УДК 004.021:004.75

РАСПРЕДЕЛЕННАЯ СИСТЕМА ПОТОКОВОЙ ОБРАБОТКИ ДАННЫХ ДЛЯ ЗАДАЧ РАСПОЗНАВАНИЯ РЕЧИ

К. Жаксылык
Магистрант кафедры ИТАС, факультета информационных технологий и управления БГУИР
kuanysh.zhk@gmail.com

В. А. Захарьев
Доцент кафедры систем управления БГУИР, кандидат технических наук
zahariev@bsuir.by

К. Жаксылык

Окончил Белорусский государственный университет информатики и радиоэлектроники. Область научных интересов связана с разработкой методов и алгоритмов построения высоконагруженных распределенных систем для потоковой обработки больших объемов данных, построение аналитических платформ на базе хранилищ данных.

В.А.Захарьев

Кандидат технических наук, доцент кафедры систем управления БГУИР. Имеет более 50 научных публикаций. Научные интересны лежат в области методов цифровой обработки сигналов и машинного обучения, а также разработки человеко-машинных интерфейсов на основе синтеза и распознавания речи.

Аннотация. Представлен обзор архитектурных решений для распределенных систем потоковой обработки данных, предназначенных для построения соврменных сервисов распознавания речи на основе глубоких нейросетевых моделей. Рассмотрены основные компоненты таких систем, включая слои хранения и обработки данных, а также особенности их программной реализации. Особое внимание уделено использованию «*Apache Kafka*» в качестве брокера сообщений для обеспечения эффективной передачи данных между компонентами системы. Показаны особенности применения «*Mlflow*» для развертывания модели распознавания речи «*Whisper*», что обеспечивает удобное управление жизненным циклом модели и ее метаданными. Рассмотрен процесс развёртывания приложения в раках концепции микросервисной архитектуры на базе системы управления вычсилительным кластером «*Kubernetes*», предоставляющей широкие возможности масштабирования вычислительных ресурсов. Полученные результаты позволяют определить ключевые характеристики распределённых систем, влияющие на эффективность работы моделей распознавания речи, работающих в реальном времени

Ключевые слова: распределенные системы, распознавание речи, микросервисная архитектура, брокер сообщений «*Apache Kafka*», система управления вычислительным кластером «*Kubernetes*», модель распознавания речь-в-текст «*Whisper*», система развёртывания нейросетевых моделей «*MLflow*».

Введение. В контексте широкого распространения цифровых коммуникаций и проникновения интеллектуальных устройств в различные сферы жизни запрос на создание эффективных и точных системы распознавания речи достиг существенного уровня. В условиях возрастающих потребностей использования речевых технологий как разработчики, так и пользователи, сталкивается с необходимостью обеспечения высокой производительности и масштабируемости моделей распознавания речи. Традиционные подходы к этой задаче часто сталкиваются с проблемами, связанными с огромным объемом и динамическим характером аудиоданных в реальном времени. В ответ на эти

вызовы интеграция распределенных вычислений и обработки потоков данных стала привлекательной парадигмой, которая обещает улучшить скорость, точность и масштабируемость систем распознавания речи.

Современные речевые системы и сервисы, обладающие хорошими показателями качества, из-за высокой степени сложности и ресурсоемкости используемых моделей, имеют распределенную архитектуру. Следовательно, возникает необходимость в развитии методов проектирования, а также, в создании архитектур и алгоритмов распределённых систем, позволяющих обеспечить: повышение отказоустойчивости подобных систем, увеличение скорости доступа к моделям распознования речи, находящимся на удалённых ресурсах, возможность их масштабирования в зависимости от нагрузки.

1 Системы потоковой обработки данных.

Рядом специалистов [1,2] были рассмотрены основные принципы построения систем потоковой обработки данных, предложен архитектурный вариант реализации распределенной системы потоковой обработки данных для задач распознавания речи на базе модели *Whisper* с *Apache Kafka* и *Kubernetes*.

