Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Отчет по преддипломной практике

Студент	А.С. Фёдоров
Руководитель	Д.И. Царёв
Консультант от кафедры ЭВМ	Е.А. Сасин
Нормоконтролер	А.С. Сидорович

МИНСК 2019

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ:
Зав. каф. ЭВМ
_____ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ОТСЛЕЖИВАНИЯ И АНАЛИЗА ПОКАЗАТЕЛЕЙ ЗАГРЯЗНЕНИЯ ОКРУЖАЮЩЕЙ СРЕДЫ НА ОСНОВЕ НЕЙРОННОЙ СЕТИ

БГУИР ДП 1-40 02 01 01 079 ПЗ

Студент	А.С. Фёдоров
Руководитель	Е.А. Сасин
Консультанты:	
от кафедры ЭВМ	Е.А. Сасин
по экономической части	А.А. Горюшкин
Нормоконтролер	А.С. Сидорович
Рецензент	

МИНСК 2019

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСИС. Кафедра: ЭВМ.
Специальность: 40 02 01 «Вычислительные машины, системы и сети».
Специализация: 40 02 01-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Б.В. Никульшин
«____» _____ 2019 г.

ЗАДАНИЕ
по дипломному проекту студента
Фёдорова Алексея Сергеевича

1 Тема проекта: «Программное средство для отслеживания и анализа показателей загрязнения окружающей среды на основе нейронной сети» – утверждена приказом по университету от 5 апреля 2019 г. № 806-с.

2 Срок сдачи студентом законченного проекта: 1 июня 2019 г.

3 Исходные данные к проекту:
- **3.1** Среда разработки: IntelliJ IDEA.
- **3.2** Используемый брокер сообщений: Apache Kafka.
- **3.3** Используемый сервер: Apache Zeppelin.
- **3.4** Используемая файловая система: локальная, HDFS.
- **3.5** Используемый сервис контейнеризации: Docker.
- **3.6** Способ отображения информации: интерактивный документ Apache Zeppelin.

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):
Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. 7. Экономическая часть. Заключение. Список использованных источников. Приложения.

5 Перечень графического материала (с точным указанием обязательных чертежей):

5.1 Вводный плакат. Плакат.

5.2 Заключительный плакат. Плакат.

5.3 Программное средство для анализа. Схема структурная.

5.4 Программное средство для анализа. Диаграмма классов.

5.5 Отправка сообщения системы. Диаграмма последовательности.

5.6 Обработка сообщения системы. Диаграмма последовательности.

6 Содержание задания по экономической части: «Технико-экономическое обоснование разработки программного средства для отслеживания и анализа показателей загрязнения окружающей среды на основе нейронной сети»

ЗАДАНИЕ ВЫДАЛ А.А. Горюшкин

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы	10	23.03 – 07.04	
Структурное проектирование	10	07.04 – 19.04	
Функциональное проектирование	20	19.04 – 28.04	
Разработка программных модулей	30	28.04 – 08.05	
Программа и методика испытаний	10	08.05 – 18.05	
Расчет экономической эффективности	10	18.05 – 25.05	
Оформление пояснительной записки	10	25.05 – 01.06	

Дата выдачи задания: 23 марта 2019 г.

Руководитель Е.А. Сасин

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ _____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С каждым днём промышленность развивается и одним из самых острых вопросов для человечества остаётся вопрос экологии. Именно по этой причине возникает необходимость в непрерывном наблюдении и анализе изменений индексов загрязнений. Для осуществления такого наблюдения необходима система, которая сможет обрабатывать данные в режиме реального времени.

Однако для такой системы будет характерно большое количество данных и большие вычислительные затраты. Именно по этой причине, зарабатываемый проект будет производить вычисления с помощью распределённой системы. Распределенная система - такая система, которая работает сразу на нескольких машинах, образующих кластер. В свою очередь кластер - это набор компьютеров или серверов, объединенных сетью и взаимодействующих между собой. Важнейшие плюсы такого подхода – это высокодоступность и отказоустойчивость.

Для обеспечивания надёжности хранимых данных также необходима отказоустойчивая система. Отказоустойчивая система - это такая система, в которой отсутствует единая точка отказа. Такую систему можно подстраивать под случающиеся отказы. Например, при отказе одной машины, остальные смогут продолжить свою работу, так что в целом кластер не отключится. В качестве машин в кластере могут выступать не только физические, но и виртуальные машины.

Целью данного дипломного проекта является разработка системы, которая позволит осуществлять мониторинг изменений индекса загрязнений окружающей среды. В качестве источника данных был выбран сервис OpenWeatherMap, который предоставляет API для получения данных загрязнений по CO, NO, SO и OZ. Однако данных от этого источника недостаточно для тренировки нейронной сети, поэтому для её тренировки был выбран отдельный датасет.

Для достижения поставленной цели нужно выполнить следующие задачи:

– разработать модуль сбора данных с сервиса OpenWeatherMap;

– разработать модуль обработки предоставляемых данных в режиме реального времени;

– разработать математическую модель, с помощью которой можно предсказывать дальнейшее изменения значения на основе предыдущих;

– разработать систему представления обработанных данных;

Дипломный проект будет реализовать следующие функции:

– сохранение полученных данных;

– представление статистики о полученных данных;

– представление ожидаемых предстоящих значений;

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Брокеры сообщений

Для передачи данных между компонентами в сложной системе используются так называемые брокеры сообщений (англ. message broker). Это программа, целью которой является приём, хранение и передача сообщений. Такой подход позволяет осуществить обработку сообщений в режиме реального времени. Традиционно брокер состоит из следующих компонентов:

– Сообщения - совокупность двоичных или текстовых данных, имеющих что-то общее в контексте программы. К этим данным перед отправкой может добавляться дополнительная информация, в зависимости от протокола.

– Очереди сообщений - объекты, хранящие сообщения в программе;

Существует множество брокеров сообщений. Наиболее известные из них:

– ActiveMQ;
– IBM MQ;
– RabbitMQ;
– Redis.

Для обеспечения схожей функциональности для кластера используется такая платформа, как Apache Kafka [1]. Данная платформа позволяет организовать систему публикации/подписки Структура kafka изображена на рисунке 1.1.

Имеется два общих класса приложений, для работы с данной платформой:

– приложения, которые осуществляют обмен сообщениями для непосредственно передачи данных между компонентами системы;

– приложения, которые преобразуют или реагируют на сообщения.

