СОДЕРЖАНИЕ

[1. Теоретическая часть 5](#_Toc154343199)

[1.1 Способы передачи информации между сервисами 5](#_Toc154343200)

[1.2 gRPC - Google Remote Procedure Call 6](#_Toc154343201)

[1.2.1 Использовать RPC и Protocol Buffers. 7](#_Toc154343202)

[1.2.2 Разновидности API в gRPC 8](#_Toc154343203)

[1.3 Многопоточность (treading.thread) 11](#_Toc154343204)

[1.4 Распределенное обучение (+ модель) (DDP, ) 11](#_Toc154343205)

[1.5 Рассмотренные библиотеки 12](#_Toc154343206)

[1.5.1 Библиотека gRPC 13](#_Toc154343207)

[1.5.2 treading 13](#_Toc154343208)

[1.5.3 DistributedDataParallel 14](#_Toc154343209)

[1.6 Docker 15](#_Toc154343210)

[1.6.1 Dockerfile 15](#_Toc154343211)

[1.6.2 docker-compose 15](#_Toc154343212)

[2 Практическая часть 16](#_Toc154343213)

[2.1 Реализация программ 16](#_Toc154343214)

[2.1.1 fastapi (bytes / Uploadfiles) 16](#_Toc154343215)

[2.1.2 grpc-all 16](#_Toc154343216)

[2.1.3 grpc-stream 16](#_Toc154343217)

[2.2 Описание остального (requirements, docker, как запускать) 16](#_Toc154343218)

[2.3 Результаты работы 16](#_Toc154343219)

СПИСОК ИСПОЛЬЗОВАННЫХ ОПРЕДЕЛЕНИЙ

HTTP (Hypertext Transfer Protocol) - это протокол передачи данных, который используется для обмена информацией между клиентом и сервером в сети Интернет. Он является основным протоколом для передачи данных в вебе и широко применяется для обеспечения связи между браузерами и веб-серверами.

HTTP/2 (Hypertext Transfer Protocol version 2) - это вторая версия протокола HTTP, который был представлен в 2015 году и представляет собой значительное обновление по сравнению с предыдущей версией, HTTP/1.1.

RPC – RPC (Remote Procedure Call) - это протокол удаленного вызова процедур.

gRPC - современная технология удаленного вызова процедур, разработанная компанией Google.

Threading - это модуль в языке программирования Python, который предоставляет инструменты для создания и управления потоками выполнения (threads).

DistributedDataParallel – (DDP) - это один из компонентов библиотеки PyTorch, предоставляющий механизм для распределенного обучения глубоких нейронных сетей на множестве устройств (например, графических процессорах) или на нескольких узлах в сети.

Docker - открытая платформа для разработки, доставки и выполнения приложений в контейнерах. Контейнеры обеспечивают стандартизированный способ упаковки приложений и всех их зависимостей, включая библиотеки, конфигурацию и код, в одном компактном пакете.

DockerFile – текстовый файл, содержащий инструкции по созданию Docker-образа. Он описывает шаги, которые Docker должен выполнить, чтобы собрать образ.

docker – compose - инструмент для определения и запуска многоконтейнерных Docker-приложений. С помощью файла docker-compose.yml можно определить конфигурацию нескольких сервисов, и Docker Compose автоматически управит их развертыванием и взаимодействием.

Encoding - (кодирование) - это процесс преобразования данных из одной формы представления в другую. В контексте информационных технологий и передачи данных, кодирование часто относится к преобразованию данных из одного формата в другой с целью эффективного представления, передачи или хранения.

Serialization - (сериализация) - это процесс преобразования структурированных данных или объектов программы в формат, который может быть легко сохранен, передан по сети или восстановлен в их исходное состояние.

Parsing - это процесс анализа строки символов или последовательности данных в соответствии с заданными правилами, с целью извлечения определенной информации или преобразования данных в структурированный формат

Decoding - (декодирование) - это процесс обратного преобразования закодированных или зашифрованных данных в их исходное состояние.

Protocol Buffers (protobuf) - это формат сериализации данных, разработанный компанией Google.

Сверточная нейронная сеть (Convolutional Neural Network, CNN) -

один из подходов нейронных сетей.

GPU - графический процессор.

FastAPI - один из вариантов реализации серверной часть приложения.

API - маршруты, для обработки запросов и отправку ответов клиентам.

Маршрутизация (Routing) - пределение АРІ-маршрутов, которые клиенты могут запрашиваь.

Request Handlers - методы обработки запросов.

JSON, HTML - форматы ответов от сервера.

Эндпоинт - маршрут в котором прописан путь до страницы, метод

обращения, а также действие.

ВВЕДЕНИЕ

Скорость передачи информации имеет широкий спектр применений, и ее ускорение необходимо и применимо практически во всех областях компьютерных наук. gRPC, высокопроизводительный фреймворк для удалённого вызова процедур (RPC), становится ключевым инструментом в обеспечении эффективности передачи данных. Методы передачи информации на его основе имеют высокую практическую значимость, превосходя традиционные методы. Возможности gRPC, вместе с алгоритмами на основе передачи данных, могут привести к значительному улучшению качества и точности результатов в различных областях.

