Имитационная модель оптимизации занятости сервера БД

Никулин Д. Д., Корецкий К. В.

В работе рассмотрено использование имитационного моделирования для анализа занятости серверов баз данных (БД) в условиях многопользовательской нагрузки. Представлена разработанная модель, оценивающая производительность серверов БД при различных параметрах нагрузки, таких как количество запросов в секунду (RPS), использование индексации и таймаутов в очереди. Эксперименты выявили ключевые факторы, влияющие на производительность системы, и предоставили рекомендации для оптимизации ресурсов серверов БД.

Ключевые слова: сервер баз данных, имитационное моделирование, оптимизация производительности, многопользовательские системы, численное моделирование, Python, JavaScript.

Введение

В настоящий момент в Российской Федерации реализуется масштабный проект по созданию «цифровой экономики». Достижение проектных планов требует разработки новых методик управления субъектами экономической деятельности – социотехническими системами (СТС) страны, которые основаны на цифровых моделях деятельности и их дальнейшей реализации.

Разработка нормативного обеспечения поэтапного перехода к цифровой экономике требует от участников экономической деятельности системно-инженерного подхода и унификации широкого спектра применяемых решений в области современных технологий процессного управления на основе математического моделирования. Создание такой нормативной базы позволит создать унифицированное цифровое информативное пространство (УЦИП) для решения задач государственного управления и мониторинга деятельности СТС, обеспечить внедрение современной интеллектуальной модели управления.

Современный уровень развития информационных технологий (ИТ) позволяет и предопределяет использование методов численного моделирования (ЧМ) динамических рабочих процессов (потоков работ) в социотехнических системах (СТС) [3]. Для СТС численное моделирование является единственной возможностью оценки качества системы и прогнозирования ее взаимодействия с внешней средой. ЧМ позволяет существенно сократить

затраты в ходе анализа, оценки, модификации и реализации архитектуры деятельности СТС.

Методология

Для анализа производительности системы использовался метод имитационного моделирования. Основные этапы включали:

1. Разработку модели, имитирующей поведение серверов баз данных при обработке пользовательских запросов.
2. Введение параметров нагрузки, таких как количество пользователей, частота запросов и сложность операций.
3. Проведение экспериментов для оценки времени отклика и уровня занятости сервера.

Имитационная модель была реализована с использованием языка Python и библиотеки SimPy, а также веб версия на JavaScript и библиотекой chart.js. Для упрощения анализа система была сведена к следующей структуре:

- Сервер обрабатывает запросы в порядке очереди (алгоритм FIFO).
- Для каждого запроса рассчитывается время обработки, зависящее от сложности операции.
- В систему включен механизм моделирования узких мест, таких как ограничение числа подключений к серверу.

Базовая модель

В ходе разработки модели были реализованы следующие функции:

– Основной блок программы:

Получает количество пользователей симуляции и RPS (для более корректного и удобного проведения эксперимента количество пользователей задавалось через интерфейс).

Запускает симуляцию

Визуализирует результаты выполнения

– Основной блок программы:

Получает количество пользователей симуляции и RPS (для более корректного и удобного проведения эксперимента количество пользователей задавалось через интерфейс).

Запускает симуляцию

Визуализирует результаты выполнения

– Функция пользовательских запросов:

Описывает процесс для одного запроса

Сохраняет время поступления запроса в очередь

Увеличивает длину очереди и сохраняет изменения

Пытается занять ресурс сервера БД, если запрос превышает допустимое время ожидания, то в одной из видов моделирования создается ошибка. Если запрос попал в обработку, то он вычитается из очереди

– Функция генерации запросов (рис. 2):

Генерирует фиксированное число запросов

Задает интервал между запросов

Генерирует уникальные идентификаторы запросов

Запускает запросы в обработку

– Функция запуска симуляции (рис. 3):

Создает среду симуляции

Создает объект сервера с определенной емкостью

Добавляет процесс генерации запросов с заданным числом пользователей

Запускает симуляцию на заданное время

По завершении выводится общее число ошибок и успешно обработанных запросов

– Функция визуализации результатов (рис. 4):

Строит график изменения запросов в секунду, занятость серверов БД, время ожидания в очереди, а также эффекты влияние индексации, выполнения хранимых функций, триггеров и кэшей на стороне сервера БД

– Сервер обработки запросов

Ссылается на симуляционную среду для синхронизации событий во времени.

Определяет пул серверов с заданной емкостью и конфигурацией.

Показывает текущую длину очереди запросов для отслеживания нагрузки.

Структура взаимодействий пользователя и сервера показана на рис. 1

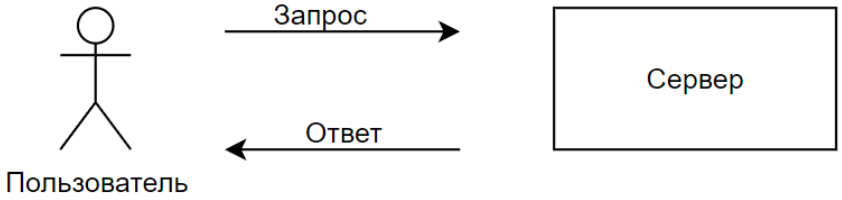

Рисунок 1 – структурная схема процесса функционирования

На рис. 2 представлена структура схемы модели системы

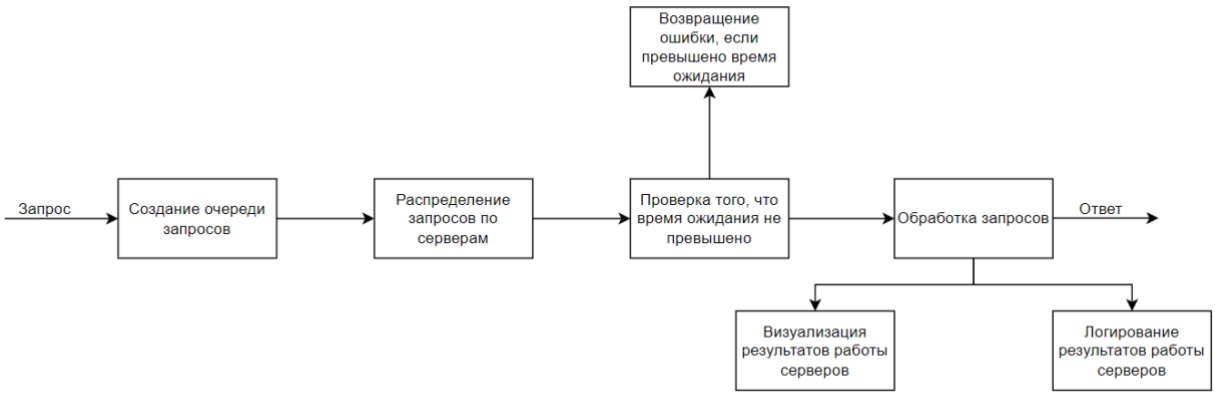

Рисунок 2 – структурная схема модели системы

Результаты

Были проведены эксперименты работоспособности системы:

- Запросы с использованием индекса: (рис. 3)

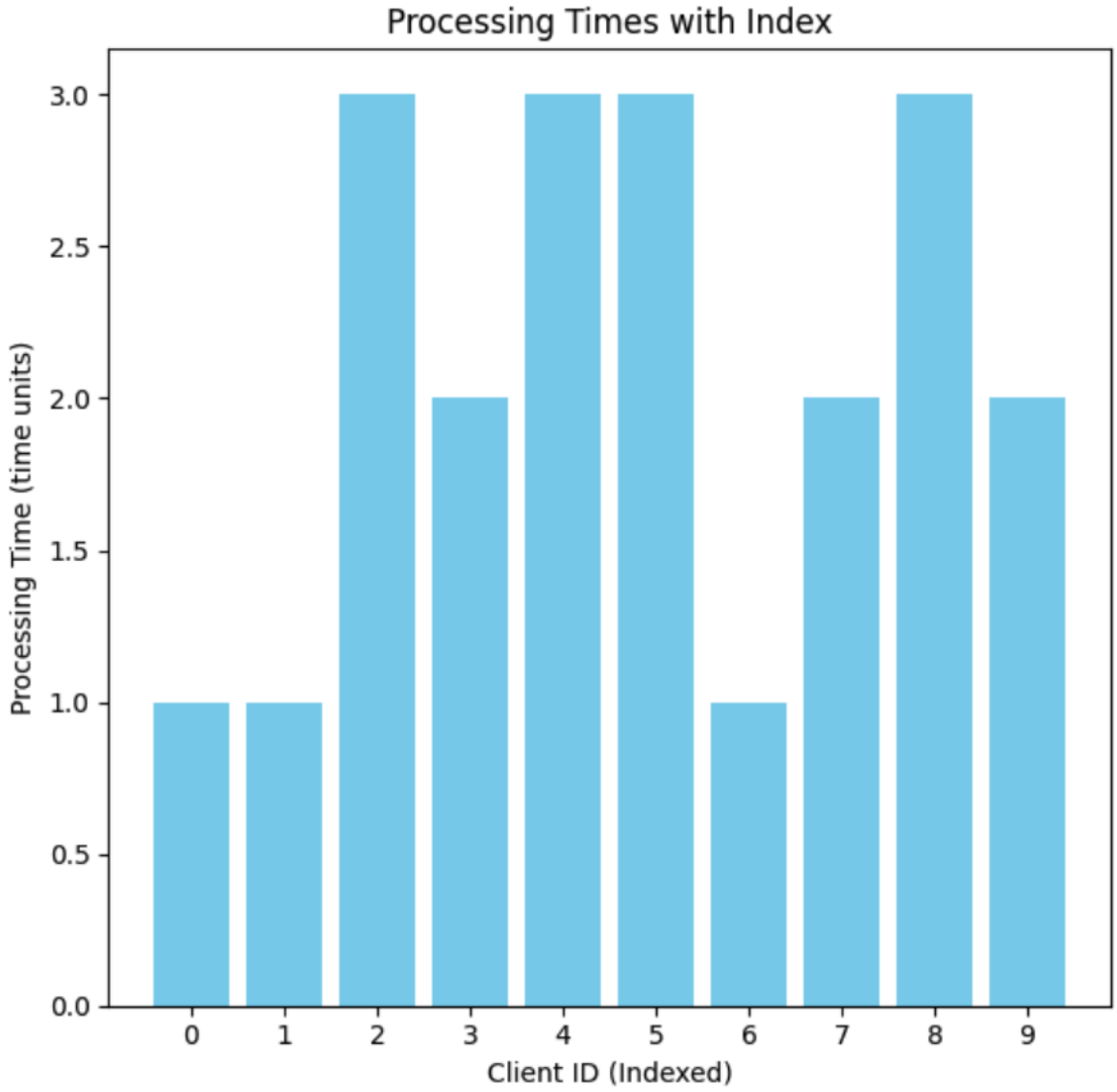

Рисунок 3 – эксперимент для 10 пользователей с индеком

Видим, что для 10 пользователей среднее время ожидания располагается в пределах случайной ошибки.

- Запросы без использования индекса: (рис. 4)

