НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
май–июнь 2026 Том 26 № 3 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
May–June 2026 Vol. 26 No 3 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2026-26-3-517-524
УДК 004.738.5

Сравнение gRPC и REST API в поиске более быстрой и совместимой архитектуры

Артем Эдуардович Баженов[1], Арина Алексеевна Маленкова[2], Никита Владимирович Майоров[3]✉

[1,2,3] Поволжский государственный университет телекоммуникаций и информатики, Самара, 443010, Российская Федерация

[1] a.bazhenov@psuti.ru, https://orcid.org/0009-0001-5887-2077
[2] kroshik0812@vk.com, https://orcid.org/0009-0005-0755-895X
[3] nikita@m7n.ru✉, https://orcid.org/0009-0002-7502-8860

Аннотация

Введение. Микросервисная архитектура стала стандартным подходом к построению высоконагруженных и эволюционирующих информационных систем. В цифровых образовательных платформах одновременно важны совместимость внешних интерфейсов и низкие задержки внутренних межсервисных вызовов. Выполнено сравнение подходов Google Remote Procedure Calling (gRPC) и Representational State Transfer Application Programming Interface (REST API) по метрикам времени отклика и пропускной способности. Сформулированы рекомендации по выбору подхода для сервисов аналитики учебных платформ. **Метод.** Объектом исследования является мини-сервис аналитики учебной платформы, который по запросу возвращает агрегированные показатели действий пользователя за заданный интервал времени. Реализованы две функционально эквивалентные версии сервиса на Python с одинаковой моделью данных и идентичной бизнес-логикой: REST-сервис на основе Hypertext Transfer Protocol (HTTP) версии 1.1, с передачей данных в формате JavaScript Object Notation и gRPC-сервис на основе HTTP/2 с передачей данных в формате Protocol Buffers. Измерения выполнялись нагрузочными скриптами на стороне клиента с фиксацией полного времени выполнения запроса в базовом сценарии фильтрации по идентификатору пользователя и интервалу. Тесты проводились при 100, 200 и 500 одновременных клиентах при длительности измерительного окна 20 с. **Основные результаты.** В качестве метрик использовались среднее время отклика, 95-й процентиль задержки и пропускная способность (число запросов в секунду). По результатам измерений при 100 одновременных клиентах подход gRPC обеспечивает меньшую среднюю задержку, тогда как при 500 клиентах REST API демонстрирует более высокую пропускную способность и меньшее среднее время отклика. При росте нагрузки 95-й процентиль задержки у gRPC остается ниже, чем у REST API, что свидетельствует о более стабильном поведении задержек в области высоких процентилей в условиях высокой конкуренции запросов. **Обсуждение.** Показано, что выбор между REST API и gRPC не является взаимоисключающим: REST API удобен для публичных интерфейсов и интеграций с внешними клиентами, а gRPC рационально использовать для внутренних вызовов между микросервисами, где критичны верхние задержки и требуется строгая контрактная спецификация. Рекомендации по выбору технологии сформулированы с учетом профиля нагрузки и требований к интеграции.

Ключевые слова

микросервисная архитектура, цифровые образовательные платформы, gRPC, REST, HTTP/2, HTTP/1.1, Protocol Buffers, JSON, пропускная способность, нагрузочное тестирование

Ссылка для цитирования: Баженов А.Э., Маленкова А.А., Майоров Н.В. Сравнение gRPC и REST API в поиске более быстрой и совместимой архитектуры // Научно-технический вестник информационных технологий, механики и оптики. 2026. Т. 26, № 3. С. 517–524. doi: 10.17586/2226-1494-2026-26-3-517-524

Comparing gRPC and REST API in search of a faster and more compatible architecture

Artem E. Bazhenov[1], Arina A. Malenkova[2], Nikita V. Maiorov[3]✉

[1,2,3] Volga Region State University of Telecommunications and Informatics, Samara, 443010, Russian Federation

[1] a.bazhenov@psuti.ru, https://orcid.org/0009-0001-5887-2077
[2] kroshik0812@vk.com, https://orcid.org/0009-0005-0755-895X
[3] nikita@m7n.ru✉, https://orcid.org/0009-0002-7502-8860

Abstract

Microservice architecture has become a standard approach to building highly loaded and evolving information systems. In digital educational platforms, compatibility of external interfaces and low delays of internal inter-service calls are important at the same time. The purpose of the work is to compare the Google Remote Procedure Calling (gRPC) and Representational State Transfer Application Programming Interface (REST API) in terms of response time and bandwidth metrics and formulate recommendations on choosing an approach for the analytics services of educational platforms. The object of the research is a mini-analytics service of the educational platform which, upon request, returns aggregated indicators of user actions for a given time interval. Two functionally equivalent versions of the service are implemented in Python with the same data model and identical business logic: a REST service based on Hypertext Transfer Protocol (HTTP) version 1.1, with data transmission in JavaScript Object Notation format and a gRPC service based on HTTP/2 with data transmission in Protocol Buffers format. The measurements were performed by load scripts on the client side with measurement of the full request execution time in the basic filtering scenario by user ID and interval. The tests were performed with 100, 200, and 500 simultaneous clients with a measuring window duration of 20 seconds. The average response time, 95th percentile latency, and throughput (number of requests per second) were used as metrics. According to the measurement results, with 100 simultaneous clients, gRPC provides lower average latency, while with 500 clients, REST demonstrates higher throughput and lower average response time. As the load increases, the 95th percentile of latency in gRPC remains lower than in REST, which indicates a more stable behavior of the "tail" of delays in conditions of high query competition. It is shown that the choice between REST and gRPC is not mutually exclusive: REST is convenient for public interfaces and integrations with external clients, and gRPC is rational to use for internal calls between microservices where overhead delays are critical and strict contractual specification is required. Recommendations on the choice of technology are formulated taking into account the load profile and integration requirements.