Использование алгоритмов на основе gRPC открывает перспективы для разработки более эффективных систем, способных работать в реальном времени и справляться с различными вызовами современного информационного обмена. Все эти факторы подчеркивают актуальность и значимость исследования в области ускоренной передачи данных, открывая новые возможности для развития и внедрения в различные сферы.

Цель данной работы - исследование, разработка и применение методов ускоренной передачи данных. Для достижения этой цели ставятся следующие задачи:

1. Провести анализ существующих методов передачи данных и выявить их основные характеристики.
2. Выбрать и обосновать подходы, основанные на gRPC и FastAPI, в качестве основных методов ускоренной передачи информации.
3. Изучить библиотеки, необходимые для реализации и оптимизации методов передачи данных.
4. Реализовать и оптимизировать разработанные методы.
5. Провести эксперименты для оценки эффективности и производительности разработанных методов.
6. Контейнеризовать разработанные методы с использованием Docker для обеспечения легкости развертывания и масштабирования.

# Теоретическая часть

## Способы передачи информации между сервисами

Микросервисная архитектура - способ организации сложных программных систем: приложение разбивается на сервисы, которые развертываются независимо, но взаимодействуют друг с другом. Эта архитектура хороша тем, что код, соответствующий модульным функциям продукта, тоже может быть модульным. При изменении части кода достаточно задействовать один модуль а не весь код. Разделение кода на микросервисы дает большую гибкость. Можно писать микросервисы на разных языках, каждый микросервис можно независимо масштабировать. Минусы микросервисов в том, что их труднее тестировать, чем монолитный код, также не стоит создавать их слишком маленькими.

Существует большое количество различных способов передачи информации между сервисами. Все они основаны на различных подходах к хранению и обмену данными. Для лучшего понимания gRPC стоит более детально рассмотреть другие протоколы, среди которых особое внимание заслуживают HTTP и его последняя версия — HTTP/2.

HTTP/1.1 поддерживает множество запросов через одно соединение, дает возможность отправлять несколько запросов без ожидания ответов, также позволяет виртуальным хостам существовать на одном сервере, но он все еще подвержен проблемам блокировок из-за одного соединения, особенно при множестве мелких файлов.

HTTP/2 был представлен в 2015 году для улучшения производительности. Множество запросов может выполняться параллельно в рамках одного соединения, разделенного на потоки. Его плюсами также является уменьшение объема передаваемых данных, возможность устанавливать приоритеты запросов, возможность сервера предварительно отправлять ресурсы, которые могут быть запрошены клиентом.

В целом HTTP/2 предоставляет значительное улучшение производительности за счет множественных потоков, сжатия заголовков и других оптимизаций. Он особенно эффективен при загрузке больших веб-страниц и ресурсов, снижая задержки и ускоряя время загрузки.

Программы обычно сталкиваются с данными в различных форматах: данные, хранящиеся в памяти, и данные, предназначенные для записи в файлы и передачи по сети. Обычные структуры данных в памяти оптимизированы для эффективного взаимодействия с центральным процессором. Однако, для передачи по сети или записи в файл, эти данные необходимо преобразовать в последовательность байтов. Этот процесс называется кодированием или сериализацией. Разнообразные форматы, такие как JSON, XML и CSV, используются для представления данных в виде последовательности байтов.

Существует несколько способов организации взаимодействия процессов по сети, включая REST и RPC.

REST - подход к проектированию, основанный на принципах НТТР. Он делает акцент на простых форматах данных, применении URL для идентификации ресурсов и использовании возможностей НТТР для управления кэшем, аутентификации и согласования типа контента. АРІ, спроектированный в соответствии с принципами REST, называют воплощающим REST (RESTful).

RPC (Remote Procedure Call) представляет собой набор технологий, который позволяет программам вызывать функции или процедуры в другом контексте или адресном пространстве – будь то на удаленных узлах или в независимой сторонней системе на том же узле. SOAP (Simple Object Access Protocol) является одной из реализаций RPC, использующей формат XML для выполнения запросов к сетевым API. Применяясь в основном по протоколу HTTP, SOAP стремится к независимости от него, избегая использования большинства его функциональных возможностей. Вместо этого SOAP сопровождается множеством дополнительных стандартов (фреймворк веб-сервисов, известный как WS-\*), обогащающих его разнообразными функциональными возможностями. API веб-сервиса SOAP описывается с использованием языка описания веб-сервисов (Web Services Description Language, WSDL), базирующегося на XML. Изначально SOAP был создан прежде всего для реализации удаленного вызова процедур (RPC).

В свою очередь, REST, основанный на принципах HTTP, становится более популярным по сравнению с SOAP, особенно в контексте интеграции сервисов между организациями, и часто ассоциируется с архитектурой микросервисов.

