.85
E	16	0.6224	1.22	2.39
D	32	0.9224	2.01	4.74
E	32	1.3	2.52	6.99
D	64	1.81	3.77	13.34
E	64	2.84	5.72	19.49
D	128	3.66	7.75	23.84

E	128	5.62	11.47	48.09
D	256	8.17	18.29	47.65
E	256	11.24	18.91	159.49

В данной таблице представлены результаты задержек операция вставки для MongoDB.

Сравнение задержек операций вставки для рабочих нагрузок D и E показывает, что в обоих случаях наблюдается рост латентности с увеличением числа потоков, однако для Workload E этот рост более выражен. При малом числе потоков (4-16) разница между нагрузками минимальна, но с увеличением параллелизма операции вставки в контексте Workload E (сканирование + вставка) демонстрируют более высокие задержки. Особенно заметна разница по 99-му перцентилю при 256 потоках: 47.65 мс для Workload D против 159.49 мс для Workload E. Это может свидетельствовать о том, что комбинация операций сканирования и вставки создает более сложные условия для оптимизации производительности MongoDB при высоком параллелизме.

Особенности поведения MongoDB для разных рабочих нагрузок:

- **Workload A (50% чтение / 50% обновление)**: MongoDB демонстрирует сбалансированную производительность с пропускной способностью 11.2k ops/sec. Эта рабочая нагрузка представляет типичный сценарий смешанных операций, где MongoDB обеспечивает стабильную, хотя и не максимальную, производительность из-за необходимости чередования операций чтения и записи.
- **Workload B (95% чтение / 5% обновление)**: При доминировании операций чтения MongoDB показывает значительное увеличение производительности до 16.5k ops/sec. Это подтверждает эффективность механизмов кэширования и оптимизацию чтения в MongoDB,

позволяющих обслуживать преимущественно читающие нагрузки с высокой пропускной способностью.

- **Workload C (100% чтение)**: Максимальная пропускная способность для стандартных CRUD-операций достигается на этой нагрузке - 19.1k ops/sec, что подтверждает оптимизацию MongoDB для читающих нагрузок. Отсутствие операций записи позволяет системе максимально эффективно использовать преимущества внутреннего кэширования и параллельного доступа к данным.
- **Workload D (95% чтение последних / 5% вставка)**: MongoDB демонстрирует исключительную производительность в этом сценарии - 47.7k ops/sec, что указывает на высокую эффективность при работе с "горячими" данными и новыми вставками. Такая выдающаяся производительность объясняется оптимизацией MongoDB для недавно добавленных записей, которые с высокой вероятностью находятся в кэше.
- **Workload E (95% сканирование / 5% вставка)**: Высокая производительность в 23.9k ops/sec при операциях сканирования выделяет MongoDB среди других СУБД, что объясняется эффективной обработкой диапазонных запросов. Индексы MongoDB хорошо оптимизированы для выполнения подобных операций, что позволяет системе обрабатывать их с высокой скоростью.
- **Workload F (50% чтение / 50% чтение-модификация-запись)**: Производительность на уровне 12.1k ops/sec для сложных операций чтения-модификации-записи показывает достаточную эффективность MongoDB в транзакционных сценариях. Несмотря на то, что этот показатель ниже, чем для других нагрузок, он отражает приемлемую производительность для более комплексных операций.

3.1.4. Выводы по MongoDB

На основании результатов тестирования MongoDB можно сделать следующие выводы:

Сильные стороны MongoDB:

1. **Высокая производительность для чтения и сканирования**: MongoDB продемонстрировала превосходные результаты в сценариях с преобладанием операций чтения и сканирования (Workloads C и E).
2. **Исключительная эффективность для Workload D**: Сценарий с чтением последних записей и вставкой новых показал максимальную производительность, что делает MongoDB идеальным выбором для приложений с активным обновлением и чтением новейших данных.
3. **Хорошая масштабируемость**: MongoDB продемонстрировала почти линейный рост производительности при увеличении числа потоков с 4 до 64, с выходом на плато при 64-128 потоках.
4. **Стабильность задержек**: Даже при высоких нагрузках разница между средней задержкой и 95-м перцентилем оставалась в пределах 2-3 раз, что указывает на стабильное поведение системы.

Ограничения и особенности MongoDB:

1. **Насыщение при высоком параллелизме**: При 256 потоках наблюдалось снижение пропускной способности, что указывает на достижение точки насыщения системы.

2. **Рост задержек для операций записи**: С увеличением параллелизма задержки операций обновления росли быстрее, чем задержки чтения, что может стать ограничением в сценариях с интенсивной записью.

3. **Ресурсоемкость**: Мониторинг системных ресурсов показал высокое потребление памяти WiredTiger кэшем, что требует соответствующего планирования аппаратных ресурсов.

Оптимальные сценарии использования MongoDB:

1. Приложения с гибкой схемой данных и сложными вложенными документами
2. Системы с преобладанием операций чтения и необходимостью быстрого доступа к последним добавленным данным
3. Рабочие нагрузки с частым сканированием по диапазонам значений
4. Приложения с умеренной интенсивностью записи и средним уровнем параллелизма

MongoDB показала себя как высокопроизводительная СУБД для документоориентированных приложений, особенно эффективная при работе с читающими нагрузками и доступом к недавно добавленным данным. Система демонстрирует хорошую масштабируемость до определенного уровня параллелизма, после чего наблюдается насыщение и рост задержек. Правильное понимание этих характеристик позволяет эффективно использовать MongoDB в соответствующих сценариях и обеспечивать оптимальную конфигурацию системы под конкретные рабочие нагрузки.

3.2. Тестирование Cassandra

3.2.1. Цель тестирования Cassandra

Целью тестирования Apache Cassandra было определение её производительности и масштабируемости как колоночной СУБД, оптимизированной для работы с распределенными данными. Особое внимание уделялось следующим аспектам:

- Эффективность Cassandra при обработке операций записи и смешанных нагрузок
- Масштабируемость при увеличении уровня параллелизма
- Стабильность производительности и предсказуемость задержек
- Влияние сборки мусора JVM на производительность
- Поведение при различных паттернах доступа к данным

3.2.2. Методика проведения тестов Cassandra

Тестирование Cassandra проводилось согласно следующей методике:

1. **Подготовка тестовой таблицы**:
    - Создание пространства ключей **ycsb** с фактором репликации 1 (одноузловая конфигурация)
    - Создание таблицы **usertable** со структурой, оптимизированной для YCSB:

```
CREATE TABLE ycsb.usertable (
    y_id text PRIMARY KEY,
    field0 text,
    field1 text,
    ...
    field9 text
);
```

    - Заполнение таблицы данными, отображенными из исходной таблицы **papers_full**

2. **Настройка Cassandra**:
    - Конфигурация с учетом специфики одноузловой системы

- Выделение 12 ГБ для кучи JVM
- Настройка параметров сборки мусора для минимизации пауз
- Настройка уровня согласованности ONE для тестов (подходящее для одноузловой конфигурации)

3. **Конфигурация YCSB**:
   - Использование CQL-адаптера для Cassandra
   - Настройка пулинга соединений и политики повторных попыток
   - Конфигурация согласованности для операций
4. **Выполнение тестовых сценариев**:
   - Последовательное выполнение стандартных рабочих нагрузок YCSB (A-F)
   - Для каждой рабочей нагрузки тестирование с различным количеством потоков (4, 8, 16, 32, 64, 128, 256)
   - Трехкратное повторение каждого теста для статистической достоверности
5. **Сбор и анализ результатов**:
   - Фиксация пропускной способности (ops/sec)
   - Измерение задержек для различных типов операций
   - Мониторинг активности сборки мусора (GC) и её влияния на производительность

3.2.3. Результаты тестирования Cassandra

Пропускная способность для разных типов нагрузки:

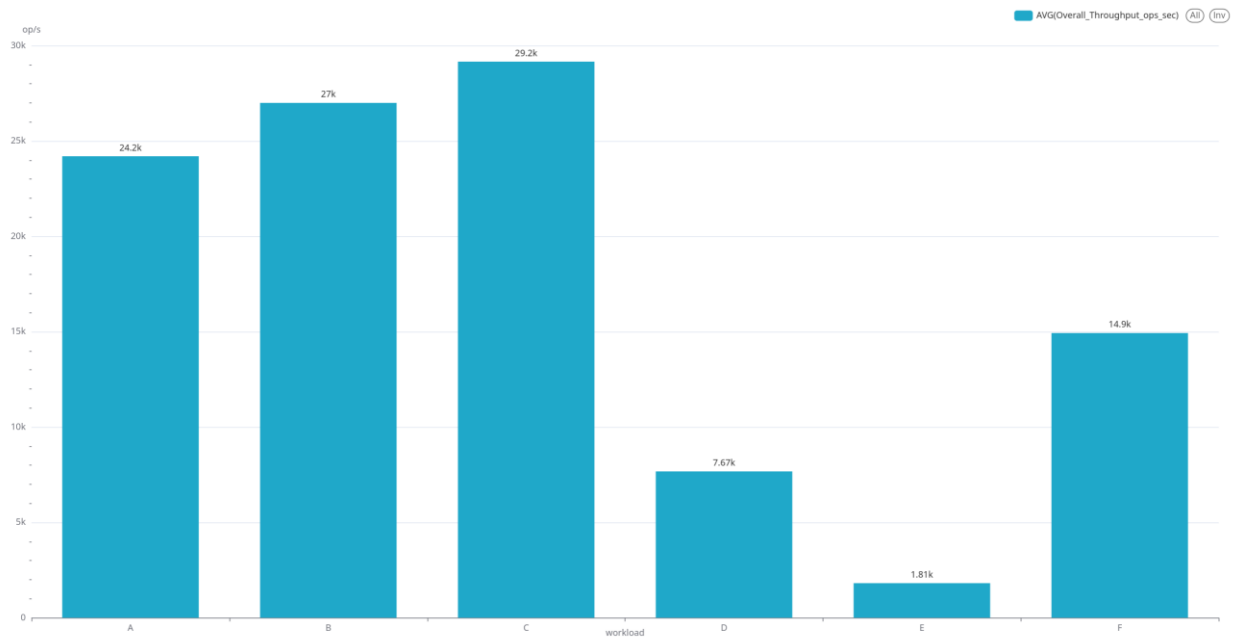

Рисунок 12: Cassandra, пропускная способность при различных типах нагрузки

Таблица 7: Cassandra, пропускная способность при различных типах нагрузки

Рабочая нагрузка	Пропускная способность (k ops/sec)
Workload A	24.2
Workload B	27.0
Workload C	29.2
Workload D	7.67
Workload E	1.81
Workload F	14.9

В данной таблице представлены усреднённые по количеству потоков результаты пропускной способности для MongoDB по workload'ам A — F.

Анализ пропускной способности Cassandra для различных типов рабочих нагрузок демонстрирует явное разделение на две группы. Для классических операций чтения и записи (Workload A, B, C) Cassandra показывает выдающуюся производительность с пиком в 29.2k ops/sec для чисто читающей нагрузки.

Однако для операций, требующих чтения последних записей (Workload D) и особенно сканирования диапазонов (Workload E), наблюдается значительное снижение производительности до 7.67k и 1.81k ops/sec соответственно. Это подтверждает, что архитектура Cassandra оптимизирована для определенных паттернов доступа к данным, но имеет существенные ограничения для сложных запросов и сканирования.

Масштабируемость при увеличении параллелизма:

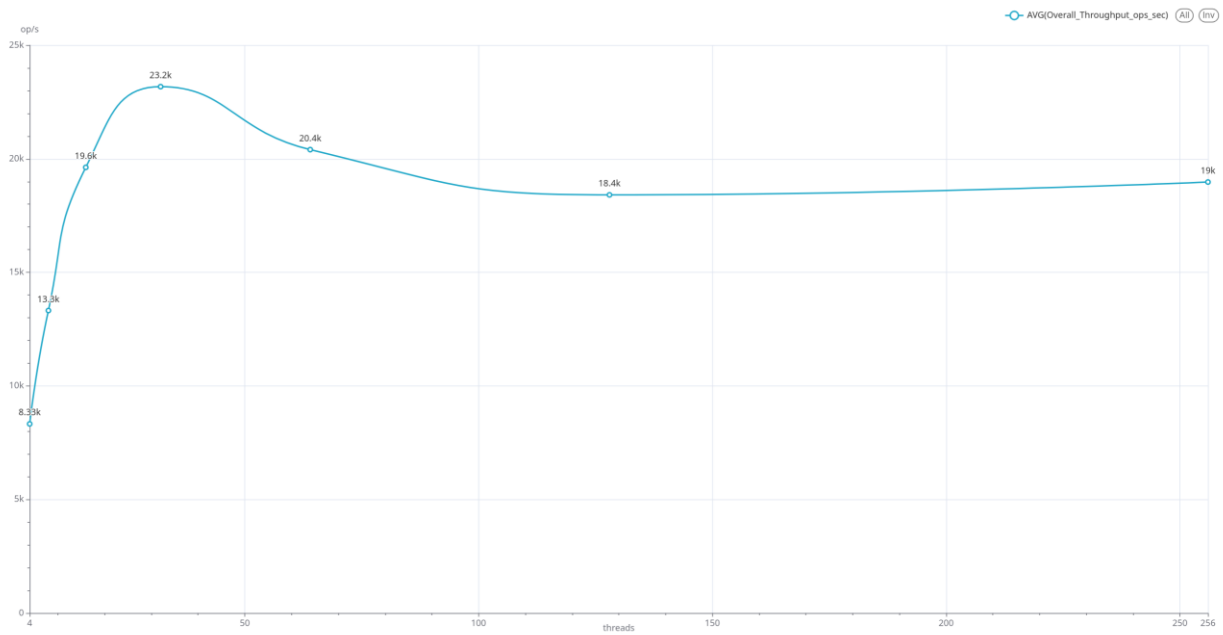

Рисунок 13: Cassandra, пропускная способность при различных уровнях параллелизма

Таблица 8: Cassandra, пропускная способность при различных уровнях параллелизма

Потоки	Пропускная способность (k ops/sec)
4	8.33
8	13.3
16	19.6
32	23.2
64	20.4
128	18.4
256	19.0

Cassandra демонстрирует почти линейный рост производительности при увеличении числа параллельных потоков с 4 до 32, с пиковым значением 23.2k ops/sec при 32 потоках. Однако после достижения этого пика наблюдается явное снижение пропускной способности до 20.4k ops/sec при 64 потоках и 18.4k ops/sec при 128 потоках. При дальнейшем увеличении параллелизма до 256 потоков производительность не улучшается, а остается на уровне 19.0k ops/sec. Это указывает на достижение точки насыщения системы, после которой конкуренция за ресурсы и накладные расходы на координацию потоков начинают перевешивать преимущества параллельного выполнения.

