УДК 004.94
DOI

# РАЗРАБОТКА УНИФИЦИРОВАННОЙ МОДЕЛИ ДЛЯ РАСЧЕТА ВРЕМЕНИ HTTP-ЗАПРОСОВ НА ПРИМЕРЕ ВЕБ-ПРИЛОЖЕНИЙ EXPRESS И FASTAPI

**Кошельков В.С., Грязев Т.А., Соколов М.А., Жуков Н.Н.**

*ФГАОУ ВО «Национальный исследовательский университет ИТМО», Санкт-Петербург,*
*e-mail: mail@koshelkov.ru, gryazev.tim@yandex.ru, lmikhailsokolovl@gmail.com, nnzhukov@itmo.ru*

Цель работы заключается в разработке унифицированной модели для расчета полного времени запроса в клиент-серверной архитектуре. После рассмотрения примера взаимодействия клиент-сервер были выдвинуты гипотезы и определены основные компоненты, влияющие на время запроса. Были определены конфигурации тестируемых систем с использованием языков программирования JavaScript и Python и баз данных PostgresSQL и MongoDB. Нагрузочное тестирование проводилось с использованием инструмента ApacheBench. По результатам тестирования были подтверждены гипотезы и рассчитана модель. Модель показала воспроизведение экспериментальных данных. Дополнительный анализ результатов выявил негативное влияние высокоуровневых инструментов для работы с базами данных на скорость доступа к данным. Исследование выявило преимущество PostgresSQL над MongoDB в рамках ограниченного тестирования. Использование JavaScript и Python показало отсутствие зависимости времени обработки запроса при обращении к базе данных PostgresSQL от языков программирования и фреймворков, что также подтвердило гипотезу о независимости отдельных компонентов в расчете времени запроса. Коэффициенты, полученные в результате эксперимента, могут быть использованы для расчетов предполагаемого времени ответа сервера в зависимости от количеств сущностей и объема сущности в байтах, при этом, опираясь на ход теоретических исследований, приведенных в данной статье, формула может быть расширена и пересчитана для специфических конфигураций систем, которые содержат большее число параметров.

**Ключевые слова: модель, веб-приложение, время запроса, методика расчета, базы данных, клиент-сервер**

# DEVELOPMENT OF GENERALIZED QUERY TIME CALCULATION MODEL USING EXPRESS AND FASTAPI WEBAPPLICATIONS

**Koshelkov V.S., Gryazev T.A., Sokolov M.A., Zhukov N.N.**

*ITMO University «ITMO National Research University», Saint Petersburg,*
*e-mail: mail@koshelkov.ru, gryazev.tim@yandex.ru, lmikhailsokolovl@gmail.com, nnzhukov@itmo.ru*

The purpose of the work is to develop a unified model for calculating the total time of an request in a client-server architecture. After considering the example of client-server interaction, hypotheses were put forward and the main components affecting the request time were identified. The configurations of the tested systems using the JavaScript and Python programming languages and the PostgresSQL and MongoDB databases were determined. Load testing was performed using the ApacheBench tool. Based on the test results, the hypotheses were confirmed and the model was calculated. The model showed a reproduction of experimental data. Additional analysis of the results revealed the negative impact of high-level database tools on the speed of data access. The study revealed the advantage of PostgresSQL over MongoDB in limited testing. The use of JavaScript and Python showed the absence of dependence of the query processing time when accessing the PostgresSQL database on programming languages and frameworks, which also confirmed the hypothesis of the independence of individual components in calculating the query time. In conclusion, the derived coefficients can be used to calculate the estimated server response time depending on the number of entities and the volume of the entity in bytes. At the same time, based on the course of theoretical research presented in this article, the formula can be expanded and recalculated for specific configurations of systems that contain a larger number of parameters.