Keywords

microservice architecture, digital educational platforms, gRPC, REST, HTTP/2, HTTP/1.1, Protocol Buffers, JSON, bandwidth, load testing

For citation: Bazhenov A.E., Malenkova A.A., Maiorov N.V. Comparing gRPC and REST API in search of a faster and more compatible architecture. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2026, vol. 26, no. 3, pp. 517–524 (in Russian). doi: 10.17586/2226-1494-2026-26-3-517-524

Введение

В современных развивающихся информационных системах, рассчитанных на переменную нагрузку, часто используется микросервисная декомпозиция [1–3]. При таком проектировании возрастает интенсивность межсервисных вызовов, и технология взаимодействия становится одним из факторов, определяющим задержки, пропускную способность и стоимость интеграции компонентов [4, 5]. В прикладных сценариях наиболее распространены два подхода: Representational State Transfer Application Programming Interface (REST API) как универсальный Hypertext Transfer Protocol (HTTP)-интерфейс и Google Remote Procedure Calling (gRPC) как контрактно-ориентированный механизм с бинарной сериализацией и более эффективным транспортом [6, 7].

В цифровых образовательных платформах и системах управления обучением один и тот же функционал может обслуживать разные типы потребителей [8, 9]. Публичные интерфейсы и интеграции с внешними системами предъявляют требования к прозрачности, совместимости и простоте диагностики. Внутренние сервисы, включая аналитику активности пользователей, чаще зависят от стабильности верхних задержек и способности выдерживать всплески параллельных обращений. Исходя из этого, задача выбора протокола для конкретного контура системы сводится к компромиссу между совместимостью и эксплуатационной эффективностью [10].

В научных работах посвященных анализу подходов REST API и gRPC описаны их преимущества, однако результаты измерений часто трудно сопоставимы: различаются языки реализации, структура полезной нагрузки, сетевые условия, модели конкурентности и способы измерения задержек. На практике это приводит к тому, что выводы в целом быстрее или проще остаются слишком общими и плохо применимыми при проектировании конкретных микросервисов.

Цель настоящей работы состоит в сравнении REST API и gRPC на примере сервиса аналитики с одинаковой бизнес-логикой и моделью данных в контролируемых условиях. Для этого реализованы два функционально эквивалентных варианта сервиса на Python и проведено нагрузочное тестирование с измерением времени отклика и пропускной способности при фиксированных параметрах прогонов, в соответствии с общими подходами к тестированию API в микросервисной архитектуре [11].

В работе впервые предложена воспроизводимая постановка эксперимента для сравнения REST API и

gRPC при неизменной бизнес-логике и одинаковых данных; выполнены измерения времени отклика и 95-го процентиля задержки, а также пропускной способности при нескольких уровнях параллелизма; сформулированы практические рекомендации по выбору подхода для разных контуров микросервисной системы.

Обзор подходов REST API и gRPC

В микросервисной архитектуре подходы REST API и gRPC применяются как два практических способа организации взаимодействия между компонентами, однако их эксплуатационные свойства отличаются. Для внешних интеграций важны совместимость, простота диагностики и прозрачность обмена, тогда как для внутренних вызовов чаще критичны устойчивость задержек при росте параллелизма и однозначность спецификации интерфейса между сервисами. В результате сравнение этих подходов целесообразно вести не на уровне общих определений, а через факторы, которые напрямую влияют на производительность и стоимость сопровождения интерфейсов.

REST API обычно строится вокруг ресурсов и стандартных операций HTTP, а полезная нагрузка в типовых веб-сценариях передается в текстовом формате JavaScript Object Notation (JSON). Это облегчает ручную проверку запросов и ответов и упрощает подключение разнородных клиентов и инструментов. Однако текстовая сериализация и разбор JSON создают дополнительные накладные расходы, которые могут проявляться при высокой интенсивности запросов и большом числе одновременных обращений, особенно в сервисах, где полезная нагрузка содержит несколько полей, но вызовы выполняются часто [4].

gRPC ориентирован на контрактное описание интерфейса и автоматическую генерацию клиентских и серверных компонентов на основе файла .proto-схемы. Такой подход уменьшает неоднозначность трактовки данных, дисциплинирует эволюцию интерфейса и снижает риск рассогласования между сервисами. Вместо текстовой сериализации используется бинарный формат Protocol Buffers, который обычно компактнее и требует меньших затрат на обработку структурированных данных [6]. В типовой конфигурации gRPC работает поверх HTTP версии 2 (HTTP/2), что предоставляет мультиплексирование запросов в одном соединении и уменьшает накладные расходы на обслуживание большого числа конкурентных вызовов.

Существенное различие проявляется и в эволюции интерфейса. В REST API часто используют версионирование или добавление необязательных полей в JSON, что снижает вероятность нарушения работы существующих клиентов, но повышает требования к дисциплине документирования и обратной совместимости на уровне соглашений. В gRPC обратная совместимость опирается на правила изменения .proto-схемы (нумерация полей, резервирование идентификаторов), что помогает избежать поломок на бинарном уровне, но требует более строгого управления схемой и релизным циклом.

