МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА МЕТОДИЧЕСКИХ УКАЗАНИЙ ПО РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ ОБРАБОТКИ ПОТОКА СОБЫТИЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 5 курса 551 группы
направления 09.03.04 Программная инженерия
факультета КНиИТ
Шишигиной Елены Николаевны

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире данные непрерывно генерируются и накапливаются практически из всех сфер человеческой жизни в огромных масштабах. К ним относится любая отрасль, связанная либо с человеческими взаимодействиями, либо с вычислениями. К основным источникам данных относят, например, Интернет (социальные сети, СМИ, различные сервисы), показания датчиков, приборов и других устройств. Мы получаем информацию, анализируем ее, выполняем над ней какие-либо действия, делаем на основании результата какие-либо выводы и создаем новые данные в качестве результатов. Каждый байт данных что-нибудь да значит — что-нибудь, определяющее дальнейшие действия. В этом и заключается актуальность настоящей работы.

Целью настоящей работы является разработка методических указаний для студентов и преподавателей направлений компьютерных наук и информационных технологий в рамках дисциплины «Технология распределенной обработки данных» по реализации приложения обработки потока событий.

Для достижения поставленной цели необходимо решить следующие задачи:

- Подбор и анализ литературы по заданной тематике;
- Анализ и подбор фреймворков, программных платформ и выбор языков программирования для взаимодействия с ними;
- Построение технологического стека;
- Исследование выбранных фреймворков и программных платформ обработки потоковых данных;
- Разработка клиентского приложения получения сообщений потока событий;
- Разработка приложения для обработки потока событий;
- Описание указаний к реализации приложения, возможных трудностей и ошибок.

В теоретической части работы необходимо описать все используемые технологии и необходимые предварительные сведения для программной реализации. В практической части работы необходимо описать разработанный программный продукт и дать подробные сведения к его реализации.

1 Теоретическая часть

1.1 Потоковая обработка данных

Потоковая обработка (англ. stream processing) — это практика выполнения действий с серией данных во время их создания [1]. Исторически сложилось так, что специалисты по обработке данных подразумевали «обработку в реальном времени», чтобы в целом говорить о данных, которые обрабатывались так часто, как это необходимо для конкретного случая использования. Но с появлением и внедрением технологий и фреймворков потоковой обработки в сочетании со снижением цен на оперативную память термин «потоковая обработка» стал использоваться более специфическим образом. Время обработки можно измерять в микросекундах (одна миллионная секунды), а не в часах или днях. Данные в реальном времени обычно относятся к данным, которые немедленно становятся доступными без задержки из исходной системы или процесса для некоторых последующих действий.

Есть три типа систем обработки потоковых данных в реальном времени: жесткого реального времени, мягкого реального времени и почти реального времени. Опознать систему жесткого реального времени довольно просто. Почти всегда они встречаются во встраиваемых системах и характеризуются очень строгими временными ограничениями, невыполнение которых может привести к полному отказу системы или привести к угрозе жизни. Система считается системой мягкого или почти реального времени в зависимости от того, как воспринимают задержку пользователи.

Во многих ситуациях вычислительная часть системы работает в режиме нежесткого реального времени, но клиенты потребляют данные не в реальном времени из-за сетевых задержек, дизайна приложения или просто потому, что клиентское приложение не запущено. Иначе говоря, мы имеем службу нежесткого реального времени и клиентов, которые потребляют данные, когда захотят. Это и называется системой потоковой обработки данных – система нежесткого реального времени, которая делает данные доступными, когда клиентское приложение хочет их видеть. Это не система мягкого или почти реального времени, это потоковая система [2]. Такая концепция потоковой обработки данных исключает всякие недоразумения, связанные с нечетким различием между системами мягкого реального времени, почти реального времени и вообще не реального времени, и позволяет сосредоточиться на проектировании систем,

которые доставляют информацию в тот момент, когда клиент ее запрашивает.

Потоковая обработка чаще всего применяется к данным, которые генерируются как серия событий, например, к данным от датчиков Интернета вещей, систем обработки платежей, а также журналов серверов и приложений. Такую обработку называют обработкой потоков событий (англ. event stream processing, ESP) [3].