**Keywords: model, web application, time request, calculation method, databases, client-server**

В контексте современных информационных технологий время отклика сервера является определяющим фактором в производительности веб-приложений. С учетом того, что пользователи предполагают мгновенный доступ к данным, задержки в загрузке могут существенно снизить их удовлетворенность от использования продукта. Это крайне негативно сказывается на позиции веб-сайта в результатах поисковой системы и его конкурентоспособности на рынке. В эпоху цифровизации, когда время приобретает особую ценность, оптимизация скорости загрузки становится ключевым аспектом для обеспечения потребностей пользователей и сохранения конкурентных преимуществ. Активное развитие сетей и методов передачи информации способствует повышению скорости клиент-серверных взаимодействий и положительно влияет на общую производительность веб-приложений.

В рамках улучшения производительности веб-приложений исследователи и раз-

работчики применяют множество методов оптимизации. Эти методы охватывают широкий спектр подходов, включая программную оптимизацию исходного кода, выбор и использование передовых технологий, а также улучшение архитектуры и компонентов веб-приложений [1]. Однако, несмотря на разнообразие доступных методик, существует затруднение в получении точных количественных показателей, которые бы отражали изменение производительности в прямой зависимости от внедряемых технологических решений. Это обусловлено множеством переменных, влияющих на производительность, и сложностью их изолированного анализа в контексте реальных условий эксплуатации веб-приложений.

В представленной работе описывается методология для вычисления общего времени HTTP-запроса, основанная на комплексном анализе тестовых данных, полученных в разнообразных условиях. Данный метод предоставляет возможность гибкого расчета временных параметров запроса для различных системных конфигураций и может быть модифицирован для применения в специфических системах. Это достигается за счет внедрения гибкой структуры, позволяющей интегрировать дополнительные составляющие, возникающие в ходе реализации конкретных веб-приложений.

Цель исследования – разработка унифицированной модели для расчета полного времени HTTP-запроса в клиент-серверной архитектуре.

**Материалы и методы исследования**

Веб-приложение состоит из клиентской и серверной части. Подавляющее большинство запросов между ними осуществляется посредством протокола HTTP, обеспечивающего обмен данными путем отправки запроса от клиента к серверу и последующим получением ответа от сервера клиентом [2].

На более низком уровне запрос можно разделить на большее количество этапов. Например, на сервере запрос может проходить через широкий ряд компонентов: веб-сервер, программная платформа, библиотеки, база данных.

В частном случае, который будет рассмотрен в этой статье, запрос можно представить в следующем виде:

1. Клиент отправляет HTTP-запрос к веб-серверу, который представляет собой приложение, написанное на JavaScript или Python с использованием Express или FastApi соответственно.

2. Веб-сервер принимает запрос и обращается к СУБД PostgreSQL или MongoDB для получения данных.

3. СУБД возвращает данные приложению, которое в свою очередь отправляет эти данные обратно клиенту.

В расчете времени запроса клиентская часть рассматривается в качестве вывода полученной информации с сервера и не учитывается как фактор, оказывавший влияние на скорость получения ответа от сервера.

Таким образом можно выделить компоненты системы, которые оказывают влияние на общее время запроса:

1. Время работы обработки запроса веб-сервером, включая программную платформу, библиотеки и другие компоненты. Обозначим его как «базовое время».

2. Время обращения к СУБД. Обозначим его как «время СУБД».

3. Время передачи запроса по сети. Обозначим его как «время сети».

В ходе данного исследования были выдвинуты следующие гипотезы:

1. Каждую из этих компонент можно рассматривать как математическую функцию, принимающую аргументы, необходимые и достаточные, в рамках данной модели, для дифференциации запроса.

2. Суммой этих функций является результирующая функция, отражающая общее время запроса.

3. Все компоненты результирующей функции независимы друг от друга.