Также важно учитывать, что подход gRPC поддерживает потоковые режимы взаимодействия, которые полезны для телеметрии и сценариев передачи последовательностей сообщений. Однако преимущества потоковых моделей зависят от характера задачи и не всегда проявляются в сценариях простого запроса и ответа. По этой причине корректное сравнение должно фиксировать, какой именно профиль взаимодействия рассматривается и какие допущения сделаны.

Результаты сравнения подходов REST API и gRPC в научных работах и инженерных отчетах часто трудно сопоставимы напрямую [12, 13], поскольку различаются языки реализации, библиотеки, модели конкурентности, структура полезной нагрузки и условия измерения. Это означает, что обобщенные выводы о превосходстве по скорости или удобству разработки имеют смысл только при явно заданных условиях эксперимента. В настоящей работе сравнение выполняется в контролируемой постановке: реализованы два функционально эквивалентных варианта одного сервиса с одинаковыми данными и вычислениями, а различаются только формат сообщений и транспорт взаимодействия. Такая постановка позволяет интерпретировать различия в измеренных метриках как следствие выбранной технологии коммуникации, а не различий бизнес-логики.

Эксперимент сравнения технологий

В качестве прикладного сценария рассматривается мини-сервис аналитики для цифровой образовательной платформы. Сервис получает от других микросервисов информацию о действиях студента (просмотры материалов, выполнение заданий, попытки тестов) и по запросу возвращает агрегированные метрики за период (общее количество действий, среднее время между действиями, количество завершенных заданий и т. п.).

Подобные сервисы характерны для платформ управления обучением, внутренних панелей аналитики и систем отчетности онлайн-курсов. В некоторых реализациях учебная подсистема разворачивается как тесно связанный комплекс, где функциональные части распространяются и запускаются совместно. Это ограничивает возможность гибко заменять или расширять отдельные элементы и усложняет интеграцию внешних программных инструментов для управления обучением [8].

В продуктивной системе подобный сервис вызывается сотни и тысячи раз в секунду, причем часто из других микросервисов. Следовательно, для анализа одинаково важны: низкая задержка; высокая пропускная способность; возможность легко интегрировать новые клиенты.

Архитектура стенда. Для эксперимента было реализовано два варианта одного и того же сервиса аналитики:

1) вариант на основе REST API на Python с JSON поверх HTTP/1.1;
2) вариант на основе gRPC на Python с использованием Protocol Buffers поверх HTTP/2.

Оба сервиса используют одну и ту же модель данных в оперативной памяти, реализуют одинаковую бизнес-логику, работают на одной машине и слушают разные порты.

Конфигурация стенда и программного обеспечения. Эксперимент выполнен на одном вычислительном узле: тестовые сервисы и генератор нагрузки запускались на одном хосте. Такая конфигурация уменьшает влияние внешней сети и позволяет сравнивать накладные расходы протокола и сериализации в контролируемых условиях.

Операционная система — macOS версии 15.6 (Build 24G84). Эксперимент выполнен на ноутбуке MacBook Pro (чип Apple M4 Pro, 12 вычислительных ядер, оперативная память 24 GB).

Интерпретатор — Python 3.9.6. Для реализации тестовых сервисов и генерации нагрузки использовались следующие программные компоненты: FastAPI 0.115.0, Uvicorn 0.30.1, grpcio 1.66.0, grpcio-tools 1.66.0, protobuf 5.29.5, pydantic 2.12.4, для обработки результатов и построения графиков — pandas 2.2.2 и matplotlib 3.9.0.

Порты и точки входа тестовых сервисов: Реализация на основе REST API принимает HTTP-запросы методом POST /api/v1/metrics на порту 8000, реализация на основе gRPC предоставляет метод MetricsService.Calculate на порту 50051.

Стенд включает следующие компоненты:

— реализация на основе REST API с обработчиком POST /api/v1/metrics;
— реализация на основе gRPC с описанием интерфейса в файле metrics.proto;
— генераторы нагрузки bench_rest.py и bench_grpc.py;
— модуль генерации и хранения событий в памяти (data.py);
— скрипт построения графиков по текстовому формату CSV (plot_results.py).

Стенд логически разделен на три группы компонентов (рис. 1): генераторы нагрузки расположены в левой части схемы, реализации на основе REST API и gRPC с общей моделью данных и логикой — в центральной части, система сбора и анализа результатов — в правой части.

1) Генератор нагрузки формирует запросы к реализации на основе REST API или gRPC, измеряет время отклика, считает Requests Per Second (RPS) и базовую статистику по задержкам.
2) Реализации аналитики на основе REST API и gRPC, развернутые параллельно.
3) Система измерений, реализованная в виде простого логгера и агрегирующих функций.

Сценарий и метрики. В эксперименте использован базовый сценарий обращения к агрегированной аналитике, характерный для цифровых образовательных платформ и систем управления обучением: клиент передает идентификатор пользователя и временной интервал, а сервис выполняет фильтрацию событий в памяти и рассчитывает агрегаты. Тело запроса содержит поля user_id, from_ts и to_ts, где from_ts и to_ts задают границы интервала в миллисекундах UNIX.