Традиционный метод конфигурации инфраструктуры для обработки больших данных часто включает в себя развертывание «*Hadoop*»-кластера и настройку соответствующих инструментов для построения платформы обработки данных. Однако данный подход сталкивается с рядом ограничений, таких как отсутствие возможности четкого разделения между уровнями хранения и вычислений, сложности в масштабировании и недостаточная изоляция сред для различных приложений. Даже при предоставлении «*Hadoop*» в качестве сервиса («*Hadoop-as-a-Service*») облачным провайдером, подход остается схожим с традиционным развертыванием на собственном оборудовании.

Однако существует альтернативный, основанный на облачных технологиях: подход к обработке больших данных, известный как «*Cloud-Native*». Этот подход не только позволяет преодолеть упомянутые ограничения, но и предоставляет дополнительные преимущества, основанные на возможностях облачных технологий. Для реализации такого подхода в качестве инфраструктуры для построения системы используется платформа управления вычислительными ресурсами «*Kubernetes*», которая совместно с другими компонентами, обеспечивает большую гибкую и масштабируемую среду для обработки и анализа данных. Далее будут рассмотрены основные концепции для построения систем потоковой обработки данных.

Потоковая обработка данных – это метод обработки информации, при котором данные анализируются по мере их непрерывного поступления, обеспечивая оперативное реагирование и принятие решений. В сравнении с традиционной пакетной обработкой, где данные группируются и обрабатываются в определенные интервалы времени (пакеты), потоковая обработка работает в режиме реального времени, что позволяет извлекать ценную информацию из потока данных немедленно. Далее представлен алгоритм и основные принципы потоковой обработки данных.

Захват данных:

Процесс начинается с извлечения информации из различных источников данных, таких как сенсоры, приложения и веб-сервисы.

Главная цель этого этапа – обеспечить эффективный и надежный сбор данных для последующей обработки.

Агрегация и преобразование:

После захвата данных они проходят через этап агрегации и преобразования. На этом этапе данные могут быть отфильтрованы, преобразованы в нужный формат или агрегированы для упрощения последующей обработки.

Обработка:

Данные, уже агрегированные и преобразованные, направляются на обработку. Здесь применяются различные алгоритмы и логика для анализа данных, выявления шаблонов, трендов и аномалий.

Принятие решений:

После обработки данных система может автоматически принимать решения на основе полученных результатов. Это может включать в себя автоматические действия или предоставление информации для принятия решений человеком.

Действие и хранение:

В зависимости от результатов обработки, система может выполнять дополнительные действия. Это может включать отправку уведомлений, запись данных в хранилище, архивирование информации или внесение изменений в рабочие процессы.

2 Анализ моделей потоковой обработки данных.

Перед тем как переходить непосредственно к рассмотрению архитектуры распределённой системы необходимо определиться с выбором базовой модели потоковый обработки данных. Можно выделить две основные такие модели: точка-точка и микросервисная модель [6].

Модель «точка-точка» в архитектуре потоковой обработки данных представляет собой подход, при котором данные передаются напрямую от одного источника к одному или нескольким конечным потребителям.

Микросервисная модель в контексте потоковой обработки данных представляет собой подход, при котором сложные системы обработки данных разделяются на небольшие, автономные компоненты, называемые микросервисами.

Произведём сравнительный анализ моделей построения распределённой системы используя метод их качественного сопоставления, который подразумевает систематическое изучение и анализ характеристик двух или более подходов, моделей или явлений с целью выявления их сходств, различий, преимуществ и недостатков. Результаты качественного анализа моделей представлены в табл. 1.

Таблица 1. Результаты качественного анализа моделей потоковой обработки данных.

Модели потовокой обработки данных	Преимущества	Недостатки
Точка-точка	Низкая задержка: поскольку данные передаются непосредственно от источника к потребителю без промежуточных этапов обработки, задержка минимальна, что особенно важно для приложений, требующих реакции в реальном времени.	Отсутствие гибкости: Такая модель не обеспечивает гибкости в управлении данными и их обработке на промежуточных этапах. Возникают сложности при добавлении новых источников или потребителей данных.

