УДК 004.4`2

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ТЕХНОЛОГИЙ REST и gRPC ДЛЯ ВЗАИМОДЕЙСТВИЯ ВЕБ-СЕРВИСОВ

Шаяхметов А.Р., студент гр. 4410, IV курс
Научный руководитель: Валитова Н.Л., к.т.н., доцент
Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ
г. Казань

Введение

В современной разработке распределённых систем выбор архитектуры API является ключевым моментом. Долгое время REST оставался наиболее распространённым подходом, тогда как технология gRPC, основанная на концепции удалённого вызова процедур, только набирает популярность в высокопроизводительных микросервисных архитектурах. Данная статья рассматривает основные концепции, сравнительный анализ и рекомендации по выбору подхода в зависимости от специфики проекта.

Основные концепции

REST – это архитектурный стиль, который использует стандартные HTTP-методы (GET, POST, PUT, DELETE) для управления ресурсами. Данные чаще всего передаются в формате JSON, что обеспечивает их удобочитаемость и простоту использования.

gRPC – это фреймворк для реализации удалённых вызовов процедур, разработанный Google [1]. Он использует бинарный протокол Protocol Buffers для сериализации данных и работает на базе HTTP/2, что позволяет эффективно реализовывать мультиплексирование, двунаправленную потоковую передачу и снижать задержки. Для gRPC характерен строгий контракт, описываемый в proto файлах, что упрощает генерацию кода для клиентов и серверов на различных языках программирования.

Архитектурные особенности

Архитектурные различия между REST и gRPC определяют их область применения и эффективность в различных сценариях. REST основан на традиционной клиент-серверной модели взаимодействия, где каждое сообщение содержит всю необходимую информацию, а сервер не хранит состояние клиента между запросами. Это упрощает масштабирование и делает архитектуру отказоустойчивой, но приводит к избыточной передаче данных [2]. REST-API следуют единообразному интерфейсу, используя стандартные методы HTTP, такие как GET, POST, PUT и DELETE, что делает их понятными

и предсказуемыми. Одним из преимуществ является кэшируемость, позволяющая снизить нагрузку на сервер за счёт повторного использования ранее полученных данных. REST также использует слоистую архитектуру, где запросы могут проходить через балансировщики нагрузки, кэши и прокси-серверы. Наиболее распространённый формат данных – JSON, который удобен для работы, но не так эффективен по объёму и скорости обработки, как бинарные форматы.

gRPC ориентирован на высокопроизводительные распределённые системы. В его основе лежит использование HTTP/2, что позволяет открывать одно соединение и отправлять несколько запросов одновременно, снижая накладные расходы на установку соединений [3], как показано на рис. 1.