В качестве фактора нагрузки изменялся уровень параллелизма, т. е. число одновременных клиентских задач, выполняющих запросы к сервису. Измерения проводились для 100, 200 и 500 одновременных клиентов при фиксированной длительности измерительного окна 20 с.

В качестве метрик использовались среднее время отклика, 95-й процентиль задержки и пропускная способность в RPS. На стороне клиента фиксировалось время выполнения каждой попытки запроса, отдельно учитывались число и доля успешных ответов.

Реализация тестовых сервисов. Для сравнения REST API и gRPC был реализован единый тестовый сервис аналитики, рассчитывающий простые агрегированные метрики по действиям пользователей в учебной системе. В памяти сервиса хранится набор событий, каждое из которых содержит идентификатор пользователя, временную метку и числовой показатель score. По запросу сервис выполняет фильтрацию событий по выбранному пользователю и заданному интервалу времени и вычисляет: количество событий, сумму и среднее значение показателя score, а также средний интервал между соседними событиями в выбранном диапазоне.

Набор данных и правила его формирования. Для воспроизводимых измерений используется синтетический набор событий, формируемый программно с фиксированным начальным значением генератора случайных чисел (seed 42). Генерируется данные для 100 пользователей со строковыми идентификаторами от «u1» до «u100», для каждого пользователя устанавливается 1000 событий, всего около 100 000 записей. Временные метки формируются равномерно внутри окна длительностью 30 дней и задаются в миллисекундах UNIX. Значение score генерируется как вещественное число в диапазоне от 0 до 5. После генерации события сортируются по пользователю и времени, что обеспечивает корректное вычисление интервалов между соседними событиями при фильтрации.

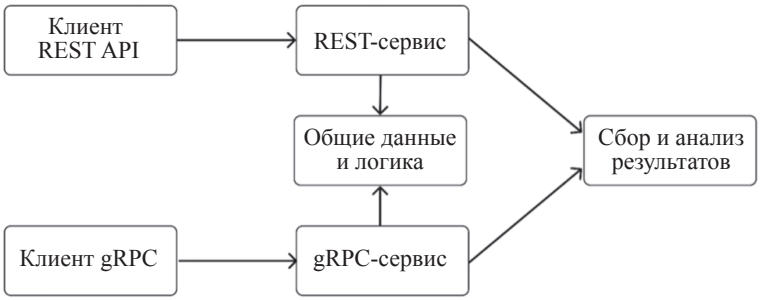

Рис. 1. Схема экспериментального стенда для сравнения REST API и gRPC
Fig. 1. Experimental bench diagram for comparing REST API and gRPC

Вариант сервиса на основе REST API. В этом варианте сервис реализован на Python и запускается как HTTP-приложение на порту 8000. Клиент обращается к методу POST /api/v1/metrics, передавая параметры запроса в формате JSON. Тело запроса содержит поля user_id, from_ts и to_ts, где user_id задает идентификатор пользователя, а from_ts и to_ts задают границы интервала времени в миллисекундах UNIX. Пример тела запроса:

```
{
  "user_id": "u1",
  "from_ts": 1700000000000,
  "to_ts": 1710000000000
}
```

В ответ сервис возвращает агрегированные значения в формате JSON со следующими полями: count (число событий), sum_score (сумма значений score), avg_score (среднее значение score) и avg_interval_ms (средний интервал между соседними событиями в миллисекундах). Пример структуры ответа:

```
{
  "count": 4592,
  "sum_score": 22828.406053670566,
  "avg_score": 4.971342781722684,
  "avg_interval_ms":
131702.69875844044
}
```

Численные значения в ответе зависят от выбранного пользователя и интервала, приведены как иллюстрация формата.

Вариант сервиса на основе gRPC. Этот вариант использует тот же набор данных и те же вычисления, но иной способ описания и вызова операций. Интерфейс задается в файле metrics.proto, где объявлен сервис MetricsService с методом Calculate, а также сообщения MetricsRequest и MetricsResponse. Запрос MetricsRequest содержит поля user_id, from_ts, to_ts, а ответ MetricsResponse возвращает те же агрегированные значения, что и реализация на основе REST API. Описание интерфейса:

```
syntax = "proto3";
package metrics;
service MetricsService {
  rpc Calculate (MetricsRequest)
returns (MetricsResponse) {}
}
message MetricsRequest {
  string user_id = 1;
  int64 from_ts = 2;
  int64 to_ts   = 3;
}
```

Клиент формирует объект MetricsRequest и вызывает метод Calculate. На сервере запрос обрабатывается той же вычислительной логикой, что и в реализации на основе REST API, после чего результат упаковывается в MetricsResponse и передается по HTTP/2 в бинарном формате Protocol Buffers.

Таким образом, обе реализации опираются на общую бизнес-логику и отличаются только транспортным уровнем и форматом сериализации данных. Это позволяет сравнивать подходы REST API и gRPC в равных условиях: при одинаковых данных и идентичной вычислительной нагрузке меняется только технология взаимодействия между клиентом и сервисом.

Нагрузочное тестирование сервисов

Для оценки производительности реализованных сервисов проведено нагрузочное тестирование реализаций на основе REST API и gRPC на одном стенде. Оба сервиса реализуют одинаковую бизнес-логику и используют общий набор событий в оперативной памяти, различия между реализациями ограничены транспортным уровнем и форматом представления сообщений: REST API использует HTTP/1.1 и JSON, а gRPC — HTTP/2 и Protocol Buffers.

