Оглавление

[2.3.4 Проблемы многозадачности 2](#_Toc168006805)

[3. РЕАЛИЗАЦИЯ ВЫШЕОПИСАННЫХ МЕТОДОВ ПЕРЕДАЧИ ДАННЫХ В КЛИЕНТ – СЕРВЕРНОЙ АРХИТЕКТУРЕ 3](#_Toc168006806)

[3.1 Реализация передачи данных по НТТР 1.1 4](#_Toc168006807)

[3.1.1 Синхронная передача данных 4](#_Toc168006808)

[3.1.2 Асинхронная передача данных 5](#_Toc168006809)

[3.1.3. Многопоточность 5](#_Toc168006810)

[3.2 Реализация передачи данных по НТТР2 6](#_Toc168006811)

[3.2.1 Добавление параметров и сертификатов 6](#_Toc168006812)

[3.2.2 gRPC 7](#_Toc168006813)

[3.3 Сокеты 9](#_Toc168006814)

[3.4 Вывод по главе 3 11](#_Toc168006815)

[ГЛАВА 4 СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РАЗЛИЧНЫХ МЕТОДОВ ПЕРЕДАЧИ ДАННЫХ 11](#_Toc168006816)

[4.1 Методология сравнения 11](#_Toc168006817)

[4.2 Сравнение НТТР1 и НТТР2 11](#_Toc168006818)

[4.3 Сравнение синхронной и асинхронной передачи 11](#_Toc168006819)

[4.4 Сравнение всего 11](#_Toc168006820)

[4.5 Выводы по главе 4 11](#_Toc168006821)

### 2.3.4 Проблемы многозадачности

#### 2.3.4.1 GIL – global interpreter lock

Глобальная блокировка интерпретатора (global interpreter lock – GIL). GIL не дает Python-процессу исполнять более одной команды байт-кода в каждый момент времени. Это означает, что, даже если имеется несколько потоков на многоядерной машине, интерпретатор сможет в каждый момент исполнять только один поток. GIL освобождается при вводе – выводе. В этом случае низкоуровневые системные вызовы работают за пределами среды выполнения Python и не зависят от него. GIL захватывается снова, только когда полученные данные переносятся в объект Python.

#### 2.3.4.2 Взаимная блокировка и состояние гонки

К проблемам реализации многозадачности через многопроцессность и многопоточность также стоит отнести взаимную блокировку и состояние гонки. Взаимная блокировка, или deadlock, это состояние в многозадачной среде, когда два или более потока находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение.

Взаимная блокировка может возникнуть, когда два или более потока пытаются получить доступ к общим ресурсам в определенном порядке. Если каждый из потоков держит ресурс, который нужен другому потоку, все они могут оказаться в состоянии вечного ожидания, так как ни один из потоков не может продолжить выполнение, пока другой поток не освободит нужный ему ресурс. Эта проблема в равной степени распространяется и на процессы.

Взаимная блокировка является серьезной проблемой, которую следует избегать при проектировании многопоточных приложений. Она может привести к тому, что приложение будет “зависать” или работать неэффективно.

Состояние гонки (или race condition) - это ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода.

Состояние гонки возникает, когда несколько потоков или процессов одновременно пытаются получить доступ к одним и тем же ресурсам (например, переменным памяти, файлам или устройствам ввода-вывода) без синхронизации. В результате, порядок, в котором операции выполняются, может влиять на конечный результат, что делает поведение программы неопределенным.

# 3. РЕАЛИЗАЦИЯ ВЫШЕОПИСАННЫХ МЕТОДОВ ПЕРЕДАЧИ ДАННЫХ В КЛИЕНТ – СЕРВЕРНОЙ АРХИТЕКТУРЕ

Клиент передает серверу данные каждого файла и его название.