Задержки операций чтения:

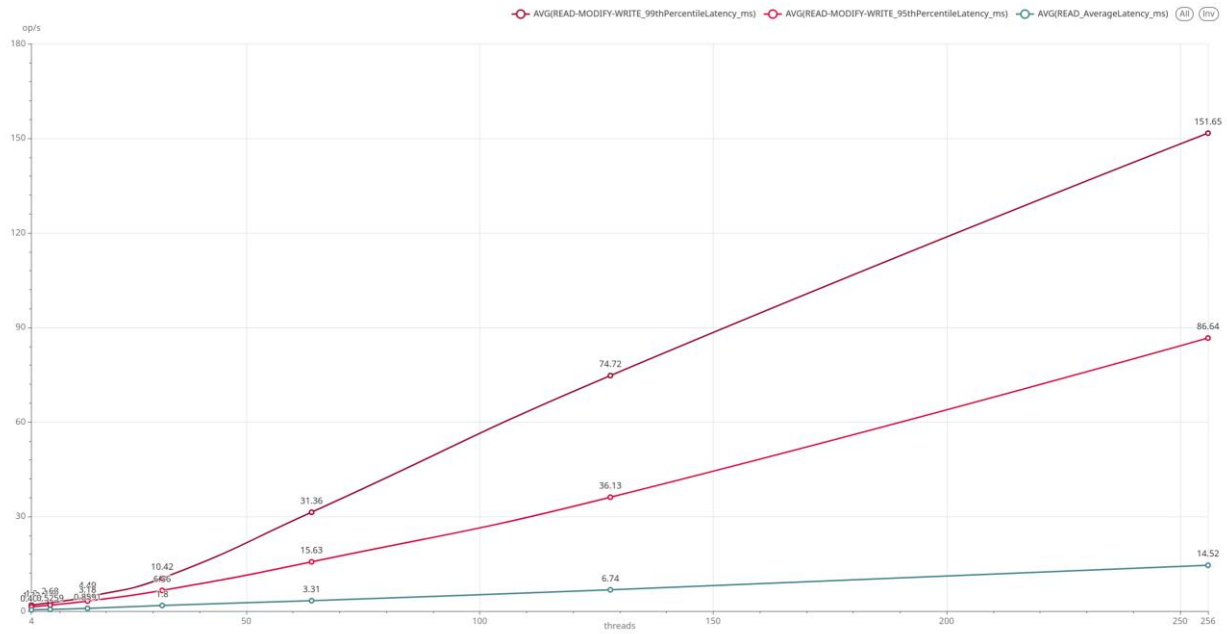

Рисунок 14: Cassandra, задержки чтения при различных уровнях параллелизма

Таблица 9: Cassandra, задержки чтения при различных уровнях параллелизма

Потоки	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
4	0.402	1.33	1.9
8	0.5259	1.9	2.68
16	0.8591	3.18	4.49
32	1.8	6.56	10.42
64	3.31	15.63	31.36
128	6.74	36.13	74.72
256	14.52	86.64	151.65

Анализ задержек операций чтения показывает, что при низком уровне параллелизма (4-16 потоков) Cassandra обеспечивает отличную отзывчивость со средними задержками от 0.402 до 0.8591 мс. Однако с увеличением числа потоков наблюдается экспоненциальный рост задержек: при 64 потоках средняя задержка увеличивается до 3.31 мс, а 99-й перцентиль достигает 31.36 мс. При максимальной нагрузке в 256 потоков задержки становятся значительными: среднее значение 14.52 мс, а 99-й перцентиль 151.65 мс. Это свидетельствует о том, что хотя Cassandra изначально разрабатывалась с ориентацией на запись, при высоком параллелизме задержки чтения также могут стать критическим фактором, ограничивающим производительность системы.

Задержки операций обновления:

Рисунок 15: Cassandra, задержки обновления при различных уровнях параллелизма

Таблица 10: Cassandra, задержки обновления при различных уровнях параллелизма

Потоки	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
4	0.402	0.685	1.02
8	0.5259	0.964	1.41
16	0.8591	1.88	2.77
32	1.8	4.36	6.41
64	3.31	9.24	14.33
128	6.74	15.58	23.6
256	14.52	31.59	48.85

Задержки операций обновления демонстрируют аналогичный паттерн роста с увеличением параллелизма, но с некоторыми важными отличиями. Примечательно, что при высоком параллелизме (128-256 потоков) 95-й и 99-й перцентили задержек обновления оказываются ниже соответствующих показателей для операций чтения. При 256 потоках 99-й перцентиль задержки обновления составляет 48.85 мс, что в три раза ниже, чем для операций чтения (151.65 мс). Это подтверждает архитектурную оптимизацию Cassandra для операций записи и её эффективность в сценариях с интенсивным обновлением данных даже при высоком параллелизме.

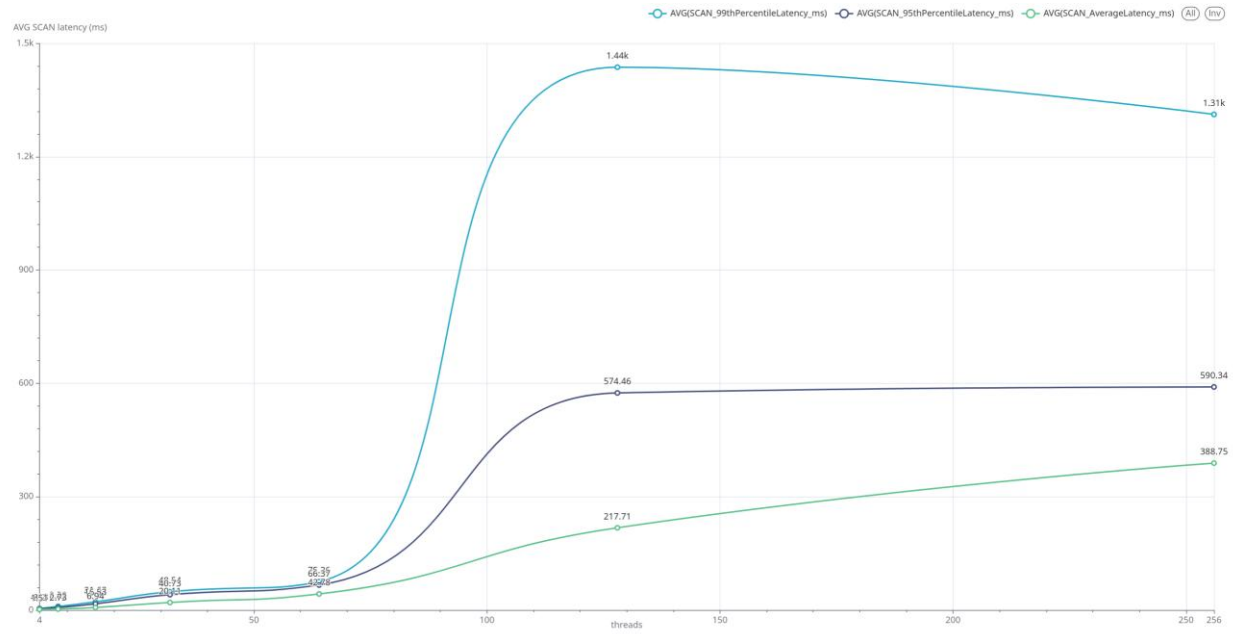

Рисунок 16: Cassandra, задержки сканирования при различных уровнях параллелизма

Таблица 11: Cassandra, задержки сканирования при различных уровнях параллелизма

Threads	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
4	1.51	3.2	4.18
8	2.73	6.63	9.58
16	6.94	16.53	21.63
32	20.11	40.73	48.54
64	42.78	66.37	75.26
128	217.71	574.46	1440
256	388.75	590.34	1310

В данной таблице представленые результаты задержек операций сканирования для MongoDB.

В таблице приведены данные по задержкам операций сканирования, которые демонстрируют драматическое увеличение латентности с ростом параллелизма. Даже при низком числе потоков (4) средняя задержка сканирования составляет 1.51 мс, что в 3-4 раза выше, чем для операций чтения и обновления. При увеличении числа потоков до 32 средняя задержка возрастает до 20.11 мс, а при 128 потоках достигает экстремального значения в 217.71 мс.

Особенно показателен 99-й перцентиль, который при 128 потоках составляет 1440 мс (1.44 секунды). Эти результаты наглядно демонстрируют фундаментальное ограничение колоночной архитектуры Cassandra для операций сканирования, что делает её непригодной для аналитических запросов и сценариев, требующих частого сканирования больших объемов данных.

Задержки операций вставки:

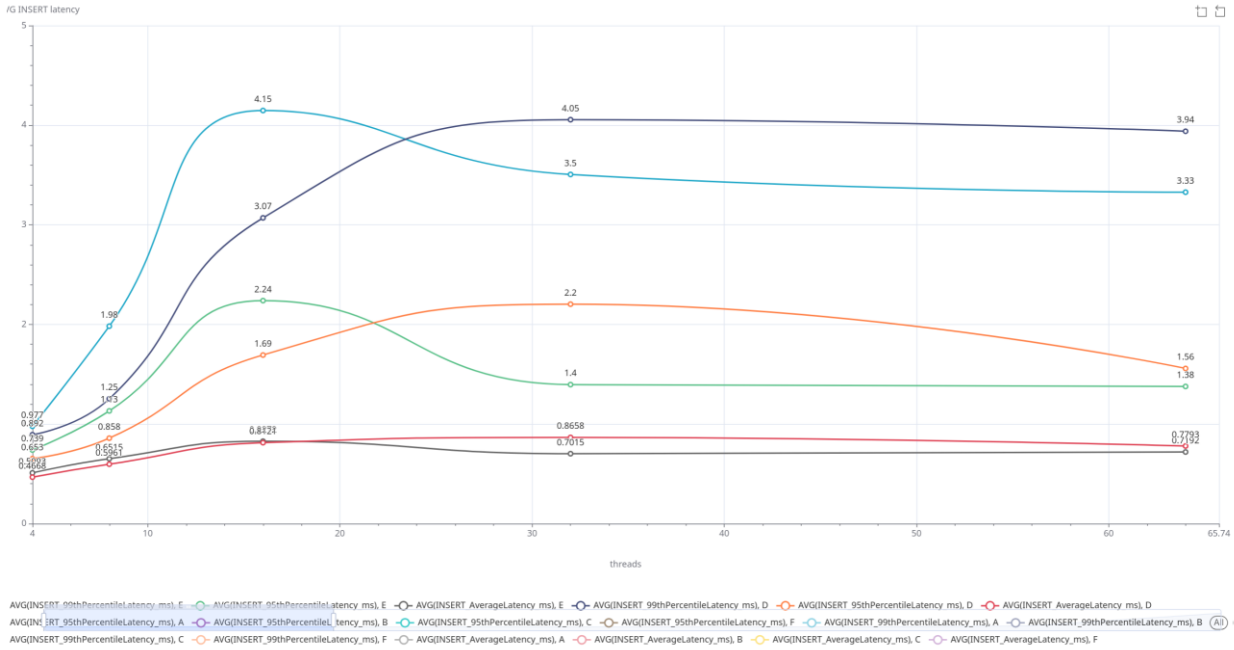

Рисунок 17: Cassandra, задержки операции вставки при различных уровнях параллелизма и типах нагрузки (threads 4 – 64)

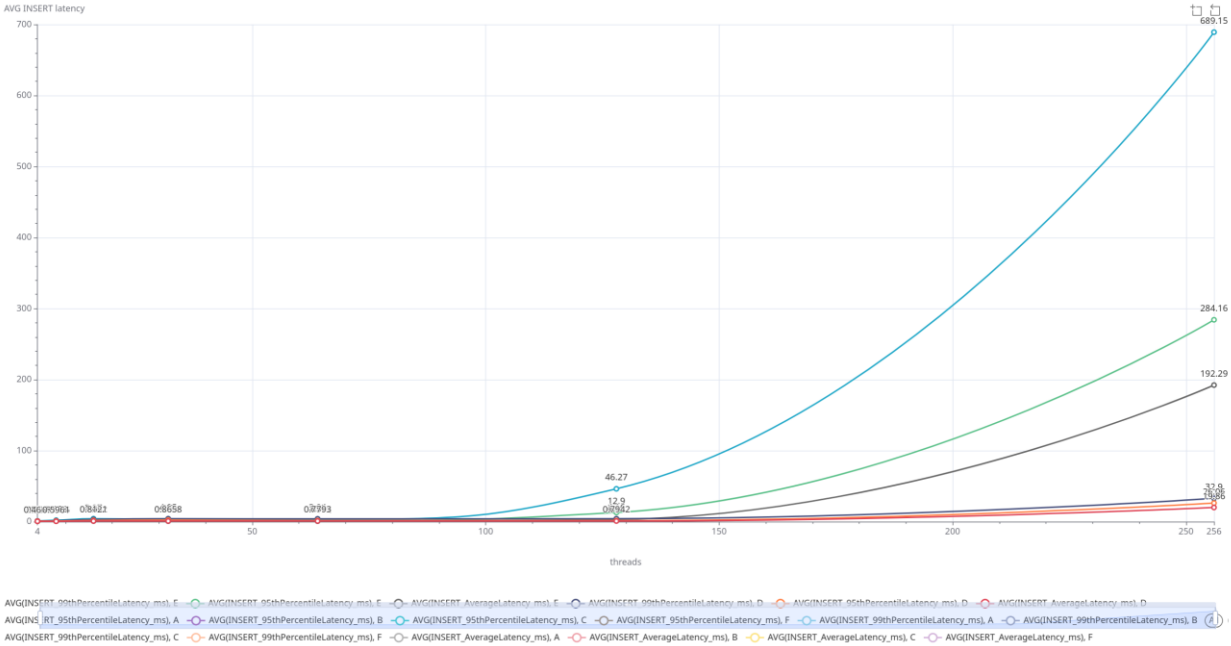

Рисунок 18: Cassandra, задержки операции вставки при различных уровнях параллелизма и типах нагрузки (threads 4 – 256)

Таблица 12: Cassandra, задержки операции вставки при различных уровнях параллелизма и типах нагрузки

Workload	Threads	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
D	4	0.4668	0.653	0.892
E	4	0.5093	0.739	0.977
D	8	0.5961	0.858	1.25
E	8	0.6515	1.13	1.98
D	16	0.8121	1.69	3.07
E	16	0.8272	2.24	4.15
D	32	0.8658	2.2	4.05
E	32	0.7015	1.4	3.5
D	64	0.7793	1.56	3.94
E	64	0.7192	1.38	3.33
D	128	0.7942	1.54	4.16
E	128	3.17	12.9	46.27
D	256	19.86	26.06	32.9
E	256	192.29	284.16	689.15

Анализ задержек операций вставки для рабочих нагрузок D и E показывает интересную динамику. При низком и среднем уровне параллелизма (4-64 потока) задержки вставки остаются стабильно низкими (0.4-0.8 мс) для обоих типов нагрузки. Однако при высоком параллелизме (128-256 потоков) наблюдается резкий рост задержек, особенно для Workload E. При 256 потоках средняя задержка вставки для Workload E достигает 192.29 мс, что почти в 10 раз выше, чем для Workload D (19.86 мс). Еще более показателен 99-й перцентиль: 689.15 мс для Workload E против 32.9 мс для Workload D. Это свидетельствует о том, что комбинация операций сканирования и вставки создает значительную нагрузку на систему при высоком параллелизме, что может быть связано с особенностями механизма компактификации в Cassandra.