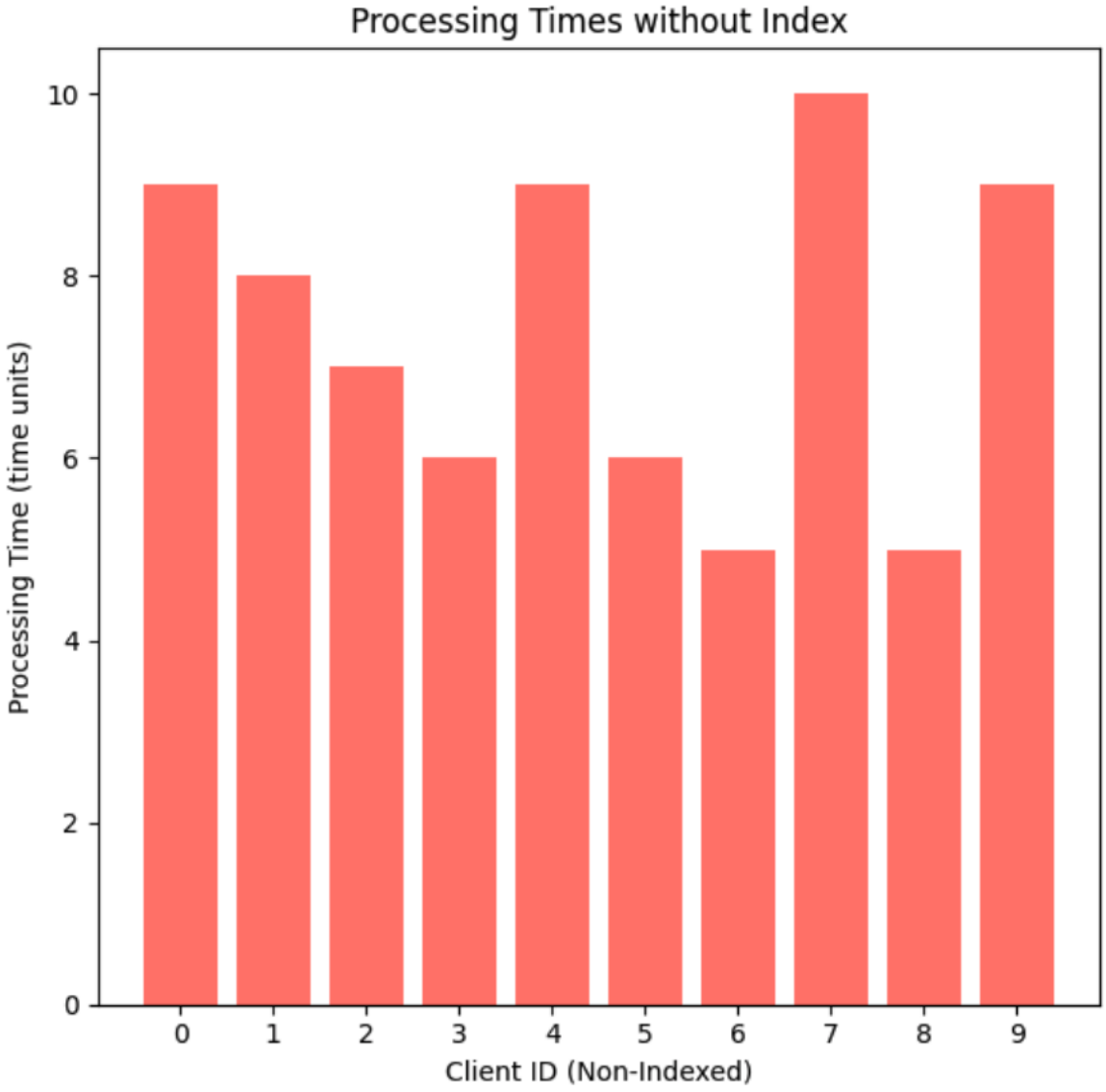

Рисунок 4 – эксперимент для пользователей без индекса

По диаграмме видно, что без использования индексации среднее время запросов для пользователей выросло и так же имеет равномерное распределение.

- Сравнение времени выполнения в зависимости от индексов (рис. 5)

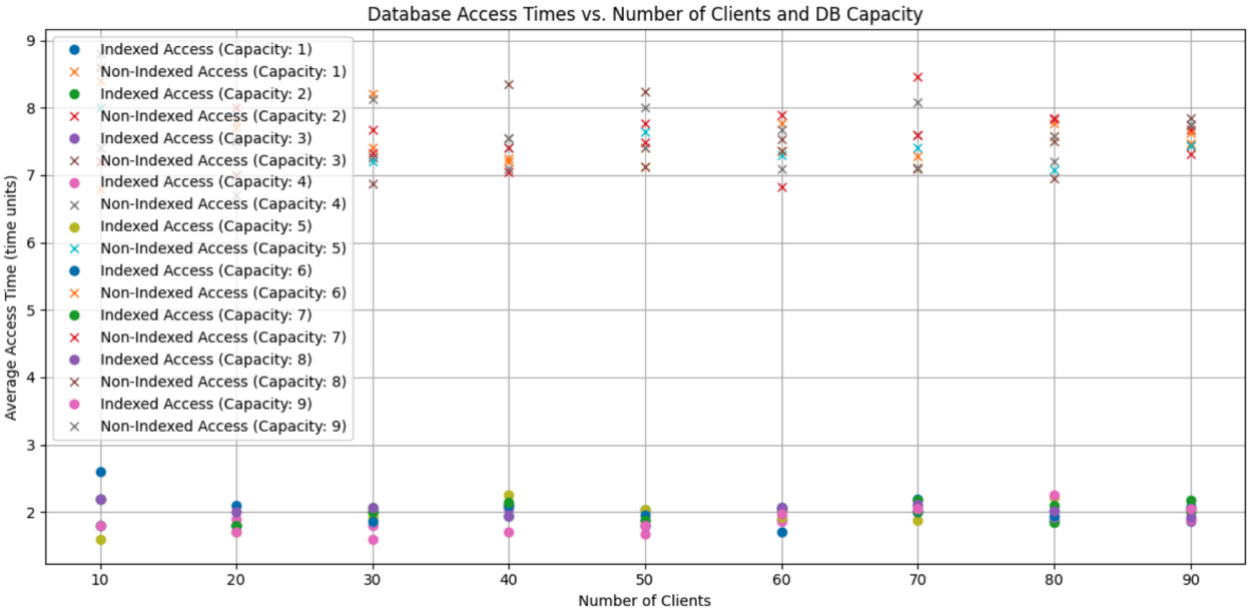

Рисунок 5 – эксперимент анализа эффективности индексации

Видно, что вне зависимости от количества пользователей, запросы с использованием индексом выполняются в несколько раз быстрее.

- Использование таймаута в очереди (рис. 6)

Для RPS=10 и квантилем выполнения 95%:

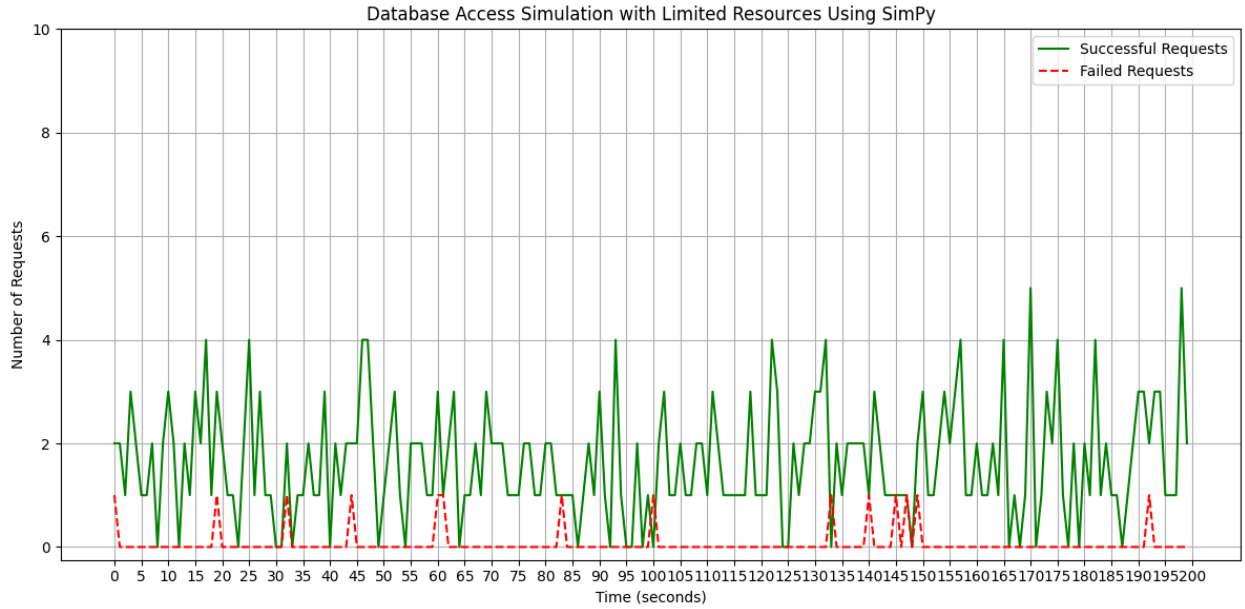

Рисунок 6 - эксперимент выполнения квантиля 95%

- Использование таймаута в очереди (рис. 7)