## gRPC - Google Remote Procedure Call

gRPC - это новый и современный фреймворк для разработки масштабируемых, современных и быстрых API и дословно переводится как система удаленного вызова процедур, разработанный компанией Google еще в 2015 году. Используется многими ведущими компаниями, такими как Google, Square n Netflix, и позволяет программистам писать микросервисы на любом языке, который они хотят, сохраняя при этом возможность легко устанавливать связь между этими сервисами. И так же является реализацией удалённого вызова процедур (RPC), но уже в качестве транспорта использует новый НТТР/2. А для описания интерфейса используется Protocol Buffers (protobuf), где описывается структура для передачи, кодирования и обмена данных между представлениями описанными выше. Protocol Buffers проще, компактнее и быстрее, чем XML, поскольку осуществляется передача бинарных данных.

### Использование RPC и Protocol Buffers.

Первое преимущество использования Protocol Buffers заключается в том, что они дают API четко определенную и самодокументированную схему. Используя JSON, необходимо задокументировать содержащиеся в нем поля и их типы. Как и в случае с любой другой документацией есть риск, что она окажется неточной, неполной или устаревшей. Из API на Protobuf, можно сгенерировать код Python. То есть код никогда не будет рассинхронизирован с документацией. Самодокументируемый код – хорошее преимущество.

Второе преимущество заключается в том, что при генерации кода Python из кода Protobuf, заодно получается некоторая базовую проверку. Например, сгенерированный код не принимает поля неправильного типа. Если для API используется HTTP и JSON, то приходится писать код, который создает запрос, отправляет его, ожидает ответа, проверяет код состояния, а также анализирует и проверяет ответ. С помощью Protobuf можно генерировать код, который выглядит так же, как обычный вызов функции, но выполняет под капотом сетевой запрос.

Третье преимущество заключается в том, что платформа gRPC более эффективна, чем использование обычных НТТР-запросов. gRPC построен на основе HTTP/2, позволяющем выполнять несколько запросов параллельно поточно- безопасным способом. Сообщения RPC хранятся в бинарном виде и весят меньше, чем JSON. Кроме того, НТТР/2 имеет встроенное сжатие заголовков, а gRPC - встроенную поддержку потоковой передачи запросов и ответов.

Вероятно, самая важная причина, заключается в том, что можно определять свой АРІ в терминах функций, а не НТТР-команд и ресурсов. B REST API необходимо решать, какие у есть ресурсы, как строить пути, как называть команды. Часто эти задачи могут быть решены множеством разных способов. Protobuf похож на JSON или XML в том, что все они являются способами форматирования данных. В отличие от JSON, Protocol Buffers имеют строгую схему и более компактны при передаче по сети.

В Protobuf для каждого типа данных, который требуется преобразовать в бинарный формат, создается соответствующее сообщение, где определяется структура данных и указывается имя и тип для каждого поля в этом сообщении. Затем, для описания логики взаимодействия между клиентом и сервером, определяется сервис и указываем методы, которые этот сервис предоставляет. Завершая этап подготовки протокола, компилируется описанные в протоколе классы. Просто введя команду с необходимой опцией для выбранного языка, компилятор автоматически генерирует код, освобождая разработчика от рутины ручного написания множества классов и методов.

### Разновидности API в gRPC

gRPC предоставляет четыре основных способа взаимодействия между клиентом и сервером, каждый из которых предназначен для различных сценариев использования.

Unary RPC (Унитарное взаимодействие) - В унитарном RPC клиент отправляет один запрос и получает один ответ от сервера. Подходит для простых запросов и ответов, когда клиент ожидает результат в ответ на свой запрос.