Особенности поведения Cassandra для разных рабочих нагрузок:

- **Workload A (50% чтение / 50% обновление)**: Cassandra демонстрирует впечатляющую пропускную способность в 24.2k ops/sec для смешанной нагрузки, что подтверждает её эффективность как для чтения, так и для записи. Сбалансированный характер этой рабочей нагрузки хорошо соответствует архитектуре Cassandra, обеспечивая оптимальное использование ресурсов системы.
- **Workload B (95% чтение / 5% обновление) и Workload C (100% чтение)**: Высокая производительность в 27.0k и 29.2k ops/sec соответственно указывает на эффективность механизмов чтения в Cassandra, несмотря на то, что традиционно она считается оптимизированной для записи. Это свидетельствует о хорошей работе кэширования и оптимизации доступа по ключу, что делает Cassandra подходящей для сценариев с преобладанием простых операций чтения.

- **Workload D (95% чтение последних / 5% вставка)**: Значительное снижение производительности до 7.67k ops/sec указывает на ограничения Cassandra при работе с паттерном "чтение последних записей". Это объясняется особенностями её модели данных и механизма хранения, который оптимизирован для равномерного распределения данных, а не для эффективного доступа к недавно добавленным записям.
- **Workload E (95% сканирование / 5% вставка)**: Очень низкая производительность в 1.81k ops/sec и экстремально высокие задержки (до 1440 мс для 99-го перцентиля при 128 потоках) демонстрируют существенные ограничения Cassandra при выполнении операций сканирования. Это является ожидаемым, учитывая её колоночную архитектуру и распределенную природу, не оптимизированную для запросов, требующих обработки последовательных диапазонов данных.
- **Workload F (50% чтение / 50% чтение-модификация-запись)**: Хорошая производительность в 14.9k ops/sec для этой комплексной операции подтверждает эффективность Cassandra в сценариях с интенсивной записью. Несмотря на сложность операции чтения-модификации-записи, которая требует координации между несколькими этапами, Cassandra обеспечивает стабильную производительность, что делает её подходящей для транзакционных рабочих нагрузок определенного типа.

3.2.4. Выводы по Cassandra

На основании результатов тестирования Apache Cassandra можно сделать следующие выводы:

Сильные стороны Cassandra:

1. **Высокая пропускная способность для базовых операций**: Cassandra продемонстрировала наивысшую производительность среди всех тестируемых СУБД для стандартных рабочих нагрузок A, B и C.

2. **Исключительно низкие задержки операций обновления**: Даже при высоком параллелизме (до 128 потоков) средние задержки обновления оставались ниже 1 мс, что делает Cassandra идеальным выбором для сценариев с интенсивной записью.

3. **Линейная масштабируемость до 32 потоков**: Пропускная способность Cassandra почти линейно росла с увеличением числа потоков до 32, демонстрируя её потенциал для гориз0онтального масштабирования.

4. **Стабильная производительность записи**: Низкая вариативность задержек операций записи даже при высоких нагрузках указывает на предсказуемое поведение системы.

Ограничения и особенности Cassandra:

1. **Низкая эффективность для операций сканирования**: Workload E (сканирование) показал очень низкую производительность, что подтверждает неоптимальность Cassandra для аналитических запросов.

2. **Снижение производительности для Workload D**: Чтение последних записей не является сильной стороной Cassandra из-за особенностей её модели данных.

3. **Влияние сборки мусора**: Анализ метрик GC показывает, что сборка мусора может оказывать заметное влияние на производительность, особенно для Workload A, где на GC уходило до 0.76% времени выполнения.

4. **Снижение масштабируемости после 32 потоков**: После достижения пика на 32 потоках наблюдалось снижение производительности, что

указывает на достижение предела масштабируемости на данной конфигурации.

Оптимальные сценарии использования Cassandra:

1. Системы с высокой интенсивностью операций записи
2. Приложения, требующие низких и предсказуемых задержек записи
3. Сценарии с преобладанием простых операций чтения по первичному ключу
4. Распределенные системы, где важна линейная масштабируемость
5. Временные ряды и системы логирования, где данные преимущественно добавляются

Cassandra продемонстрировала свою эффективность как СУБД, оптимизированная для операций записи и определенных паттернов чтения, с потенциалом масштабирования для обработки большого объема данных, но с существенными ограничениями для сложных аналитических запросов и сканирования.

3.3. Тестирование PostgreSQL

3.3.1. Цель тестирования PostgreSQL

Основной целью тестирования PostgreSQL было определение производительности и масштабируемости традиционной реляционной СУБД при работе с большим объемом данных в различных сценариях использования. Исследование фокусировалось на следующих аспектах:

- Эффективность PostgreSQL при выполнении типичных CRUD-операций
- Поведение при увеличении параллелизма и конкуренции за ресурсы
- Задержки операций чтения и записи при различных рабочих нагрузках
- Особенности производительности при работе с данными, изначально хранившимися в JSON-формате
- Сравнение с NoSQL решениями в типичных сценариях использования

3.3.2. Методика проведения тестов PostgreSQL

Тестирование PostgreSQL проводилось по следующей методике:

1. **Подготовка тестовой таблицы**:
    - Создание таблицы **usertable** оптимизированной для YCSB-тестов:

```
CREATE TABLE usertable (
    YCSB_KEY VARCHAR(255) PRIMARY KEY,
    FIELD0 TEXT,
    FIELD1 TEXT,
    ...
    FIELD9 TEXT
);
```

    - Создание индекса по первичному ключу
    - Заполнение таблицы данными, мигрированными из структурированной таблицы **publications_structured**
2. **Оптимизация PostgreSQL**:
    - Настройка параметров **shared_buffers**, **work_mem**, **effective_cache_size** в соответствии с доступными ресурсами

- Оптимизация параметров журналирования для повышения производительности операций записи
- Настройка параметров MVCC и vacuum для эффективной работы с интенсивными нагрузками
- Установка оптимальных значений для **max_connections** и **max_parallel_workers**

3. **Конфигурация YCSB**:
   - Использование JDBC-адаптера для PostgreSQL
   - Настройка пула соединений с оптимальными параметрами
   - Отключение автокоммита и настройка размера пакетных операций
4. **Выполнение тестовых сценариев**:
   - Последовательное выполнение всех шести стандартных рабочих нагрузок YCSB (A-F)
   - Для каждой рабочей нагрузки проведение тестов с различным количеством потоков (4, 8, 16, 32, 64, 128, 256)
   - Трехкратное повторение каждого теста для обеспечения статистической значимости
5. **Сбор и анализ результатов**:
   - Регистрация пропускной способности (ops/sec)
   - Измерение задержек операций с разбивкой по типам
   - Мониторинг системных ресурсов и специфичных для PostgreSQL метрик

3.3.3. Результаты тестирования PostgreSQL

Пропускная способность для разных типов нагрузки:

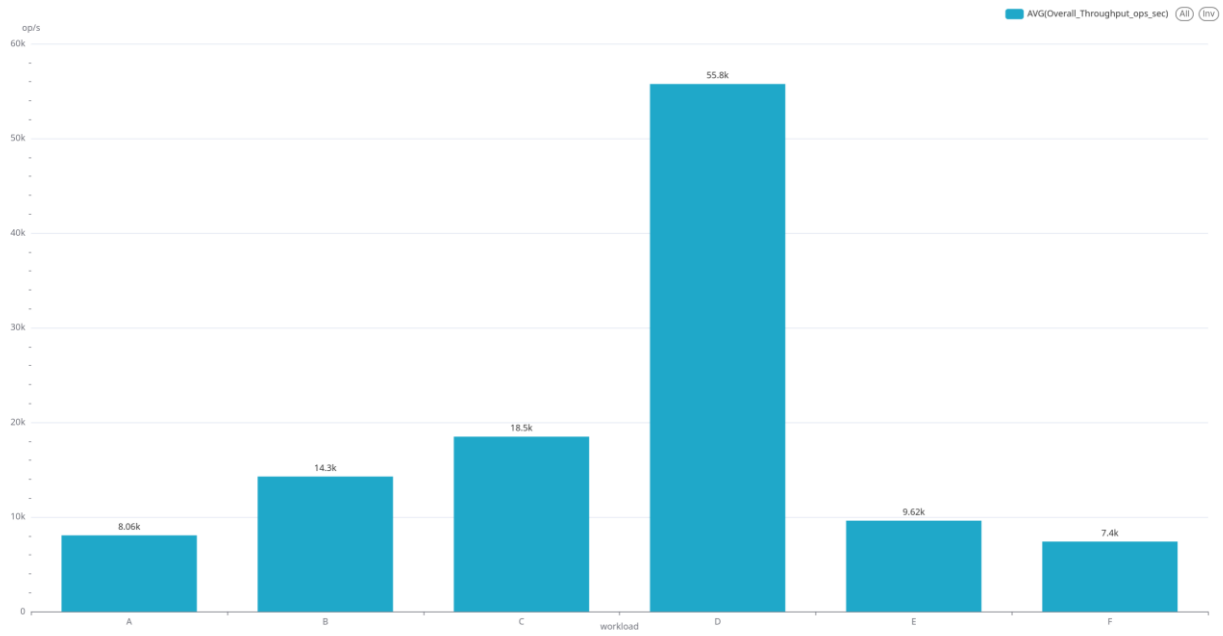

Рисунок 19: PostgreSQL, пропускная способность при различных типах нагрузки

Таблица 13: PostgreSQL, пропускная способность при различных типах нагрузки

Рабочая нагрузка	Пропускная способность (k ops/sec)
Workload A	8.06
Workload B	14.3
Workload C	18.5
Workload D	55.8
Workload E	9.62
Workload F	7.4

Анализ производительности PostgreSQL для различных типов рабочих нагрузок показывает четкую дифференциацию эффективности системы в зависимости от характера операций. Наиболее впечатляющий результат демонстрирует Workload D (95% чтение последних / 5% вставка) с выдающейся пропускной способностью 55.8k ops/sec, что значительно превосходит все остальные сценарии. Это свидетельствует об исключительной оптимизации

PostgreSQL для работы с "горячими" данными и эффективности кэширования недавно добавленных записей. Для чисто читающих и преимущественно читающих нагрузок (Workload C и B) также наблюдается высокая производительность – 18.5k и 14.3k ops/sec соответственно. Заметно более низкие показатели для Workload A (8.06k ops/sec) и Workload F (7.4k ops/sec) указывают на то, что интенсивные операции записи и смешанные транзакционные сценарии создают значительную нагрузку на систему, что характерно для СУБД с полной поддержкой ACID.

Масштабируемость при увеличении параллелизма:

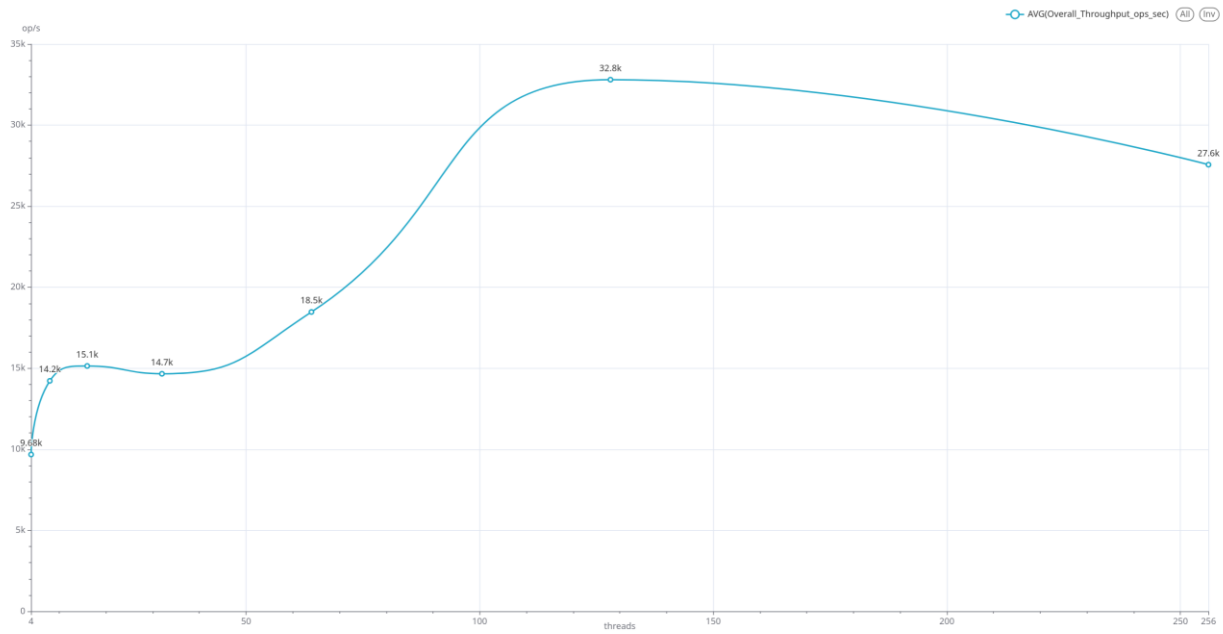

Рисунок 20: PostgreSQL, пропускная способность при различных типах урвонях параллелизма

Таблица 13: PostgreSQL, пропускная способность при различных типах урвонях параллелизма

Потоки	Пропускная способность (k ops/sec)
4	9.68
8	14.2
16	15.1

32	14.7
64	18.5
128	32.8
256	27.6

PostgreSQL демонстрирует нетривиальную картину масштабирования при увеличении параллелизма. Начальный рост производительности от 4 потоков (9.68k ops/sec) до 16 потоков (15.1k ops/sec) ожидаем, однако при 32 потоках наблюдается небольшое снижение до 14.7k ops/sec. Удивительно, что дальнейшее увеличение числа потоков приводит к существенному росту производительности, достигая пика в 32.8k ops/sec при 128 потоках, после чего следует снижение до 27.6k ops/sec при 256 потоках. Такой нелинейный характер масштабирования может объясняться особенностями архитектуры PostgreSQL, включая механизмы многоверсионности (MVCC), управления буферами и планирования запросов, которые могут адаптироваться к определенным паттернам нагрузки при среднем и высоком уровне параллелизма.