Kafka - это быстрая, расширяемая, долговечная и отказоустойчивая система обмена сообщениями. Обычно она используется для замены традиционных брокеров сообщений наподобие Java Message Service (JMS) и Advanced Message Queuing Protocol (AMQP) из-за своей высокой пропускной способности, надёжности и использования репликаций. Может использоваться вместе с такими сервисами, как Storm, Spark, Samza, Flink и другими. Такая комбинация позволяет анализировать и обрабатывать данные в режиме реального времени.

Рассматриваемая платформа имеет три важных свойства:

– публикация и подписка к потоку записей схожа на очередь сообщений;

– сохранение потока записей в отказоустойчивой и надёжной системе;

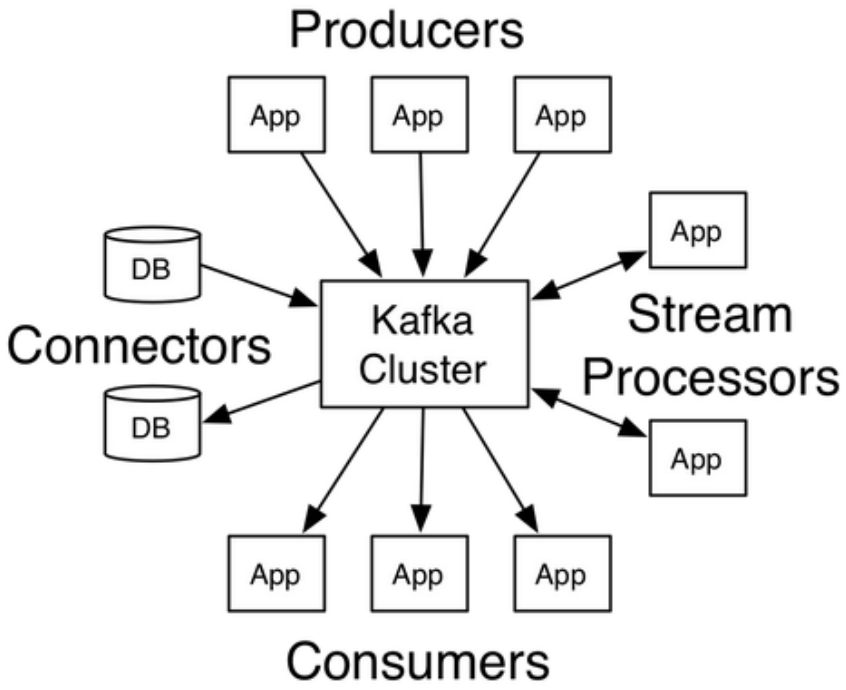

Рисунок 1.1 – Структура брокера сообщений Apache Kafka [1]

– обработка потока записей как только они появляются.

А также имеет ещё несколько особенностей, что отличает её от выше упомянутых аналогов:

– сбор и мониторинг метрик;
– отслеживание активности с помощью веб-сервиса;
– использование репликаций в системе логирования.

Раздел (англ. partition) в kafke представляет из себя упорядоченную неизменяемую последовательность. Каждое сообщение в таком разделе ассоциируется с числом, которое обозначает номер сообщения в последовательности раздела. Это число называется смещением (англ. offset). Смещение используется для сохранения позиции получателя при получении сообщения. В свою очередь, каждый топик состоит из одного или более раздела. Смещение в разделе для каждого получателя хранится в отдельном топике.

Репликации никогда не используются для записи или чтения. Их единственная задача - заменить лидирующую реплику в случае её отказа. Один из брокеров в кластере выступает в качестве контроллера, который выбирается автоматически из активных членов кластера. В задачи этого контроллера входит связывание разделов с брокерами и контроль их отключения.

Разделом всегда владеет один брокер. Такой брокер называется лидером раздела (англ. leader of the partition). Остальные брокеры называются последователями (англ. follower). Все сообщения имеют определённое количество реплик, тем самым при размещении на кластере -

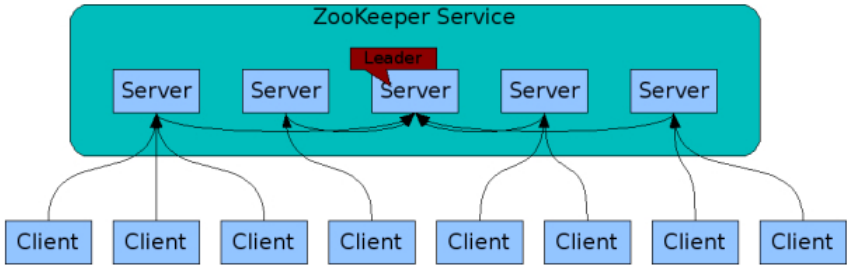

Рисунок 1.2 – Структура координирующего сервиса ZooKeeper [2]

такая система становится отказоустойчивой, ведь при выходе из строя некоторой машины - на остальных машинах останутся реплики сообщений, что позволит продолжить работу. Все последователи постоянно синхронизируются с лидером и копируют актуальную информацию. Также брокеры делятся на активные и устаревшие. Устаревшей реплика становится, если в течении определённого времени она не отослала специальный сигнал heartbeat координатору или в течении определённого времени не скопировала актуальную информацию от лидера. Как только лидер выходит из строя - одна из активных последующих реплик становится новым лидером.

В качестве менеджера выступает zookeeper [2]. Его структура изображена на рисунке 1.2. С помощью данного сервиса достигается отказоустойчивость системы.

1.2 Контейнеризация

Для обеспечения работы всех выше перечисленных сервисов, необходим инструмент, который позволит обеспечить работу кластера на одной машине. Для такой задачи отлично подойдёт docker [3]. Docker - это платформа для разработки, развёртки, и запуска приложений в контейнерах. Использование Linux контейнеров для развёртки приложений называется контейнеризацией.