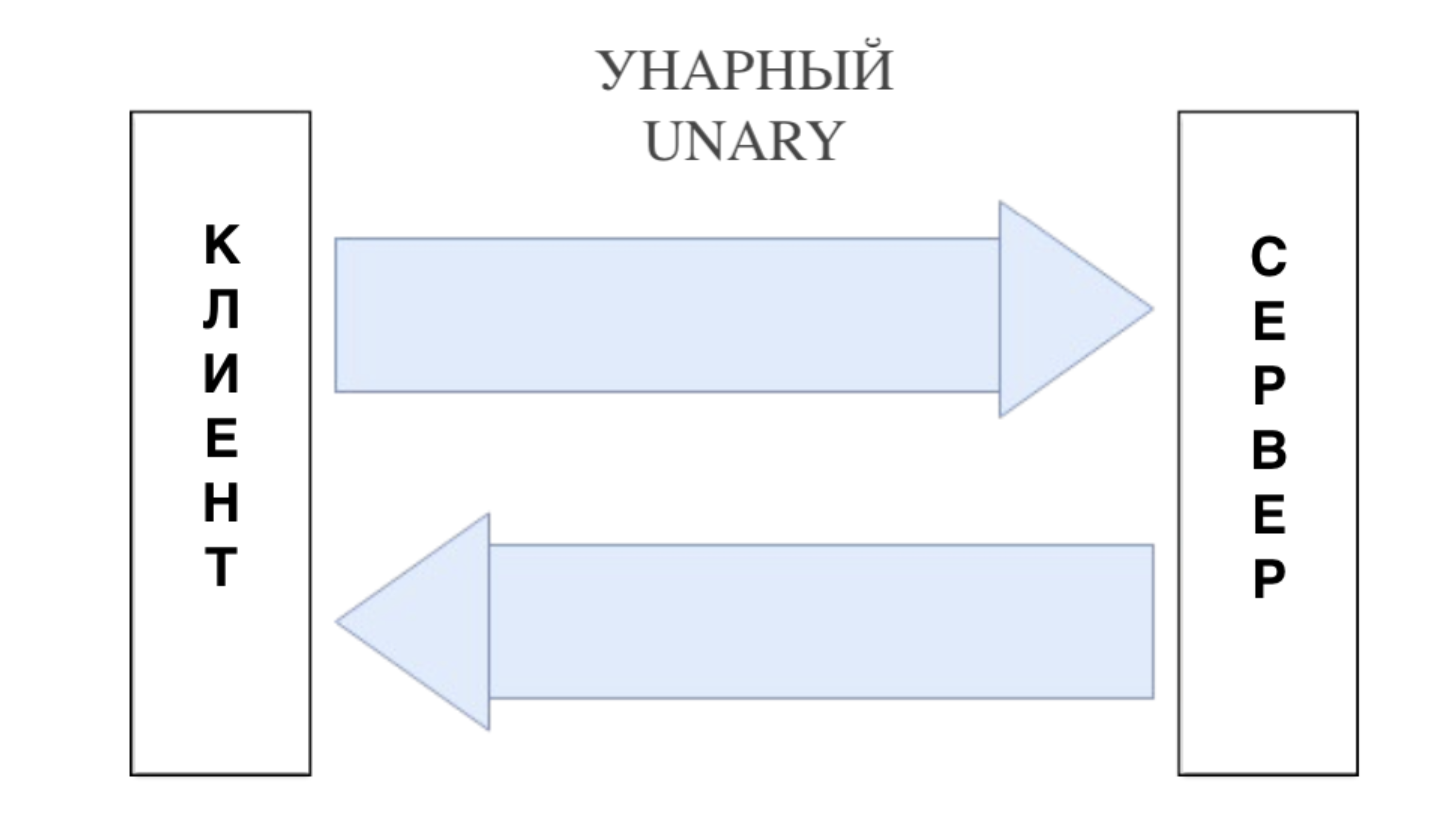


Рисунок. 1 – иллюстрация унарного способа передечи

Server Streaming RPC (Потоковое взаимодействие от сервера) - Клиент отправляет один запрос, а сервер отвечает потоком сообщений. Используется, когда клиенту нужно получить последовательность результатов от сервера, например, при потоковой передаче данных или мониторинге.