Продолжение таблицы 1

Модели потовокой обработки данных	Преимущества	Недостатки
	Простота реализации: Эта модель является относительно простой в реализации и управлении, так как не требует сложной логики маршрутизации и обработки данных.	Отсутствие возможности анализа и преобразования данных: поскольку данные передаются непосредственно, без промежуточной обработки, они не могут быть анализированы или преобразованы на этапе передачи.
	Эффективное использование ресурсов: поскольку данные напрямую доставляются от источника к потребителю, нет необходимости хранить данные на промежуточных узлах, что способствует экономии ресурсов.	Ограниченная масштабируемость: при увеличении количества источников и потребителей модель «точка-точка» может столкнуться с ограничениями масштабируемости из-за необходимости поддерживать прямые соединения между каждой парой источник-потребитель
Микросервисная модель	Гибкость: Модель «микросервисы» позволяет гибко масштабировать и развивать систему путем добавления, удаления или изменения микросервисов.	Мониторинг и отладка: Следить за работой нескольких микросервисов и находить ошибки может быть более сложно по сравнению с другими моделями
	Изоляция ошибок: Изоляция микросервисов позволяет ограничить влияние ошибок в одном компоненте на другие части системы, обеспечивая надежность и стабильность работы.	
	Независимая разработка: Команды разработчиков могут работать над отдельными микросервисами независимо друг от друга, ускоряя процесс разработки.	

Окончание таблицы 1

Модели потовокой обработки данных	Преимущества	Недостатки
	Технологическое разнообразие: Разные микросервисы могут быть реализованы с использованием различных технологий, что позволяет выбирать наилучшие инструменты для каждой задачи.	

Исходя из результатов анализа, представленных выше, можно сделать выводы что микросервисная модель представляет собой привлекательную и грамотную выборку в контексте потоковой обработки данных, при условии ее правильной реализации.

3 Архитектура распределённой системы потоковой обработки.

Типовая архитектура потоковой обработки данных выделяет два ключевых слоя, необходимых для организации системы:

– слой хранения (*storage layer – SL*);

– слой обработки (*processing layer – PL*).

Слой хранения должен обеспечивать возможность организации высокой пропускной способности для операций ввода/вывода для больших потоков данных. Слой обработки отвечает за потребление данных, поступающих из слоя хранения, обработку этих данных и оповещение *SL* об обновлении данных [9].

Типовая двухслойная архитектура, состоящая из слоя хранения и слоя обработки, имеет свои ограничения, особенно в контексте распределенных систем и обработки больших объемов данных. Например, она может столкнуться с проблемами масштабирования, управления ресурсами и обеспечения отказоустойчивости в условиях высокой нагрузки. Кроме того, она может оказаться недостаточно гибкой для обработки разнообразных источников данных и реализации сложных алгоритмов обработки.

Для организации распределенной обработки потоков данных на множестве независимых вычислительных устройств, требуется некоторое расширение представленной модели. Исходя из этих принципов и из принципов построения данных систем, была составлена архитектура системы, состоящая из пяти слоев (см. рис. 1) [10]:

1 Слой потребления потока данных, включает операции по получению потоков данных из первоначальных источников и передаче в систему обработки или хранения данных [8]. Это могут быть различные *IoT* – решения, датчики, *WebSocket* соединения, системы логирования, брокеры сообщений, очереди и т.п.