В этой главе описывается реализация следующих методов передачи данных:

* Синхронная передача данных по HTTP/1 и HTTP/2
* Асинхронная передача данных по HTTP/1 и HTTP/2
* Многопоточная передача данных
* Передача данных через gRPC
* Передача данных с использованием сокетов

Для наглядности, термин "клиент" будет применяться к серверу, отправляющему данные, а "сервер" - к серверу, принимающему данные. В процессе передачи данных клиент отправляет серверу не только сами файлы, но и их названия. Это обеспечивает дополнительный контекст и упрощает управление файлами на стороне сервера. Все методы оптимизированы для эффективной передачи большого количества файлов.

Измерение времени передачи осуществляется с помощью библиотеки `time` в момент начала и окончания передачи каждого файла. Итоговое время передачи рассчитывается путем суммирования времени передачи всех файлов, а затем вычисляется среднее значение из десяти тестов.

Тестирование проводилось на компьютере с процессором Intel i3, \_ ГБ оперативной и \_ ГБ видеопамяти, при передаче 1000 изображений общим размером 6.47 ГБ.

Сервер, принимающий данные, реализован единообразно для всех методов передачи. Он отвечает за прием файлов и подтверждение успешного получения данных.

Для обработки входящих данных разработана функция, которая управляет данными, обрабатывает ошибки и возвращает статус-ответ. В клиент – серверной архитектуре FastAPI сервер функционирует асинхронно, что требует использования асинхронных функций и ключевых слов, описанных в разделе 2.3.6. Для связывания функций с соответствующими запросами были применены декораторы маршрутизации.

Разработанное ASGI – приложение запускалось посредством функции «run» гиперсервера Uvicorn с указанием экземпляра приложения, хоста и порта.

## 3.1 Реализация передачи данных по НТТР 1.1

### 3.1.1 Синхронная передача данных

Синхронная передача данных реализована через специально разработанную функцию, использующую библиотеку «requests». Эта функция осуществляла POST – запросы к серверу, передавая данные и путь к серверному приложению в качестве параметров.

Передача файлов происходила последовательно: сервер начинал обработку следующего файла только после подтверждения приема предыдущего. Этот процесс описан в разделе 2.3.2 и занял в среднем 8.02940 секунд.

### 3.1.2 Асинхронная передача данных

3.1.2.1Асинхронный клиент был создан для улучшения производительности передачи данных. Используя асинхронные функции для чтения и открытия файлов, которые описаны в разделе 2.3.6, каждый файл обрабатывался как отдельная задача в цикле событий. Задачи выполнялись конкурентно в одном потоке, что позволяло эффективно использовать время ожидания и сократило время передачи до 4.8172 секунд.

3.1.2.2 После внесения изменений в серверную часть с использованием модуля UploadFile, время передачи увеличилось до 31.453 секунд. Это было связано с дополнительными возможностями управления файлами и передачи сопутствующих данных, описанными в разделе 2.2.

### 3.1.3. Многопоточность

Внедрение многопоточности в клиентскую часть программы привело к изменению методики передачи данных. Процесс работы этого метода иллюстрируется блок-схемой на Рисунке 14.

БЛОК СХЕМА

Рисунок 14 - Реализация клиента в многопоточной передаче

Используя библиотеку httpx, был создан объект клиента, для каждого файла из списка инициировался отдельный поток, который отправлял файл на сервер. Это действие осуществлялось с применением специального класса для многопоточности, упомянутого в разделе 2.3.3, где также указывалось количество потоков.

Метод submit использовался для назначения исполняемых функций потокам и их запуска. Он возвращает объект Future, представляющий собой асинхронное выполнение этой функции. Для управления потоками были разработаны – функция, которая возвращает завершенные задачи по мере их выполнения, а также метод для извлечения результата получения готовых задач.

В этой реализации все задачи выполнялись конкурентно, переключаясь между потоками для оптимизации времени выполнения. Общее время, затраченное на передачу данных, составило 6.0617 секунд. Передача осуществлялась по протоколу HTTP 1.1. Детали реализации и результаты передачи по протоколу HTTP 2 описаны в следующем разделе.