Контейнеры всё больше и больше становятся популярными по следующим причинам:

– гибкость. Даже самые сложные приложения могут быть контейнеризированы;

– легковесность. Контейнеры используют и разделяют ядро хоста;

– взаимозаменяемость. Возможность развертывать обновления и обновления на лету;

– портативность. Возможность собрать локально, разворачивать в облаке, и запускать везде;

– расширяемость. Возможность увеличивать и автоматически распределять реплики контейнера;

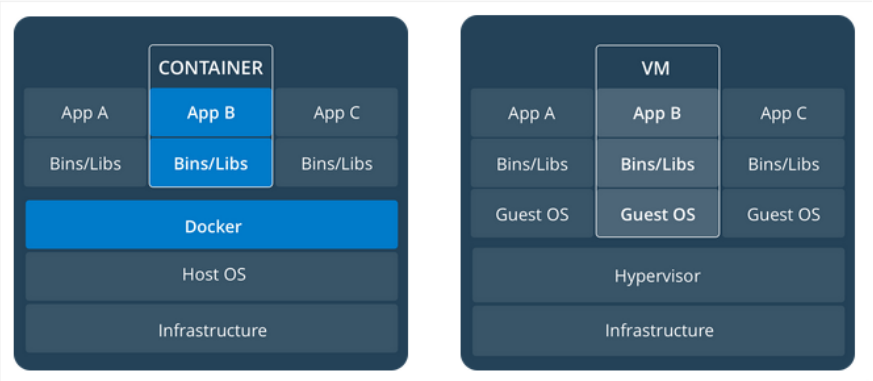

Рисунок 1.3 – Работа docker с обычной контейнеризацией и с использованием виртуальной машины [3]

– наращиваемость. Возможность наращивать сервисы на лету.

Контейнер запускается с помощью образа. Образ это исполняемый пакет, который включает в себя всё необходимое для работы приложения - само приложение, окружение, библиотеки, переменные окружения, файлы конфигурациии. Контейнер это исполняемая сущность образа.

Контейнер изначально работает в Linux и разделяет ядро хост-машины с другими контейнерами. Он запускает отдельный процесс, занимая не больше памяти, чем любой другой исполняемый файл, что делает его легким. В отличие от виртуальной машины работает полнофункциональная «гостевая» операционная система с виртуальным доступом к ресурсам хоста через гипервизор. Как правило, виртуальные машины предоставляют среде больше ресурсов, чем требуется большинству приложений.

Структура работы докера при обычной контейнеризации и с использованием виртуальной машины продемонстрированы на рисунке 1.3

Для обеспечения взаимодействия между контейнерами отлично подходит способ создания виртуальной сети, в которой и запускаются необходимые контейнеры, а также метод пробрасыввания портов, что позволит по необходимому порту пересылать данные между запущенными контейнерами. Также существует более удобный вариант работы с несколькими контейнерами - docker-compose. Данный инструмент позволяет удобно оформить настройки планируемого кластера с помощью файла конфигурации. Благодаря таким инструментам есть возможности добиться работы системы, идентичной в реальных условиях, когда вся эта система будет размещена на полноценном кластере.

1.3 Произведение вычислений

Для обработки вычислений на кластере существует отличный продукт под названием Apache Spark [4]. Apache Spark — это фреймворк с помощью которого можно создавать приложения для обработки данных на кластере.

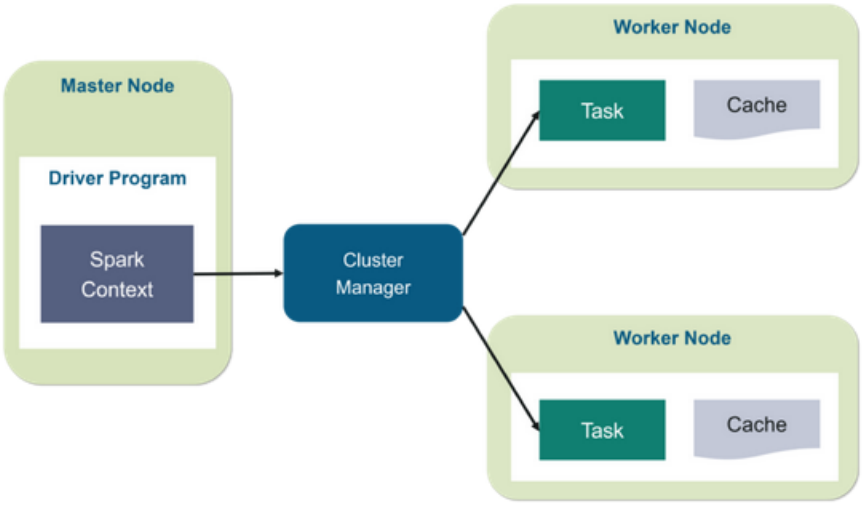

Рисунок 1.4 – Архитектура Apache Spark [5]

Он отвечает за распределенное по всему кластеру выполнение приложения. Данный фреймворк раскидывает код по всем узлам кластера, разбивает на подзадачи, создаёт план их выполнения и следит за их выполнением. Если на каком либо узле произошел сбой, и подзадача завершилась ошибкой, то она будет перезапущена. Для данной цели в кластере определяется мастер (англ. spark-master) и рабочие (англ. spark-worker). Мастер отправляет приложение на рабочие машины и отслеживает их выполнение. Когда рабочий завершит обработку своей задачи, он отсылает результат обратно на мастер [5]. Архитектура Apache Spark представлена на рисунке 1.4.

Результаты вычислений представляются в виде RDD (англ. Resilient Distributed Dataset). В актуальных версиях spark также применяются аналоги RDD, которые имеют другую структуру и интерфейс (DataFrame и DataSet). Эти объекты являются ленивыми и вычисляются только на этапе исполнения. Для отказоустойчивости все вычисления представляют из себя направленный ацикличный граф, благодаря которому, из предыдущих результатов всегда можно получить последующий. Благодаря такому подходу при потере результатов текущих вычислений есть возможность их восстановить из предыдущих.

Все операции над RDD подразделяются на:

– трансформации;

– действия.

Трансформация - преобразование, результатом которого является новый RDD. Примерами такого преобразования служат функции map и filter. Действия - операции, применяемые для материализации результата. Например сохранение в файл или подсчёт количества записей.

spark поддерживает несколько менеджеров кластера:

– автономный (англ. standalone). Простой кластерный менеджер, включённый в spark, что позволяет легко развернуть кластер;

– Apache Mesos. Общий кластерный менеджер, который также может запускать Hadoop MapReduce и сервисные приложения;

– Hadoop YARN. Ресурсный менеджер, используемый в Hadoop 2;

– Kubernetes. Система с открытым исходным кодом для автоматизации развёртки, расширения, и управления конейнтеризированными приложениями.

При использовании автономного режима - spark самостоятельно будет контролировать ресурсы кластера и запускать приложения. Локальный режим запускает приложение как один процесс, что позволяет осуществлять отладку. Yarn это ресурсный менеджер, входящий в экосистему Hadoop [6]. Mesos это альтернативный менеджер ресурсов кластера.