В качестве аргументов, достаточных для исследуемой модели, функции, были выбраны количество запрашиваемых сущностей и размер одной сущности. Таким образом была сформирована следующая результирующая функция (1) для расчета времени запроса:

$$T(n,S) = T_0(n,S) + T_c(n,S) + T_{бд}(n,S), \quad (1)$$

где $n$ – количество запрашиваемых сущностей;

$S$ – размер одной сущности (байт).

MongoDB и PostgreSQL представляют собой два различных типа баз данных. MongoDB является базой данных NoSQL типа, обладающей гибкой моделью данных. Это достигается за счет того, что все данные хранятся в виде документов JSON, обеспечивая быстрое извлечение, репликацию и анализ данных. В отличие от MongoDB, PostgreSQL является объектно-реляционной базой данных, хранящей данные в виде таблиц с рядами и столбцами. В ходе различных исследований было выявлено, что реляционные системы управления базами данных (СУБД) обычно показывают высокую производительность при выполнении обширных запросов к большим объемам данных, в то время как объектно-ориентированный доступ оказывается более эффек-

тивным для малых объемов данных, включая те, которые имеют сложную структуру [3]. Проведение тестирования с использованием этих разных по свойствам работы СУБД позволило получить более широкое представление о специфике их поведения при разной нагрузке [4].

В качестве программных платформ для реализации веб-сервера были выбраны Node.js и Python.

Для веб-сервера на Node.js был использован фреймворк Express, обладающий упрощенной структурой и обеспечивающий обширный набор функциональных возможностей для создания как мобильных, так и веб-приложений. Веб-сервер на Python был разработан с использованием FastAPI. Данный инструмент хорошо подходит для создания высокопроизводительных API и использует библиотеку asyncio для обеспечения асинхронной обработки запросов. Основными его достоинствами являются высокая скорость работы, типизация данных и автоматическая генерация спецификации OpenAPI.

Зачастую работа сервера с реляционными и нереляционными базами данных строится соответственно через ORM и ODM библиотеки [5]. Для работы сервера с базой данных MongoDB на базе Node.js использовалась ODM Mongoose. Для подключения к MongoDB через сервер на FastAPI использовался официальный драйвер Pymongo. В случае с PostgresSQL применялись npm-пакет «pg» для Node.js и «psycopg2» для Python.

Для оценки производительности системы был выбран инструмент ApacheBench, который является утилитой командной строки, предназначенной для тестирования и измерения производительности веб-серверов [6].

Выбор ApacheBench в качестве инструмента для оценки производительности системы основан на следующих преимуществах: возможность тестирования производительности веб-серверов, работающих по протоколу HTTP, возможность генерации и отправки запросов, сбор и обработка данных, анализ и визуализация результатов, генерация и предоставление отчетов и др. [7]. Тестовый клиент ApacheBench был развернут на отдельном сервере с операционной системой Linux, имеющей 1 ядро процессора, 2 ГБ оперативной памяти и 30 ГБ дискового пространства.

Версии технологий, используемых в тестировании, приведены в табл. 1.

На базе Node.js и Express был спроектирован и развернут веб-сервер, обрабатывающий входящие GET-запросы [8]. В качестве ответа на запрос отправлялся список произвольных данных. Для проведения серии тестов были подготовлены абсолютно идентичные списки для всех типов запросов. Список содержал 10000 записей, каждая из которых представляет собой объект с несколькими полями разных типов.

Были проведены серии тестов, представленные в табл. 2.

Тестирование каждой из представленных выше баз данных проводилось в тысячу повторных одиночных GET-запросов для усреднения получаемых данных. В данном исследовании клиент и сервер находились в одной локальной сети. Результаты замеров приведены на рис. 1–3.

На рис. 1 продемонстрирована независимость компонент друг от друга за счет того, что изменение Node.js на Python не оказало влияния на время обращения к PostgreSQL.