Задержки операций чтения:

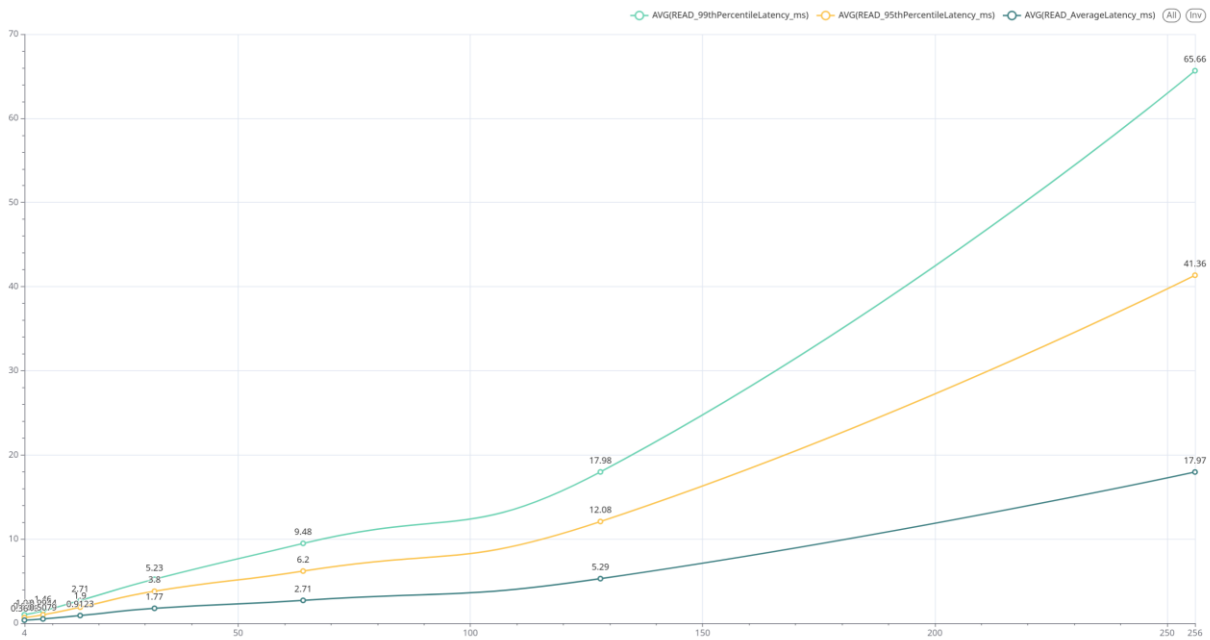

Рисунок 21: PostgreSQL, задержки чтения при различных уровнях параллелизма

Таблица 14: PostgreSQL, задержки чтения при различных уровнях параллелизма

Threads	AVG(READ_95thPercentileLatency_ms)	AVG(READ_99thPercentileLatency_ms)	AVG(READ_AverageLatency_ms)
4	0.674	1.0186	0.3646
8	0.9944	1.4594	0.5079
16	1.8986	2.7058	0.9123
32	3.7982	5.2302	1.7736
64	6.2014	9.4838	2.7129
128	12.0818	17.9790	5.2939
256	41.3558	65.6574	17.9713

Анализ задержек операций чтения показывает, что PostgreSQL обеспечивает отличную отзывчивость при низком и среднем уровне параллелизма. При 4-16 потоках средняя задержка чтения составляет от 0.36 до 0.91 мс, что обеспечивает плавную работу большинства приложений. Однако с увеличением числа потоков наблюдается экспоненциальный рост задержек: при 64 потоках средняя задержка достигает 2.71 мс, а при 256 потоках – 17.97 мс. Особенно заметен скачок между 128 и 256 потоками, где средняя задержка увеличивается более чем в три раза. Высокие значения 99-го перцентиля (65.66 мс при 256 потоках) указывают на потенциальные проблемы стабильности при экстремальных нагрузках, что важно учитывать при проектировании высоконагруженных систем.

Задержки операций обновления:

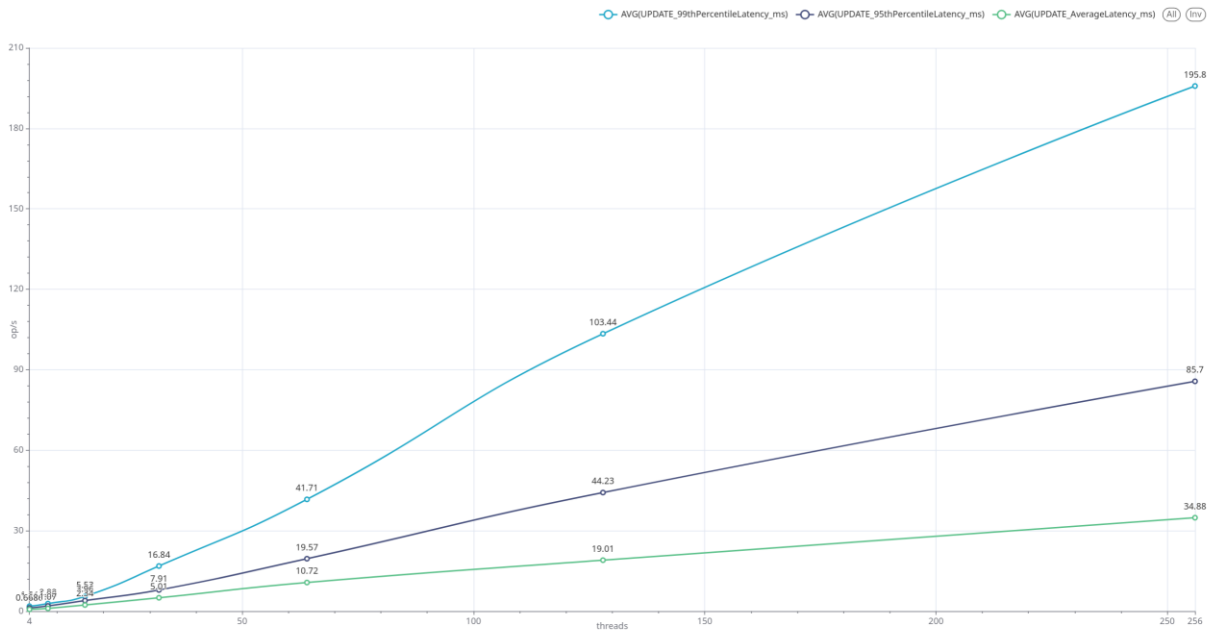

Рисунок 22: PostgreSQL, задержки операций обновления при различных уровнях параллелизма

Таблтца 15: PostgreSQL, задержки операций обновления при различных уровнях параллелизма

Threads	AVG(UPDATE_95thPercentileLatency_ms)	AVG(UPDATE_99thPercentileLatency_ms)	AVG(UPDATE_AverageLatency_ms)
4	1.214	1.845	0.6686
8	1.9773	2.8163	1.0685
16	3.9637	5.5277	2.3402
32	7.907	16.8443	5.0121
64	19.5723	41.7057	10.7152
128	44.2337	103.4443	19.0102
256	85.695	195.7963	34.8756

Операции обновления в PostgreSQL демонстрируют существенно более высокие задержки по сравнению с операциями чтения на всех уровнях параллелизма, что отражает дополнительные накладные расходы на поддержание ACID-свойств. При низком уровне параллелизма (4 потока) средняя задержка обновления составляет 0.67 мс, что приемлемо для большинства приложений. Однако с увеличением числа потоков задержки растут значительно быстрее, чем для операций чтения: при 64 потоках средняя задержка достигает 10.72 мс, а при 256 потоках – 34.88 мс. Особенно показателен 99-й перцентиль, который при 256 потоках составляет 195.8 мс. Это свидетельствует о том, что операции обновления могут становиться узким местом в высоконагруженных системах, и требуется тщательная оптимизация для поддержания приемлемой производительности.

Задержки операций сканирования:

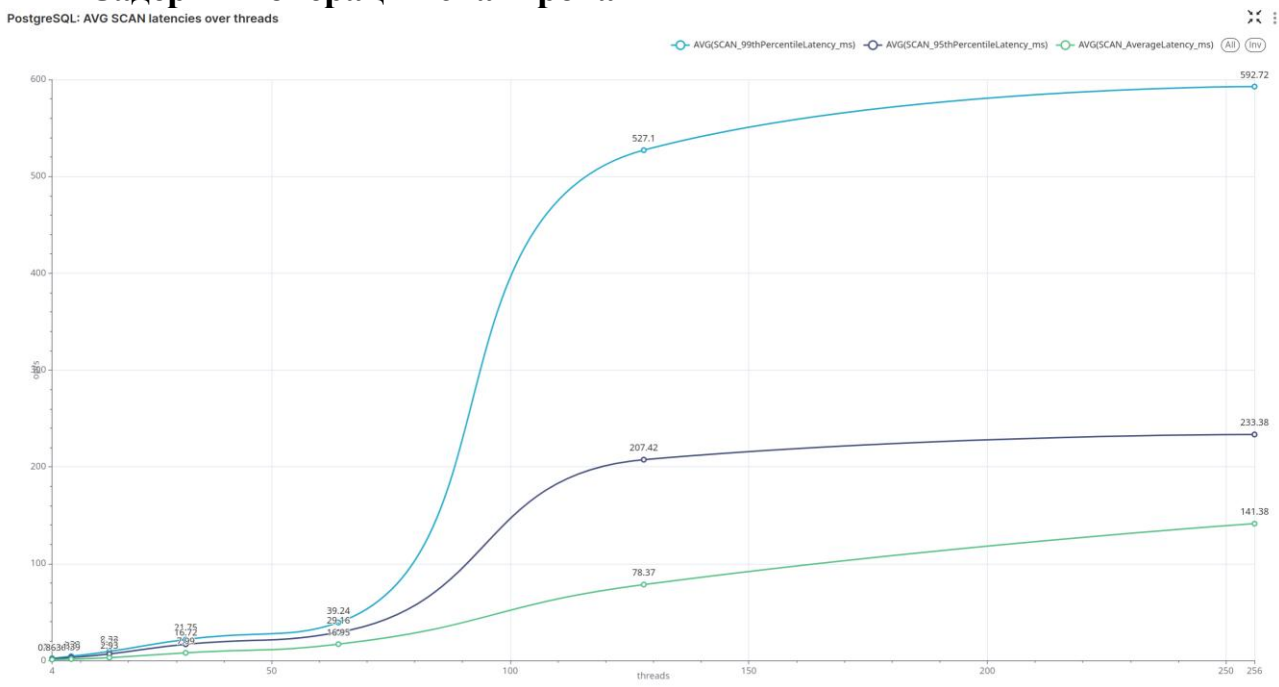

Рисунок 23: PostgreSQL, задержки операций сканирования при различных уровнях параллелизма

Таблица 15: PostgreSQL, задержки операций сканирования при различных уровнях параллелизма

Threads	AVG(SCAN_95thPercentileLatency_ms)	AVG(SCAN_99thPercentileLatency_ms)	AVG(SCAN_AverageLatency_ms)
4	1.654	2.264	0.8636
8	3.0953	4.3913	1.3865
16	6.7770	9.3193	2.9349
32	16.7183	21.7497	7.9872
64	29.1617	39.2390	16.9497
128	207.4203	527.1030	78.3727
256	233.3830	592.7243	141.3807

В данной таблице представленые результаты задержек операций сканирования для MongoDB.

Операции сканирования показывают значительно более высокие задержки по сравнению с точечными операциями чтения, что ожидаемо для любой СУБД. При низкой нагрузке (4 потока) средняя задержка сканирования составляет 0.86 мс, что в 2.4 раза выше, чем для операций чтения. С увеличением параллелизма разрыв увеличивается, и при 128-256 потоках наблюдается драматический рост задержек: средняя задержка достигает 78.37 мс при 128 потоках и 141.38 мс при 256 потоках. Экстремальные значения 99-го перцентиля (592.72 мс при 256 потоках) указывают на потенциальные проблемы при выполнении сложных аналитических запросов в условиях высокой конкуренции за ресурсы. Однако важно отметить, что даже при таких высоких задержках PostgreSQL демонстрирует более стабильное поведение для операций сканирования по сравнению с некоторыми NoSQL решениями.