1.4 Визуализация данных

Для визуализации данных существует такой проект, как Apache Zeppelin [7]. Apache Zeppelin - это веб-приложение наподобие проекта Jupyter Notebook с открытым исходным кодом, который позволяет осуществлять интерактивный анализ данных. Данный notebook интегрирован с такими распределёнными, универсальными системами, как Apache Spark, Apache Flink и с многими другими. Zeppelin позволяет создавать удобные, интерактивные документы с помощью таких языков, как SQL, Scala, R и Python, напрямую из браузера.

Особенности данного проекта:

– интерактивный интерфейс. Apache Zeppelin имеет интерактивный интерфейс, который позволяет мгновенно увидеть результаты анализа;

– использование браузера. Создание интерактивных документов напрямую в браузере, что позволяет использовать его удалённо. Также это позволяет эксперементировать с различными типами диаграмм для представления результатов;

– интегрирование. Интеграция со многими различными инструментами с открытым исходным кодом, предназначенными для обработки большого количества данных. Например Spark, Flink, Hive, Ignite, Lens и Tajo;

– динамические формы. Динамическое создание форм ввода прямо в интерактивном документе;

– сотрудничество и обмен. Сообщество разработчиков предоставляет доступ к новым источникам данных, которые постоянно добавляются и распространяются через их лицензию Apache 2.0 с открытым исходным кодом;

– интерпретатор. Концепция интерпретатора позволяет использовать любой язык либо приложения для обработки данных. В настоящее время Apache Zeppelin поддерживает множество интерпретаторов, таких как:

Apache Spark, Python, JDBC, Markdown и Shell.

Благодаря таким свойствам, данный инструмент отлично подойдёт для интерактивного способа представления данных.

1.5 Непрерывная интеграция

Так как разрабатываемый продукт является проектом с открытым исходным кодом, то важной частью процесса разработки является процесс непрерывной интеграции. Для данной цели используется такой инструмент, как Jenkins [8]. Данный инструмент из себя представляет java веб-приложение, которое позволяет запускать заготовленные команды, и отслеживать их исполнение. С помощью jenkins появляется возможность проводить интеграционные тесты для какой-либо ветки в системе git из удалённого репозитория.

Команды, которые должен исполнить jenkins прописываются либо прямо в настройках, либо берутся из удалённого репозитория, к которому jenkins имеет доступ. Все команды исполняются на машине, в которой запущена данная платформа. Однако для создания определённого системного окружения отлично подойдёт Docker и его аналог docker-compose, которые были рассмотрены выше.

С помощью такой системы непрерывной интеграции, всегда можно узнать работоспособность конкретной версии продукта. Эта возможность крайне важна при разработке сообществом.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты разрабатываемой системы и выработав список требований необходимых для разработки системы, разбиваем систему на функциональные блоки(модули). Это необходимо для обеспечения гибкой архитектуры. Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В разрабатываемом веб-приложении можно выделить следующие блоки:

- блок загрузки данных;
- блок преобразования данных;
- блок передачи сообщений;
- блок приёма сообщений;
- блок анализа данных;
- блок сохранения данных;
- блок визуализации данных.

Взаимосвязь между основными компонентами проекта отражена на структурной схеме ГУИР.400201.079 С1.

2.1 Блок загрузки данных

Блок загрузки данных предназначен для получения информацию из удалённого источника. В качестве такого источника выступает сервис OpenWeatherMap [9]. Данный сервис поддерживает множество application programming interfaces (API), для выборки данных. Конкретно для текущих целей используются API для следующих данных:

- индекс монооксида углерода (CO);
- индекс озона (O_3);
- индекс диоксида серы (SO_2);
- индекс диоксида азота (NO_2).

Рассматриваемый блок использует каждый вышеперечисленный API для получения данных. Посредством использования HyperText Tranfer Protocol (HTTP) запроса, блок получает необходимые данные. Данные предоставляются в формате JavaScript Object Notation (JSON). Данный формат позволяет легко получить необходимые поля предоставляемых данных.

У используемого сервиса есть ограничение в виде максимального количества запросов по API в единицу времени. На текущий момент это ограничение составляет 10 запросов в минуту. Для обхода данного ограничения, используется локальное хранилище уже полученных данных. С помощью такого метода, данный блок будет обращаться к внешнему серверу только для получения данных, которых нету в локальном

хранилище. Если же требуемый запрос уже был совершён и был сохранён, то вместо обращения к серверу, блок отправит данные из локального хранилища.

Также, из-за использования модульной структуры данный блок можно заменить блоком, который будет предоставлять локальные данные. Таким образом при отсутствии должного количества данных, есть возможность воспользоваться локальным датасетом. Полученные данным блоком данные отправляются в блок преобразования данных.

2.2 Блок преобразования данных

Блок преобразования данных осуществляет преобразования исходных данных, полученных в формате JSON в необходимые для дальнейшего анализа данные. Текущая структура JSON данных, предоставляемая сервисом OpenWeatherMap имеет множество полей, которые несут дополнительную информацию. Например, в таком формате хранятся данные о исследуемом индексе на различных высотах с разным атмосферным давлением. Как указано в документации к данному сервису, для анализа рекомендуется брать значения между 215 и 0.00464 гектопаскалей (hPa). Поэтому целью рассматриваемого блока и является выбор необходимых значений из предоставляемого формата. Для каждого индекса используется свой трансформатор, так как структура JSON для некоторых индексов отличается. Все эти данные преобразуются в цифровые значения и перенаправляются в блок передачи сообщений.

2.3 Блок передачи сообщений

Блок передачи данных предназначен для отправки подготовленных данных в очередь сообщений. Данная операция необходима для того, чтобы абсолютно любые компоненты системы смогли получить обработанные данные и обработать их своим способом. В качестве очереди сообщений выступает, описанная в предыдущем разделе, платформа kafka. Данная платформа позволит остальным компонентам системы своевременно получить подготовленные данные и начать их обработку.

Одними из важнейших особенностей платформы kafka являются наличие топиков, и представление сообщения как пары ключ-значение. Также на каждый предоставляемый индекс будет использоваться отдельны топик. И из-за использования системы подписки, остальные компоненты системы могут отлавливать только интересующие их индексы. Такой метод позволяет использовать один источник данных для всех компонентов системы. Тем самым обеспечив разрабатываемой системе гибкость и вертикальную расширяемость.

Для обеспечения передачи используется kafka producer API [10]. Это требует определённой конфигурации. В частности, необходимы следующие параметры, которые предоставляются в файлах конфигурации приложения:

– адреса kafka серверов;
– идентификатор клиента;
– сериализатор ключа;
– сериализатор значения;
– количество необходимых подтверждений;
– тип сжатия;
– количество попыток;
– размер пакета;
– максимальный рамер запроса;
– время ожидания запроса;
– протокол безопасности.