Для RPS=10 и квантилем выполнения 70%:

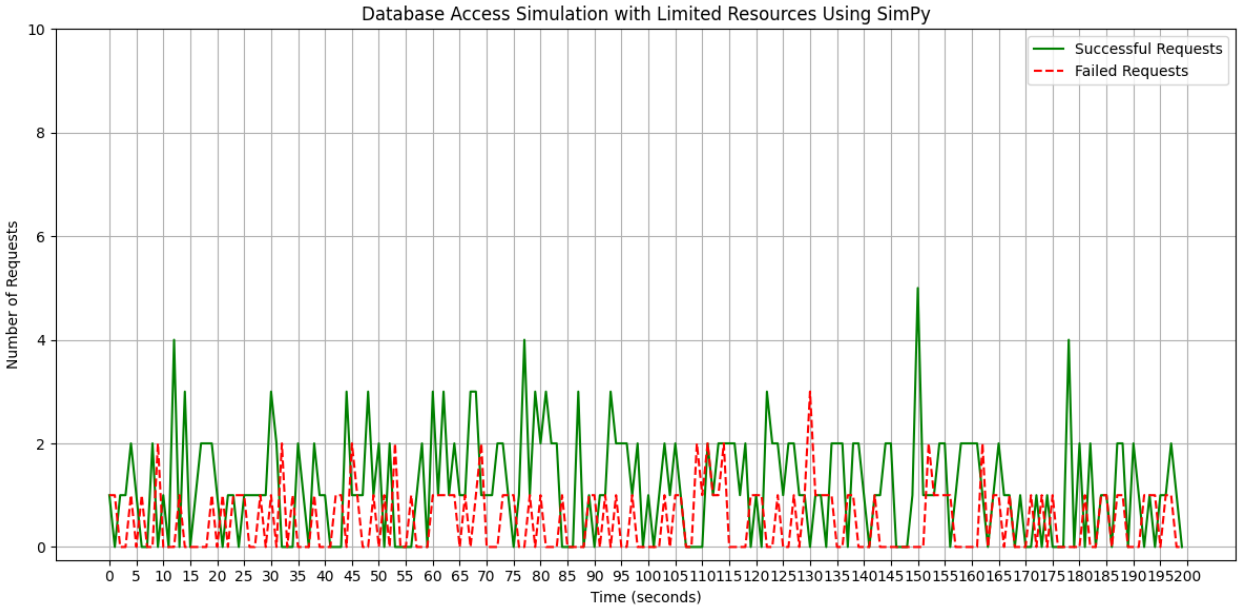

Рисунок 7 - эксперимент выполнения квантиля 70%

Из графиков использования таймаута в очереди видно, что пиковая нагрузка на сервер БД падает при наличии таймаута в очереди. Использование таймаута может повлиять на эффективность обработки запросов, т.к. конечная нагрузка становится ниже, а время ожидания в Эта гипотеза доказывается путем моделирования так же графика времени ожидания.

- Использование таймаута в очереди (рис. 8)

Для RPS=10 и квантилем выполнения 95%:

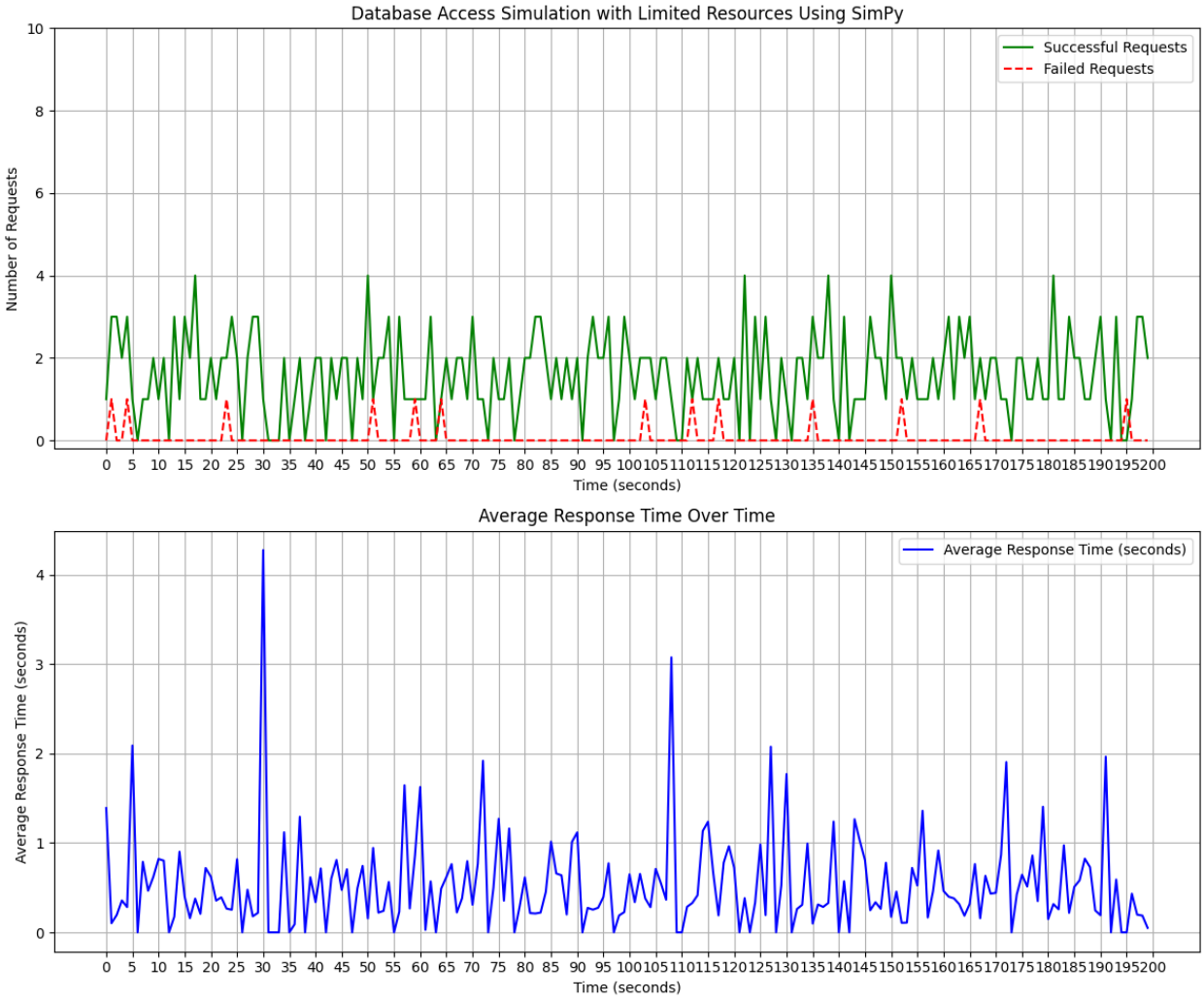

Рисунок 8 - эксперимент выполнения квантиля 95%

- Использование таймаута в очереди (рис. 9)