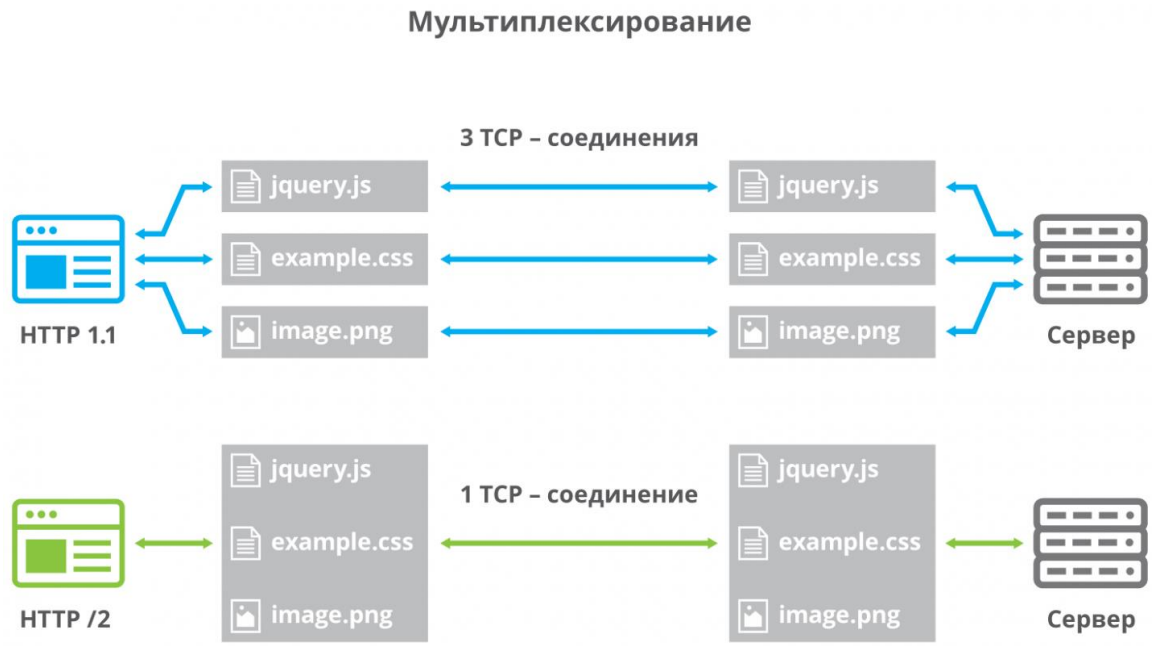

Рис. 1. Схема работы механизма мультиплексирования в HTTP/2

Вместо текстового формата JSON gRPC использует бинарную сериализацию на основе Protocol Buffers, что делает передачу данных более компактной и ускоряет их обработку. Одним из ключевых преимуществ является поддержка потокового взаимодействия, позволяющая серверу и клиенту обмениваться данными в реальном времени [4]. В отличие от REST, где взаимодействие строго ограничено моделью «запрос-ответ», gRPC поддерживает клиентский, серверный и двунаправленный стриминг. Код для клиента и сервера автоматически генерируется на основе файлов описания интерфейсов взаимодействия (.proto), что упрощает работу в многоязычной среде и сокращает вероятность ошибок. Благодаря оптимальному использованию сетевых ресурсов и низким задержкам gRPC особенно эффективен для микросервисных архитектур [5], и высоконагруженных систем. Ниже в таблице 1 представлено сравнение основных технических различий в обеих технологиях

Таблица 1.

Сравнительная таблица технических характеристик REST и gRPC

Характеристика	REST	gRPC
Протокол	HTTP/1.1	HTTP/2
Формат данных	JSON, XML	Protocol Buffers (бинарный формат)
Модель взаимодействия	Запрос-ответ	Унарные, потоковые, двунаправленные вызовы
Производительность	Ограничена пропускной способностью HTTP/1.1	Высокая, оптимизирована для нагрузок
Поддержка браузеров	Полная	Ограниченная (требуется gRPC-web)
Генерация кода	Зависит от сторонних инструментов	Встроенная генерация через protoc

Для более наглядного понимания сильных и слабых сторон обоих подходов ниже в таблице 2 представлена сравнительная таблица ключевых характеристик REST и gRPC, их преимущества и недостатки в контексте различных сценариев применения.

Таблица 2.

Преимущества и недостатки REST и gRPC

Технология	Преимущества	Недостатки
REST	1. Простота и широкая распространённость. 2. Человекочитаемый формат данных (JSON) упрощает отладку и интеграцию. 3. Гибкость и независимость клиента и сервера. 4. Использование стандартных HTTP-методов упрощает интеграцию	1. Малая эффективность при передаче больших объёмов данных. 2. Модель «запрос-ответ» не всегда подходит для реального времени. 3. Не поддерживает двунаправленный стриминг
gRPC	1. Высокая производительность благодаря бинарной сериализации и HTTP/2. 2. Поддержка унарных и потоковых вызовов.	1. Более высокая сложность внедрения. 2. Ограниченная поддержка

Технология	Преимущества	Недостатки
	3. Автоматическая генерация кода по файлам .proto. 4. Эффективное использование сетевых ресурсов.	браузерами (требуется gRPC-web). 3. Бинарный формат менее удобен для отладки.

Сценарии применения

REST отлично подходит для публичных API, веб-приложений и микросервисов, где важна простота интеграции, человекочитаемость и широкая поддержка стандартных HTTP-методов. Примеры включают приложения для CRUD-операций, мобильные и веб-сервисы с низкими требованиями к задержкам.

gRPC рекомендуется для внутренних API в микросервисных архитектурах, систем, требующих высокой производительности и эффективной потоковой передачи данных (например, в системах реального времени, IoT-решениях и распределённых вычислительных системах).

Заключение

Выбор между REST и gRPC зависит от специфических требований проекта. REST остаётся удобным выбором для веб-приложений благодаря простоте, совместимости и широкой поддержке, тогда как gRPC предоставляет высокую производительность и гибкость в средах с высокой нагрузкой и сложными схемами взаимодействия. Разработчики и архитекторы должны учитывать особенности приложения, требования к производительности и возможности поддержки различных платформ для принятия оптимального решения.

Список литературы:

1. About gRPC. Who is using gRPC and why. [Электронный ресурс] — Режим доступа. — URL: https://grpc.io/about/ (дата обращения 20.03.2025)
2. Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. [Электронный ресурс] — Режим доступа. — URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата обращения 21.03.2025)
3. HTTP/2 и HTTP/3. [Электронный ресурс] — Режим доступа. — URL: https://ru.hexlet.io/courses/http_protocol/lessons/http_x/theory_unit (дата обращения 21.03.2025)
4. Core concepts, architecture and lifecycle. [Электронный ресурс] — Режим доступа. — URL: https://grpc.io/docs/what-is-grpc/core-concepts/ (дата обращения 21.03.2025)

5. Mohamed Hassan. Choosing the Right Communication Protocol for your Web Application. [Электронный ресурс] — Режим доступа. — URL: https://arxiv.org/abs/2409.07360 (дата обращения 21.03.2025)