Ещё одним не маловажным фактором выбором kafka является её отказоустойчивость. Все эти сообщения будут храниться на кластере. Тем самым, все отправленные данные не потеряются при внезапном отключении машины в кластере.

2.4 Блок приёма сообщений

Блок приёма сообщений отвечает за своевременное принятие сообщений из платформы kafka. Как только данные были получены, они сразу же начинают обработку. Такое поведение называется потоковой обработкой (англ. stream processing).

На самом деле обрабатываеются данные, полученные за установленный период времени. Данный промежуток времени называется окном (англ. window). Данные, полученные за такое окно называется пакетом (англ. micro batch). При обработке в Apache Spark каждый такой пакет будет представлять из себя отдельный RDD, над которым и будут в дальнейшем производиться все операции.

Данному блоку необходимо обеспечить получение данных из конкретных топиков для дальнейшей обработки. Таким образом, есть возможность установить, какие именно данные начнут обрабатываться. Как было отмечено выше, все полученные индексы используются в дальнейшей обработке, так что этот блок должен принимать данные по каждому индексу.

Источником данных служит уже рассмотренная выше платформа kafka. С помощью установки смещения, есть возможность установить, с какого момента данные будут поступать в систему. Тем самым можно использовать только самые актуальные данные, либо использовать предыдущие данные в качестве датасета.

Так как все приёмники объединяются в группы (англ. consumer group),

то есть возможность создания блока, который будет параллельно обрабатывать те же самые данные, с точно таким же смещением. Как раз с помощью данной особенности, данные и обрабатываются параллельно при использовании Apache Spark. Так как код отправляется на активные машины в кластере, то и каждая машина будет параллельно получать данные из kafka платформы, а так как все они входят в одну и ту же принимающую группу, то и данные они будут получать идентичные. Такой подход обеспечивает вертикальную масштабируемость системы. Полученные сообщения отправляются в блок анализа данных и в блок сохранения данных.

2.5 Блок анализа данных

Блок анализа данных предназначен для обработки полученных данных. Данный блок одержит математическую модель, представляющую из себя регрессионную нейронную сеть. На основе полученных данных производится её обучение. Так как данные подаются в рассматриваемый блок постоянно в режиме реального времени, то и внутренняя нейронная сеть обучается так же в режиме реального времени.

Подобный поход позволяет постоянно обновлять математическую модель, чтобы она была актуальной. Однако в таком подходе есть и недостаток. Речь идёт о проблеме переобучения. Данная проблема заключается в ухудшении аппроксимирующей способности нейронной сети при слишком сильном обучении. Для решения данной проблемы используются ограничения в виде максимального количества результатов, на которые будет опираться математическая модель. Это позволит нейронной сети «забывать» старые данные и использовать только актуальные значения.

Весь этот блок тесно сотрудничает с блоком визуализации данных, так как он используется в интерактивном документе Apache Zeppelin. Результаты обработки поступают на блок визуализации данных.

2.6 Блок сохранения данных

Блок сохранения данных предназначен для сохранения полученных данных на диске. Так как в качестве обработчика используется целый кластер, то и сохранение возможно осуществить как в локальной файловой системе, так и в распределённой файловой системе платформы Hadoop (англ. Hadoop Distributed File System (HDFS)). Сохранение данных нужно для восстановления предыдущих данных после перезапуска системы. В течении жизненного цикла приложения в данном блоке нет необходимости, так как все данные успешно хранятся в платформе kafka. Однако при полной перезагрузке системы, придётся заново получать данные из

внешнего веб-сервиса OpenWeatherMap. Данный блок предназначен как раз для того, чтобы этого избежать такой проблемы.

При использовании экосистемы Hadoop появляется возможность сохранения данных в HDFS. В отличие от локальной файловой системы, данная система является распределённой, что обеспечивает её отказоустойчивость.

Основные свойства HDFS:

– Предназначена для хранения большого количества больших файлов (порядка десятков гигабайт).

– Оптимизирована для поддержки потокового доступа к данным (англ. high-streaming read).

– Ориентирована использование кластера большого размера.

– Отказоустойчивость.

Все накопленные данные сохраняются в локальном хранилище, что позволяет сразу загрузить уже полученные данные. А из платформы kafka получать только новые данные, которые ещё не были загружены. Таким образом достигается оптимизация приложения и возможность восстановления процесса обработки при перезапуске системы.

2.7 Блок визуализации данных

Блок визуализации данных предназначен для интерактивного представления обработанных результатов. Для подобного представления используется такая система, как Apache Zeppelin. Как уже отмечалось, Apache Zeppelin использует интерактивные документы, которые используют концепцию интерпретатора, что позволяет моментально визуализировать результаты вычислений. Данная система крайне схожа с проектом Jupyter Notebook по концепции. Она также, как и Jupyter использует интерактивные документы (англ. notebooks).

Данный блок как раз представляет из себя интерактивный документ в системе Apache Zeppelin. Такой подход позволяет использовать совершенно разные способы предоставления данных, а так же в режиме реального времени визуализировать текущие результаты.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробно функционирование программы. Для этого проведем анализ основных модулей программы и рассмотрим их зависимости. А также проанализируем все модули, которые входят в состав кода программы, и рассмотрим назначение всех методов и переменных этих модулей. В разрабатываемом приложении можно выделить следующие модули:

- модуль конфигурации загрузчика;
- модуль хранения загрузчика;
- модуль внешнего подключения загрузчика;
- модуль обработки загрузчика;
- модуль отправки загрузчика;
- модуль приёма обработчика;
- модуль анализа обработчика;
- модуль сохранения обработчика;
- модуль визуализации обработчика.

Изначально пользователю предоставляется интерактивный документ на Apache Zepelin, в котором он может просмотреть обработанные данные.

3.1 Классы модуля конфигурации загрузчика

Для конфигурации приложения используются файлы конфигурации. Классы данного модуля как раз представляют объекты конфигураций, которые создаются на основе файлов конфигураций. Главным объектом конфигурации является класс `ApplicationConfig`. Он содержит в себе все остальные конфигурации, необходимые для работы всего приложения. Сам объект этого класса строится с помощью метода `parse_from_file()`. Этот класс включает в себя более специфические конфигурации, представленные классами: `APIConfig`, `FSConfig`, `KafkaProducerConfig`.