Для RPS=10 и квантилем выполнения 20%:

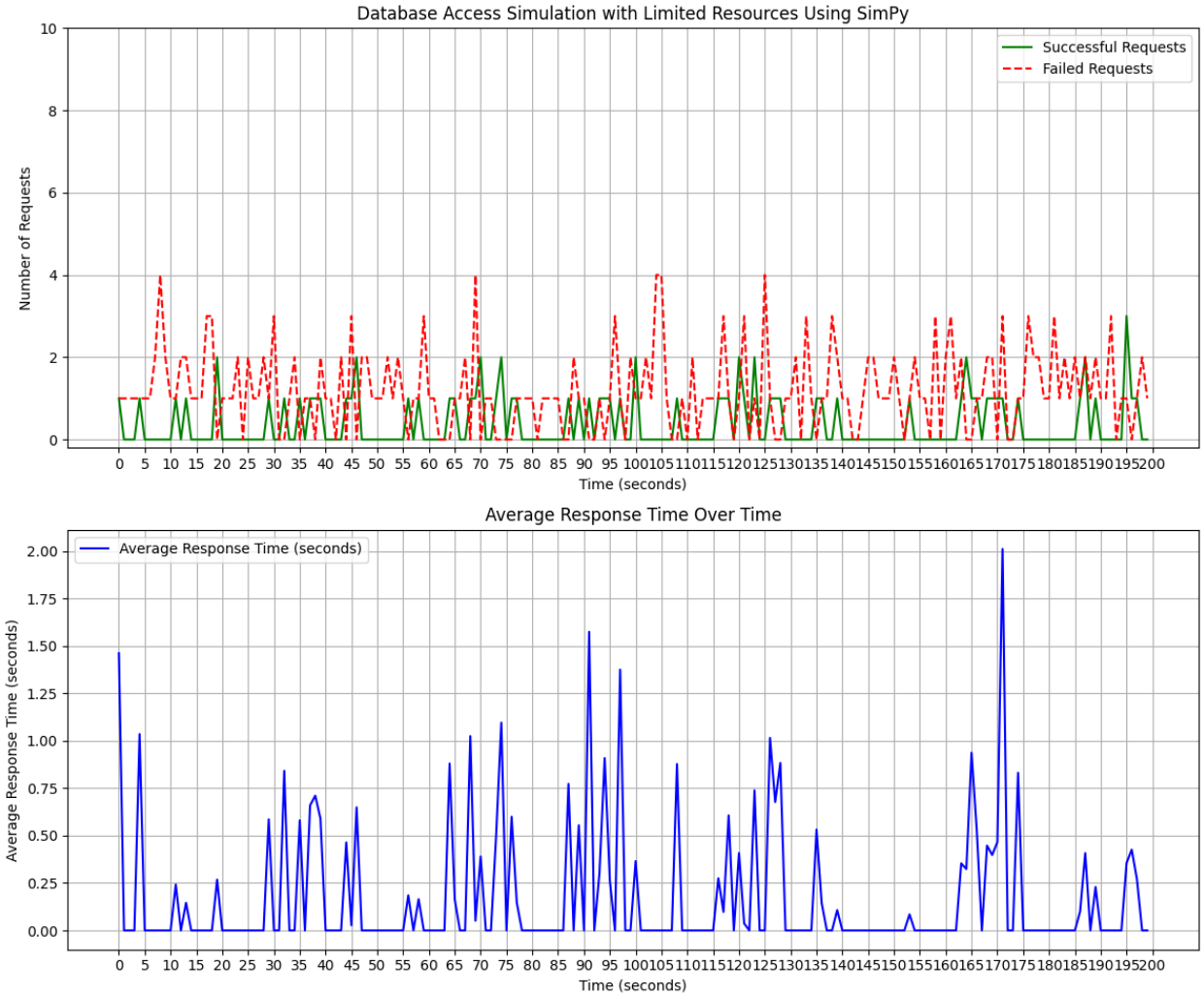

Рисунок 9 - эксперимент выполнения квантиля 20%

Как и ожидалось, время ожидания падает.

- Real-time симуляция запросов (рис. 10)