Итак, можно составить некоторую архитектуру потоковой системы обработки событий, которая будет состоять из:

— Звена сбора данных;
— Звена очереди сообщений;
— Звена анализа;
— Звена долгосрочного хранения;
— Звена хранилища данных в памяти;
— Доступа к данным.

1.2 Технологический стек

1.2.1 Apache Kafka

Apache Kafka — это платформа распределенной потоковой передачи событий с открытым исходным кодом, используемая тысячами компаний для высокопроизводительных конвейеров данных, потоковой аналитики, интеграции данных и критически важных приложений [4]. Kafka — это распределенная система, состоящая из серверов и клиентов, которые обмениваются данными через высокопроизводительный сетевой протокол TCP. Его можно развернуть на аппаратном обеспечении, виртуальных машинах и контейнерах как в локальной, так и в облачной среде.

Производители (producer) — это те клиентские приложения, которые публикуют (записывают) события в Kafka, а потребители (consumer) — это те, которые подписываются на эти события (читают и обрабатывают). В Kafka производители и потребители полностью отделены друг от друга и не зависят друг от друга, что является ключевым элементом дизайна для достижения высокой масштабируемости, которой славится Kafka. Например, производителям никогда не нужно ждать потребителей. Kafka предоставляет различные гарантии, например возможность обрабатывать события ровно один раз.

События организованы и надежно хранятся в темах (topic). Очень упро-

щенно, тема (или просто топик) похожа на папку в файловой системе, а события — это файлы в этой папке. События в теме можно читать сколь угодно часто — в отличие от традиционных систем обмена сообщениями, события не удаляются после использования. Вместо этого вы определяете, как долго Kafka должен сохранять ваши события с помощью настройки конфигурации для каждого топика, после чего старые события будут отброшены. Производительность Kafka практически не зависит от размера данных, поэтому хранить данные в течение длительного времени — это нормально [5].

Альтернативами Apache Kafka считаются такие фреймворки, как RabbitMQ, Apache Pulsar, Apache Flume и другие [6].

1.2.2 Apache Spark

Apache Spark — это фреймворк с открытым исходным кодом для параллельной обработки и анализа слабоструктурированных данных в оперативной памяти. Он разработан для обеспечения скорости вычислений, масштабируемости и программируемости, необходимых для больших данных, особенно для потоковых данных, графических данных, машинного обучения и приложений искусственного интеллекта [7]. Spark — это инструмент обработки данных. Он позволяет выполнять различные операции с распределенными коллекциями данных, но не предусматривает их распределенного хранения. Механизм аналитики Spark обрабатывает данные от 10 до 100 раз быстрее, чем альтернативы. Он масштабируется за счет распределения обработки между большими кластерами компьютеров со встроенным параллелизмом и отказоустойчивостью. Он даже включает API-интерфейсы для языков программирования, популярных среди аналитиков и специалистов по обработке данных, включая Scala, Java, Python и R.

Каждое Spark-приложение состоит из управляющего процесса — драйвера (Driver) — и набора распределённых рабочих процессов — исполнителей (Executors). Driver запускает главный метод приложения, в нем создается SparkContext — объект, который отвечает за реализацию операций с кластером — автономным диспетчером кластеров Spark или другими диспетчерами кластеров, такими как Hadoop YARN, Kubernetes или Mesos, — для распределения и отслеживания выполнения по узлам. Он также создает устойчивые распределенные наборы данных (RDD), которые являются ключом к замечательной скорости обработки Spark. Устойчивые распределенные наборы данных (RDD)

— это отказоустойчивые коллекции элементов, которые могут быть распределены между несколькими узлами в кластере и работать с ними параллельно. RDD — это фундаментальная структура Apache Spark.

Spark загружает данные, ссылаясь на источник данных или распараллеливая существующую коллекцию с помощью метода SparkContext parallelize в RDD для обработки. После загрузки данных в RDD Spark выполняет преобразования и действия с RDD в памяти. Spark также хранит данные в памяти, если в системе не заканчивается память или пользователь не решает записать данные на диск для сохранения [8].