3.1.1 Класс `APIConfig`

Данный класс представляет из себя конфигурацию для работы с сервисом OpenWeatherMap. Сам объект класса создаётся из списка строк, которые представляют собой строки файла конфигурации, ответственные за настройку API. Конфигурация содержит в себе следующие поля:

- `api_key`;
- `host`.

Для использования сервиса OpenWeatherMap необходимо предоставить ключ, который выдаётся при оформлении подписки на определённый сервис API. Данный параметр как раз и является этим

ключом. Этот ключ передаётся в качестве параметра при использовании GET запроса к сервису. Как раз для этого ключа и используется параметр `api_key` в конфигурации.

Параметр `host` устанавливает корневой путь к сервису OpenWeatherMap. Относительно этого пути и строятся все HTTP запросы для получения данных.

Для создания экземпляра рассматриваемого класса, необходимо вызвать статический метод `parse_from_lines()`, который принимает список строк, из которых и будет строиться конфигурация. Для получения ключа для использования API и адреса сервиса OpenWeatherMap из конфигурации, необходимо воспользоваться методами `get_api_key()` и `get_host()` соответственно.

3.1.2 Класс `FSConfig`

Так как для всего приложения используется полноценный кластер, то и необходима возможность использования приложения в экосистеме Hadoop. В этой экосистеме используется HDFS, как уже неоднократно отмечалось. Если для работы с локальной файловой системой можно обойтись без каких-либо конфигураций, то для HDFS необходимо знать её корневой адрес. Данный адрес и является полем `host` в рассматриваемой конфигурации и задаётся с помощью файлов конфигурации.

Для создания экземпляра этого класса, необходимо вызвать статический метод `parse_from_lines()`, который принимает список строк, из которых и будет строиться конфигурация. Для получения адреса к корневому каталогу в используемой локальной системе из конфигурации, необходимо воспользоваться методами `get_host()`.

3.1.3 Класс `KafkaProducerConfig`

Данный класс представляет из себя конфигурацию для работы с платформой kafka. Сам объект класса создаётся из списка строк, которые представляют собой строки файла конфигурации, ответственные за настройку kafka producer. Конфигурация содержит в себе следующие поля:

- `bootstrap_servers`;
- `client_id`;
- `key_serializer`;
- `value_serializer`;
- `acks`;
- `compression_type`;
- `retries`;
- `batch_size`;

– max_request_size;
– request_timeout_ms;
– security_protocol.

Параметр bootstrap_servers указывает адреса kafka серверов в формате «host[:port]».

Параметр client_id указывает уникальный идентификатор клиента. Данный идентификатор используется для логирования.

Параметр key_serializer указывает имя класса сериализатора для ключа. Данный параметр необходим для возможности передачи пользовательских объектов в качестве ключа.

Аналогичным является параметр value_serializer, который указывает имя класса сериализатора для значения. Этот параметр необходим для возможности передачи пользовательских объектов в качестве значения. Также kafka предоставляет набор стандартных классов для сериализации и десериализации. Например org.apache.kafka.common.serialization.StringSerializer и org.apache.kafka.common.serialization.StringDeserializer.

Параметр acks указывает требуемое количество подтверждений от kafka серверов. Возможные значения:

– 0. Означает, что подтверждения не требуются;
– 1. Используется по умолчанию. Означает, что требуется хотя бы одно подтверждение;
– all. Означает, что для каждого запроса необходимо подтверждение.

Параметр compression_type указывает тип сжатия для всех отправленных данных, либо его отсутствие. Возможные значения:

– gzip;
– snappy;
– lz4;
– None. Используется по умолчанию.

Параметр retries указывает количество попыток отправить данные заново при неудаче.

Параметр batch_size указывает максимальный размер пакета для отправки. При превышении данного размера, данные будут разбиты на несколько пакетов.

Параметр max_request_size указывает максимальный размер запроса.

Параметр request_timeout_ms указывает максимальное время ожидания ответа при отправке пакета в миллисекундах. При превышении данного времени - считается, что пакет не был доставлен.

Параметр security_protocol указывает используемый протокол для связи с серверами kafka. Возможные значения:

– PLAINTEXT. Используется по умолчанию;

– SSL;
– SASL_PLAINTEXT;
– SASL_SSL.

Для того, чтобы создать экземпляр класса, следует использовать статический метод parse_from_lines(), который принимает список строк с конфигурацией kafka producer из файлов конфигурации.

3.1.4 Класс KafkaConsumerConfig

Данный класс представляет из себя конфигурацию для работы с платформой kafka. Сам объект класса создаётся из списка строк, которые представляют собой строки файла конфигурации, ответственные за настройку kafka consumer. Конфигурация содержит в себе следующие поля:

– bootstrap_servers;
– client_id;
– group_id;
– key_deserializer;
– value_deserializer;
– auto_offset_reset;
– enable_auto_commit;
– auto_commit_interval_ms;
– session_timeout_ms;
– security_protocol.

Параметр bootstrap_servers указывает адреса kafka серверов в формате «host[:port]».

Параметр client_id указывает уникальный идентификатор клиента. Данный идентификатор используется для логирования.

Параметр group_id указывает уникальный идентификатор группы получателей. Получатели с одинаковым идентификатором группы будут иметь одинаковое смещение, что позволит считывать одинаковые данные для всех получателей в группе.

Параметр key_deserializer указывает имя класса десериализатора для ключа. Данный параметр необходим для возможности передачи пользовательских объектов в качестве ключа.

Аналогичным является параметр value_deserializer, который указывает имя класса десериализатора для значения. Этот параметр необходим для возможности передачи пользовательских объектов в качестве значения. Также kafka предоставляет набор стандартных классов для сериализации и десериализации. Например org.apache.kafka.common.serialization.StringSerializer и org.apache.kafka.common.serialization.StringDeserializer.

Параметр auto_offset_reset указывает правила для сброса

смещения для получателя при переподключении. Может принимать следующие значения:

– `earliest`. При переподключении данные будут браться с самого начала раздела;

– `latest`. Используется по умолчанию. При переподключении данные будут браться с последнего сохранённого смещения.

Параметр `enable_auto_commit` указывает на возможность использования автоматического сохранения используемого смещения. Если данный параметр включен, то будет происходить автоматическое сохранение текущего смещения в разделе с определёнnным интервалом. По умолчанию включен.

Параметр `auto_commit_interval_ms` указывает интервал в миллисекундах, с которым будет происходить сохранение текущего смещения в разделе. По умолчанию установлен в 5000 миллисекунд.