**Таблица 1**

Версии тестируемых технологий

| Node.js | Python | PostgreSQL | MongoDB | Express | PG | Mongoose |
|---|---|---|---|---|---|---|
| 19.8.1 | 3.12 | 16.2 | 7.0.2 | 4.18.2 | 8.11.5 | 8.3.2 |

**Таблица 2**

Тестируемые конфигурации систем

| Базы данных | Среда выполнения | Фреймворк |
|---|---|---|
| PostgresSQL | Node.js | Express |
| | Python | FastAPI |
| MongoDB | Node.js | Express |
| | Python | FastAPI |

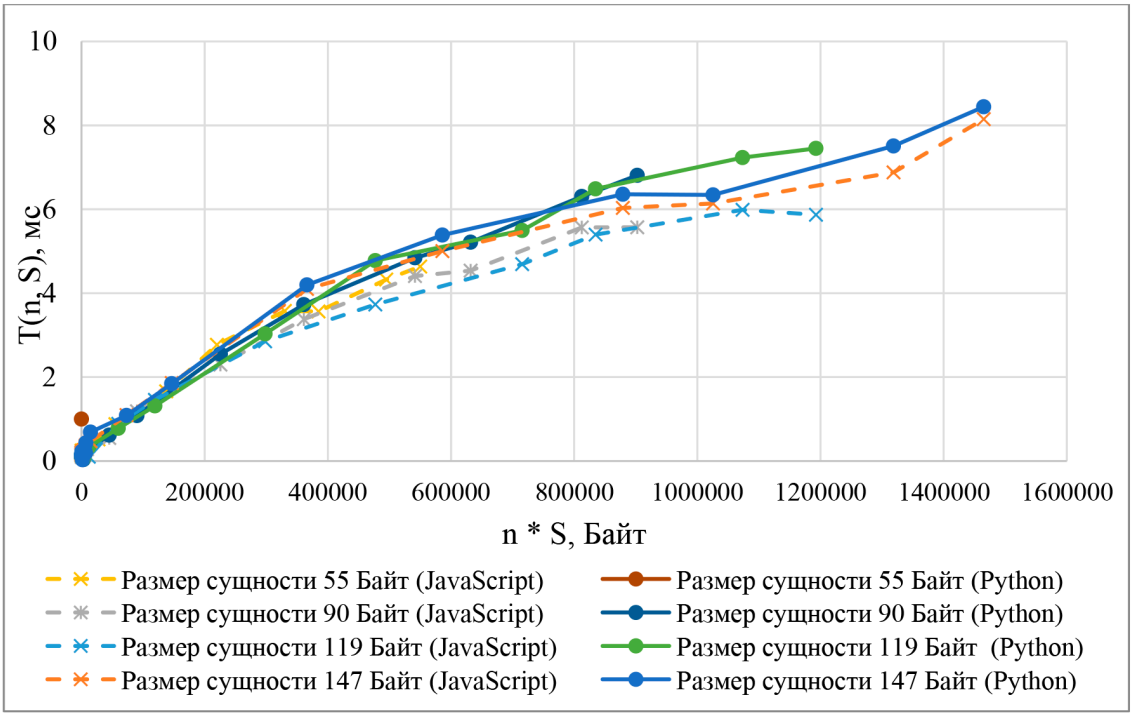


*Рис. 1. Результаты расчета СУБД компоненты PostgreSQL*

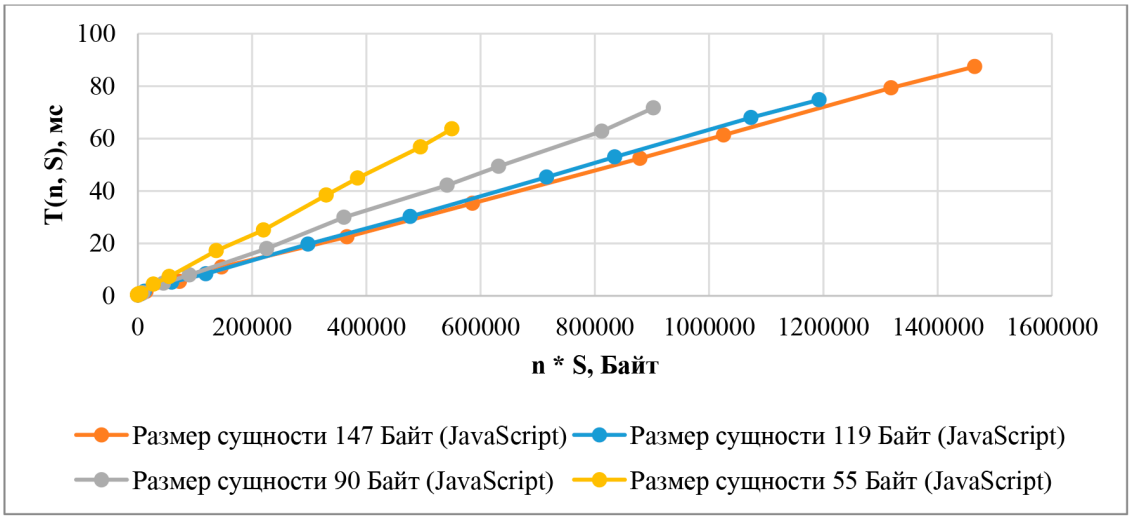


*Рис. 2. Результаты расчета СУБД компоненты MongoDB на Node.js*

Приведенные рис. 2 и 3 показывают, что время ответа MongoDB прямо пропорционально увеличению количества сущностей и обратно пропорционально размеру сущности. При рассмотрении результатов для PostgreSQL, приведенных