Каждый набор данных в RDD разделен на логические разделы, которые могут быть вычислены на разных узлах кластера. Кроме того, пользователи могут выполнять два типа операций RDD: преобразования и действия. Преобразования — это операции, применяемые для создания нового RDD. Действия используются для указания Apache Spark применить вычисление и передать результат обратно драйверу. Помимо RDD, Spark обрабатывает два других типа данных: DataFrames и DataSets. DataFrames являются наиболее распространенными интерфейсами структурированного программирования приложений и представляют собой таблицу данных со строками и столбцами. DataSets — это расширение DataFrames, обеспечивающее объектно-ориентированный программный интерфейс. По умолчанию наборы данных представляют собой набор строго типизированных объектов, в отличие от DataFrames [9].

С развитием потребности к сбору, анализу и обработке больших данных появляются новые фреймворки. Некоторые большие корпорации разрабатывают собственный продукт с учетом внутренних задач и потребностей. Если говорить о фреймворках, популярных среди широкого круга пользователей, то кроме уже упомянутого Hadoop можно назвать Apache Flink, Apache Storm и Apache Samza. Каждый фреймворк имеет свои слабые и сильные стороны. Пока ни один из них не является универсальным и не может заменить остальные.

2 Практическая часть

В рамках данной работы было решено реализовать приложение потоковой обработки данных в учебных целях — на локальной машине, то есть, по сути, на одном кластере. Разработанное приложение легко масштабируется и модифицируется. Оно наглядно показывает основные ступени обработки информации, знакомит с основными понятиями и приемами работы в области обработки потоковых данных. В начале нужно было определить, с какими потоками данных будет работать разрабатываемое приложение. Множество различных веб-сервисов предоставляют свои потоки событий, самые популярные — это, конечно, социальные сети, такие, как Twitter, Facebook, Instagram, LinkedIn, Instagram, Reddit и так далее. Для реализации приложения был выбран поток событий фонда Викимедиа.

2.1 Wikimedia EventStreams

Фонд Викимедиа (Wikimedia Foundation) — некоммерческая организация, размещающая веб-сайты, известные как «проекты Викимедиа» [10]. Все основные проекты Фонда Викимедиа разрабатываются совместно пользователями со всего мира с помощью программного обеспечения MediaWiki. Все материалы выпускаются под бесплатной лицензией Creative Commons, что означает, что любой контент проекта может свободно использоваться, редактироваться, копироваться и распространяться в соответствии с условиями лицензии. У Викимедии множество сервисов, от каждого из которых приходит поток тех или иных событий. Так как все сервисы находятся в свободном доступе для внесения изменений, существует необходимость отслеживать потоки происходящих событий. В связи с этим разработчиками Викимедии был разработан веб-сервис EventStreams, который предоставляет непрерывные потоки структурированных данных о событиях. Этот веб-сервис, в свою очередь, основан на IBM Event Streams. Данный веб-сервис предоставляет потоки через HTTP с использованием кодирования передачи по частям в соответствии с протоколом Server-Sent Events (SSE), поддерживается Apache Kafka. EventStreams можно использовать напрямую через HTTP, но чаще используется клиентская библиотека SSE — своя для каждого языка программирования [11].

Веб-сервис Wikimedia EventStreams предоставляет множество потоков событий. Полный список предоставляемых потоков можно найти по адре-

су stream.wikimedia.org/?doc. Все доступные пути URI потока начинаются с /v2/stream. Потоки адресуются либо индивидуально, например, /v2/stream/revision-create, либо в виде списка потоков, разделенных запятыми, для составления, например, /v2/stream/page-create,page-delete,page-undelete. Потоком, предоставляющую наиболее разнородную информацию о различных типах событий является поток recentchange.