Нагрузка генерировалась отдельными программами на Python: bench_rest.py для реализации на основе REST API и bench_grpc.py для реализации на основе gRPC. Модель параллелизма реализована с использованием потоков: при уровне параллелизма N создаются N потоков, каждый из которых в течение заданного окна выполняет последовательные запросы максимально часто, без пауз между запросами. В программе bench_rest.py используются синхронные HTTP-запросы библиотеки requests, в программе bench_grpc.py — вызовы метода сервиса через Python-библиотеку grpcio. Таймаут клиентского вызова установлен равным 5 с.

В проведенных измерениях использовалось измерительное окно длительностью 20 с и уровни параллелизма 100, 200 и 500 одновременных клиентов.

Сбор и расчет метрик. Измерения выполнялись на стороне клиента. Для каждой попытки запроса фиксировались моменты начала и окончания, после чего вычислялось время отклика в миллисекундах. В массив задержек включались все попытки, в том числе завершившиеся ошибкой или таймаутом (в этом случае измеренное время отклика близко к установленному таймауту). Успешность учитывалась отдельно: для реализации на основе REST API успешными считались ответы со статусом 200, для реализации на основе gRPC — вызовы метода Calculate, завершившиеся без исключения.

В качестве метрик использовались среднее время отклика, медиана и процентильные оценки (95-й и 99-й процентили), рассчитанные по отсортированному массиву времен отклика. Пропускная способность вычислялась как отношение общего числа попыток запросов на длительность измерительного окна в секундах. Дополнительно фиксировалось число успешных ответов и их доля, что позволяет интерпретировать результаты не только по скорости, но и по устойчивости обработки запросов при росте нагрузки.

Результаты каждого прогона сохранялись в два CSV-файла: rest_results.csv и grpc_results.csv. Каждая строка содержит параметры прогона и агрегированные значения: технологию, длительность окна, уровень па-

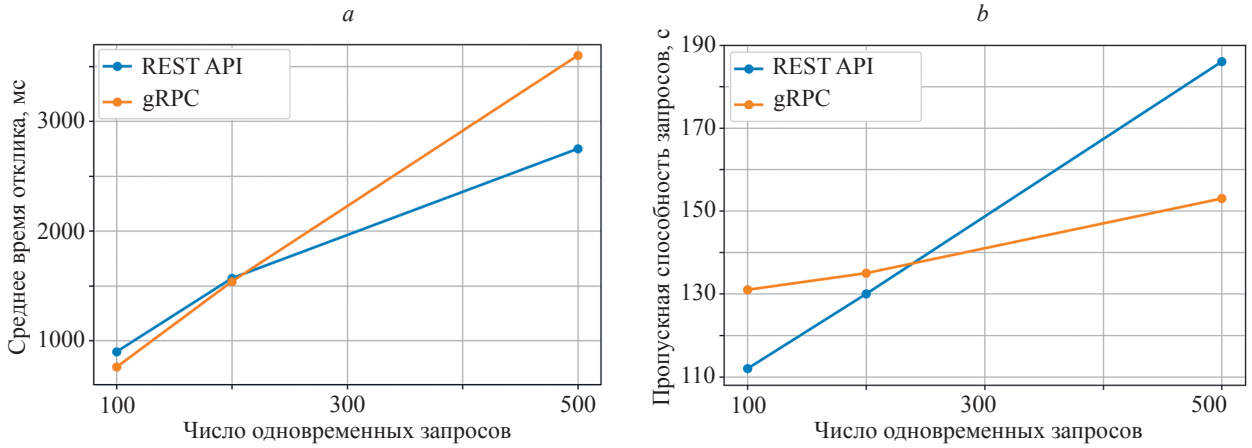

Рис. 2. Зависимости среднего времени отклика (*a*) и пропускной способности запросов (*b*) для реализаций на основе REST API и gRPC от числа одновременных запросов

Fig. 2. Dependences of the average response time (*a*) and request throughput (*b*) for implementations based on REST API and gRPC on the number of simultaneous requests

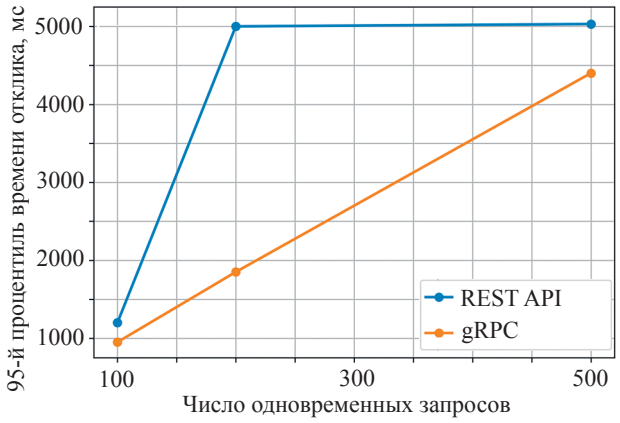

Рис. 3. Изменение 95-го процентиля времени отклика для реализаций на основе REST API и gRPC при росте нагрузки

Fig. 3. Change in the 95th percentile of response time for implementations based on REST API and gRPC with increasing load

раллелизма, число попыток запросов, число успешных запросов, пропускную способность, среднее время отклика и процентильные оценки задержки (включая медиану, 95-й и 99-й процентили). На основе полученных файлов были построены графики зависимости среднего времени отклика, пропускной способности и 95-го процентиля задержки от числа одновременных клиентов (рис. 2 и 3).