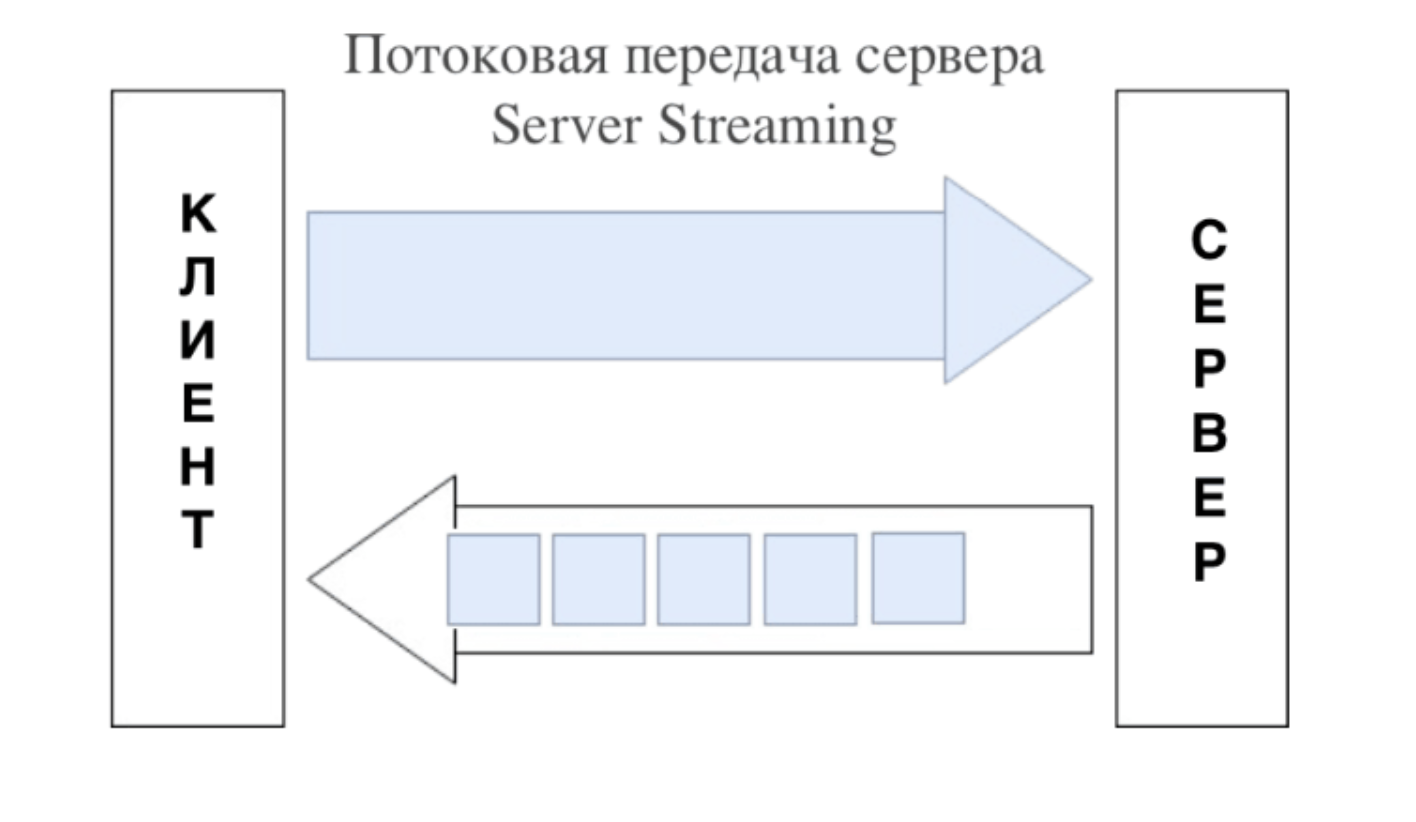


Рисунок. 2 – иллюстрация потоковой передачи от сервера

Client Streaming RPC (Потоковое взаимодействие от клиента) - Клиент отправляет поток сообщений, а сервер отвечает одним ответом. Применяется, когда клиент отправляет большой объем данных на сервер, но ожидает только один ответ, как в случае передачи файла.

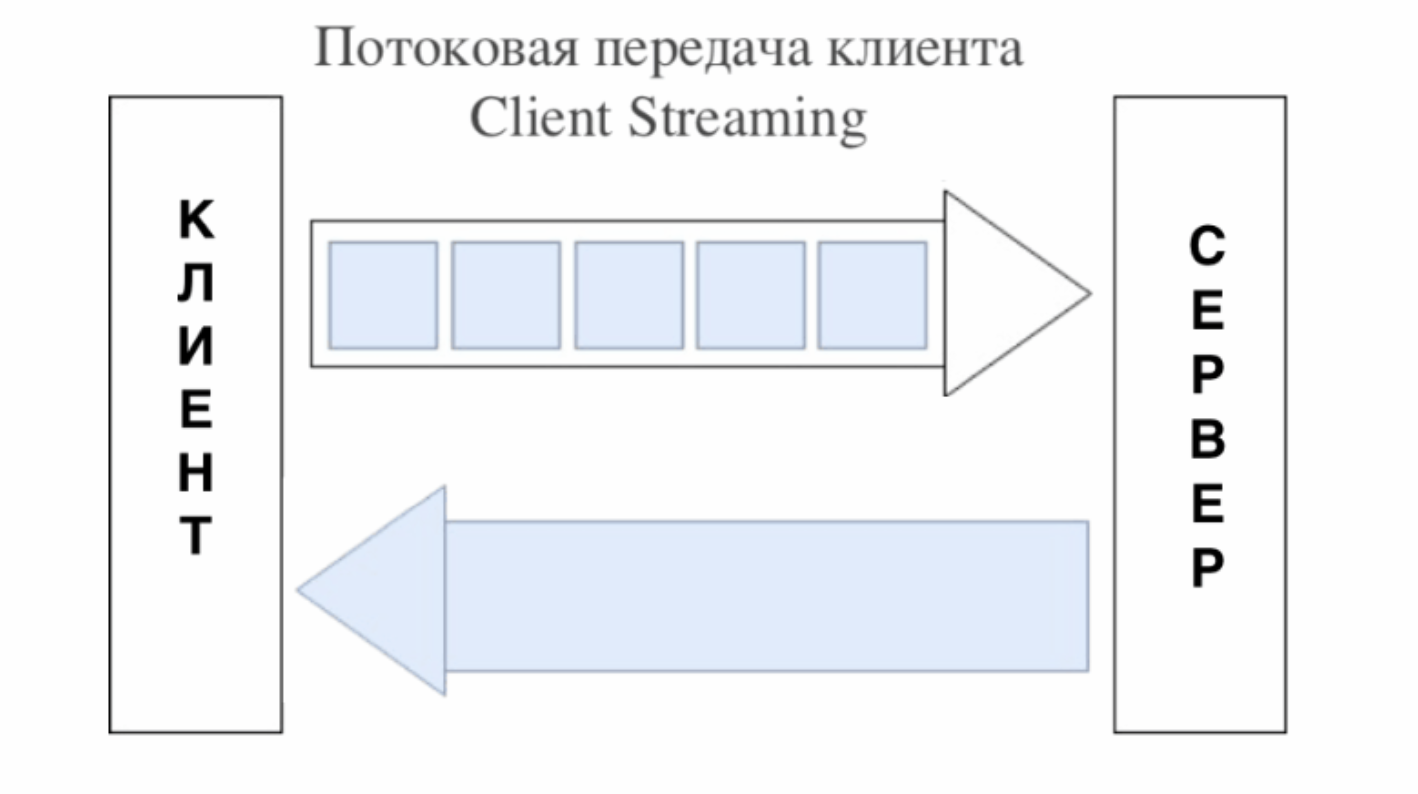


Рисунок. 3 – иллюстрация потоковой передачи от клиента

Двунапра Bidirectional Streaming RPC (Двустороннее потоковое взаимодействие) - Клиент и сервер обмениваются потоками сообщений независимо друг от друга. Эффективно использовать, когда клиент и сервер могут отправлять данные асинхронно, что полезно для чат-приложений, потокового обмена данными и реактивных систем.