на рис. 1, наблюдается явная зависимость времени запроса от произведения количества сущностей на размер сущности. При варьировании значений аргументов с сохранением их произведения не наблюдается закономерного изменения времени запроса. Поскольку функция для расчета времени запроса к MongoDB не может быть выражена как функция одной переменной, в расчетах модели она участвовать не будет, и во всех последующих расчетах в качестве СУБД будет использоваться PostgreSQL.

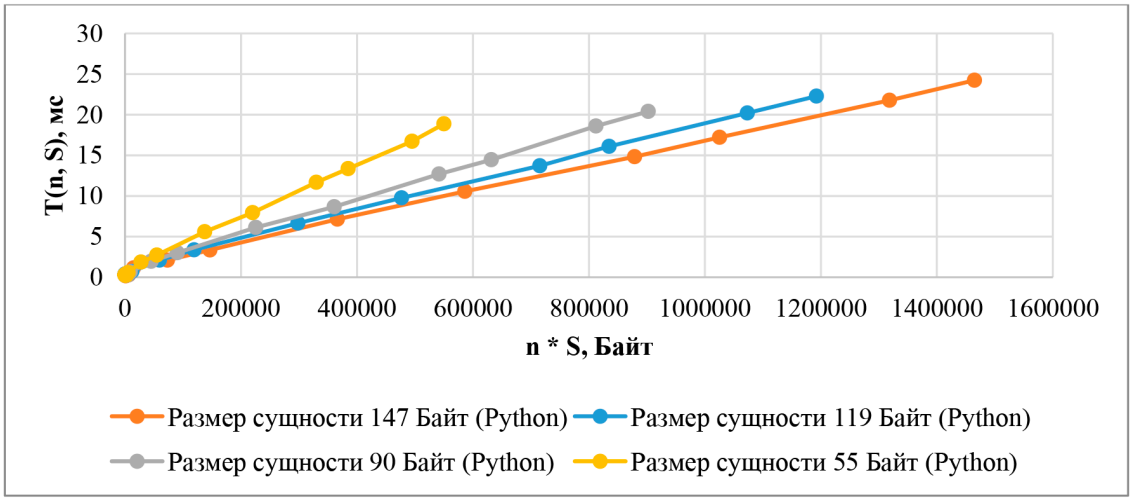


*Рис. 3. Результаты расчета СУБД компоненты MongoDB на Python*

Также следует отметить, что само время доступа к MongoDB сильно отличается для Node.js и Python ввиду использования концептуально разных библиотек для доступа к БД, поэтому специфика библиотеки будет отражаться на компоненте, отвечающей за время обращения к СУБД.