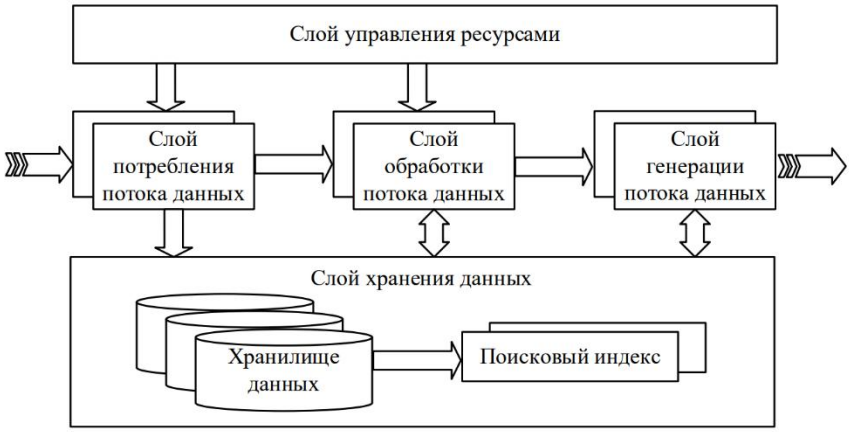

Рисунок 1. Архитектура потоковой системы обработки данных

2 Слой обработки потока данных, это слой, на котором выполняются приложения для обработки потоковых данных. На нем могут размещаться как отдельные приложения обработки данных, так и платформы обработки потоков данных (*Data Stream Processing Engines – DSPE*) [1]. Например, *Azure Databricks*, единая, открытая платформа аналитики для создания, развертывания, совместного использования и обслуживания корпоративных данных, аналитики и решений искусственного интеллекта в масштабе.

3 Слой хранения данных, обеспечивает хранение сообщений, промежуточных и финальных результатов обработки, выявленных паттернов и информации из потока данных [2]. Различные решения могут обеспечить поддержку такого слоя, включая хранилища «ключ-значение», реляционные и *NoSQL* базы данных, базы данных в памяти [3].

4 Слой управления ресурсами отвечает за организацию работы и взаимодействие вычислительных узлов, узлов хранения, а также за управлением временем жизни данных ресурсов (включая выделение новых и освобождение занятых ресурсов) для поддержки обработки больших объемов потоков данных [4]. В качестве примера может выступать служба для координации распределённых систем, организованная на основе резидентной базы данных категории «ключ – значение» *Apache Zookeper*, *Kubernetes* для оркестровки контейнеризированных приложений.

5 Слой генерации потока данных отвечает за формирование результирующего потока данных и его отправку последующим потребителям. В качестве таких потребителей могут выступать сторонние приложения, другие системы обработки данных, системы визуализации и мониторинга и др. [3].

4 Основные компоненты архитектуры и особенности их реализации.

В качестве основного компонента предлагаемой архитектуры слоя потребления данных в работе предлагается использовать брокер сообщений «*Apache Kafka*», представляющий собой одновременно северное программное обеспечение, реализующее функции распределённого брокера (очереди) сообщений, а также соответствующий фреймворк для различных языков программирования, позволяющих подключаться к серверу и передавать сообщения. Работа с *Apace Kafka* основана на использовании следующих основных программных абстракций и компонентов:

– брокеры;
– продюсеры;
– потребители.

Брокер – это экземпляр сервера «*Apache Kafka*» (также известный как узел *Kafka*), который размещает именованные потоки записей, называемые темами. Брокер принимает сообщения от продюсеров и сохраняет их в темы. В свою очередь, он позволяет потребителям извлекать сообщения из темы.

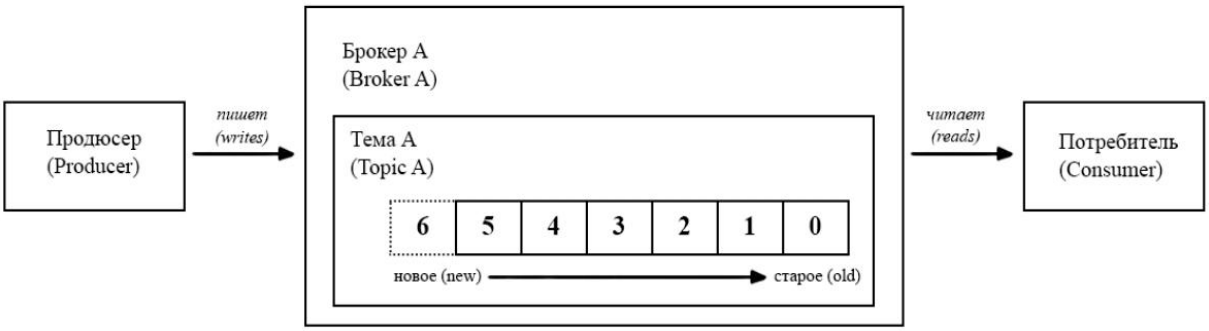

Рисунок 2. Архитектура брокера сообщений на основе «*Apache Kafka*»

В случае использования базового варианта архитектуры кластера «*Kafka*» может функционировать с одним брокером сообщений. Такой вариант использования приводит к появлению в системе единой точки отказа, что может нарушить связность компонентов в системе: препятствовать продюсерам отправлять сообщения, а потребителям получать их. Чтобы обеспечить доступность, нам нужны запасные брокеры для резервного копирования.