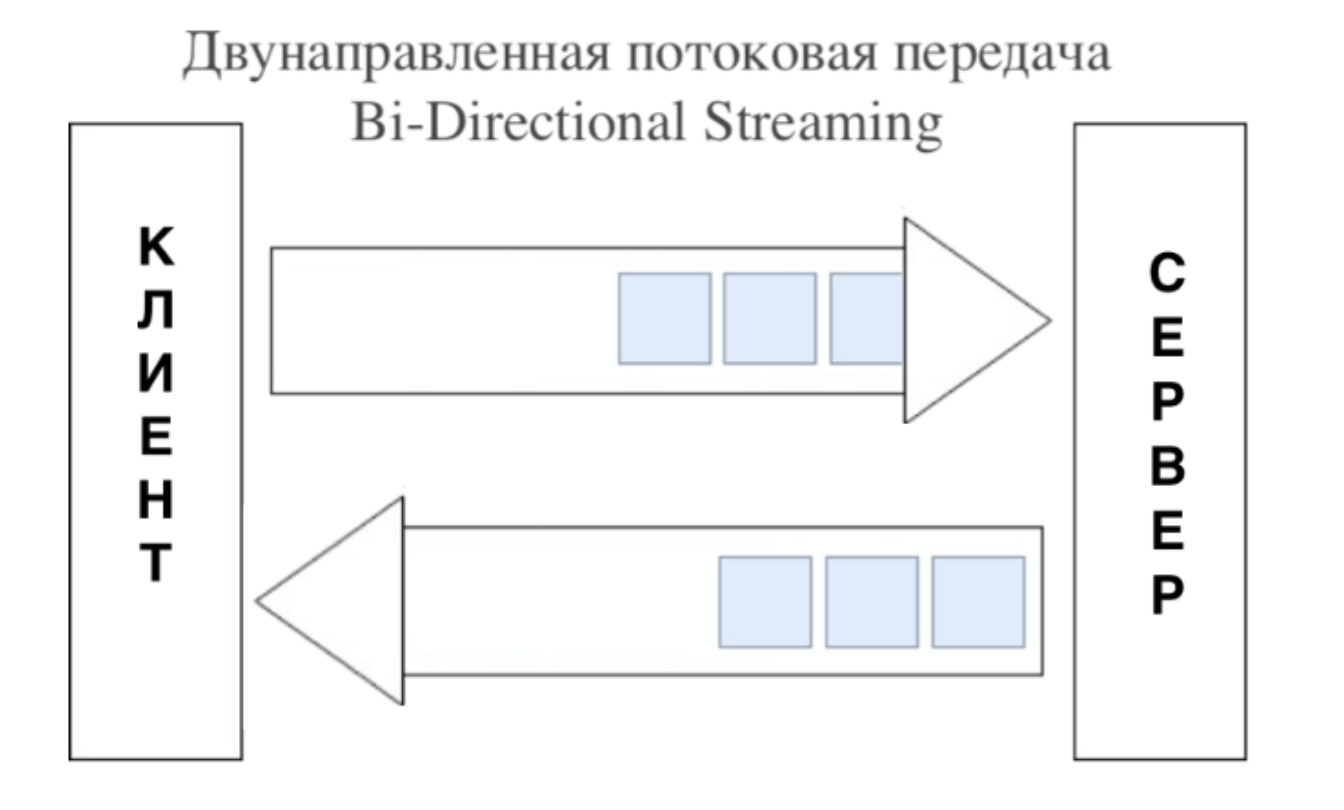


Рисунок. 4 – иллюстрация двунаправленной потоковой передачи

Каждый из этих подходов предоставляет гибкость в выборе метода взаимодействия в зависимости от требований конкретной задачи. Кроме того, эти способы взаимодействия поддерживаются множеством языков программирования, что делает gRPC мощным инструментом для разработки распределенных систем с разнообразными потребностями.

## Многопоточность (treading.thread)

Многопоточность - это концепция, которая позволяет программе выполнять несколько потоков (или задач) одновременно. Потоки представляют собой подзадачи, которые работают параллельно и могут улучшить эффективность выполнения программы. В Python для работы с многопоточностью используется встроенный модуль threading. Поток представляет собой легковесный процесс, который работает параллельно с другими потоками в рамках одного процесса. В Python потоки создаются с использованием класса Thread из модуля threading. Thread - это основной класс в модуле threading, который представляет собой поток выполнения. Потоки создаются, и каждый из них выполняет код параллельно с другими потоками. Создание потока в Python осуществляется путем создания экземпляра класса Thread и передачи целевой функции (функции, которую выполнит поток) в качестве аргумента.