**Результаты исследования и их обсуждение**

По результатам замеров были выведены и рассчитаны функции компонент для Node.js (2)–(4) и Python (5)–(7):

$$T_0(n,S) = k_0 \cdot n \cdot S + C_0 = 0 \cdot n \cdot S + 0{,}16 = 0{,}16, \quad (2)$$

$$T_c(n,S) = k_c \cdot n \cdot S + C_c = 0.5206 * 10^{-5} \frac{\text{мс}}{\text{байт}} \cdot n \cdot S + 0.0178 \quad (3)$$

$$T_{бд}(n,S) = k_{бд3}(n \cdot S)^3 + k_{бд2}(n \cdot S)^2 + k_{бд1} \cdot n \cdot S + C_{бд} =$$

$$= 0.528 * 10^{-17} \frac{\text{мс}}{\text{байт}^3} \cdot (n \cdot S)^3 - 0.1244 * 10^{-10} \cdot (n \cdot S)^2 + 0.1309 * 10^{-4} \frac{\text{мс}}{\text{байт}} \cdot n \cdot S + 0.166, \quad (4)$$

$$T_0(n,S) = k_0 \cdot n \cdot S + C_0 = 0 \cdot n \cdot S + 0.278 = 0.27\,, \quad (5)$$

$$T_{\tilde{n}}(n,S) = k_{\tilde{n}} \cdot n \cdot S + C_c = 5.760 * 10^{-6} \cdot n \cdot S + 5.705 * 10^{-2}, \quad (6)$$

$$T_{бд}(n,S) = k_{бд3}(n \cdot S)^3 + k_{бд2}(n \cdot S)^2 + k_{бд1} \cdot n \cdot S + C_{бд} =$$

$$= 0.360 * 10^{-17} \cdot (n \cdot S)^3 - 0.1112 * 10^{-10} \cdot (n \cdot S)^2 + 0.142 * 10^{-4} \cdot n \cdot S + 0.103\,, \quad (7)$$

где $C_0$ – базовая константа (мс/байт);
$k_c$ – коэффициент сети (мс/байт);
$C_c$ – константа сети (мс/байт);
$k_{бдn}$ – коэффициент базы данных (мс/байт$^n$);
$C_{бд}$ – константа базы данных (мс/байт).

После расчета коэффициентов для функций компонент построим результирующие функции для Node.js на рис. 4 и для Python на рис. 5. На рис. 4 и 5 точками обозначены экспериментальные данные, а сплошными линиями рассчитанные функции для выведенной модели.