Компонент «*Kafka*» является приложением с состоянием, что означает, что ему нужно сохранять данные и поддерживать свое состояние. Развертывание приложения с состоянием представляет собой довольно сложную задачу. Для решения данной проблемы в рамках предложенной архитектуры воспользуемся системой управления вычислительными ресурсами «Kubernetes», которая эффективно позволяет справиться с задачей создания и управления группой персистентных (постоянных) томов.

Из представленной схемы на рисунке 3 можем выделить следующие ресурсы:

1 Кластер *Kafka* с тремя брокерами. Основное преимущество такого кластера *Kafka* заключается в его доступности. В случае если один из брокеров сообщений выйдет из строя, *Kafka* всё равно способна обрабатывать запросы своих клиентов (как потребителей, так и продюсеров).

2 В сочетании с *StatefulSet* мы используем так называемый «безымянный» сервис (*headless service*), чтобы у каждого пода *Kubernetes* были свои собственные *DNS*-имена. Это необходимо, так как *IP*-адреса подов могут изменяться, и нам нужно обеспечить возможность правильного обращения к подам.

3 Каждый из подов брокеров использует собственное постоянное хранилище на базе механизма *Kubernetes* «*persistent volume*».

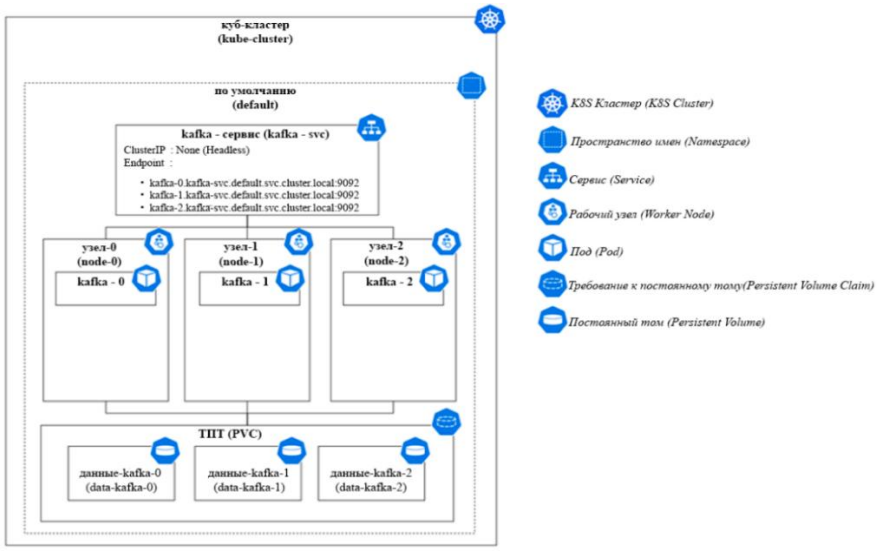

Рисунок 3. Диаграмма развертывания брокера сообщений «*Apache Kafka*», по средствам системы управления вычислительными ресурсами «*Kuberntes*»

Далее представлен Манифест для создания мультиброкерского кластера *Kafka* в *Kubernetes*:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: kafka
  labels:
    app: kafka-app
spec:
  serviceName: kafka-svc
  replicas: 3
  selector:
    matchLabels:
      app: kafka-app
  template:
    metadata:
      labels:
        app: kafka-app
    spec:
      containers:
        - name: kafka-container
          image: doughgle/kafka-kraft
          ports:
            - containerPort: 9092
            - containerPort: 9093
          env:
            - name: REPLICAS
              value: «3»
            - name: SERVICE
              value: kafka-svc
            - name: NAMESPACE
```