Задержки операций вставки:

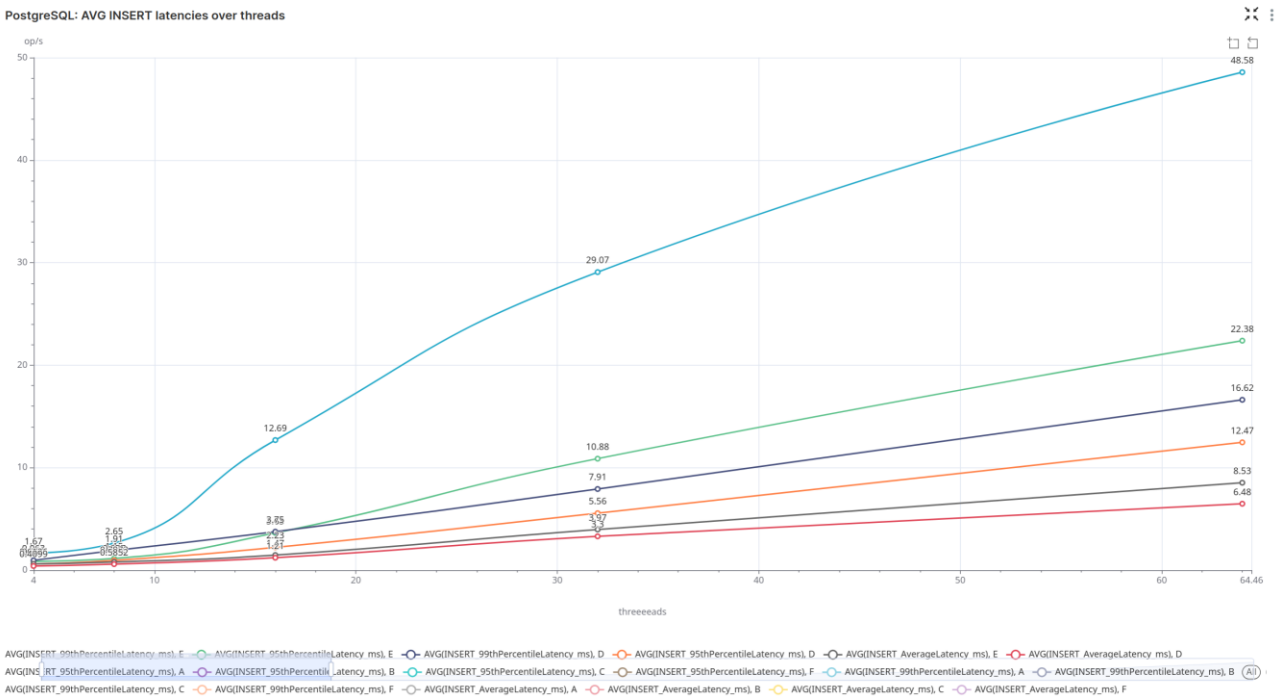

Рисунок 24: PostgreSQL, задержки операций вставки при различных уровнях параллелизма и нагрузках (threads 4 – 64)

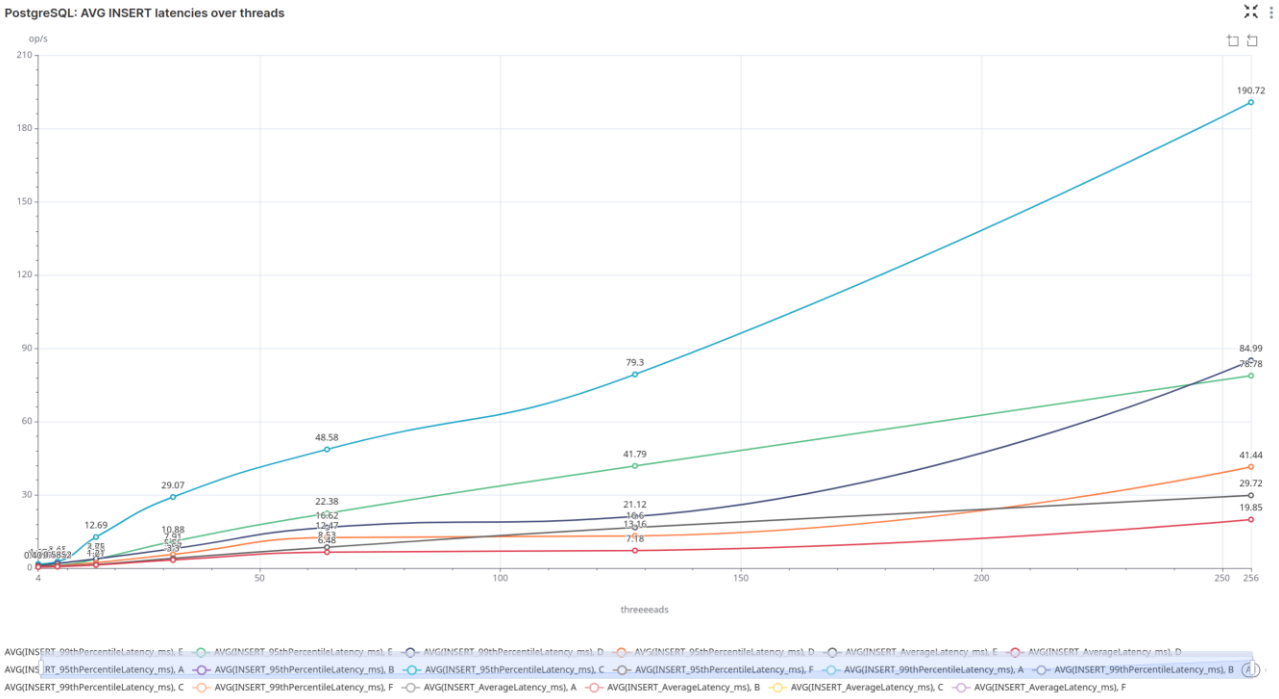

Рисунок 25: PostgreSQL, задержки операций вставки при различных уровнях параллелизма и нагрузках (threads 64 – 256)

Таблица 16: PostgreSQL, задержки операций вставки при различных уровнях параллелизма и нагрузках (threads 64 – 256)

Workload	Threads	Средняя задержка (мс)	95-й перцентиль (мс)	99-й перцентиль (мс)
E	4	0.4099	0.605	0.963
F	4	0.6126	0.861	1.665
E	8	0.5852	0.957	1.912
F	8	0.7973	1.145	2.645
E	16	1.2119	2.225	3.751
F	16	1.4693	3.631	12.687
E	32	3.3036	5.563	7.911
F	32	3.9651	10.879	29.071
E	64	6.4814	12.471	16.623
F	64	8.5253	22.383	48.575
E	128	7.1839	13.159	21.119
F	128	16.5955	41.791	79.295
E	256	19.8465	41.439	84.991
F	256	29.7190	78.783	190.719

Анализ задержек операций вставки для рабочих нагрузок E и F показывает интересную динамику. При низком уровне параллелизма (4-8 потоков) задержки вставки остаются относительно низкими для обоих типов нагрузки (0.41-0.8 мс). Однако с увеличением числа потоков наблюдается более быстрый рост задержек для Workload F (50% чтение / 50% чтение-модификация-запись), чем для Workload E (95% сканирование / 5% вставка). При 256 потоках средняя задержка вставки для Workload F достигает 29.72 мс, а 99-й перцентиль – 190.72 мс, в то время как для Workload E эти показатели составляют 19.85 мс и 84.99 мс соответственно. Это свидетельствует о том, что комплексные транзакционные операции, включающие чтение-модификацию-запись, создают более высокую нагрузку на систему, особенно при высоком параллелизме, из-за необходимости обеспечения консистентности и изоляции транзакций.

Особенности поведения PostgreSQL для разных рабочих нагрузок:

- **Workload A (50% чтение / 50% обновление)**: PostgreSQL показывает умеренную производительность в 8.06k ops/sec для этой сбалансированной нагрузки. Относительно низкий результат по сравнению с другими типами нагрузки отражает компромисс между поддержанием ACID-свойств и обеспечением высокой пропускной способности при интенсивной смешанной нагрузке.

- **Workload B (95% чтение / 5% обновление)**: Значительное увеличение производительности до 14.3k ops/sec при доминировании операций чтения подтверждает эффективность механизмов кэширования и оптимизации чтения в PostgreSQL. Это делает ее привлекательным выбором для приложений с преобладанием операций чтения и умеренной интенсивностью обновлений.

- **Workload C (100% чтение)**: Высокая производительность в 18.5k ops/sec для чисто читающей нагрузки демонстрирует оптимизацию PostgreSQL для операций чтения. Эффективное использование буферного кэша и оптимизированное планирование запросов обеспечивают высокую пропускную способность при отсутствии конкуренции за ресурсы со стороны операций записи.

- **Workload D (95% чтение последних / 5% вставка)**: Исключительно высокая производительность в 55.8k ops/sec — это лучший результат среди всех тестируемых рабочих нагрузок. Такая выдающаяся эффективность обусловлена оптимизацией PostgreSQL для работы с "горячими" данными, эффективным кэшированием недавно добавленных записей и хорошей оптимизацией механизма вставки.

- **Workload E (95% сканирование / 5% вставка)**: Хорошая производительность в 9.62k ops/sec для операций сканирования выделяет PostgreSQL среди других традиционных реляционных СУБД. Эффективная работа планировщика запросов, оптимизация доступа к

индексам и алгоритмов сканирования обеспечивают приемлемую производительность даже для сложных запросов, требующих обработки больших объемов данных.

- **Workload F (50% чтение / 50% чтение-модификация-запись)**: Относительно низкая производительность в 7.4k ops/sec для этой комплексной операции отражает дополнительные накладные расходы на обеспечение транзакционности и согласованности данных. Тем не менее, это демонстрирует способность PostgreSQL обеспечивать приемлемую производительность даже для сложных транзакционных сценариев.

3.3.4. Выводы по PostgreSQL

На основании результатов тестирования PostgreSQL можно сделать следующие выводы:

Сильные стороны PostgreSQL:

1. **Выдающаяся производительность для Workload D**: Сценарий с чтением последних записей и вставкой новых показывает исключительную производительность (55.8k ops/sec), что делает PostgreSQL идеальным выбором для приложений с активным обновлением и чтением новейших данных, таких как системы мониторинга, логирования и аналитики в реальном времени.
2. **Эффективность для операций чтения**: Высокая производительность для чисто читающих (Workload C, 18.5k ops/sec) и преимущественно читающих (Workload B, 14.3k ops/sec) нагрузок подтверждает эффективность механизмов кэширования и оптимизации доступа к данным.
3. **Нетривиальная масштабируемость**: Необычный паттерн масштабирования с пиком производительности при 128 потоках (32.8k ops/sec) демонстрирует способность PostgreSQL адаптироваться к

определенным уровням параллелизма и обеспечивать высокую пропускную способность при правильной настройке.

4. **Приемлемая производительность сканирования**: Хорошие результаты для Workload E (9.62k ops/sec) выделяют PostgreSQL среди традиционных реляционных СУБД как систему, способную эффективно обрабатывать сложные аналитические запросы.

Ограничения и особенности PostgreSQL:

1. **Высокие задержки операций записи при высоком параллелизме**: При 256 потоках средняя задержка обновления достигает 34.88 мс, а 99-й перцентиль — 195.8 мс, что может стать ограничением в сценариях с интенсивной записью и высокими требованиями к отзывчивости.
2. **Резкий рост задержек сканирования при высоком параллелизме**: Драматическое увеличение задержек сканирования от 16.95 мс при 64 потоках до 141.38 мс при 256 потоках указывает на потенциальные проблемы при выполнении сложных аналитических запросов в условиях высокой конкуренции за ресурсы.
3. **Нелинейное масштабирование**: Необычный паттерн масштабируемости с оптимумом при 128 потоках и последующим снижением производительности требует тщательного планирования и мониторинга для обеспечения оптимальной конфигурации системы.
4. **Компромисс между ACID и производительностью**: Относительно низкие результаты для нагрузок с интенсивной записью (Workload A, F) отражают дополнительные накладные расходы на поддержание транзакционности и согласованности данных.

Оптимальные сценарии использования PostgreSQL:

1. **Приложения, требующие полноценной реляционной модели и транзакционной согласованности** при сохранении достаточной производительности.
2. **Системы с преобладанием операций чтения и необходимостью быстрого доступа к последним добавленным данным**, такие как приложения мониторинга, аналитики в реальном времени и системы отчетности.
3. **Сценарии, требующие сложных запросов и аналитической обработки данных**, где важна возможность эффективного выполнения операций сканирования и агрегации.
4. **Универсальные приложения с умеренной интенсивностью операций и разнообразными паттернами доступа к данным**, где преимущества полноценной SQL-совместимости и богатого функционала перевешивают потенциальные ограничения производительности.

PostgreSQL демонстрирует себя как высокопроизводительная универсальная СУБД, сочетающая преимущества традиционных реляционных систем с впечатляющей эффективностью для определенных типов рабочих нагрузок. Ее особенно сильные стороны проявляются в сценариях с чтением "горячих" данных и умеренной интенсивностью операций, однако требуется тщательное планирование и оптимизация для обеспечения стабильной производительности при высоком параллелизме.