Частота поступления запросов: 5

Среднее время обслуживания: 2

Количество серверов: 2

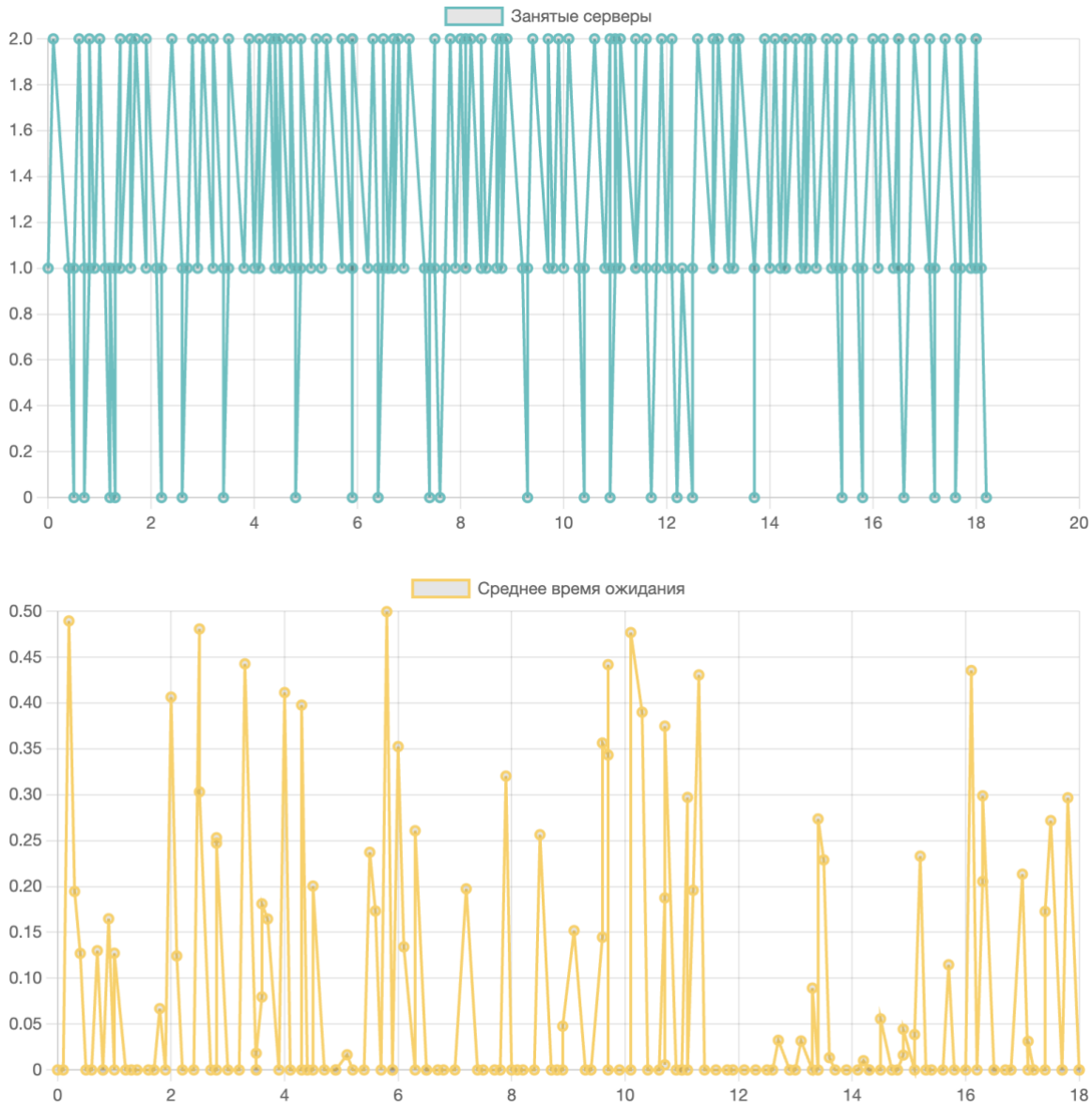

Рисунок 10 - эксперимент недостатка RPS

По графикам в реальном времени видно, что один из серверов БД работает не все время, следовательно для данной конфигурации можно повысить RPS.

- Real-time симуляция запросов (рис. 11)

Частота поступления запросов: 10

Среднее время обслуживания: 2

Количество серверов: 2

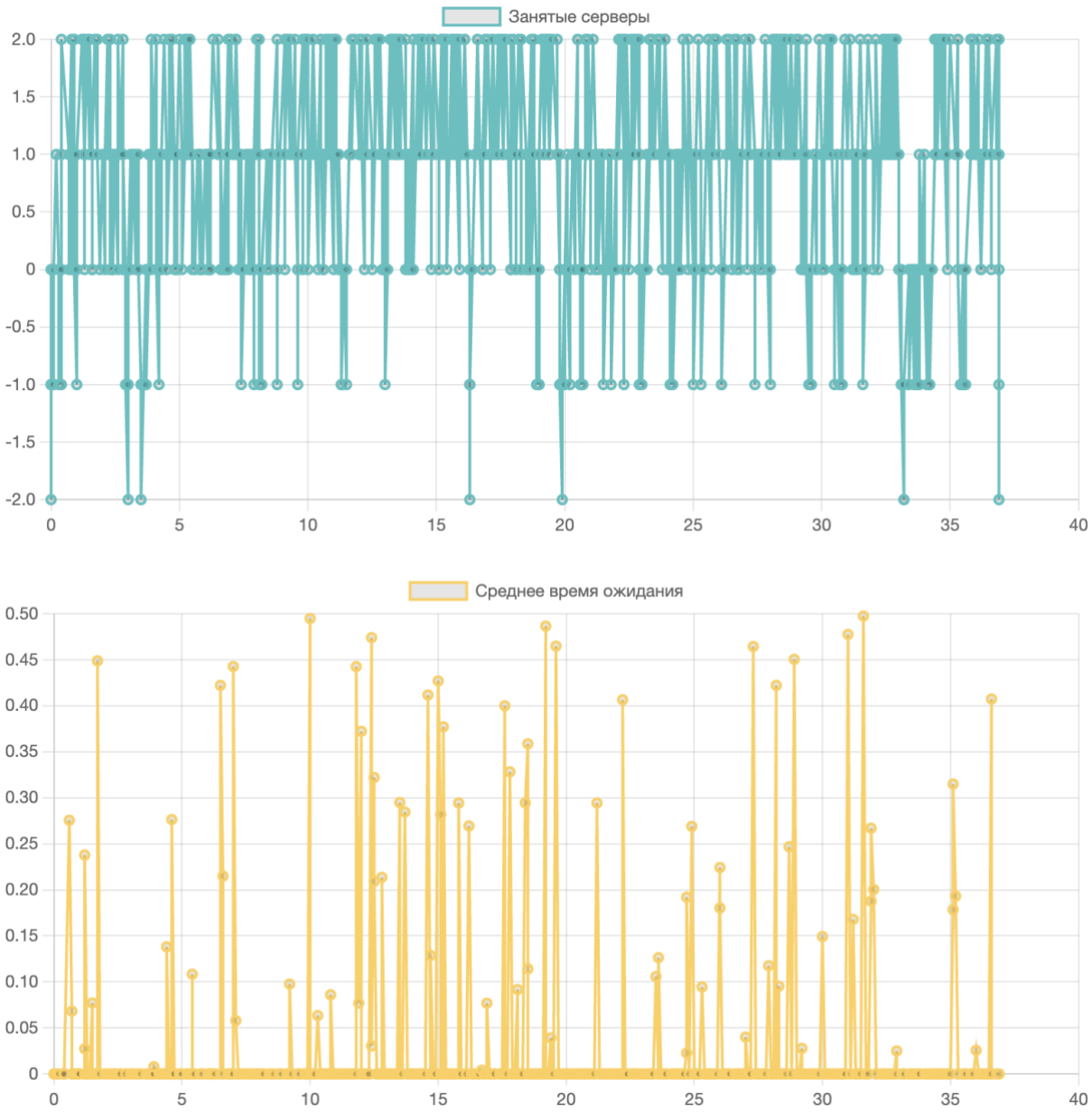

Рисунок 11 - эксперимент минимизации простоя

Теперь сервера заняты больший процент времени, время простоя минимизируется

- Real-time симуляция запросов (рис. 12)