```
          value: default
        - name: SHARE_DIR
          value: /mnt/kafka
        - name: CLUSTER_ID
          value: kb-WV7dt8Au9ixOk4YbRm5tL
        - name: DEFAULT_REPLICATION_FACTOR
          value: «3»
        - name: DEFAULT_MIN_INSYNC_REPLICAS
          value: «2»
      volumeMounts:
        - name: data
          mountPath: /mnt/kafka
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes:
          - «ReadWriteOnce»
        resources:
          requests:
            storage: «1Gi»
```

Обработка входящих потоков моделью распознавания речи. После конфигурации Kubernetes кластера и Apache Kafka, необходимо настроить платформу по обработки входящих потоков и дальнешей процесс распознавания речи. Ниже приведен блок кода для потребителя в Apache Kafka.

```
from kafka import KafkaProducer
topic = "audio-stream"

def publish_audio():
    """
    Отправляет аудиопоток с микрофона в указанную тему Kafka.
    """
    # Начало работы продюсера
    producer = KafkaProducer(bootstrap_servers='localhost:9092')
    # Настройка аудиопотока
    CHUNK = 1024
    FORMAT = pyaudio.paInt16
    CHANNELS = 1
    RATE = 44100
    audio = pyaudio.PyAudio()
    stream = audio.open(format=FORMAT, channels=CHANNELS,
                        rate=RATE, input=True,
                        frames_per_buffer=CHUNK)
    print("Отправка аудиопотока...")

    try:
        while True:
            data = stream.read(CHUNK)
            producer.send(topic, data)
```

```
            time.sleep(0.1)  # Спящий режим для управления скоростью потоковой передачи

        except KeyboardInterrupt:
            print("\nЗавершение работы.")
            stream.stop_stream()
            stream.close()
            audio.terminate()
            producer.close()
            sys.exit(1)

    if __name__ == '__main__':
        """
        Продюсер будет отправлять аудиопоток с микрофона на сервер Kafka.
        """
        publish_audio()
```

Эффективное развертывание моделей машинного обучения в производственной среде является критическим фактором, поскольку любая задержка может привести к устареванию данных и утрате воспроизводимости экспериментов. Однако часто процесс передачи моделей от специалистов по данным к инженерам данных требует значительного времени, что может препятствовать оперативному внедрению новых моделей. Решить эту проблему помогает практика *MLOps*, которая стандартизирует процесс разработки моделей машинного обучения.

Одним из ключевых инструментов в рамках практики *MLOps* является *MLflow* – платформа для машинного обучения и *Data Science*. *MLflow* предоставляет средства для эффективного управления жизненным циклом моделей, включая наблюдение экспериментов, моделей и артефактов. Кроме того, *MLflow* обеспечивает возможность оперативного вывода моделей в производство и предоставления их в виде сервиса всего за несколько минут.

В качестве модели распознавания речи будет использоваться *Whisper* от лаборатории *OpenAI*. *Whisper*, разработанный *OpenAI*, представляет собой универсальную модель *ASR*, обученную для точного преобразования речи в текст. Его особенность заключается в том, это она была обучена на 680 000 часов многоязычных и многозадачных контролируемых данных, собранных из Интернета [5].

Преимущества использования модели *Whisper* с *MLflow*:

Отслеживание экспериментов: облегчает отслеживание конфигураций модели и ее производительности для достижения оптимальных результатов.

Управление моделями: централизует различные версии моделей *Whisper*, улучшая доступность.

Воспроизводимость: обеспечивает согласованность транскрипций путем отслеживания всех компонентов, необходимых для воспроизведения поведения модели.

Развертывание: упрощает развертывание моделей *Whisper* в различных производственных средах, обеспечивая эффективное применение.

Перед тем как приступить к процессу аудиотранскрипции с помощью модели *Whisper* необходимо выполнить несколько подготовительных шагов, чтобы обеспечить плавное и эффективное преобразование аудио в текст.

Получение аудио. Первым шагом является получение аудиофайла для работы. Мы используем общедоступный аудиофайл от *NASA*. Этот образец аудио обеспечивает практический пример для демонстрации возможностей транскрипции *Whisper*.

Инициализация модели и конвейера. Мы загружаем модель *Whisper*, а также ее токенизатор и извлекатель признаков из библиотеки *Transformers*. Эти компоненты необходимы для обработки аудиоданных и преобразования их в формат, понятный модели *Whisper* для транскрипции. Затем мы создаем конвейер транскрипции, используя модель *Whisper*. Этот конвейер упрощает процесс подачи аудиоданных в модель и получения транскрипции.

Настройка среды *MLflow*. Помимо настройки модели и аудиоданных, мы инициализируем среду *MLflow*. *MLflow* используется для отслеживания и управления нашими экспериментами, предлагая организованный способ документирования процесса транскрипции и результатов [7].

Нижеприведенный кодовый блок охватывает эти начальные шаги настройки, обеспечивая основу для нашей задачи аудиотранскрипции с моделью *Whisper*. А также создает *KafkaConsumer* для получения данных и сообщений, созданных *KafkaProducer*:

```
import json
from kafka import KafkaConsumer
import mlflow
import requests
import transformers

# Функция для обработки аудио и распознавания речи с использованием Whisper и MLflow
def process  audio(audio):
    # Установка задачи, которую будет использовать наша реализация конвейера
    task = "автоматическое-распознавание-речи"

    # Определение экземпляра модели
    architecture = "openai/whisper-large-v3"

    # Загрузка компонентов и необходимой конфигурации для Whisper ASR из Hugging Face Hub
    model = transformers.WhisperForConditionalGeneration.from_pretrained(architecture)
    tokenizer = transformers.WhisperTokenizer.from_pretrained(architecture)
    feature_extractor = transformers.WhisperFeatureExtractor.from_pretrained(architecture)
    model.generation_config.alignment_heads = [[2, 2], [3, 0], [3, 2], [3, 3], [3, 4], [3, 5]]

    # Создание конвейера для ASR с использованием модели Whisper
    audio_transcription_pipeline = transformers.pipeline(
        task=task, model=model, tokenizer=tokenizer, feature  extractor=feature  extractor
    )

    # Проверка того, что наш конвейер способен обработать аудиофайл и провести транскрибирование
    transcription = audio_transcription_pipeline(audio)
    return transcription

# Настройки потребителя Kafka
topic = "audio-stream"
bootstrap_servers = 'localhost:9092'

# Настройка Kafka потребителя
```

```
consumer = KafkaConsumer(topic,
                         bootstrap_servers=bootstrap_servers,
                         value_deserializer=lambda m: m)

# Настройка MLflow
mlflow.set_tracking_uri("http://mlflow-server:5000")
mlflow.set_experiment("эксперимент-распознавания-речи")