## 3.2 Реализация передачи данных по НТТР/2

### 3.2.1 Добавление параметров и сертификатов

Реализация передачи по протоколу HTTP/2 была адаптирована из предыдущей версии с ключевым отличием: включением сертификатов безопасности. Сертификаты, состоящие из ключа и сертификата, были самоподписными для целей исследования.

Запуск сервера, в данном случае, осуществлялся через гиперсервер Hypercorn, как описано в разделе 2.1.

Результаты передачи данных по протоколу НТТР/2 синхронным и асинхронным способами, а также многопоточностью, указаны в таблице 3.

Таблица 3 – Результаты передачи данных по НТТР/2

|  |  |  |
| --- | --- | --- |
| Синхронная передача данных | Асинхронная передача данных | Многопоточность |
| 40.0234 секунд | 33.595 секунд | 38.376 секунд |

Из результатов следует, что асинхронная передача данных по протоколу НТТР/2 значительно быстрее синхронной, которая в свою очередь уступает многопоточности на 3 секунды.

### 3.2.2 gRPC

gRPC использует передачу данных по протоколу HTTP/2 по умолчанию и не требует доступа к веб – ресурсам, что исключает необходимость создания сертификатов безопасности при отсутствии API Gateway.

В текущей работе реализованы три из четырех возможных способов передачи данных между сервисами, описанных в разделе 2.5.

Программирование на gRPC начинается с описания proto файла, изображенного на Рисунке 15.

РИСУНОК

Рисунок 15 - Реализация proto файла

Он определяет интерфейс gRPC – сервиса и методы удаленной передачи данных, как показано в пунктах 1 – 4 на Рисунке 15. Эти методы включают унарную передачу, потоковую передачу запросов от клиента к серверу и двунаправленную потоковую передачу. Сообщения запроса и ответа имеют определенный формат, а каждое поле сообщения идентифицируется уникальным номером. Это продемонстрированно в пунктах 5 – 6.

Для автогенерации серверного и клиентского кода из описанного выше файла структуры предусмотрен специальный компилятор, описанный в разделе 2.5.

На серверной стороне необходимо реализовать логику сервиса, переопределив базовый класс сгенерированного каркаса. Для этого были реализованы удаленные функции на языке Python, представленные на Рисунок 16.

Сервер gRPC прослушивает клиентские запросы и возвращает ответы сервиса.

РИСУНОК

Рисунок 16 - Серверная реализация получения данных и отправки ответов.

Сервер gRPC прослушивает клиентские запросы и возвращает ответы сервиса. Его реализация изображена на Рисунке 17.

РИСУНОК

Рисунок 17 – Реализация запуска сервера

В разработанном сервере регистрируется сгенерированный ранее сервис "FileTransferService".

Используя структуру приложения в proto файле также можно сгенерировать клиентский шаблон, который предоставляет доступ к тем же методам, что и сервер. Эти методы затем преобразуются в удаленные вызовы, направленные к серверу.

Клиентская реализация включает установление соединения с сервером и использование клиентского шаблона для вызова удаленных методов. Детали этого процесса демонстрируются на Рисунках 18 и 19 ниже.

РИСУНОК

Рисунок 18 –Реализация клиентской стороны

РИСУНОК

Рисунок 19 – Вызов удаленного метода на клиенте

Результаты передачи данных, полученные при унарной передаче, потоковой передачи от клиента серверу и двунаправленной потоковой передачи, указаны в таблице 4.

Таблица 4 - Результаты передачи данных с помощью архитектуры gRPC.

|  |  |  |
| --- | --- | --- |
| Унарная передача данных | Потоковая передача данных | Двунаправленная передача данных |
| 7.72575 секунд | 5.30880 секунд | 6.30878 секунд |

По полученным результатам видно, что потоковая передача данных от клиента быстрее двух других методов. Это связано с тем, что сервер отвечает лишь одним сообщением после получения всех данных от клиента. Двунаправленная передача быстрее унарной, поскольку клиент и сервер обмениваются данными и ответами без ожидания завершения друг друга.