Частота поступления запросов: 10

Среднее время обслуживания: 10

Количество серверов: 2

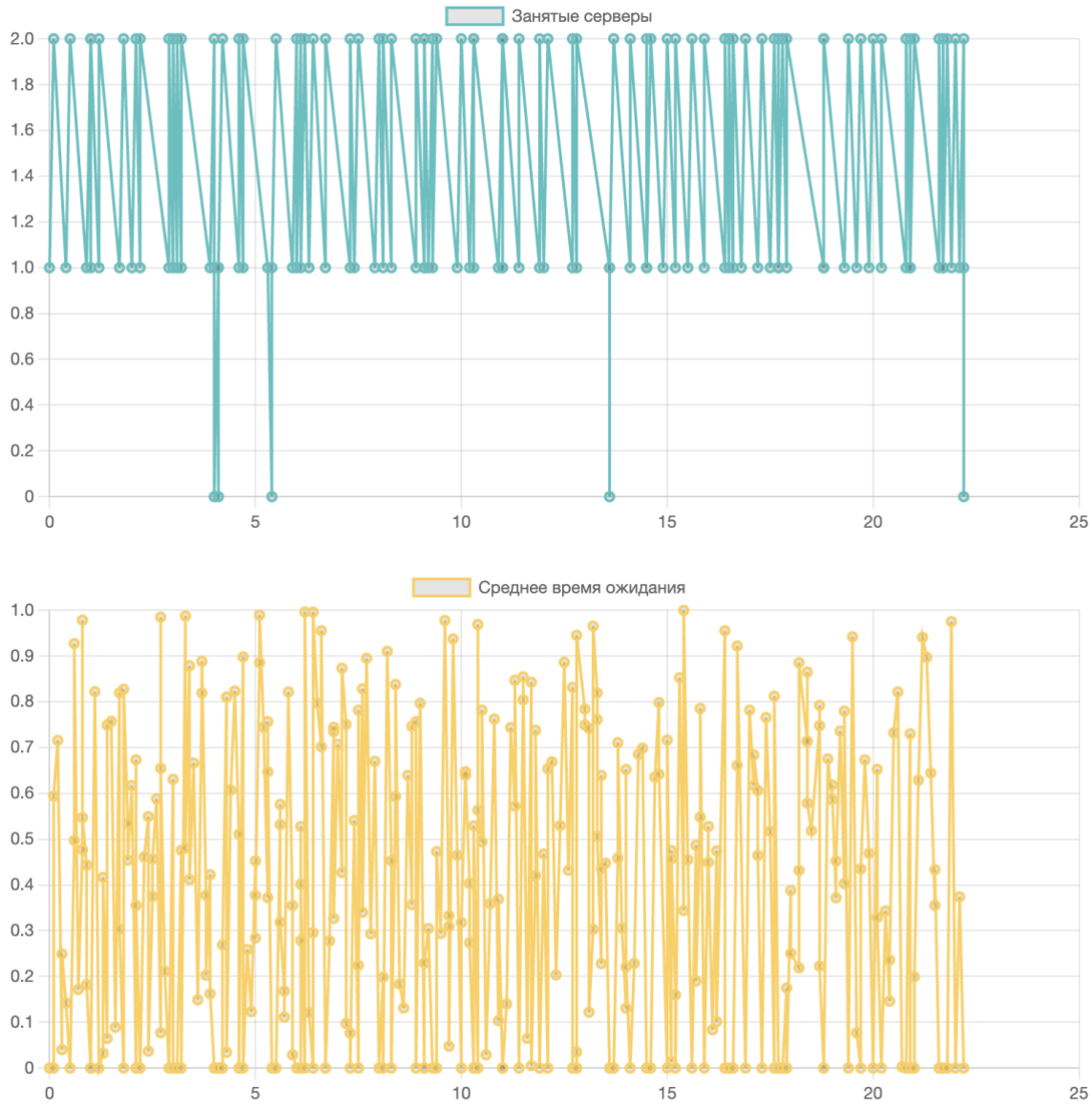

Рисунок 12 - эксперимент повышения времени ожидания

Также при повышении времени обработки растет время ожидания в очереди и процент занятости серверов. Преимущество использования симуляции в реальном времени позволяет исследовать конфигурации симуляции в сравнении. Рассмотрим повышение количества серверов БД:

- Real-time симуляция запросов (рис. 13)

Частота поступления запросов: 10

Среднее время обслуживания: 2

Количество серверов: 10

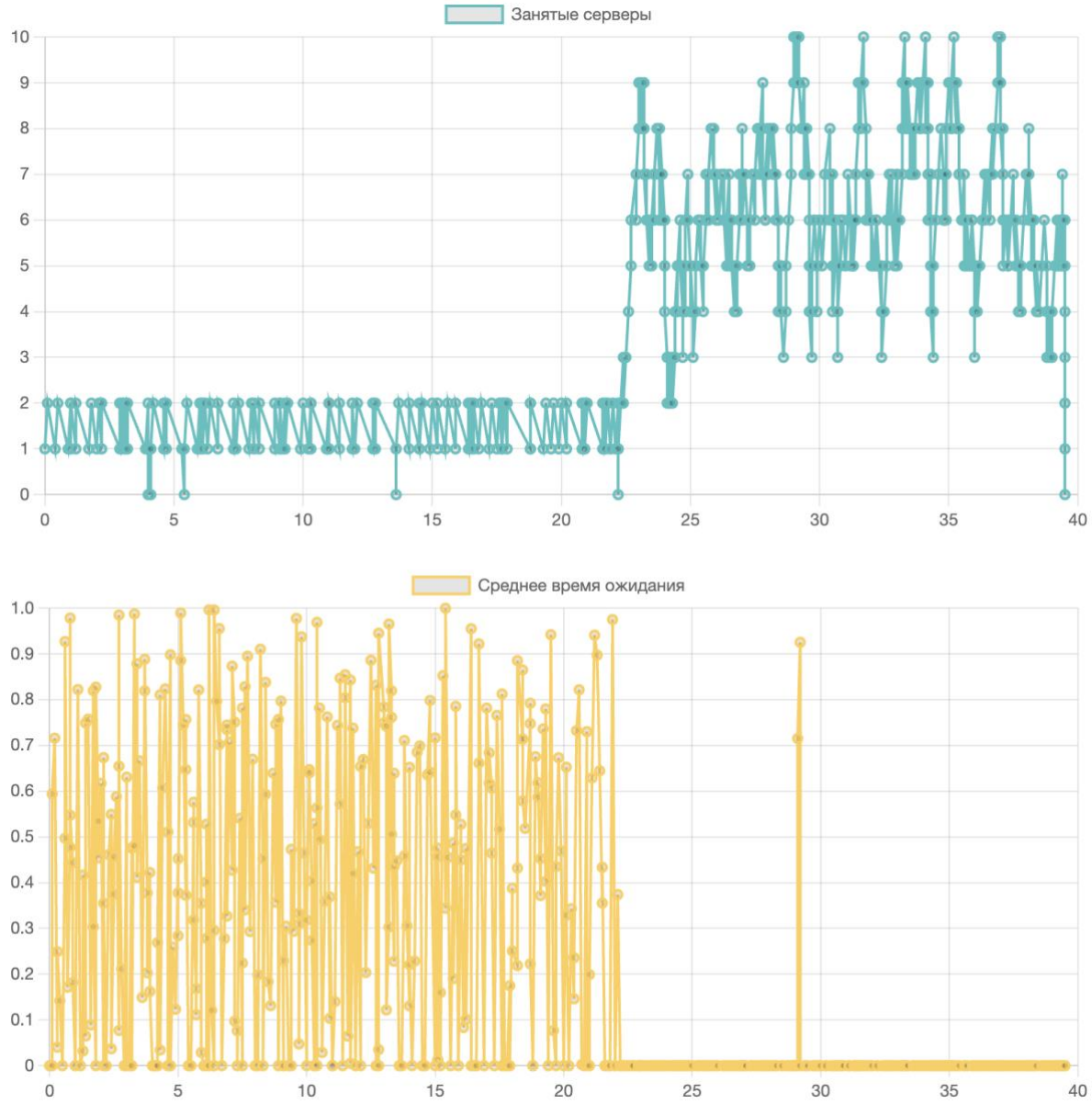

Рисунок 13 - эксперимент повышения количества серверов

В данной конфигурации время ожидания в очереди минимизируется, в то время как трафик между всеми серверами распределяется равномерно. Такая настройка позволяет, например, обеспечить равномерный износ серверного оборудования в промышленном использовании и оптимизацию времени ожидания.