Параметр `session_timeout_ms` указывает максимальное время ожидания специального сигнала, который отсылается бокером получателю. При превышении данного времени - считается, что брокер не доступен. По умолчанию установлен в 10000 миллисекунд.

Параметр `security_protocol` указывает используемый протокол для связи с серверами kafka. Возможные значения:

– `PLAINTEXT`. Используется по умолчанию;

– `SSL`;

– `SASL_PLAINTEXT`;

– `SASL_SSL`.

Для того, чтобы создать экземпляр класса, следует использовать статический метод `parse_from_lines()`, который принимает список строк с конфигурацией kafka consumer из файлов конфигурации.

3.2 Классы модуля хранения загрузчика

Для сохранения данных может использоваться как локальная файловая система, так и распределённая (HDFS). Для обеспечения работы с файловой системой, необходимо реализовать базовые операции, которые указаны в интерфейсе `FileSystemAdapter`. Его реализация `DefaultFileSystem` предназначена для произведения операций с помощью локальной файловой системы. А реализация `DistributedFileSystem` предназначена для произведения операция с помощью распределённой файловой системы (HDFS).

3.2.1 Интерфейс `FileSystemAdapter`

Предназначен для осуществления операций с файловой системой. Содержит следующие методы:

- `write_file()`;
- `append_to_file()`;
- `read_file()`;
- `remove_file()`;
- `mkdir()`;
- `ls()`;
- `is_exist()`;
- `to_file_path()`.

Метод `write_file()` принимает относительный путь в файловой системе и массив байтов. Предназначен для записи переданных байтов в файл. Если переданный файл не существует, то он создаётся. Если переданный файл уже существует, то он будет перезаписан. В результате ничего не возвращает.

Метод `append_to_file()` принимает относительный путь в файловой системе и массив байтов. Предназначен для добавления байтов в файл. Если переданный файл не существует, то он будет создан В результате ничего не возвращает.

Метод `read_file()` принимает относительный путь в файловой системе. Предназначен для чтения байтов из файла. Если файл не существует, то возвращает `null`. В результате возвращает массив считанных байт из файла.

Метод `remove_file()` принимает относительный путь в файловой системе. Предназначен для удаления файла. В результате возвращает `true` в случае успеха и `false` если файл не существовал.

Метод `mkdir()` принимает относительный путь в файловой системе. Предназначен для создания директории. В результате ничего не возвращает.

Метод `ls()` принимает относительный путь в файловой системе. Предназначен для получения всех существующих файлов в директории. В результате возвращает список файлов, которые находятся в переданной директории. В случае, если такой директории не существует, то возвращает `null`.

Метод `is_exist()` принимает относительный путь в файловой системе. Предназначен для проверки существования пути в файловой системе. В результате возвращает `true`, если путь существует и `false`, если путь не существует.

Метод `to_file_path()` принимает относительный путь в файловой системе к директории, географическую широту, географическую долготу, год. Предназначен для получения пути к файлу, который содержит результат

запроса с переданными параметрами. В результате возвращает путь к файлу.

3.2.2 Класс `DefaultFileSystem`

Предназначен для осуществления операций с локальной файловой системой. Имплементирует следующие методы:

- `write_file()`;
- `append_to_file()`;
- `read_file()`;
- `remove_file()`;
- `mkdir()`;
- `ls()`;
- `is_exist()`;
- `to_file_path()`.

Метод `write_file()` принимает относительный путь в локальной файловой системе и массив байтов. Предназначен для записи переданных байтов в файл. Если переданный файл не существует, то он создаётся. Если переданный файл уже существует, то он будет перезаписан. В результате ничего не возвращает.

Метод `append_to_file()` принимает относительный путь в локальной файловой системе и массив байтов. Предназначен для добавления байтов в файл. Если переданный файл не существует, то он будет создан В результате ничего не возвращает.

Метод `read_file()` принимает относительный путь в локальной файловой системе. Предназначен для чтения байтов из файла. Если файл не существует, то возвращает `null`. В результате возвращает массив считанных байт из файла.

Метод `remove_file()` принимает относительный путь в локальной файловой системе. Предназначен для удаления файла. В результате возвращает `true` в случае успеха и `false` если файл не существовал.

Метод `mkdir()` принимает относительный путь в локальной файловой системе. Предназначен для создания директории. В результате ничего не возвращает.

Метод `ls()` принимает относительный путь в локальной файловой системе. Предназначен для получения всех существующих файлов в директории. В результате возвращает список файлов, которые находятся в переданной директории. В случае, если такой директории не существует, то возвращает `null`.

Метод `is_exist()` принимает относительный путь в локальной файловой системе. Предназначен для проверки существования пути в локальной файловой системе. В результате возвращает `true`, если путь существует и `false`, если путь не существует.

Метод `to_file_path()` принимает относительный путь в локальной файловой системе к директории, географическую широту, географическую долготу, год. Предназначен для получения пути к файлу, который содержит результат запроса с переданными параметрами. В результате возвращает путь к файлу.

3.2.3 Класс `DistributedFileSystem`

Предназначен для осуществления операций с распределённой файловой системой (HDFS). Имплементирует следующие методы:

- `write_file()`;
- `append_to_file()`;
- `read_file()`;
- `remove_file()`;
- `mkdir()`;
- `ls()`;
- `is_exist()`;
- `to_file_path()`.

Метод `write_file()` принимает относительный путь в распределённой файловой системе (HDFS) и массив байтов. Предназначен для записи переданных байтов в файл. Если переданный файл не существует, то он создаётся. Если переданный файл уже существует, то он будет перезаписан. В результате ничего не возвращает.

Метод `append_to_file()` принимает относительный путь в распределённой файловой системе (HDFS) и массив байтов. Предназначен для добавления байтов в файл. Если переданный файл не существует, то он будет создан В результате ничего не возвращает.

Метод `read_file()` принимает относительный путь в распределённой файловой системе (HDFS). Предназначен для чтения байтов из файла. Если файл не существует, то возвращает `null`. В результате возвращает массив считанных байт из файла.

Метод `remove_file()` принимает относительный путь в распределённой файловой системе (HDFS). Предназначен для удаления файла. В результате возвращает `true` в случае успеха и `false` если файл не существовал.

Метод `mkdir()` принимает относительный путь в распределённой файловой системе (HDFS). Предназначен для создания директории. В результате ничего не возвращает.

Метод `ls()` принимает относительный путь в распределённой файловой системе (HDFS). Предназначен для получения всех существующих файлов в директории. В результате возвращает список файлов, которые находятся в переданной директории. В случае, если такой

директории не существует, то возвращает `null`.