## 3.3 Сокеты

В разработанной системе серверный компонент инициирует сокет, настроенный на работу с протоколом TCP. Это достигается путем конфигурации сокета для прослушивания определенного хоста и порта, что позволяет серверу ожидать и обрабатывать входящие подключения от клиентов.

В процессе передачи файлов между клиентом и сервером используется согласованный протокол обмена данными. Сначала клиент упаковывает размер передаваемого файла-изображения в первые 4 байта, за которыми следует содержимое файла. Это позволяет серверу точно определить объем данных для чтения. При получении файла сервер извлекает указанный размер и в цикле принимает данные, пока не будет получен полный объем файла. Специализированные функции на обеих сторонах гарантируют целостность и полноту передачи, обеспечивая, что все байты будут переданы и приняты, независимо от количества необходимых итераций.

Использование динамического размера буфера, равного размеру файла, позволяет избежать необходимости настройки параметров буфера и упрощает процесс передачи. В результате экспериментов было установлено, что среднее время передачи данных составляет 3.8661 секунды.

В альтернативном подходе с использованием статического размера буфера было выявлено, что оптимальное время передачи данных достигается при передаче 9 МБ (9000000 байт) за один раз, что составляет 3.1 секунды для файла размером 6.47 ГБ. Однако, увеличение размера буфера приводит к повышенному потреблению памяти, что требует тщательного подбора этого параметра для каждого конкретного случая.

## 3.4 Вывод по главе 3

3.3.1 В данной главе были проведены исследования по различным технологиям реализации клиента и сервера для передачи больших данных. Определены ключевые моменты, влияющие на скорость передачи данных, а также библиотеки и методы непосредственно позволяющий проводить исследование. Все результаты получены экспериментальным путем с соблюдением требований поставленных во вступлении этой главы. Далее необходимо сравнить полученные по каждому способу результаты сделать непосредственные выводы, о чем и пойдет речь в следующей главе.

# 4. СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РАЗЛИЧНЫХ МЕТОДОВ ПЕРЕДАЧИ ДАННЫХ

В этой главе представлено сравнение производительности различных методов передачи данных, рассмотренных в предыдущих разделах. Основываясь на теоретических основах, изложенных в главах 1 и 2, и практической реализации, описанной в главе 3, был проведен анализ, для определения наиболее эффективные подходов.

Методология сравнения будет включать в себя анализ времени передачи данных. Для наглядности результатов использованы диаграммы.

## 4.1 Сравнение НТТР 1.1 и НТТР/2

Асинхронная передача данных по HTTP 1.1 затратила на 40% меньше времени чем синхронная передача по тому же самому протоколу, которая составляла 8.029 секунд. Процентное уменьшение было получено по формуле:

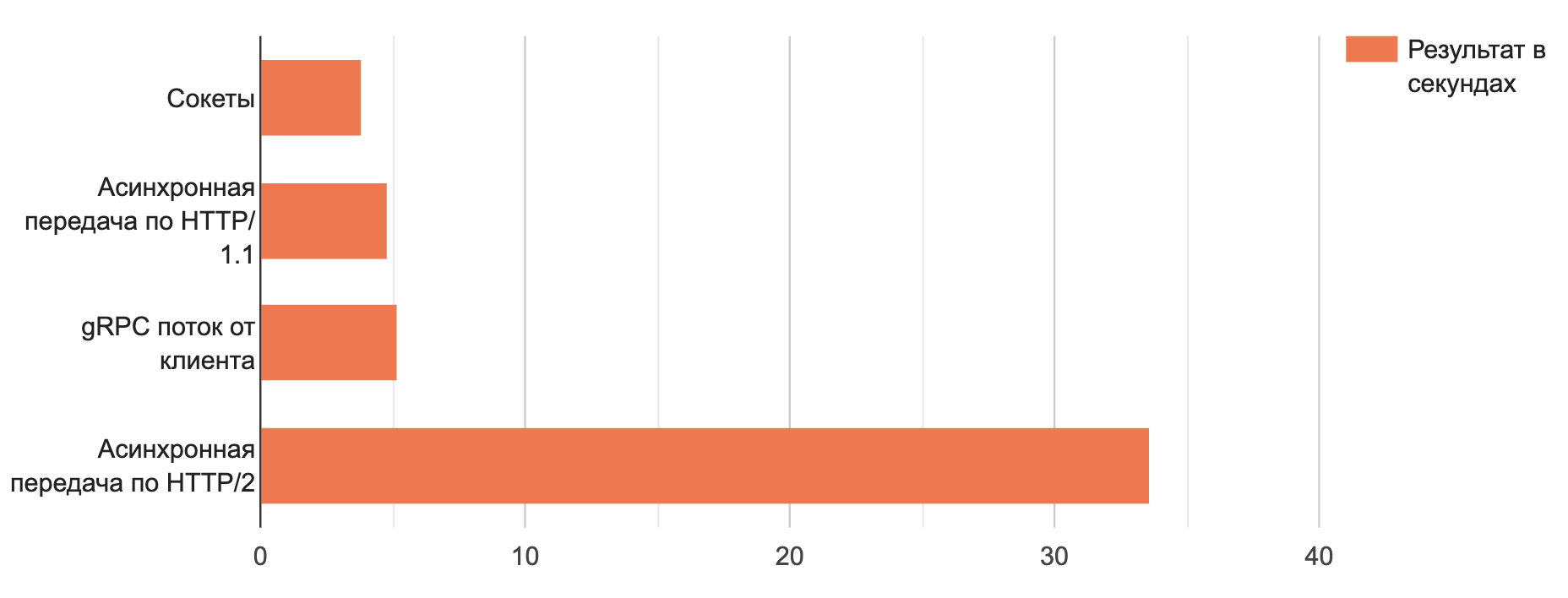
Данный метод также занял на 20.55% меньше времени чем многопоточность, которая оказалась быстрее синхронной передачи на 24.5%. Самым долгим оказался метод с применением технологии UploadFile занявший на 291.7% времени больше чем синхронная передача, а это почти в 3 раза дольше. Анализ показывает, что асинхронные методы и многопоточность обеспечивают лучшую производительность по сравнению с синхронными методами.