- Real-time симуляция запросов (рис. 14)

Частота поступления запросов: 10

Среднее время обслуживания: 2

Количество серверов: 10

Использование индекса

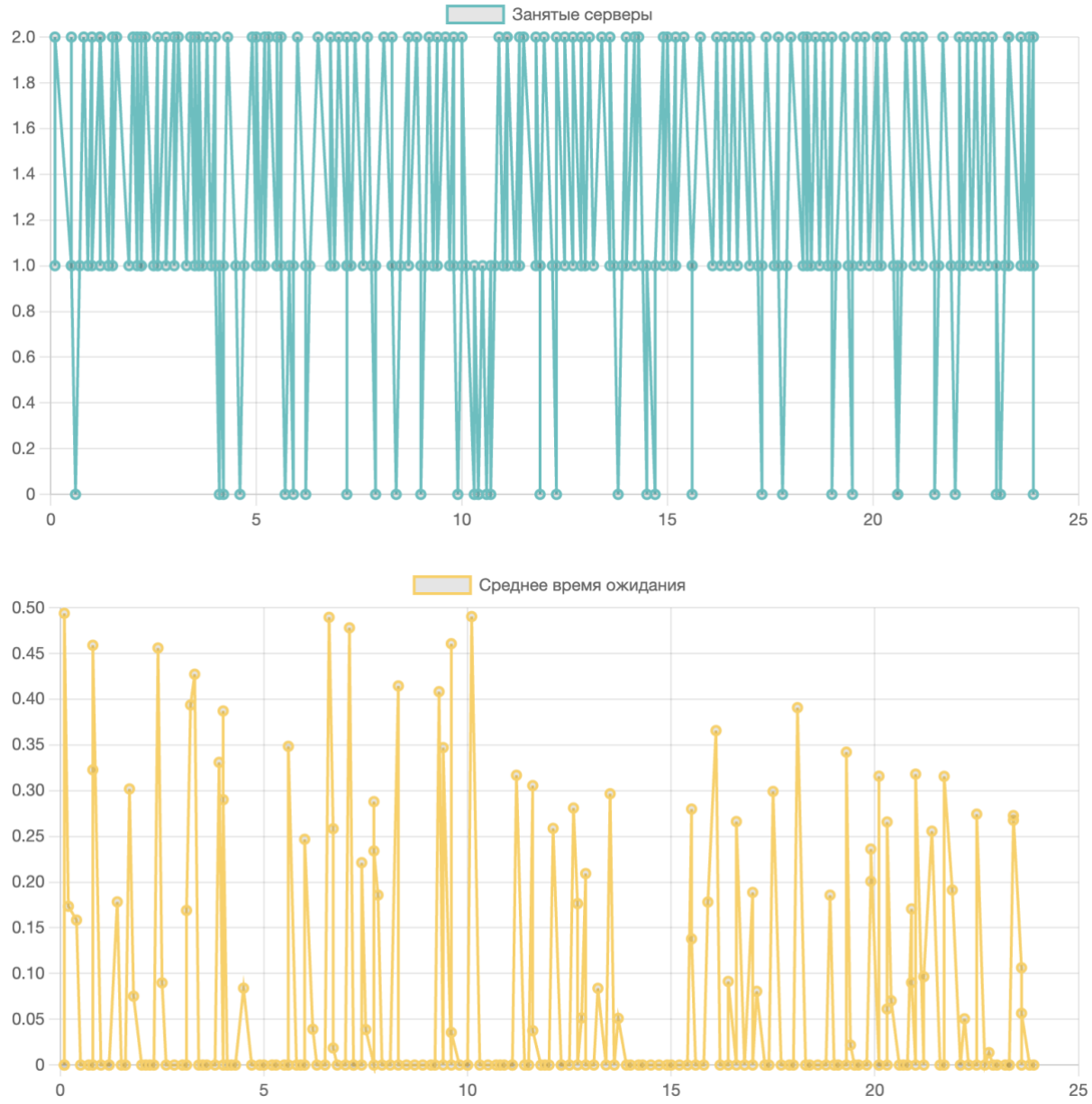

Рисунок 14 - эксперимент использования индекса

С середины процесса моделирования был включен индекс. По графику среднего времени ожидания видна оптимизация максимального времени в 2 раза. Аналогичный результат будет также при использовании механизма кэширования одинаковых запросов.

- Real-time симуляция запросов (рис. 15)

Частота поступления запросов: 10

Среднее время обслуживания: 2

Количество серверов: 10

Использование функций

Рисунок 15 - эксперимент использования функций

Использование функций ожидаемо повышают время обработки, т.к. в БД хранимая функция может быть представлена как внутренний ограниченный ресурс.

Заключение

В ходе выполнения работы было проведено несколько экспериментов, имитирующих модель оптимизации занятости сервера БД. По результатам экспериментов видно, что малое количество серверов способно обработать малое количество поступающих запросов. При большом количестве запросов сервера не успевают их обрабатывать и поэтому возникают ошибки. Для оптимизации запросов могут быть использованы различные конфигурации с использованием индексов, триггеров, хранимых функций. Каждый способ оптимизации может увеличить одну метрику и уменьшить другую. Поэтому для нахождения оптимального решения необходимо выполнять моделирование системы с учетом большего количества параметров и реальных требований сервера БД.

Литература

1. Бершадский А. М., Курилов Л. С., Финогеев А. Г. Исследование стратегий балансировки нагрузки в системах распределенной обработки данных //Известия высших учебных заведений. Поволжский регион. Технические науки. – 2009. – №. 4. – С. 38-48.
2. Кантор И. Современный учебник JavaScript //Современный учебник Java Script [Электронный ресурс]/И. Кантор.–2014.–Режим доступа: http://learn. javascript. ru.–Дата доступа. – 2014. – Т. 12.
3. SymPy 1.13.3 documentation [Электронный ресурс]
4. СТАТЬИ П. О. ТРЕБОВАНИЯ к оформлению статей //ПЕДАГОГИКА СОЦИОКИНЕТИКА. – 2019. – С. 140.
5. Черненький В. М. Алгоритмический метод описания дискретных процессов функционирования систем //Информационно-измерительные и управляющие системы. – 2016. – Т. 14. – №. 12. – С. 11-21.
6. Da Rocha H. Learn Chart. js: Create interactive visualizations for the web with chart. js 2. – Packt Publishing Ltd, 2019.