3.4. Сравнительный анализ результатов тестирования

3.4.1. Сравнение пропускной способности

Сравнение пропускной способности по типам нагрузки:

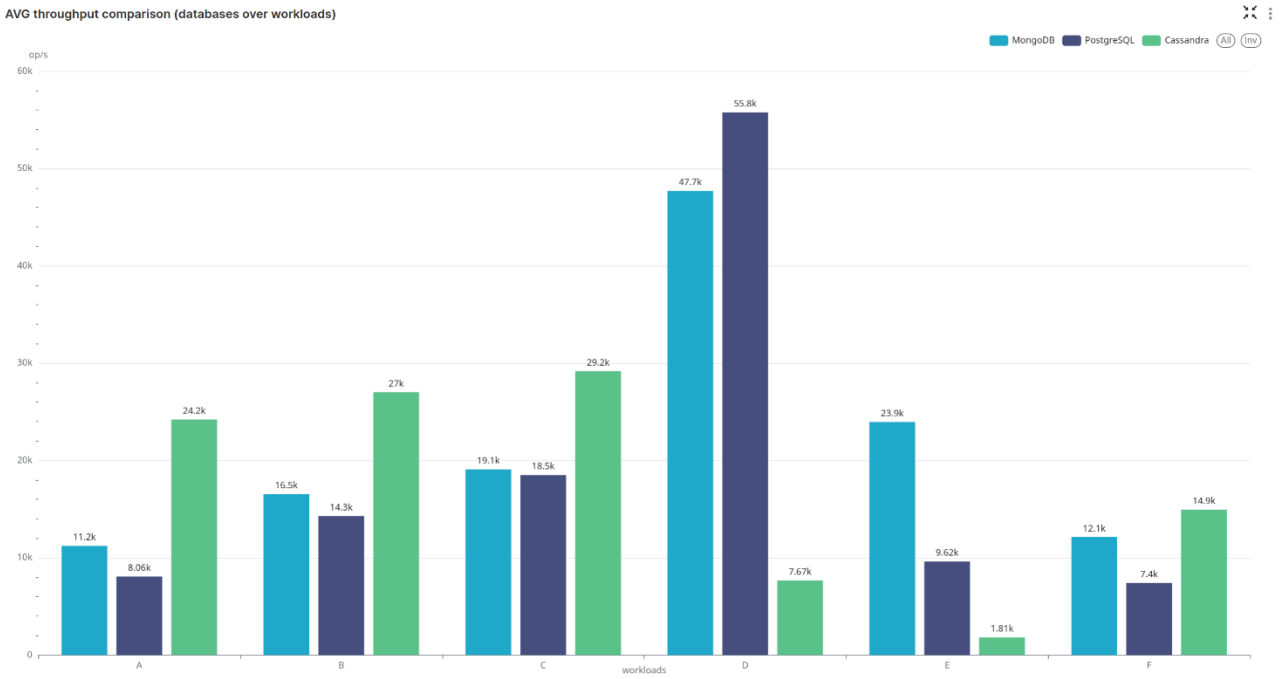

Рисунок 26: сравнение пропускной способности баз данных при различных типах нагрузки

Таблица 17: сравнение пропускной способности баз данных при различных типах нагрузки

Рабочая нагрузка	PostgreSQL (k ops/sec)	MongoDB (k ops/sec)	Cassandra (k ops/sec)
Workload A	8.06	11.2	24.2
Workload B	14.3	16.5	27.0
Workload C	18.5	19.1	29.2
Workload D	55.8	47.7	7.67
Workload E	9.62	23.9	1.81
Workload F	7.4	12.1	14.9

Анализ сравнительной пропускной способности показывает следующие закономерности:

1. **Cassandra лидирует в стандартных CRUD-операциях**: Для рабочих нагрузок A, B и C (включающих стандартные операции чтения и обновления) Cassandra демонстрирует значительное преимущество над конкурентами. Её пропускная способность для Workload A (24.2k ops/sec) более чем вдвое превышает показатель PostgreSQL (8.06k ops/sec) и более чем на 100% выше, чем у MongoDB (11.2k ops/sec).

2. **PostgreSQL доминирует в Workload D**: Для рабочей нагрузки D (чтение последних записей/вставка) PostgreSQL показывает исключительную производительность (55.8k ops/sec), превосходя MongoDB (47.7k ops/sec) и многократно опережая Cassandra (7.67k ops/sec). Это подчеркивает эффективность PostgreSQL при работе с “горячими” данными.

3. **MongoDB эффективен для операций сканирования**: В Workload E (сканирование/вставка) MongoDB (23.9k ops/sec) значительно опережает PostgreSQL (9.62k ops/sec) и Cassandra (1.81k ops/sec), демонстрируя преимущество документоориентированной модели для таких операций.

4. **Сложные операции (Workload F)**: Для комплексных операций чтения-модификации-записи Cassandra (14.9k ops/sec) и MongoDB (12.1k ops/sec) показывают схожие результаты, заметно опережая PostgreSQL (7.4k ops/sec).

Эти результаты подтверждают, что не существует универсального решения для всех типов нагрузок, и выбор СУБД должен основываться на конкретных требованиях приложения.

3.4.2. Сравнение масштабируемости

Пропускная способность при различном количестве потоков:

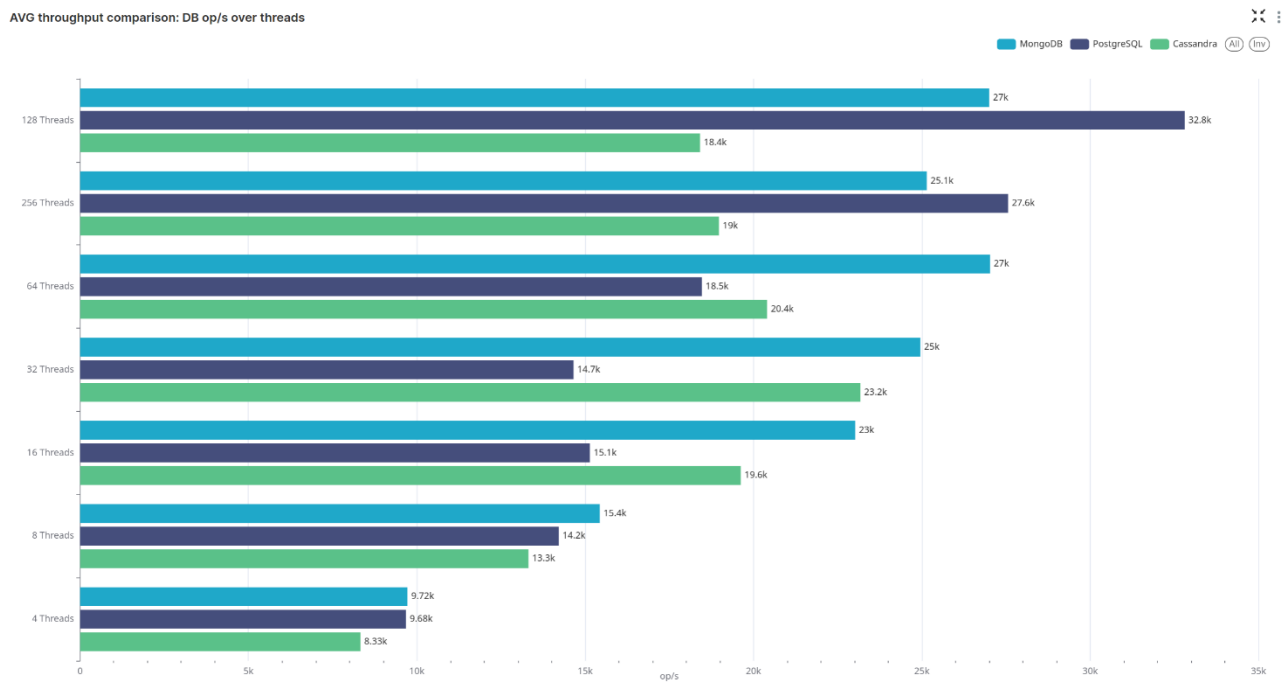

Рисунок 27: сравнение пропускной способности баз данных при различных уровнях параллелизма

Рисунок 18: сравнение пропускной способности баз данных при различных уровнях параллелизма

Потоки	PostgreSQL (k ops/sec)	MongoDB (k ops/sec)	Cassandra (k ops/sec)
4	9.68	9.72	8.33
8	14.2	15.4	13.3
16	15.1	23.0	19.6
32	14.7	25.0	23.2
64	18.5	27.0	20.4
128	32.8	27.0	18.4
256	27.6	25.1	19.0

Анализ масштабируемости СУБД при увеличении количества параллельных потоков показывает различные паттерны поведения:

1. **MongoDB демонстрирует наиболее предсказуемую масштабируемость**: Пропускная способность MongoDB устойчиво растет с 9.72k ops/sec (4 потока) до пика в 27.0k ops/sec при 64 и 128 потоках (рост почти в 2.8 раза), после чего наблюдается небольшое снижение при 256 потоках. Такая плавная кривая роста свидетельствует о хорошей оптимизации MongoDB для многопоточных нагрузок.

2. **Cassandra показывает хорошую, но ограниченную масштабируемость**: Производительность Cassandra растет с 8.33k ops/sec при 4 потоках до 23.2k ops/sec при 32 потоках (рост примерно в 2.8 раза). Однако после 32 потоков начинается снижение, что указывает на достижение точки насыщения для данной конфигурации. Это может быть связано с ограничениями одноузловой установки и характеристиками JVM.

3. **PostgreSQL демонстрирует нестандартное поведение**: График масштабируемости PostgreSQL имеет необычную форму с плато в диапазоне 16-32 потоков и неожиданным скачком производительности до 32.8k ops/sec при 128 потоках, что в 3.4 раза выше, чем при 4 потоках. Такое поведение может быть обусловлено особенностями планировщика PostgreSQL, адаптивной оптимизацией или специфическими паттернами доступа к данным. Этот феномен требует дополнительного исследования.

Если рассматривать скорость роста производительности при увеличении числа потоков с 4 до 32 (типичный диапазон для многих приложений), эффективность масштабирования составляет: - MongoDB: 2.57x (наилучший показатель) - Cassandra: 2.78x (близко к идеальному для данного диапазона) - PostgreSQL: 1.52x (наименее эффективное масштабирование в этом диапазоне)

Эти результаты подтверждают репутацию NoSQL решений как систем с хорошей горизонтальной масштабируемостью, хотя необычное поведение PostgreSQL на высоких уровнях параллелизма заслуживает дополнительного внимания.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы были решены следующие задачи:

- исследованы архитектурные особенности и принципы работы реляционной (PostgreSQL), документоориентированной (MongoDB) и колоночной (Cassandra) СУБД;
- разработана и применена единая методология сравнительного тестирования производительности с использованием реального набора данных объемом более 12 ГБ и универсального бенчмарка YCSB;
- проведена комплексная серия тестов для оценки пропускной способности и задержек каждой СУБД при шести различных типах рабочих нагрузок и семи уровнях параллелизма;
- выполнен детальный сравнительный анализ масштабируемости систем при увеличении числа параллельных потоков;
- получены графики зависимостей, наглядно демонстрирующие сильные и слабые стороны каждой технологии;
- сформулированы практические рекомендации по выбору оптимальной СУБД для конкретных сценариев использования.

Проведённое исследование убедительно доказывает, что не существует универсально лучшей СУБД, а выбор должен основываться на преобладающем характере нагрузки приложения.

Apache Cassandra демонстрирует наивысшую пропускную способность для стандартных CRUD-операций и смешанных нагрузок (Workloads A, B, C, F), что делает её идеальным выбором для систем с высокой интенсивностью операций чтения и записи, где важна стабильность и низкие задержки.

В то же время PostgreSQL показывает исключительную, многократно превосходящую конкурентов производительность в сценариях чтения последних добавленных данных (Workload D). Это делает его оптимальным решением для приложений, работающих с "горячими" данными, таких как системы мониторинга, аналитики в реальном времени и логирования.

MongoDB, в свою очередь, является лидером в сценариях, требующих сканирования диапазонов данных (Workload E), и демонстрирует наиболее плавную и предсказуемую масштабируемость. Преимущества его документоориентированной модели и гибкой схемы делают его сильным кандидатом для приложений с разнообразными и эволюционирующими структурами данных.

Таким образом, выбор конкретной СУБД должен основываться не на общих представлениях о производительности, а на тщательном анализе преобладающих паттернов нагрузки в приложении и требований к масштабируемости. Полученные в работе результаты предоставляют объективную основу для принятия такого взвешенного решения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Прамодкумар Дж. Садаладж, Фаулер Мартин NoSQL. Новая методология разработки нереляционных баз данных; Вильямс - М., 2021.- 192 с.

2. Руководство по NoSQL для разработчиков [Электронный ресурс]. URL: https://javarush.ru/groups/posts/467-rukovodstvo-po-nosql-dlja-razrabotchikov (дата обращения: 10.09.2023).

3. Использование NoSQL [Электронный ресурс]. URL: https://www.oracle.com/ru/database/nosql/what-is-nosql/ (дата обращения: 11.09.2023).

4. Dan McCreary, Ann Kelly. Making Sense of NoSQL: A guide for managers and the rest of us. - Manning Publications, 2013. - 312 p. - ISBN 978-1-61729-107-4.

5. Shashank Tiwari. Professional NoSQL. - John Wiley & Sons Inc, 2011. — 384 p. - ISBN 9780470942246.

6. Редмонд Э., Уилсон Д. Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL = MongoDB in Action. — ДМК Пресс, 2013. — 384 с. — ISBN 978-5-94074-866-3.

7. Neo4j – Текст. Изображение: электронные // Neo4j: [сайт]. – URL: Дистрибутив и документация – https://neo4j.com/ (дата обращения: 19.04.2024).

8. Еремеев А.П., Панявин Н.А. Унификация модели представления данных и преобразование форматов на основе нереляционной СУБД Neo4j. – Программные продукты и системы / Software & Systems 4, 2022. – 549-556 с. – DOI: 10.15827/0236-235X.140.

9. Карпентер Дж., Хьюитт Э. Cassandra. Полное руководство = Cassandra: The Definitive Guide. — ДМК-Пресс, 2016. — 400 с. — ISBN 978-5-97060-453-3.

10. Lakshman A., Mali P. Cassandra - A Decentralized Structured Storage System. – ACM SIGOPS Operating Systems Review 44(2), 2010. – 35-40 с. – DOI:10.1145/1773912.1773922.

11. MongoDB – Краткое руководство. – Текст. Изображение : электронные // Уроки и статьи по программированию и IT: [сайт]. –

URL:https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo (дата обращения: 19.02.2024).

12. Chodorow Kristina. MongoDB: The Definitive Guide, 2013. – 216c. – ISBN-13 978-9351102694.

13. Что такое документная база данных? – Текст. Изображение : электронные // Сервисы облачных вычислений – Amazon Web Services (AWS): [сайт]. – URL: https://aws.amazon.com/ru/nosql/document/ (дата обращения: 19.02.2024).

14. Робинсон Я., Вебер Д., Эифрем Э. Графовые базы данных: новые возможности для работы со связанными данными / пер. с англ. Р.Н. Рагимова; науч. Ред. Кисилев А.Н. – 2-е изд. – М.: LVR Пресс, 2016. – 256 с.: ил. ISBN 978-5-97060- 201-0.

15. Fernandez J. R., El-Sheikh E. M. CluSandra: A Framework and Algorithm for Data Stream Cluster Analysis. – International Journal of Advanced Computer Science, Vol. 2, 2011. – 13 с. – DOI: 10.14569/IJACSA.2011.021115.

16. MongoDB Clusters. [Электронный ресурс] – URL: https://www.mongodb.com/resources/products/fundamentals/clusters (дата обращения: 05.04.2024).

17. Sahbudin M. A., Scarpa M. L., Serrano S. MongoDB Clustering using K-means for Real-Time Song Recognition. – Conference: IEEE 2019 International Conference on Computing, Networking and Communications (ICNC), 2019. – 350-354 с. – DOI:10.1109/ICCNC.2019.8685489.

18. Fotopoulos G., Koloveas P. Comparing Data Store Performance for Full-Text Search: to SQL or to NoSQL. – Conference, 2023. – 8 с. – DOI:10.5220/0012089200003541.