2.1.1 Поток событий recentchange

Поток событий recentchange веб-сервиса Wikimedia EventStreams транслирует все изменения во всех общедоступных вики. Запрос /v2/stream/recentchange запустит поток данных в формате SSE. Этот формат лучше всего интерпретировать с помощью клиента EventSource [12]. Если его не использовать, необработанный поток будет по-прежнему удобочитаемым и будет выглядеть следующим образом (см. рисунок 1):

```
event: message
id: [{"topic":"eqiad.mediawiki.recentchange","partition":0,"timestamp":1532031066001},
{"topic":"codfw.mediawiki.recentchange","partition":0,"offset":-1}]
data: {"event": "data", "is": "here"}
```

Рисунок 1 – Необработанное событие из потока recentchange Wikimedia EventStream

Event может принимать значение message для событий данных и error для событий ошибок. Id представляет собой массив в формате JSON метаданных. Поле id можно использовать, чтобы сообщить EventStreams, что нужно начать потребление с более ранней позиции в потоке. Это позволяет клиентам автоматически возобновлять работу с того места, где они остановились, если они были отключены [13]. EventStreams не имеет возможности фильтрации на стороне сервера — фильтрация получаемой информации производится в клиентском приложении.

2.2 Установка и настройка Apache Kafka

Apache Kafka представляет собой Java-приложение, которое может работать на множестве операционных систем, в числе которых Windows, MacOS, Linux и др. Чаще всего Kafka устанавливают на операционную систему Linux. Linux также является рекомендуемой операционной системой для развертывания Kafka общего назначения, на ней легче и быстрее происходит установка

и настройка. В связи с этим было решено установить виртуальную машину Oracle VM VirtualBox 6.1 с операционной системой Ubuntu 18.04 LTS. Основной памяти виртуальной машине необходимо выделить не менее 3 Гб. Вообще говоря, рекомендуется работать с такими фреймворками, как Kafka и Spark, имея хотя бы 16 Гб оперативной памяти на основной машине. Также необходимо настроить сетевые параметры виртуальной машины — для того, чтобы сообщения с клиентского приложения, запущенного на основной машине, могли дойти до фреймворка, запущенного на виртуальной машине. Для этого нужно выбрать тип подключения «NAT» и создать правило проброса портов.

Прежде чем начать установку Kafka, необходимо установить и настроить среду Java. Рекомендуется использовать Java 8, причем это может быть версия, как включенная в вашу операционную систему, так и непосредственно загруженная с сайта java.com. Хотя Kafka будет работать и с Java Runtime Edition, при разработке утилит и приложений удобнее использовать полный Java Development Kit (JDK).

Apache Kafka использует ZooKeeper для хранения метаданных о кластере Kafka, а также подробностей о клиентах-потребителях. Zookeeper представляет из себя распределенное хранилище ключ-значение (key-value store), гарантирующий надежное консистентное (consistency) хранение информации за счет синхронной репликации между узлами, контроля версий, механизма очередей (queue) и блокировок (lock). За счет использования оперативной памяти и масштабируемости обладает высокой скоростью. Хотя ZooKeeper можно запустить и с помощью сценариев, включенных в дистрибутив Kafka, установка полной версии хранилища ZooKeeper из дистрибутива очень проста. Загрузить и установить Apache Kafka можно, следуя пошаговым советам на официальном сайте kafka.apache.org/quickstart. После установки Kafka на компьютер, сначала запускается служба Zookeeper, после — сервер Kafka.

2.3 Клиентское приложение получения сообщений потока событий

Клиентское приложение получения сообщений потока событий recentchange Wikimedia EventStreams написано на языке Python 3.8. Данное приложение с помощью клиентской библиотеки sseclient связывается с потоком по адресу https://stream.wikimedia.org/v2/stream/recentchange. Sseclient — это клиентская библиотека Python для перебора потоков HTTP

Server Sent Event, также известная как EventSource, по имени интерфейса Javascript внутри браузеров. Класс SSEClient принимает URL-адрес при инициализации и затем является итератором сообщений, поступающих с сервера [14]. Событие приходит с сервера Wikimedia в формате JSON.

В главной функции приложения для его инициализации указывается адрес (в нашем случае — локальный, в формате «хост:порт») продюсера сообщений Apache Kafka, наименование топика, в который будут публиковаться сообщения и URL-адрес потока событий Wikimedia. Публикация сообщений в Apache Kafka возможна благодаря подключенной клиентской библиотеке kafka.