# Обработка аудио и распознавание речи для каждого сообщения из Kafka
for message in consumer:
    audio_data = message.value  # Получение аудиоданных из Kafka
    # Обработка аудио с использованием Whisper и MLflow
        with mlflow.start_run():
            transcription = process_audio(audio_data)
            # Логирование результатов в MLflow
            mlflow.log_text("transcription", transcription["text"])
```

Заключение. В статье были рассмотрены существенные аспекты создания распределённых систем потоковой обработки информации для систем распознавания речи. Представлены существующие и перспективные методы построения систем потоковой обработки, включая архитектурный вариант с использованием *Apache Kafka* и *Kubernetes* для задач распознавания речи. Выполнен сравнительный качественный анализ моделей «точка-точка» и микросервисной архитектуры, их преимуществ и недостатков в контексте распределенных систем. Предложена архитектура распределённой системы потоковой обработки, включая слои хранения и обработки данных. Показаны их основные ограничения и возможности расширения для дальнейшего улучшения. Детально рассмотрены ключевые компоненты архитектуры и особенности их реализации, такие как: брокер сообщений *Apache Kafka* и система управления вычислительными ресурсами *Kubernetes*, и их роль в обеспечении масштабируемости и отказоустойчивости системы. Приведены программные артефакты описания процесса конфигурации и использования модели распознавания речи *Whisper* для обработки и транскрибирования аудиоданных.

Разработанная система демонстрирует, как использование *Apache Kafka, Kubernetes* и *MLflow* вместе с моделью распознавания речи *Whisper* от *OpenAI* улучшает масштабируемость, отказоустойчивость и точность систем распознавания речи. Система применима в различных сферах, не только непосредственно для распознавания речи, но и, например, для зада автоматизации обработки обращений и создание автоматических субтитров. Отмечено, что в процессе реализации важно учитывать потребности в вычислительных ресурсах, безопасность данных и предотвращение сбоев. Дальнейшее усовершенствование системы может включать оптимизацию архитектуры, улучшение обработки данных и интеграцию с другими сервисами. Результаты исследования подтверждают значимость развития распределенных систем обработки данных для построения эффективного высоконагруженных распределённых сервисов обработки пользовательских данных.

Список литературы

[1] Alaasam A.B.A. et al. Analytic Study of Containerizing Stateful Stream Processing as Microservice to Support Digital Twins in Fog Computing // Programming and Computer Software. 2020. Vol. 46, no. 8. P. 511–525. DOI:10.1134/S0361768820080083

[2] Isah H. et al. A Survey of Distributed Data Stream Processing Frameworks // IEEE Access. IEEE, 2019. Vol. 7, no. October. P. 154300–154316. DOI:10.1109/ACCESS.2019.2946884.Meehan J., Zdonik S. Data Ingestion for the Connected World // Cidr. 2017.

[3] Gualtieri M., Yuhanna N. The forrester wave: Big data streaming analytics, Q1 2016 // Forrester research. Cambridge, MA, USA, 2016. 15 p

[4] Henning S., Hasselbring W. Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures // Big Data Research. 2021. Vol. 25. DOI:10.1016/j.bdr.2021.100209.

[5] Real-time full-text search with Luwak and Samza - Confluent [Electronic resource]. URL: https://www.confluent.io/blog/real-time-full-text-searchwith-luwak-and-samza/ (accessed: 27.04.2021).

[6] Dias de Assunção M., da Silva Veith A., Buyya R. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions // Journal of Network and Computer Applications. Elsevier Ltd, 2018. Vol. 103. P. 1–17. DOI:10.1016/j.jnca.2017.12.001.

[7] Kleppmann, Martin. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. « O'Reilly Media, Inc.», 2017.

[8] Tanenbaum, Andrew S. Distributed systems principles and paradigms. 2007.

[9] Vitillo, Roberto. Understanding Distributed Systems: What every developer should know about large distributed applications. Roberto Vitillo, 2022.

[10] Joshi, Unmesh. "Patterns of distributed systems." martinfowler. com (2020).

Авторский вклад

Жаксылык Куаныш – руководство исследованием по разработке распределенной системы потоковой обработки данных для задач распознавания речи. Тестирование распределенной системы, развертывание кластера Kubernetes, Apache Kafka, оценка результатов.

Захарьев Вадим Анатольевич – постановка задачи исследования, описание принципа работы платформы MLflow и модели распознавания речи Whisper, формирование структуры статьи.

DISTRIBUTED STREAM DATA PROCESSING SYSTEM FOR SPEECH RECOGNITION TASKS

K. Zhaksylyk
Master's student, Department of Information Technologies of Automated Systems, Faculty of Information Technologies and Management of BSUIR

V.A. Zahariev
Associate Professor of the Department of Control Systems, Faculty of Information Technologies and Management of BSUIR, PhD of Technical Sciences, Associate Professor

Abstract. The review of architectural solutions for distributed streaming data processing systems aimed at speech recognition tasks is presented. The main components of such systems are considered, including the architectural structure of the part, as well as software implementation. Particular attention is paid to using Apache Kafka as a message broker to ensure efficient data transfer. It also describes how to use MLflow to deploy the Whisper speech recognition model, which allows for easy management of the model's lifecycle and metadata. A microservice architecture based on Kubernetes is considered, providing scalability and application management. The results obtained allow us to determine the key directions for the development of distributed streaming data processing systems for efficient and accurate speech recognition in real time.

Keywords: microservice architecture, message broker Apache Kafka, Kubernetes, Whisper, MLflow, speech recognition, distributed systems.