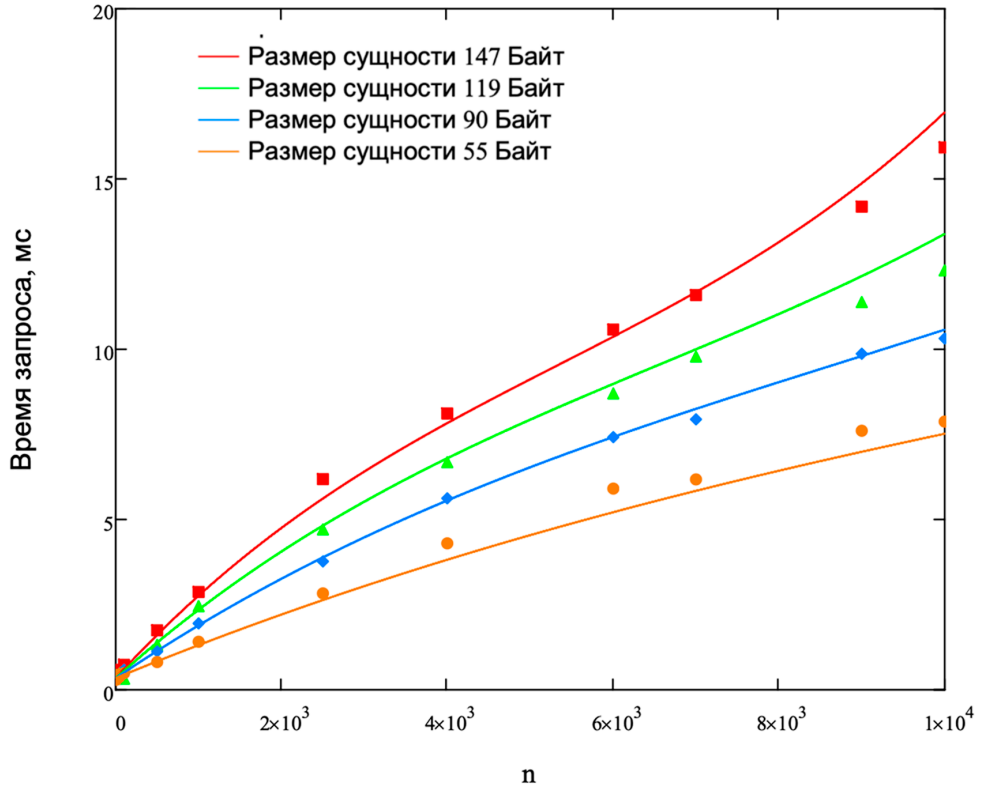


*Рис. 4. Результаты сопоставления данных для Node.js*

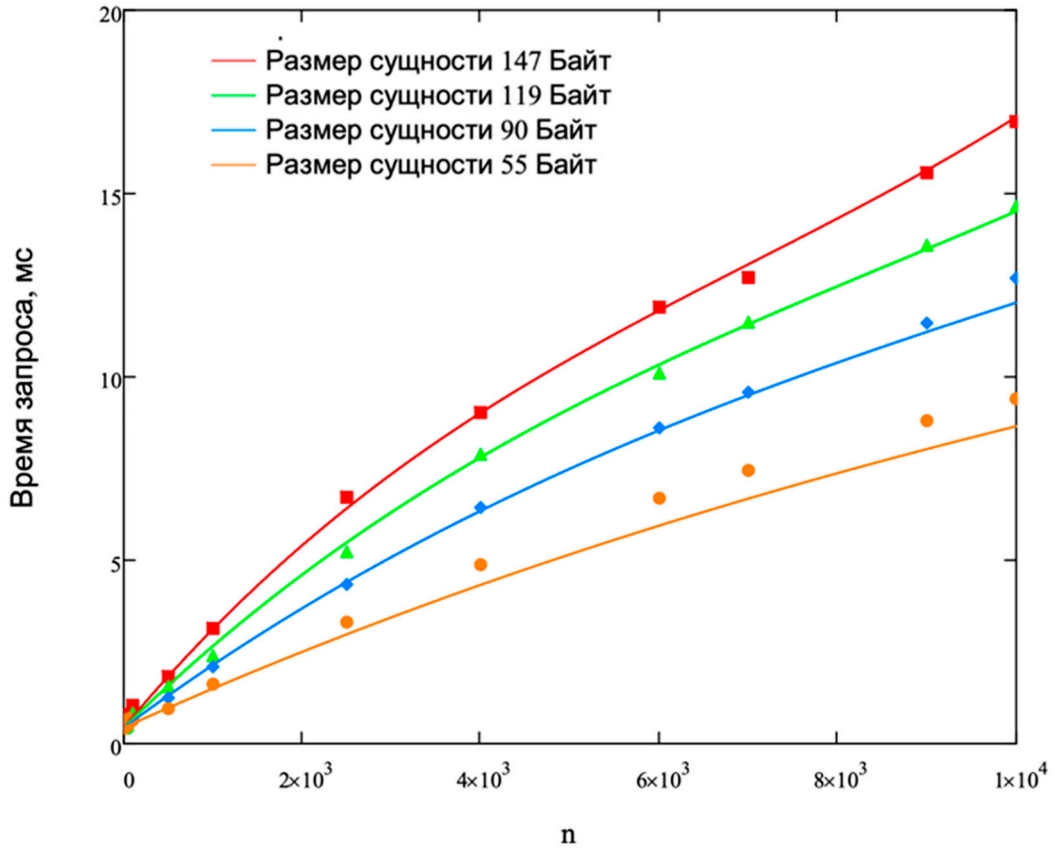


*Рис. 5. Результаты сопоставления данных для Python*

## Заключение

При проведении тестирования было выявлено, что время получения результата от MongoDB зависит от используемой ODM. В том числе фактическое использование ODM увеличивает время получения данных в отличие от более низкоуровневых подходов доступа.

При сравнении скорости ответа от PostgreSQL и MongoDB наблюдается явное преимущество первого при числе сущностей до 10000. Другие исследования также демонстрируют преимущество PostgreSQL при больших объемах данных. PostgreSQL отличается независимостью времени запроса от числа запрашиваемых сущностей, в отличие от MongoDB, где наблюдается прямая зависимость от количества сущностей и обратная зависимость от размера сущности.

Спроектированная модель на основе гипотез продемонстрировала видимое совпадение экспериментальных и теоретических данных. Также были подтверждены все выдвинутые в рамках исследования гипотезы. В проведенном исследовании была сформулирована и выведена модель расчета времени запроса на основе гипотез. Все выдвинутые гипотезы били экспериментально подтверждены.

Также исследование выявило преимущество PostgreSQL над MongoDB в рамках ограниченной серии тестов. Стоит учитывать, что результаты могут отличаться от разных конфигураций систем и при этом сохранять общую закономерность. Использование ODM для доступа к MongoDB способно снижать скорость получения данных.