Параметры воспроизведения эксперимента

Для проведения эксперимента использованы следующие параметры: длительность измерительного окна — 20 с; уровни параллелизма — 100, 200 и 500 одновременных клиентов; модель конкурентности — многопоточность (threading): *N* потоков, каждый выполняет последовательные запросы без пауз между ними; таймаут клиентского запроса — 5 с; данные — 100 пользователей, 1000 событий на пользователя, фиксированный seed 42, временное окно генерации 30 дней; точки входа — POST /api/v1/metrics (порт 8000) и MetricsService.Calculate (порт 50051).

Процедура воспроизведения. Для воспроизведения эксперимента необходимо подготовить окружение Python 3.9.6 и установить библиотеки в версиях, приведенных в подразделе «Конфигурация стенда и программного обеспечения». Далее выполняются следующие шаги.

1. Сгенерировать Python-стабы для gRPC по файлу metrics.proto командой python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. metrics.proto.
2. Запустить реализацию на основе REST API на порту 8000 командой uvicorn rest_server:app --host 127.0.0.1 --port 8000 и реализацию на основе gRPC на порту 50051 командой python3 grpc_server.py.
3. Выполнить прогоны длительностью 20 с при 100, 200 и 500 одновременных клиентах, используя клиентские программы bench_rest.py и bench_grpc.py.
4. Результаты сохраняются в файлы rest_results.csv и grpc_results.csv и используются для построения графиков (рис. 2 и 3).

Сравнение результатов

Как видно из рис. 2, *a*, при 100 одновременных клиентах среднее время отклика у реализации на основе gRPC ниже, чем у реализации на основе REST API. При увеличении параллелизма до 200 клиентов средние значения становятся сопоставимыми. При 500 клиентах среднее время отклика у gRPC возрастает сильнее, чем у REST API, что показывает: по одной только средней задержке нельзя однозначно объявить победителя для всех режимов нагрузки.

На рис. 2, *b* заметно изменение пропускной способности при росте числа одновременных клиентов. При 100–200 клиентах значения RPS у двух реализаций близки, а при 500 клиентах более высокое значение RPS демонстрирует реализация на основе REST API. Однако этот показатель следует рассматривать вместе с

задержками, поскольку рост RPS может сопровождаться ухудшением качества обслуживания.

Наиболее показательные различия видны на рис. 3. При 200 и 500 одновременных клиентах 95-й процентиль времени отклика у REST API резко увеличивается и достигает значений около 5 с, т. е. значительная часть запросов начинает выполняться заметно дольше среднего. У gRPC рост 95-го процентиля происходит более плавно, и верхние задержки остаются ниже. Таким образом, при высокой конкуренции запросов gRPC обеспечивает более предсказуемое время отклика и меньшую долю очень медленных ответов, что особенно важно для внутренних межсервисных вызовов и аналитических сценариев, чувствительных к задержкам. При этом REST API сохраняет практическую привлекательность для простых внешних интеграций за счет привычного HTTP-подхода и широкой совместимости [14].

Преимущества и недостатки технологий в рассматриваемой экспериментальной постановке. Реализация на основе REST API показывает себя как простой и удобный вариант для разработки, отладки и интеграции. Такой сервис легко проверить через браузер или командную строку cURL, ответы в формате JSON удобно читать и логировать, фронтенд-разработчикам не нужно осваивать дополнительные инструменты. При умеренной нагрузке подход REST API полностью учитывает потребности учебной платформы, и дополнительные несколько миллисекунд выигрыша не критичны.

Отметим, что проведенный эксперимент показал ограничения подхода REST API на высоких нагрузках [15]. Текстовая сериализация и особенности HTTP/1.1 приводят к росту задержек и более раннему насыщению по RPS. gRPC, напротив, потребует больше усилий на старт — нужно описать .proto-контракты, настроить сервер и клиентов, разобраться с HTTP/2, но в обмен даст более низкие задержки, лучшую масштабируемость и строгую типизацию.

В результате сравнения видно, что REST API и gRPC решают разные классы задач и нередко используются совместно. Если обратить внимание на скорость, строгую типизацию и контролируемую эволюцию интерфейса, то выигрывает подход gRPC. Когда важнее совместимость, простая отладка и минимальный порог входа для клиентов, то практичнее REST API [12].

Практические рекомендации. Исходя из эксперимента, можно сформулировать несколько практических выводов. Для внутренних вызовов между микросервисами, где критичны задержки и пропускная способность, рационально использовать gRPC, так как он дает меньшее время отклика, лучше держит высокий RPS и обеспечивает строгий контроль контрактов. Если основными потребителями API являются веб-интерфейсы, мобильные приложения [16] и внешние информационные системы, более удобен REST API. Связка HTTP и JSON проще для людей и инструментов, быстрее поддерживаются изменения, а требования к инфраструктуре мягче, что на ранних этапах проекта часто важнее максимальной эффективности протокола.