## Распределенное обучение (+ модель) (DDP, )

Распределенное обучение (Distributed Training) - это подход в глубоком обучении, который позволяет обучать модели на нескольких устройствах или узлах сети. Распределенное обучение может включать в себя распределение данных, модели или вычислений. В PyTorch для реализации распределенного обучения используется модуль torch.distributed, а одним из его компонентов является DistributedDataParallel (DDP).

Распределенное обучение может происходить на нескольких узлах (несколько машин) и/или нескольких устройствах (несколько GPU) на одном узле. Топология определяет, как узлы и устройства связаны между собой.

DistributedDataParallel (DDP) - это один из инструментов в библиотеке PyTorch, предназначенный для распределенного обучения. Он позволяет обучать модель на нескольких устройствах (GPU) и синхронизировать параметры модели между устройствами.

В начале обучения процессы инициализируются и формируют группу процессов. Это гарантирует, что процессы могут обмениваться данными и синхронизировать обучение.

Модель реплицируется на все устройства (GPU), участвующие в обучении. Каждая реплика обрабатывает свой уникальный поднабор данных, что увеличивает эффективность обучения.

DDP обеспечивает автоматическую синхронизацию параметров модели между всеми репликами. Это происходит после каждого шага обновления весов, чтобы гарантировать, что все реплики имеют одинаковые веса.

Градиенты вычисляются для каждой реплики, и DDP автоматически обрабатывает их суммирование и распределение для выполнения группового градиентного спуска.

DDP поддерживает распределенные оптимизаторы, такие как torch.optim.SGD и torch.optim.Adam, что позволяет эффективно выполнять оптимизацию градиента на нескольких устройствах.

DistributedDataParallel (DDP) в PyTorch облегчает реализацию распределенного обучения, обеспечивая синхронизацию параметров и оптимизацию градиента на нескольких устройствах. Это полезный инструмент для обучения моделей на многих GPU или в распределенной среде, что может существенно ускорить процесс обучения.

## Рассмотренные библиотеки

Библиотеки, используемые в языке Python, предоставляют множество инструментов и функций. На прошлой практике были рассмотрены используемые в данной работе библиотеки для разработки алгоритмов машинного обучения, обработки изображений и выполнения различных вычислений. В этой практической работе добавлены библиотеки для работы с gRPC, а также потоковому распределению и параллельному обучению. Ниже приведено описание нескольких популярных библиотек.

### Библиотека gRPC

Принципы работы: Определение Сервисов с использованием Proto. Сервисы и сообщения описываются в файлах Proto с использованием простого и ясного синтаксиса. С помощью компилятора Proto генерируются стабы (stubs) для клиентской и серверной сторон. Stubs предоставляют интерфейс для вызова удалённых процедур. Клиент и сервер создают канал для обмена данными. Канал представляет собой соединение между клиентом и сервером. Сервер реализует сервис, а затем привязывает его к каналу. Это позволяет серверу прослушивать входящие запросы. Клиент использует стаб для вызова удалённых процедур на сервере через канал. Параметры и результаты сериализуются в бинарном формате. gRPC поддерживает асинхронные вызовы, что позволяет эффективно использовать ресурсы и обеспечивает отзывчивость системы.

### treading

threading - это модуль в языке программирования Python, который предоставляет инструменты для создания и управления потоками выполнения (threads). Потоки представляют собой легковесные, независимые потоки выполнения внутри одного процесса. Они обеспечивают многозадачность, позволяя программам выполнять несколько задач одновременно.

Поток создается путем наследования от класса Thread и переопределения метода run(), в котором определяется код, который будет выполняться в потоке. Потоки предоставляют способ реализации многозадачности в Python. Можно создавать несколько потоков, каждый из которых выполняет свою задачу независимо. Потоки могут конфликтовать за общие ресурсы, и для избежания проблем синхронизации доступа к этим ресурсам используются механизмы синхронизации, такие как блокировки (Lock), условия (Condition), семафоры и т.д. Потоки имеют методы для управления их жизненным циклом, такие как start() для запуска потока и join() для ожидания завершения потока.

Модуль threading предоставляет удобный способ работать с потоками в Python, но также следует помнить о некоторых ограничениях и особенностях, таких как Global Interpreter Lock (GIL), который может ограничивать эффективность использования потоков в многозадачных сценариях.

### DistributedDataParallel

DistributedDataParallel (DDP) - это один из компонентов библиотеки PyTorch, предоставляющий механизм для распределенного обучения глубоких нейронных сетей на множестве устройств (например, графических процессорах) или на нескольких узлах в сети.

Библиотека PyTorch поддерживает распределенное обучение, которое может включать в себя распределение данных, модели и вычислений между различными узлами и устройствами. DistributedDataParallel - это часть этой поддержки и предоставляет возможность обучения нейронных сетей на нескольких GPU с использованием распределенных вычислений.

DistributedDataParallel используется для обучения модели на нескольких узлах и/или GPU. Каждый процесс, выполняющий обучение, работает с подмножеством данных и обменивается градиентами с другими процессами для обновления весов модели. В начале обучения несколько процессов инициализируются и запускаются на доступных устройствах (GPU). Каждый процесс будет обрабатывать свое уникальное подмножество данных.