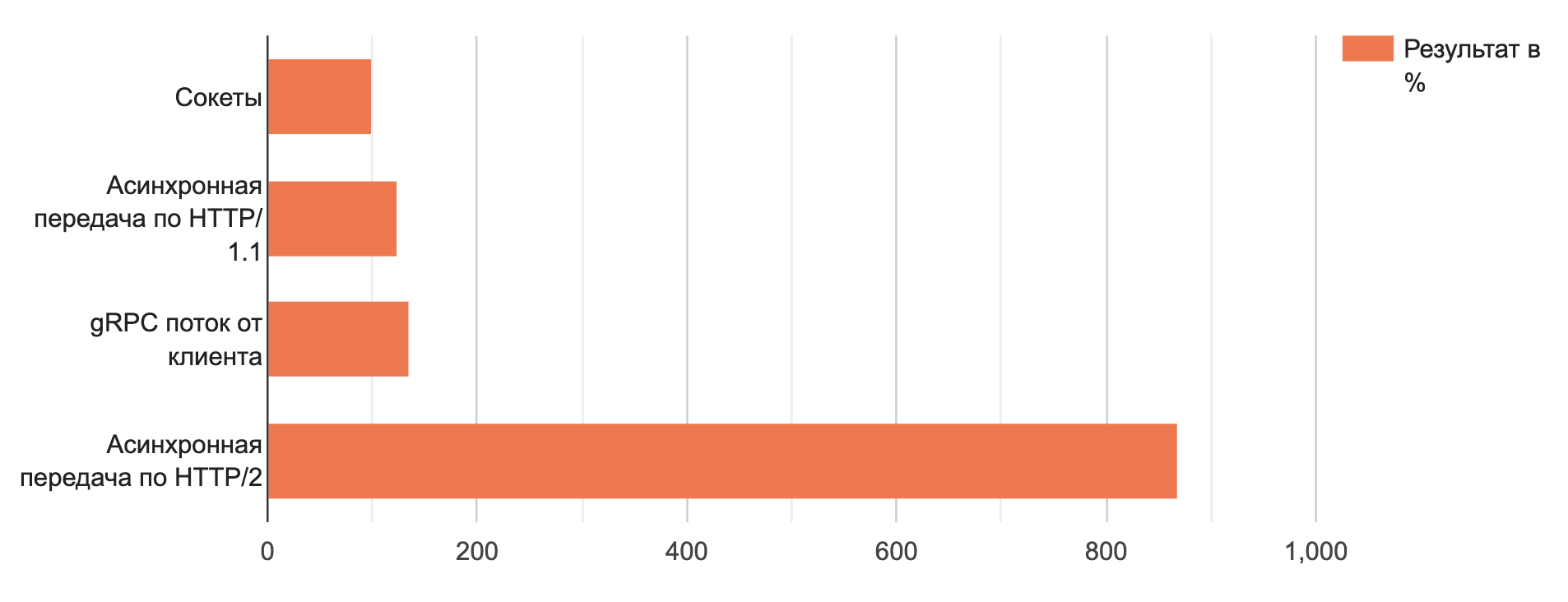
Синхронная передача по протоколу HTTP/2 составляла 40.0234 секунд, что на 16.1% дольше чем асинхронная передача данных HTTP/2 и на 4.1% дольше многопоточности. Процентное соотношение между различными методами передачи данных сохраняется для обоих протоколов.

Сравнение способов передачи данных с использованием gRPC показало, что унарная передача занявшая 7.725 секунд на 32.5% медленнее чем потоковая передача от клиента и на 18.3% медленнее чем двунаправленная передача. Потоковая передача от клиента оказалась самым быстрым методом, сокращая время передачи на 32.5% по сравнению с унарной передачей. По сравнению с асинхронной передачей по HTTP/2, преимущество составило 84.49%. Это подтверждает, что gRPC ускоряет работу HTTP/2 за счет отсутствия необходимости в передаче и проверки сертификатов безопасности с каждым переданным файлом, о чем было сказано в разделе 3.2.2. По этой же самой причине передача данных по HTTP/1.1 показала лучшие результаты по сравнению с HTTP/2

## 4.3 Общие результаты сравнения

Самым быстрым вариантом оказалась передача с использованием сокетов, с результатом в 3.866 секунд. Если взять его за 100%, то можно сравнить остальные методы относительно этого значения. Результаты сравнения отображены на диаграмме 1.