Функцией клиентской библиотеки json `json.loads()` поступающие строки в формате JSON десериализуются в объект Python с помощью таблицы преобразования `data`. Десериализация нужна для того, чтобы из поступающих данных о событии выбрать лишь нужные нам пары «ключ-значение». Клиентское приложение берет такие поля каждого события, как имя пользователя, создавшего событие, имя страницы, которая была изменена, время события, URL страницы, тип изменения, добавленный комментарий, домен, на котором произошло изменение и информация, является ли пользователь ботом. Далее данные отправляются на публикацию в Apache Kafka.

Перед запуском клиентского приложения необходимо запустить сервер Kafka и, если не был создан ранее топик, создать его, как было описано выше.

2.4 Apache Spark

Установка фреймворка Apache Spark производится с официального сайта `spark.apache.org/downloads.html`. При разработке приложения в рамках данной работы использовалась версия 2.4.4. Установка фреймворка производилась на основную машину под управлением операционной системы Windows 10.

Установка Spark предполагает также установку Apache Hadoop (это возможно произвести с официального сайта `https://hadoop.apache.org/`), который включает в себя как элемент обработки информационных данных MapReduce, так и элемент хранения информационных данных Hadoop Distributed File System (HDFS). Spark не включает в себя независимую систему, позволяющую управлять файлами, по этой причине он требует внедрения HDFS.

Использование фреймворка Apache Spark при разработке приложения

предполагает установление некоторого количества зависимостей для корректной сборки проекта, которые будут описаны далее. В связи с этим для удобства разработки было принято решение использовать IDE IntelliJ IDEA 2019, создав в нем Maven-проект.

2.5 Приложение обработки потока событий

Итак, реализуем приложение обработки потока событий с помощью описанного выше фреймворка Apache Spark на языке Java 15. Для этого создадим Maven-проект для облегчения работы с зависимостями и библиотеками, за версиями и совместимостями которых нужно следить.

Отправной точкой разрабатываемого приложения с использованием фреймворка Apache Spark является SparkSession — создание распределенной системы для исполнения будущих вычислений. Чтобы создать базовый SparkSession, используется SparkSession.builder() с возможными конфигурациями. После необходимо указать, откуда брать данные для чтения. На вход подадим созданный нами ранее топик Kafka. Это осуществляется с помощью метода SparkSession.readStream(). Поступающий поток записывается в Dataset.

Фреймворк Spark, а точнее его библиотека Spark Streaming, с которой мы работаем, каждый интервал времени (здесь — заданный по умолчанию) будет брать из топика Kafka некоторое количество сообщений о событиях и записывать в память в качестве Dataset. Каждый такой набор сообщений в интервалах времени является микро-батчем (micro-batch). С каждым микробатчем будут производиться операции преобразования и обработки, которые мы опишем далее.

Так как данные в топике Kafka хранятся в формате JSON, необходимо будет их распарсить, то есть применить схему, по которой данные должны быть разобраны по полям и храниться в виде «таблицы» в объекте Dataset.

С помощью метода Dataset.writeStream() содержимое потокового набора данных сохраняется во внешнее хранилище, в том числе — выводится на консоль. Далее с помощью библиотеки SQL Spark сделаем выборку из поступающих данных по домену — нас интересуют, например, события, происходящие только на русскоязычном домене wikipedia.org. Над содержимым потокового набора данных Dataset производится SQL-операция выборки, после выполнения которой приложение с помощью движка Java DataBase Connection связывается с базой данных MySQL 8.0. Для этого указывается URL-адрес MySQL-

сервера, в нашем случае — локальный, который содержит также информацию о базе данных, куда будет производиться выгрузка — её наименование, использование кодировки и тайм-зона для корректного соединения. Также необходимо указать название таблицы в базе данных для выгрузки (если она не создана, приложение её создаст самостоятельно с указанным наименованием), логин и пароль учетной записи администратора базы данных.