19. Citation Network Analysis. [Электронный ресурс] – URL: https://www.kaggle.com/code/virajjayant/citation-network-analysis/input (дата обращения: 05.01.2024).

20. Горячкин Б.С. Эргономический анализ систем обработки информации и управления // Вестник евразийской науки. 2017. Т. 9. №3. С. 72.

21. Елисеева Е.А., Горячкин Б.С., Виноградова М.В., Черненький М.В. Оценка времени выполнения поисковых запросов в nosql и объектно-реляционной базах данных. Динамика сложных систем - XXI век. 2022. Т. 16. № 2. С. 44-51.

22. Данилов А.М., Гарькина И.А. Интерполяция, аппроксимация, оптимизация: анализ и синтез сложных систем. Пенза.: ПГУАС. 2014.

23. Сравнительное исследование производительности графовой и документной СУБД Д.Ю. Уткин, Е. А. Кучин, Е.А. Елисеева, Г.И. Ревунков, сборник ИИАСУ’23, Москва, 27–28 апреля 2023 г.

24. Желтова А., Ваксина И.Р., Уткин Д.Ю., Варламов О.О. Автоматическое создание МБЗ с помощью выборки правил из текстовых инструкций с применением нейросети, МИВАР’24, 2024 г.

25. Колосов А. П., Богатырев М. Ю. Система полнотекстового поиска по длинным запросам. – SmartBear Software, 2016. – 6 с.

26. Eric W. Brown. Execution Performance Issues in Full-Text Information Retrieval. — University of Massachusetts Amherst: Computer Science Department, 1996. — 179 с. — (Technical Report 95-81).

27. Петрухин А. Н., Дворецкий А. Ю. Цифровой поиск как основа реализации словаря полнотекстовой базы данных // Вопросы радиоэлектроники. — Москва: Центральный научно-исследовательский институт экономики, систем управления и информации «Электроника. — ISSN 0233-9950.

28. Sonal R. Neo4j High Performance. — Packt Publishing Ltd, 2015. — ISBN 978-1-78355-516-1.

29. Gan G. Data clustering: theory, algorithms, and applications. Philadelphia, Pa. Alexandria, Va: SIAM, Society for Industrial and Applied Mathematics American Statistical Association. – 2007. - ISBN 9780898716238.

30. Everitt BS, Landau S., Leese M. Cluster Analysis (Fourth ed.). London: Arnold. - 2001. - ISBN 0-340-76119-9.

ПРИЛОЖЕНИЕ А

В приложение А выпускной квалификационной работы входят:

А.1. Общая сводная таблица, содержащая ключевые метрики производительности (пропускная способность и средние задержки) для всех трёх исследуемых СУБД (Cassandra, MongoDB, PostgreSQL) по всем рабочим нагрузкам и уровням параллелизма. Это позволяет быстро сравнить системы между собой в различных сценариях.

А.2. Детализированные таблицы для Cassandra

А.3. Детализированные таблицы для MongoDB

А.4. Детализированные таблицы для PostgreSQL

А.5. Сравнительная таблица пропускной способности: итоговая таблица, в которой для наглядности сопоставляются только показатели пропускной способности (ops/sec) всех трех СУБД.

А.1. Общая сводная таблица

Database	Workload	Threads	Throughput (ops/sec)	READ Latency (ms)	UPDATE Latency (ms)	INSERT Latency (ms)
Cassandra	A	4	11740	0.371	0.302	N/A
Cassandra	A	8	19423	0.439	0.37	N/A
Cassandra	A	16	28963	0.597	0.483	N/A
Cassandra	A	32	33925	1.131	0.704	N/A
Cassandra	A	64	26660	3.968	0.721	N/A
Cassandra	A	128	24440	9.567	0.688	N/A
Cassandra	A	256	24228	15.26	5.464	N/A
Cassandra	B	4	10738	0.369	0.36	N/A
Cassandra	B	8	18612	0.422	0.432	N/A
Cassandra	B	16	28562	0.545	0.577	N/A
Cassandra	B	32	37211	0.835	0.873	N/A
Cassandra	B	64	32696	1.966	0.8	N/A
Cassandra	B	128	30232	4.317	0.755	N/A
Cassandra	B	256	30940	8.291	4.391	N/A
Cassandra	C	4	10346	0.382	N/A	N/A
Cassandra	C	8	16894	0.466	N/A	N/A
Cassandra	C	16	28525	0.548	N/A	N/A
Cassandra	C	32	40100	0.775	N/A	N/A
Cassandra	C	64	38990	1.592	N/A	N/A
Cassandra	C	128	34921	3.571	N/A	N/A
Cassandra	C	256	34342	7.268	N/A	N/A
Cassandra	D	4	7689	0.509	N/A	0.467
Cassandra	D	8	10637	0.734	N/A	0.596
Cassandra	D	16	10959	1.445	N/A	0.812
Cassandra	D	32	6378	5.143	N/A	0.866
Cassandra	D	64	6146	10.711	N/A	0.779
Cassandra	D	128	5494	24.033	N/A	0.794
Cassandra	D	256	6420	40.03	N/A	19.86
Cassandra	E	4	2729	N/A	N/A	0.509
Cassandra	E	8	3039	N/A	N/A	0.652
Cassandra	E	16	2405	N/A	N/A	0.827
Cassandra	E	32	1667	N/A	N/A	0.702
Cassandra	E	64	1569	N/A	N/A	0.719
Cassandra	E	128	610	N/A	N/A	3.173
Cassandra	E	256	673	N/A	N/A	192.287
Cassandra	F	4	6721	0.411	0.355	N/A
Cassandra	F	8	11291	0.489	0.422	N/A
Cassandra	F	16	18319	0.609	0.502	N/A
Cassandra	F	32	19770	1.28	0.625	N/A
Cassandra	F	64	16357	3.564	0.591	N/A
Cassandra	F	128	14781	8.24	0.647	N/A

Cassandra	F	256	17245	12.097	5.101	N/A
MongoDB	A	4	6835	0.572	0.592	N/A
MongoDB	A	8	9818	0.797	0.824	N/A
MongoDB	A	16	10560	1.492	1.522	N/A
MongoDB	A	32	11900	2.661	2.702	N/A
MongoDB	A	64	13358	4.042	5.489	N/A
MongoDB	A	128	13174	6.801	12.512	N/A
MongoDB	A	256	12847	10.598	28.941	N/A
MongoDB	B	4	8060	0.489	0.573	N/A
MongoDB	B	8	13048	0.605	0.697	N/A
MongoDB	B	16	16621	0.953	1.054	N/A
MongoDB	B	32	16463	1.929	2.058	N/A
MongoDB	B	64	19918	3.191	3.317	N/A
MongoDB	B	128	20892	6.079	6.224	N/A
MongoDB	B	256	20660	12.5	8.395	N/A
MongoDB	C	4	8876	0.448	N/A	N/A
MongoDB	C	8	14528	0.548	N/A	N/A
MongoDB	C	16	19670	0.809	N/A	N/A
MongoDB	C	32	18946	1.681	N/A	N/A
MongoDB	C	64	23146	2.75	N/A	N/A
MongoDB	C	128	25199	5.043	N/A	N/A
MongoDB	C	256	23159	10.94	N/A	N/A
MongoDB	D	4	17555	0.224	N/A	0.366
MongoDB	D	8	27770	0.286	N/A	0.471
MongoDB	D	16	51501	0.303	N/A	0.551
MongoDB	D	32	60366	0.514	N/A	0.922
MongoDB	D	64	62366	0.952	N/A	1.811
MongoDB	D	128	60238	1.953	N/A	3.664
MongoDB	D	256	53974	4.431	N/A	8.167
MongoDB	E	4	10029	N/A	N/A	0.371
MongoDB	E	8	16808	N/A	N/A	0.462
MongoDB	E	16	27159	N/A	N/A	0.622
MongoDB	E	32	29371	N/A	N/A	1.299
MongoDB	E	64	28851	N/A	N/A	2.836
MongoDB	E	128	28495	N/A	N/A	5.62
MongoDB	E	256	26899	N/A	N/A	11.244
MongoDB	F	4	6979	0.431	0.277	N/A
MongoDB	F	8	10637	0.563	0.369	N/A
MongoDB	F	16	12615	1.025	0.473	N/A
MongoDB	F	32	12701	2.188	0.642	N/A
MongoDB	F	64	14508	3.346	2.077	N/A
MongoDB	F	128	14022	5.068	7.984	N/A
MongoDB	F	256	13342	6.473	25.07	N/A
PostgreSQL	A	4	6704	0.412	0.776	N/A
PostgreSQL	A	8	8309	0.592	1.326	N/A
PostgreSQL	A	16	8039	1.081	2.889	N/A
PostgreSQL	A	32	8334	1.963	5.7	N/A
PostgreSQL	A	64	8732	3.157	11.449	N/A

PostgreSQL	A	128	9008	6.48	21.816	N/A
PostgreSQL	A	256	7282	26.127	43.848	N/A
PostgreSQL	B	4	10057	0.379	0.691	N/A
PostgreSQL	B	8	13865	0.55	1.023	N/A
PostgreSQL	B	16	14086	1.067	2.341	N/A
PostgreSQL	B	32	14414	2.045	5.394	N/A
PostgreSQL	B	64	17975	3.092	12.103	N/A
PostgreSQL	B	128	17980	6.418	19.346	N/A
PostgreSQL	B	256	11594	21.071	38.324	N/A
PostgreSQL	C	4	11308	0.352	N/A	N/A
PostgreSQL	C	8	17807	0.447	N/A	N/A
PostgreSQL	C	16	18025	0.884	N/A	N/A
PostgreSQL	C	32	16533	1.929	N/A	N/A
PostgreSQL	C	64	24512	2.595	N/A	N/A
PostgreSQL	C	128	22024	5.778	N/A	N/A
PostgreSQL	C	256	19197	13.192	N/A	N/A
PostgreSQL	D	4	18870	0.199	N/A	0.41
PostgreSQL	D	8	29945	0.247	N/A	0.585
PostgreSQL	D	16	30201	0.49	N/A	1.212
PostgreSQL	D	32	29368	0.965	N/A	3.304
PostgreSQL	D	64	40828	1.29	N/A	6.481
PostgreSQL	D	128	129782	0.619	N/A	7.184
PostgreSQL	D	256	111267	1.205	N/A	19.847
PostgreSQL	E	4	5842	N/A	N/A	0.613
PostgreSQL	E	8	8346	N/A	N/A	0.797
PostgreSQL	E	16	12257	N/A	N/A	1.469
PostgreSQL	E	32	11182	N/A	N/A	3.965
PostgreSQL	E	64	10510	N/A	N/A	8.525
PostgreSQL	E	128	9610	N/A	N/A	16.595
PostgreSQL	E	256	9574	N/A	N/A	29.719
PostgreSQL	F	4	5289	0.482	0.54	N/A
PostgreSQL	F	8	7038	0.704	0.857	N/A
PostgreSQL	F	16	8236	1.04	1.79	N/A
PostgreSQL	F	32	8104	1.966	3.943	N/A
PostgreSQL	F	64	8257	3.431	8.594	N/A
PostgreSQL	F	128	8436	7.175	15.868	N/A
PostgreSQL	F	256	6457	28.262	22.454	N/A

А.2. Детализированные таблицы для Cassandra

Cassandra - Workload A Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	11740.0	416881.0	0.371	0.528	0.302	0.398	0.763
8.0	19423.0	251974.0	0.439	0.591	0.37	0.509	0.224
16.0	28963.0	168979.0	0.597	0.982	0.483	0.819	0.256
32.0	33925.0	144260.0	1.131	2.789	0.704	1.511	0.358
64.0	26660.0	183574.0	3.968	16.143	0.721	1.566	0.685
128.0	24440.0	200252.0	9.567	20.223	0.688	1.415	0.341
256.0	24228.0	202003.0	15.26	28.127	5.464	7.939	0.612

Cassandra - Workload B Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	10738.0	455775.0	0.369	0.552	0.36	0.516	0.588
8.0	18612.0	262960.0	0.422	0.592	0.432	0.605	0.246
16.0	28562.0	171350.0	0.545	0.902	0.577	0.947	0.593
32.0	37211.0	131522.0	0.835	1.912	0.873	1.781	0.53
64.0	32696.0	149685.0	1.966	10.191	0.8	1.803	0.42
128.0	30232.0	161885.0	4.317	14.335	0.755	1.602	0.665
256.0	30940.0	158180.0	8.291	17.567	4.391	5.903	0.681

Cassandra - Workload C Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	10346.0	473025.0	0.382	0.599	N/A	N/A	0.448
8.0	16894.0	289698.0	0.466	0.655	N/A	N/A	0.246
16.0	28525.0	171573.0	0.548	0.922	N/A	N/A	0.569
32.0	40100.0	122046.0	0.775	1.757	N/A	N/A	0.616
64.0	38990.0	125522.0	1.592	8.131	N/A	N/A	0.954
128.0	34921.0	140148.0	3.571	10.519	N/A	N/A	0.572
256.0	34342.0	142512.0	7.268	16.191	N/A	N/A	0.742

Cassandra - Workload D Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	7689.0	130048.0	0.509	0.913	N/A	N/A	0.14
8.0	10637.0	94008.0	0.734	1.511	N/A	N/A	0.179
16.0	10959.0	91248.0	1.445	3.661	N/A	N/A	0.446
32.0	6378.0	156784.0	5.143	12.695	N/A	N/A	0.426
64.0	6146.0	162707.0	10.711	19.839	N/A	N/A	0.296
128.0	5494.0	182011.0	24.033	35.263	N/A	N/A	0.35
256.0	6420.0	155755.0	40.03	53.055	N/A	N/A	0.492

Cassandra - Workload E Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	2729.0	1162236.0	N/A	N/A	N/A	N/A	0.3
8.0	3039.0	1162278.0	N/A	N/A	N/A	N/A	0.108
16.0	2405.0	1162252.0	N/A	N/A	N/A	N/A	0.139
32.0	1667.0	1162285.0	N/A	N/A	N/A	N/A	0.151
64.0	1569.0	1162331.0	N/A	N/A	N/A	N/A	0.173
128.0	610.0	1163520.0	N/A	N/A	N/A	N/A	0.036
256.0	673.0	1162903.0	N/A	N/A	N/A	N/A	0.041