DistributedDataParallel автоматически обрабатывает агрегацию градиентов между различными процессами. Это позволяет каждому процессу обновлять веса модели на основе градиентов, вычисленных на его подмножестве данных. Модель в каждом процессе синхронизируется с другими процессами после каждого шага обновления. Это обеспечивает согласованность весов модели на всех процессах.

PyTorch обеспечивает инфраструктуру для обмена данными между процессами и узлами, а DistributedDataParallel эффективно использует эту инфраструктуру для синхронизации данных.

## Docker

Docker использует технологию контейнеризации, где приложение и все его зависимости (библиотеки, среда выполнения, настройки) упаковываются в контейнер. Это обеспечивает изолированное и независимое окружение выполнения для приложения.

Основным строительным блоком в Docker является образ. Образ представляет собой статичный пакет, содержащий все необходимое для запуска приложения. Образ создается из файла Dockerfile, который содержит инструкции для построения образа.

### Dockerfile

Dockerfile — это текстовый файл, в котором описываются шаги по созданию образа. Образ представляет собой упакованный и готовый к запуску контейнер, который включает в себя все необходимые зависимости и приложение. В Dockerfile указываются базовый образ, необходимые зависимости, команды для установки приложения и прочее.

Основные компоненты структуры Dockerfile. Указывается базовый образ, на основе которого строится ваш образ. Обычно это минимальный образ с необходимыми инструментами и зависимостями. Устанавливается текущая рабочая директория внутри контейнера, где будет выполняться команды. Копирует файлы из локальной системы внутрь контейнера. После этого запускаются команды для установки зависимостей, сборки приложения и выполнения других действий внутри контейнера, также определяются переменные окружения, которые могут использоваться в процессе сборки образа и выполнения контейнера, и объявляет порт, который будет открыт в контейнере и доступен извне. В завершении определяется команда, которая будет выполнена при запуске контейнера. Это может быть исполняемый файл, команда командной строки и так далее.

### docker-compose

docker-compose - это инструмент для определения и запуска многоконтейнерных Docker-приложений. С его помощью можно описать конфигурацию вашего приложения, включая контейнеры, сети, объемы и другие аспекты, в файле docker-compose.yml, а затем легко запустить все контейнеры одной командой.

# Практическая часть

В данной практической работе были рассмотрены несколько методов передачи данных между двумя серверами. Для удобства сервер отправляющий данные будет назван клиентам, а сервер получающий их – сервером.

Задача передачи стереопар изображений с клиента на сервер и сравнение результатов была реализована несколькими способами рассмотренными далее.

## Реализация программ

### fastapi (bytes / Uploadfiles)

В качестве опыта для получения информации о скорости передачи стереопары изображений клиент будет брать из заранее подготовленной папки и передавать их на сервер. В качестве средства передачи выступает FastApi , а данные будут передаваться в виде UploadFiles.

В FastAPI существует специальный компонент для обработки загрузки файлов, который называется UploadFile. Этот компонент позволяет легко получать файлы, отправленные в API. Компонент UploadFile автоматически обрабатывает загрузку файла и предоставляет информацию о файле, такую как его имя (filename).

Вторым способом будем передавать данные в виде bytes.

Bytes используется для передачи бинарных данных напрямую в теле запроса. Например, это может быть полезно, когда нужно передать изображение или другие бинарные данные в теле запроса.

Основное отличие между bytes и UploadFile заключается в том, что bytes используется для прямой передачи бинарных данных в теле запроса, тогда как UploadFile предназначен для удобства обработки файлов, загруженных через механизм форм. Далее реализованы 4 способа передачи данных с помощью gRPC.

### grpc-all

Была написана программа, которая реализует передачу данных всеми способами доступными в gRPC, ее задача на одних и тех же данных проверить скорость передачи изображений.

### grpc-stream

Третья программа сочетает в себе унарную передачу из прошлой программы, и разделение на потоки. В ней на клиенте создаются два потока, первый – получает стереопары и передает их на клиента, а второй поток это клиент передающий изображения на сервер по мере их обработки сервером.

## Результаты работы

ЗАКЛЮЧЕНИЕ

В данной исследовательской работе были успешно рассмотрены и выполнены поставленные цели и задачи. Актуальность темы была подтверждена в ходе обзора различных методов и подходов, применяемых для передачи данных между сервисами.

В ходе анализа преимуществ и недостатков каждого метода были выявлены ключевые особенности, позволяющие извлекать пользу из каждого.

Изучение популярных библиотек, таких как grpc, threading, и DistributedDataParallel, позволило освоить инструменты, способствующие более эффективной разработке и управлению ресурсами. Рассмотренные принципы работы этих библиотек предоставили полезный фундамент для решения задачи, а также позволили изучить особенности параллельного и распределенного программирования.

В работе были подробно описаны различные методы, включая сравнение использования технологий FastAPI и gRPC для передачи изображений между микросервисами. Каждый выбранный подход был обоснован, и преимущества каждой технологии были выделены.

Полученные результаты понимание полезности каждого способа. В целом, эти результаты открывают перспективы для использования их в дальнейших проектах.СПИСОК ЛИТЕРАТУРЫ