Заключение

Проведено нагрузочное сравнение двух функционально эквивалентных реализаций одного сервиса: реализации на основе Representational State Transfer Application Programming Interface с использованием Hypertext Transfer Protocol версии 1.1 и формата JavaScript Object Notation, а также реализации на основе Google Remote Procedure Calling с использованием Hypertext Transfer Protocol версии 2 и формата Protocol Buffers. По результатам измерений различия зависят от уровня параллелизма. При 100 одновременных клиентах Google Remote Procedure Calling обеспечивает меньшее среднее время отклика. При увеличении нагрузки до 200–500 клиентов по среднему времени отклика и пропускной способности в данной постановке Representational State Transfer Application Programming Interface демонстрирует сопоставимые или более высокие значения.

При этом по 95-му процентилю время отклика при 200 и 500 одновременных клиентах значения для Representational State Transfer Application Programming Interface заметно выше и приближаются к уровню клиентского таймаута, тогда как у Google Remote Procedure Calling верхние задержки растут более плавно и остаются ниже. Следовательно, при высокой конкуренции запросов Google Remote Procedure Calling дает более предсказуемое качество обслуживания за счет меньших задержек на верхних процентилях, даже если средние значения в отдельных режимах оказываются сопоставимыми.

Практически это означает, что для внутренних межсервисных вызовов, где критичны предсказуемость и устойчивость задержек, предпочтителен Google Remote Procedure Calling. Для сценариев, ориентированных на максимальную пропускную способность и средние показатели в рамках рассматриваемого запроса, Representational State Transfer Application Programming Interface в данной реализации показал конкурентоспособный результат и требует выбора с учетом требований к интерфейсу и окружению.

Литература

1. Кравченко Д.А. Микросервисная архитектура // Интерактивная наука. 2022. № 4 (69). С. 43–44. doi: 10.21661/r-556565
2. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. СПб.: Питер, 2022. 544 с.
3. Ньюмен С. Создание микросервисов. СПб.: Питер, 2016. 300 с.
4. Бужин И.Г., Деревянкин А.Ю., Антонова В.М., Перевалов А.П., Миронов Ю.Б. Эффективность фреймворков для передачи ин-

References

1. Kravchenko D.A. Microservice architecture. *Interactive Science*, 2022, no. 4 (69), pp. 43–44. (in Russian). doi: 10.21661/r-556565
2. Richardson C. *Microservices Patterns*. Manning, 2019, 520 p.
3. Newman S. *Building Microservices*. O'Reilly Media, 2015, 250 p.
4. Buzhin I.G., Derevyankin A. Yu., Antonova V.M., Perevalov A.P., Mironov Yu.B. The effectiveness of frameworks for transmitting information in a virtualized communication network with a

формации в виртуализированной сети связи с микросервисной архитектурой // Наукоёмкие технологии в космических исследованиях Земли. 2023. Т. 15. № 2. С. 23–32. doi: 10.36724/2409-5419-2023-15-2-23-32
5. Ирбитский И.С., Романенков А.М., Стульников К.Т., Удалов Н.Н. Анализ микросервисных архитектур и моделей // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. 2022. № 3. С. 83–90. doi: 10.37882/2223-2966.2022.03.15
6. Пацей Н.В., Шитько А.М. Интеграция микросервисов на основе RPC // Эпоха науки. 2021. № 27. С. 32–37. doi: 10.24412/2409-3203-2021-27-32-37
7. Вальдивия Х.А., Лора-Гонсалес А., Лимон К., Кортес-Вердин К., Очаран-Эрнандес Х.О. Паттерны микросервисной архитектуры: многопрофильный обзор литературы // Труды Института системного программирования РАН. 2021. Т. 33. № 1. С. 81–96. doi: 10.15514/ISPRAS-2021-33(1)-6
8. Босенко Т.М. Реализация микросервисной архитектуры в системе управления процессом обучения // Вестник МГПУ. Серия «Информатика и информатизация образования». 2024. № 2 (68). С. 106–114. doi: 10.25688/2072-9014.2024.68.2.10
9. Босенко Т.М. Анализ микросервисной архитектуры в среде e-learning с многовариантным доступом к учебным материалам // Международный научно-исследовательский журнал. 2024. № 10 (148). С. 98. doi: 10.60797/IRJ.2024.148.65
10. Sangabriel-Alarcón J., Ocharán-Hernández J., Limón X., Cortés-Verdín M. Domain-driven design in microservices architecture // Proceedings of the Institute for System Programming of the RAS. 2024. V. 36. N 6. P. 39–58. doi: 10.15514/ISPRAS-2024-36(6)-3
11. Васильев Б.Я. Методика тестирования API в микросервисной архитектуре продукта // Международный журнал гуманитарных и естественных наук. 2025. № 4-1 (103). С. 154–159. doi: 10.24412/2500-1000-2025-4-1-154-159
12. Андрианов А.В., Калинин А.С., Лаптева М.Г. Использование gRPC в современных микросервисных архитектурах: плюсы и минусы по сравнению с REST // Вестник науки. 2025. Т. 2. № 6 (87). С. 1575–1583.
13. Kvancz B., Forstner B. Performance review of using gRPC for inter-service communication in microservice architecture // Proc. of the 2nd Workshop on Intelligent Infocommunication Networks, Systems and Services (WI2NS2). 2024. P. 31–35. doi: 10.3311/wins2024-006
14. Орлов В.А., Гавриленко А.В. Интеллектуальные голосовые помощники для служб технической поддержки // Успехи кибернетики. 2025. Т. 6. № 3. С. 96–104.
15. Салтанов Д.С., Арсентьева Н.В. Сравнительный анализ архитектурных подходов к разработке REST API для высоконагруженных систем // Форум молодых учёных. 2025. № 6 (106). С. 329–340.
16. Нематилаев Б.К., Ташполотов Ы., Омурбекова Г.К. Исследование методологии облачных технологий, совместимых с платформой Android и их возможностей // Вестник науки. 2025. Т. 3. № 4 (85). С. 854–861.