Cassandra - Workload F Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	6721.0	728191.0	0.411	0.604	0.355	0.516	0.387
8.0	11291.0	433466.0	0.489	0.679	0.422	0.595	0.214
16.0	18319.0	267161.0	0.609	1.003	0.502	0.845	0.305
32.0	19770.0	247556.0	1.28	3.557	0.625	1.196	0.35
64.0	16357.0	299195.0	3.564	11.311	0.591	1.102	0.267
128.0	14781.0	331103.0	8.24	16.167	0.647	1.239	0.324
256.0	17245.0	283789.0	12.097	20.943	5.101	6.491	0.653

А.3. Детализированные таблицы для MongoDB

MongoDB - Workload A Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	6835.0	715987.0	0.572	1.038	0.592	1.068	0.73
8.0	9818.0	498458.0	0.797	1.798	0.824	1.836	0.344
16.0	10560.0	463464.0	1.492	3.719	1.522	3.775	0.438
32.0	11900.0	411259.0	2.661	7.367	2.702	7.435	0.521
64.0	13358.0	366381.0	4.042	12.599	5.489	19.087	0.628
128.0	13174.0	371506.0	6.801	22.863	12.512	45.311	0.759
256.0	12847.0	380951.0	10.598	41.567	28.941	90.239	0.777

MongoDB - Workload B Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	8060.0	607181.0	0.489	0.827	0.573	0.937	0.886
8.0	13048.0	375083.0	0.605	1.079	0.697	1.198	0.393
16.0	16621.0	294447.0	0.953	2.285	1.054	2.421	0.664
32.0	16463.0	297273.0	1.929	4.887	2.058	5.063	0.632
64.0	19918.0	245707.0	3.191	9.271	3.317	9.423	0.991
128.0	20892.0	234259.0	6.079	18.351	6.224	18.591	1.181
256.0	20660.0	236889.0	12.5	31.071	8.395	23.791	1.182

MongoDB - Workload C Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	8876.0	551360.0	0.448	0.761	N/A	N/A	1.077
8.0	14528.0	336864.0	0.548	0.983	N/A	N/A	0.397
16.0	19670.0	248806.0	0.809	2.027	N/A	N/A	0.722
32.0	18946.0	258322.0	1.681	4.447	N/A	N/A	0.811
64.0	23146.0	211446.0	2.75	8.027	N/A	N/A	1.101
128.0	25199.0	194216.0	5.043	15.791	N/A	N/A	1.449
256.0	23159.0	211322.0	10.94	28.143	N/A	N/A	1.554

MongoDB - Workload D Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	17555.0	278780.0	0.224	0.336	N/A	N/A	1.192
8.0	27770.0	176235.0	0.286	0.425	N/A	N/A	0.396
16.0	51501.0	95029.0	0.303	0.6	N/A	N/A	0.729
32.0	60366.0	81073.0	0.514	1.109	N/A	N/A	1.064
64.0	62366.0	78473.0	0.952	1.965	N/A	N/A	1.184
128.0	60238.0	81246.0	1.953	4.027	N/A	N/A	1.303
256.0	53974.0	90674.0	4.431	10.055	N/A	N/A	1.511

MongoDB - Workload E Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	10029.0	487992.0	N/A	N/A	N/A	N/A	2.909
8.0	16808.0	291184.0	N/A	N/A	N/A	N/A	1.032
16.0	27159.0	180204.0	N/A	N/A	N/A	N/A	1.191
32.0	29371.0	166627.0	N/A	N/A	N/A	N/A	1.254
64.0	28851.0	169632.0	N/A	N/A	N/A	N/A	1.543
128.0	28495.0	171755.0	N/A	N/A	N/A	N/A	1.915
256.0	26899.0	181946.0	N/A	N/A	N/A	N/A	1.698

MongoDB - Workload F Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	6979.0	701280.0	0.431	0.747	0.277	0.387	1.742
8.0	10637.0	460089.0	0.563	1.175	0.369	0.533	0.601
16.0	12615.0	387958.0	1.025	2.641	0.473	0.922	0.688
32.0	12701.0	385345.0	2.188	5.867	0.642	1.572	0.735
64.0	14508.0	337332.0	3.346	10.127	2.077	10.639	0.871
128.0	14022.0	349027.0	5.068	15.767	7.984	34.783	1.061
256.0	13342.0	366808.0	6.473	20.415	25.07	108.479	1.109

А.4. Детализированные таблицы для PostgreSQL

PostgreSQL - Workload A Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	6704.0	730035.0	0.412	0.767	0.776	1.464	0.343
8.0	8309.0	588975.0	0.592	1.148	1.326	2.473	0.149
16.0	8039.0	608781.0	1.081	2.187	2.889	4.631	0.047
32.0	8334.0	587237.0	1.963	3.935	5.7	8.591	0.056
64.0	8732.0	560485.0	3.157	6.427	11.449	24.479	0.096
128.0	9008.0	543333.0	6.48	14.711	21.816	53.279	0.114
256.0	7282.0	672054.0	26.127	60.735	43.848	108.799	0.112

PostgreSQL - Workload B Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	10057.0	486626.0	0.379	0.652	0.691	1.086	0.468
8.0	13865.0	352989.0	0.55	1.013	1.023	1.76	0.094
16.0	14086.0	347435.0	1.067	2.053	2.341	4.267	0.069
32.0	14414.0	339528.0	2.045	3.991	5.394	9.335	0.113
64.0	17975.0	272276.0	3.092	6.867	12.103	21.263	0.159
128.0	17980.0	272191.0	6.418	14.775	19.346	38.879	0.183
256.0	11594.0	422114.0	21.071	48.479	38.324	76.607	0.167

PostgreSQL - Workload C Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	11308.0	432782.0	0.352	0.616	N/A	N/A	0.502
8.0	17807.0	274846.0	0.447	0.773	N/A	N/A	0.286
16.0	18025.0	271523.0	0.884	1.664	N/A	N/A	0.08
32.0	16533.0	296024.0	1.929	3.711	N/A	N/A	0.112
64.0	24512.0	199663.0	2.595	5.839	N/A	N/A	0.162
128.0	22024.0	222217.0	5.778	13.447	N/A	N/A	0.188
256.0	19197.0	254945.0	13.192	31.087	N/A	N/A	0.175

PostgreSQL - Workload D Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	18870.0	259356.0	0.199	0.445	N/A	N/A	1.023
8.0	29945.0	163435.0	0.247	0.632	N/A	N/A	0.245
16.0	30201.0	162052.0	0.49	1.526	N/A	N/A	0.162
32.0	29368.0	166645.0	0.965	3.509	N/A	N/A	0.212
64.0	40828.0	119870.0	1.29	5.071	N/A	N/A	0.338
128.0	129782.0	37710.0	0.619	1.725	N/A	N/A	1.196
256.0	111267.0	43985.0	1.205	3.727	N/A	N/A	1.075

PostgreSQL - Workload E Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	5842.0	837774.0	N/A	N/A	N/A	N/A	0.791
8.0	8346.0	586382.0	N/A	N/A	N/A	N/A	0.695
16.0	12257.0	399276.0	N/A	N/A	N/A	N/A	1.147
32.0	11182.0	437671.0	N/A	N/A	N/A	N/A	1.104
64.0	10510.0	465674.0	N/A	N/A	N/A	N/A	1.48
128.0	9610.0	509263.0	N/A	N/A	N/A	N/A	1.578
256.0	9574.0	511174.0	N/A	N/A	N/A	N/A	2.056

PostgreSQL - Workload F Detailed Performance

Threads	Throughput (ops/sec)	Runtime (ms)	READ Avg Latency (ms)	READ 95th Latency (ms)	UPDATE Avg Latency (ms)	UPDATE 95th Latency (ms)	GC Time (%)
4.0	5289.0	925285.0	0.482	0.89	0.54	1.092	0.416
8.0	7038.0	695374.0	0.704	1.406	0.857	1.699	0.15
16.0	8236.0	594252.0	1.04	2.063	1.79	2.993	0.055
32.0	8104.0	603893.0	1.966	3.845	3.943	5.795	0.093
64.0	8257.0	592698.0	3.431	6.803	8.594	12.975	0.121
128.0	8436.0	580134.0	7.175	15.751	15.868	40.543	0.155
256.0	6457.0	757950.0	28.262	62.751	22.454	71.679	0.125

А.5. Сравнительная таблица пропускной способности: итоговая таблица, в которой для наглядности сопоставляются только показатели пропускной способности (ops/sec) всех трех СУБД.

Workload	Threads	Cassandra	MongoDB	PostgreSQL
A	4	11740	6835	6704
A	8	19423	9818	8309
A	16	28963	10560	8039
A	32	33925	11900	8334
A	64	26660	13358	8732
A	128	24440	13174	9008
A	256	24228	12847	7282
B	4	10738	8060	10057
B	8	18612	13048	13865
B	16	28562	16621	14086
B	32	37211	16463	14414
B	64	32696	19918	17975
B	128	30232	20892	17980
B	256	30940	20660	11594
C	4	10346	8876	11308
C	8	16894	14528	17807
C	16	28525	19670	18025
C	32	40100	18946	16533
C	64	38990	23146	24512
C	128	34921	25199	22024
C	256	34342	23159	19197
D	4	7689	17555	18870
D	8	10637	27770	29945
D	16	10959	51501	30201
D	32	6378	60366	29368
D	64	6146	62366	40828
D	128	5494	60238	129782
D	256	6420	53974	111267
E	4	2729	10029	5842
E	8	3039	16808	8346
E	16	2405	27159	12257
E	32	1667	29371	11182
E	64	1569	28851	10510
E	128	610	28495	9610
E	256	673	26899	9574
F	4	6721	6979	5289
F	8	11291	10637	7038
F	16	18319	12615	8236
F	32	19770	12701	8104

F	64	16357	14508	8257
F	128	14781	14022	8436
F	256	17245	13342	6457

ПРИЛОЖЕНИЕ В

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

Утверждаю
Научный руководитель
_____ Виноградова М.В.
"__"_____2025 г.

Исследование технологий поиска в кластерных базах данных

Техническое задание
(вид документа)

писчая бумага
(вид носителя)

4
(количество листов)

ИСПОЛНИТЕЛЬ:

_____ Кучин Елисей
Анатольевич
"__"_____2025 г.

Москва – 2025

1. Наименование

Исследование производительности постреляционных баз данных с применением технологий тестирования.

2. Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу, подписанное руководителем выпускной работы и утверждённое заведующим кафедрой ИУ5 МГТУ им. Н.Э. Баумана.

3. Исполнитель

Студент второго курса магистратуры группы ИУ5-44М Кучин Е.А.

4. Цель работы

Целью работы является проведение комплексного сравнительного исследования производительности и масштабируемости постреляционных СУБД (MongoDB, Cassandra) и реляционной СУБД (PostgreSQL). Исследование направлено на выявление их сильных и слабых сторон при обработке больших объемов данных в различных сценариях использования, а также на формирование практических рекомендаций по выбору оптимальной технологии в зависимости от характера нагрузки.

5. Требования к решаемым задачам

5.1 Задачи

5.1.1. Исследовать архитектурные особенности СУБД PostgreSQL, MongoDB и Cassandra.

5.1.2. Провести сравнительный анализ технологий тестирования СУБД и обосновать выбор инструментария для исследования.

5.1.3. Подготовить и проанализировать реальный набор данных объёмом 12 ГБ для проведения тестов.

5.1.4. Разработать и настроить единое тестовое окружение для обеспечения сопоставимости результатов.

5.1.5. Разработать методики загрузки и подготовки данных для каждой из исследуемых СУБД.

5.1.6. Оптимизировать конфигурации СУБД для достижения высокой производительности.

5.1.7. Разработать методику проведения экспериментов с использованием бенчмарка YCSB.

5.1.8. Провести серию экспериментов по оценке производительности (пропускная способность, задержки) и масштабируемости.

5.1.9. Разработать скрипт для автоматизированного сбора и агрегации результатов тестирования.

5.1.10. Провести сравнительный анализ и визуализацию полученных результатов.

5.1.11. Сформулировать выводы и практические рекомендации по выбору СУБД.

5.1.12. Оформить техническую документацию по результатам работы.

5.2 Требования к функциональным характеристикам

Разработанный комплекс средств и методик должен обеспечивать выполнение следующих функций::

5.2.1. Загрузка исходного набора данных в СУБД PostgreSQL, MongoDB и Cassandra.

5.2.2. Выполнение стандартизированных тестов производительности (workloads YCSB) на каждой из СУБД при различных уровнях параллелизма.

5.2.3. Сбор и агрегация метрик производительности (пропускная способность, задержки операций) и системных показателей.

5.2.4. Представление результатов в виде, пригодном для анализа и визуализации (таблицы, графики).

5.3 Требования к входным и выходным данным

5.3.1 Требования к входным данным

- Исходный набор данных в формате JSON, содержащий метаданные научных публикаций (объем >12 ГБ).
- Конфигурационные файлы для бенчмарка YCSB, определяющие параметры тестов (тип нагрузки, количество потоков, распределение запросов и т.д.).

5.3.2 Требования к выходным данным

Структурированные данные (в формате CSV) с результатами тестов, включающие метрики пропускной способности (ops/sec) и задержек операций (мс) для каждой СУБД, нагрузки и уровня параллелизма.

Визуализации (графики, диаграммы), иллюстрирующие сравнительную производительность и масштабируемость систем.

6. Этапы работы

График выполнения отдельных этапов работ приведен в соответствии с приказом об организации учебного процесса в 2023/2025 учебном году.

Таблица 1: Этапы разработки

№	Наименование этапа и содержание работ	Сроки исполнения
1	Разработка и утверждение ТЗ	Февраль 2025 г.
2	Исследование предметной области	Февраль — Март 2025 г.
3	Подготовка данных и настройка тестового окружения	Март 2025 г.
4	Разработка и реализация тестовых сценариев	Март — Апрель 2025 г.
5	Проведение экспериментов и сбор результатов	Апрель — Май 2025 г.
6	Анализ результатов и оформление документации	Май — Июнь 2025 г.
7	Защита работы	Июнь 2025 г.

7. Техническая документация

По окончании работы предъявляется следующая техническая документация:

1. Техническое задание;
2. Расчётно-пояснительная записка;
3. Графический материал по проекту в формате презентации.

8. Порядок приема работы

Приём и контроль программного изделия осуществляется в соответствие с методикой испытаний.

9. Дополнительные условия

Данное техническое задание может уточняться в установленном порядке.