Выведенные коэффициенты могут быть использованы для расчетов предполагаемого времени ответа сервера в зависимости от количеств сущностей и объема сущности в байтах. При этом, опираясь на ход теоретических исследований, приведенных в данной статье, формула может быть расширена и пересчитана для специфических конфигураций систем, которые содержат большее число параметров.

## Список литературы

1. Медведев Ю.С., Терехов В.В. Проектирование интерактивных web-приложений // Современные проблемы науки и образования. 2015. № 1–1. URL: https://science-education.ru/ru/article/view?id=17486 (дата обращения: 29.03.2024).

2. Лиманов Н.И., Селезнев И.А. Анализ эффективности клиент-серверной архитектуры // Бюллетень науки и практики. 2022. № 7. С. 392–396. DOI: 10.33619/2414-2948/80/37.

3. Сорокин В.Е. Об эффективности наследования таблиц в СУБД PostgreSQL // Программные продукты и системы. 2016. № 3. С. 15–23.

4. Josefin Lindvall, Adam Sturesson. A comparison of latency for MongoDB and PostgreSQL with a focus on analysis of source code // JTH, Department of Computer Science and Informatics. 2021. P. 74.

5. Ершов Т.А., Голубничий А.А. Введение в Object-Relational Mapping, работа с данными PostgreSQL с использованием языка Python и SQLAlchemy // Научно-технический вестник Поволжья. 2024. № 1. С. 151–154.

6. Щербаков А.Б. Сравнительный анализ серверных фреймворков на платформе Node.js // Альманах научных работ молодых ученых Университета ИТМО. 2018. С. 323–325.

7. Srilekha Pothuganti, Samanth Malepati. Comparative analysis of load balancing in cloud platforms for an online bookstore web application using Apache Benchmark // Department of Computer Science. 2023. P. 51.

8. Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. 2-е изд. СПб.: Питер, 2021. 336 с.