microservice architecture. *H&ES Research*, 2023, vol. 15, no 2, pp. 23–32. (in Russian). doi: 10.36724/2409-5419-2023-15-2-23-32
5. Irbitskiy I.S., Romanenkov A.M., Stulnikov K.T., Udalov N.N. Analysis of microservice architectures and models. *Modern Science: actual problems of theory and practice. Series: Natural and Technical Sciences*. 2022, no. 3, pp. 83–90. (in Russian). doi: 10.37882/2223-2966.2022.03.15
6. Patsei N.V., Shytska A.M. Microservices integration based on RPC. *Era of Science*, 2021, no. 27, pp. 32–37. (in Russian). doi: 10.24412/2409-3203-2021-27-32-37
7. Valdivia J., Lora-Gonzalez A., Limón X., Cortes-Verdin K., Ocharán-Hernández J. Patterns related to microservice architecture: a multivocal literature review. *Proceedings of the Institute for System Programming of the RAS*, 2021, vol. 33, no. 1, pp. 81–96. (in Russian). doi: 10.15514/ISPRAS-2021-33(1)-6
8. Bosenko T.M. Implementation of microservice architecture in the training process management system. *Bulletin of the Moscow City Pedagogical University. Series "Pedagogy and Psychology"*, 2024, no. 2 (68), pp. 106–114. (in Russian). doi: 10.25688/2072-9014.2024.68.2.10
9. Bosenko T.M. Analysis of microservice architecture in e-learning environment with multivariate access to learning materials. *International Research Journal*, 2024, no. 10 (148), pp. 98. (in Russian). doi: 10.60797/IRJ.2024.148.65
10. Sangabriel-Alarcón J., Ocharán-Hernández J., Limón X., Cortés-Verdín M. Domain-driven design in microservices architecture. *Proceedings of the Institute for System Programming of the RAS*, 2024, vol. 36, no. 6, pp. 39–58. doi: 10.15514/ISPRAS-2024-36(6)-3
11. Vasiliev B.Y. API testing methodology in the micro-service architecture of the product. *International Journal of Humanities and Natural Sciences*, 2025, no. 4-1 (103), pp. 154–159. (in Russian). doi: 10.24412/2500-1000-2025-4-1-154-159
12. Andrianov A.V., Kalinin A.S., Lapteva M.G. Using gRPC in modern microservice architectures: pros and cons compared to REST. *Science Bulletin*, 2025, vol. 2, no. 6 (87), pp. 1575–1583. (in Russian)
13. Kvancz B., Forstner B. Performance review of using gRPC for inter-service communication in microservice architecture. *Proc. of the 2nd Workshop on Intelligent Infocommunication Networks, Systems and Services (WI2NS2)*, 2024, pp. 31–35. doi: 10.3311/wins2024-006
14. Orlov V.A., Gavrilenko A.V. Intelligent voice assistants for technical support services. *Russian Journal of Cybernetics*, 2025, vol. 6, no. 3, pp. 96–104. (in Russian)
15. Saltanov D.S., Arsentyeva N.V. Comparative analysis of architectural approaches to REST API development for high-load systems. *Forum of Young Scientists*, 2025, no. 6 (106), pp. 329–340. (in Russian)
16. Nimatilaev B.K., Tashpolotov Y., Omurbekova G.K. Research methodology of cloud computing platform-compatible technologies Android and their features. *Science Bulletin*, 2025, vol. 3, no. 4 (85), pp. 854–861. (in Russian)

Авторы

Баженов Артем Эдуардович — старший преподаватель, Поволжский государственный университет телекоммуникаций и информатики, Самара, 443010, Российская Федерация, https://orcid.org/0009-0001-5887-2077, a.bazhenov@psuti.ru

Маленкова Арина Алексеевна — студент, Поволжский государственный университет телекоммуникаций и информатики, Самара, 443010, Российская Федерация, https://orcid.org/0009-0005-0755-895X, kroshik0812@vk.com

Майоров Никита Владимирович — студент, Поволжский государственный университет телекоммуникаций и информатики, Самара, 443010, Российская Федерация, https://orcid.org/0009-0002-7502-8860, nikita@m7n.ru

Authors

Artem E. Bazhenov — Senior Lecturer, Volga Region State University of Telecommunications and Informatics, Samara, 443010, Russian Federation, https://orcid.org/0009-0001-5887-2077, a.bazhenov@psuti.ru

Arina A. Malenkova — Student, Volga Region State University of Telecommunications and Informatics, Samara, 443010, Russian Federation, https://orcid.org/0009-0005-0755-895X, kroshik0812@vk.com

Nikita V. Maiorov — Student, Volga Region State University of Telecommunications and Informatics, Samara, 443010, Russian Federation, https://orcid.org/0009-0002-7502-8860, nikita@m7n.ru

Статья поступила в редакцию 19.11.2025
Одобрена после рецензирования 24.04.2026
Принята к печати 22.05.2026

Received 19.11.2025
Approved after reviewing 24.04.2026
Accepted 22.05.2026