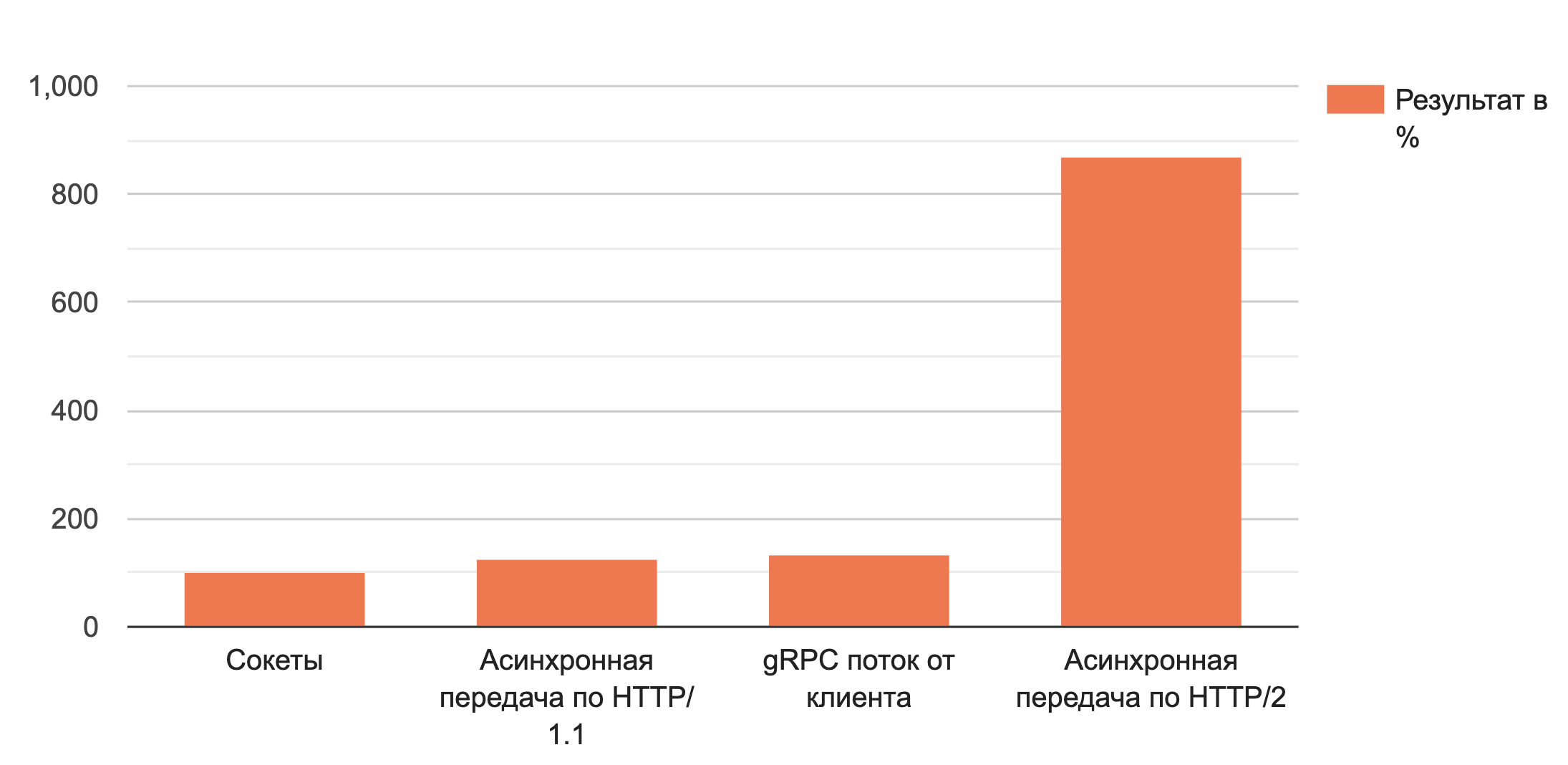


Диаграмма 1 – Сравнение лучших полученных результатов

Исходя из расчетов, видно, что асинхронная передача данных по HTTP/1.1 занимает на 24.6% больше времени, чем передача данных с использованием сокетов. gRPC поток от клиента занимает – на 34.8% больше, а асинхронная передача по HTTP/2 значительно медленнее, занимая на 768.9% больше времени, то есть почти в 8 раз.

## 4.4 Выводы по главе 4

Из проведенного сравнения следует, что асинхронные методы передачи данных и многопоточность значительно превосходят синхронные методы по скорости передачи данных. Сокеты демонстрируют наилучшую производительность, особенно в сравнении с традиционными HTTP-методами. gRPC выделяется своей эффективностью, особенно при потоковой передаче данных от клиента, что делает его предпочтительным выбором для сценариев, требующих высокой производительности. HTTP/2, несмотря на свои теоретические преимущества, не показал ожидаемого улучшения производительности в данном исследовании, что может быть связано с дополнительной нагрузкой на обработку сертификатов безопасности.