Например, за буквально три минуты работы приложения всего было обработано более трёх тысяч сообщений о событиях на всех сервисах Wikimedia, и более двухсот сообщений из них выбрано с помощью SQL-инструкции (то есть это те события, которые происходили только в рускоязычном домене только одного из сервисов Wikimedia — Wikipedia.org) и сохранено в базу данных.

ЗАКЛЮЧЕНИЕ

Данные окружают нас повсюду, и каждый день в сети появляются новые источники данных. В ближайшем будущем с созданием системы обработки потоковой информации столкнется практически каждая организация или предприятие. Исходя из этого, ознакомление студентов направлений компьютерных наук и информационных технологий с технологиями обработки потоковых данных приобретает особую актуальность в настоящее время.

В первой главе рассмотрены теоретические аспекты области потоковой передачи данных, необходимые для усвоения студентами в рамках дисциплины «Технология распределенной обработки данных». Во второй главе работы подробно описаны шаги реализации приложения обработки потока событий. Данное приложение основано на технологиях распределенной обработки, передачи и анализа слабоструктурированных данных.

Таким образом, можно считать, что в ходе выполнения дипломной работы были реализованы все поставленные задачи и достигнута обозначенная цель:

- Проанализированы и подобраны фреймворки, программные платформы и выбраны языки программирования для взаимодействия с ними;
- Построен технологический стек;
- Выбранные фреймворки и программные платформы обработки потоковых данных исследованы;
- Разработано клиентское приложение получения сообщений потока событий;
- Разработано приложение для обработки потока событий;
- Описаны указания к реализации приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Densmore, J.* Data Pipelines Pocket Reference / J. Densmore. — USA, Boston: O'Reilly Media, Inc., 2021. — 276 pp.

2 *Калашников, Д.* Проектирование системы потоковой обработки данных / Д. Калашников. — Кишинёв: OmniScriptum Publishing KS, 2018. — 80 с.

3 Определение потоковой передачи данных [Электронный ресурс]. — URL: https://aws.amazon.com/ru/streaming-data/ (Дата обращения 10.05.2021). Загл. с экр. Яз. рус.

4 *Нархид, Н.* Apache Kafka. Потоковая обработка и анализ данных / Н. Нархид, Ш. Гвен, Т. Палино. — СПб: Питер, 2019. — 320 с.

5 Apache Kafka Documentation [Электронный ресурс]. — URL: https://kafka.apache.org/documentation (Дата обращения 14.05.2021). Загл. с экр. Яз. рус.

6 *Беджек, Б.* Kafka Streams в действии. Приложения и микросервисы для работы в реальном времени / Б. Беджек. — СПб: Питер, 2019. — 304 с.

7 *Maas, G.* Stream Processing with Apache Spark / G. Maas, F. Garillot. — USA, Boston: O'Reilly Media, Inc., 2019. — 452 pp.

8 Apache Spark Overview [Электронный ресурс]. — URL: https://spark.apache.org/docs/2.4.4/ (Дата обращения 14.05.2021). Загл. с экр. Яз. рус.

9 *Mehrotra, S.* Apache Spark Quick Start Guide / S. Mehrotra, A. Grade. — UK, Birmingham: Packt Publishing, 2019. — 253 pp.

10 Фонд Викимедиа [Электронный ресурс]. — URL: https://meta.wikimedia.org/wiki/Wikimedia_Foundation (Дата обращения 06.05.2021). Загл. с экр. Яз. рус.

11 Wikimedia EventStreams [Электронный ресурс]. — URL: https://wikitech.wikimedia.org/wiki/Event_Platform/EventStreams (Дата обращения 20.05.2021). Загл. с экр. Яз. рус.

12 API:Recent changes stream [Электронный ресурс]. — URL: https://www.mediawiki.org/wiki/API:Recent_changes_stream (Дата обращения 13.05.2021). Загл. с экр. Яз. рус.

13 Manual:recentchanges table [Электронный ресурс]. — URL: https://www.mediawiki.org/wiki/Manual:Recentchanges_table (Дата обращения 13.05.2021). Загл. с экр. Яз. рус.

14 Python sseclient [Электронный ресурс]. — URL: https://pypi.org/project/sseclient (Дата обращения 12.05.2021). Загл. с экр. Яз. рус.