Метод `is_exist()` принимает относительный путь в распределённой файловой системе (HDFS). Предназначен для проверки существования пути в распределённой файловой системе. В результате возвращает `true`, если путь существует и `false`, если путь не существует.

Метод `to_file_path()` принимает относительный путь в распределённой файловой системе к директории, географическую широту, географическую долготу, год. Предназначен для получения пути к файлу, который содержит результат запроса с переданными параметрами. В результате возвращает путь к файлу.

В качестве аргумента в конструкторе принимает экземпляр класса `FSConfig`, в котором располагаются необходимые параметры для работы с файловой системой. Корневая директория, в которой будут располагаться все файлы проекта, получается из конфигурации с помощью метода `get_dir`. Адрес хоста получается из конфигурации с помощью метода `get_host()`.

3.3 Классы модуля конфигурации загрузчика

Для получения данных из сервиса OpenWeatherMap необходимо использовать HTTP запросы. Структура запросов различаются для каждого типа API, которые используются для получения разных индексов загрязнения.

3.3.1 Класс `PollutionDumper`

Данный класс используется для получения данных из внешнего сервиса OpenWeatherMap. Для подключения к нему, необходимы следующие параметры:

– `api_key`. Ключ для получения доступа к api сервиса;
– `host`. Адрес самого сервиса.

Все эти параметры хранятся в файлах конфигурации приложения. После загрузки конфигурации, создаётся экземпляр класса `APIConfig`, который как раз и содержит необходимые для доступа параметры. При помощи методов `get_api_key()` и `get_host()` этого класса возможно получить необходимые параметры: ключ для api и адрес используемого сервиса соответственно. Экземпляр данного класса используется в конструкторе класса `PollutionDumper`.

Также для пропуска уже полученных данных, необходим инструмент работы с файловой системой. В качестве данного инструмента, подойдёт интерфейс `FileSystemAdapter`. В зависимости от конфигурации, мы сможем работать либо с локальной файловой системой с помощью

экземпляра класса `DefaultFileSystem`, либо с распределённой файловой системой (HDFS) при помощи экземпляра класса `DistributedFileSystem`. Благодаря данному объекту, мы сможем использовать один и тот же интерфейс, не завися от того, как развёрнуто приложение - в локальном режиме или с использованием кластера Hadoop.

Рассматриваемый класс имеет следующие методы:

– `dump()`;
– `to_address()`.

Метод `dump()` принимает географическую широту, географическую долготу и год. Сначала метод с помощью используемого адаптера файловой системы ищет в ней файл, в котором находятся данные с переданными параметрами. В случае, если такой файл существует, то возвращает данные из этого файла. Если такой файл не был найден, то происходит обращение к внешнему сервису OpenWeatherMap для получения данных по переданным географическим координатам и переданному году. Для их получения составляется HTTP запрос с переданными аргументами и отправляется на сервер. В случае, если такие данные есть на сервере, то полученные данные будут записаны в соответствующий файл в используемой файловой системе. Если же таких данных нет и сервер вернул ошибку, то будет выброшено исключение, которое далее будет залогировано вызываемым классом. Возвращает найденные или полученные с сервиса данные.

Метод `to_address()` принимает географическую широту, географическую долготу и год. Формирует адрес HTTP запроса для получения данных по конкретному индексу. Так как адреса запросов разные у каждого индекса, то данный метод будет переопределяться у каждого конкретного дочернего класса. Возвращает адрес для HTTP запроса.

3.3.2 Класс `NODumper`

Данный класс используется для получения индекса загрязнения диоксидом азота (NO_2) из внешнего сервиса OpenWeatherMap. Для подключения к нему, необходимы следующие параметры:

– `api_key`. Ключ для получения доступа к api сервиса;
– `host`. Адрес самого сервиса.

Все эти параметры хранятся в файлах конфигурации приложения. После загрузки конфигурации, создаётся экземпляр класса `APIConfig`, который как раз и содержит необходимые для доступа параметры. При помощи методов `get_api_key()` и `get_host()` этого класса возможно получить необходимые параметры: ключ для api и адрес используемого сервиса соответственно. Экземпляр данного класса используется в конструкторе родительского класса `PollutionDumper`.

Также для пропуска уже полученных данных, необходим инструмент

работы с файловой системой. В качестве данного инструмента, подойдёт интерфейс `FileSystemAdapter`. В зависимости от конфигурации, мы сможем работать либо с локальной файловой системой с помощью экземпляра класса `DefaultFileSystem`, либо с распределённой файловой системой (HDFS) при помощи экземпляра класса `DistributedFileSystem`. Благодаря данному объекту, мы сможем использовать один и тот же интерфейс, не завися от того, как развёрнуто приложение - в локальном режиме или с использованием кластера Hadoop.

Рассматриваемый класс имеет следующие методы:

– `dump()`;
– `to_address()`.

Метод `dump()` принимает географическую широту, географическую долготу и год. Сначала метод с помощью используемого адаптера файловой системы ищет в ней файл, в котором находятся данные с переданными параметрами. В случае, если такой файл существует, то возвращает данные из этого файла. Если такой файл не был найден, то происходит обращение к внешнему сервису OpenWeatherMap для получения данных по переданным географическим координатам и переданному году. Для их получения составляется HTTP запрос с переданными аргументами и отправляется на сервер. В случае, если такие данные есть на сервере, то полученные данные будут записаны в соответствующий файл в используемой файловой системе. Если же таких данных нет и сервер вернул ошибку, то будет выброшено исключение, которое далее будет залогировано вызываемым классом. Возвращает найденный или полученный с сервиса индекс загрязнения диоксидом азота.

Метод `to_address()` принимает географическую широту, географическую долготу и год. Формирует адрес HTTP запроса для получения индекса загрязнения диоксидом азота. Возвращает адрес для HTTP запроса.

3.3.3 Класс `SODumper`

Данный класс используется для получения индекса загрязнения диоксидом серы (SO_2) из внешнего сервиса OpenWeatherMap. Для подключения к нему, необходимы следующие параметры:

– `api_key`. Ключ для получения доступа к api сервиса;
– `host`. Адрес самого сервиса.

Все эти параметры хранятся в файлах конфигурации приложения. После загрузки конфигурации, создаётся экземпляр класса `APIConfig`, который как раз и содержит необходимые для доступа параметры. При помощи методов `get_api_key()` и `get_host()` этого класса возможно получить необходимые параметры: ключ для api и адрес используемого

сервиса соответственно. Экземпляр данного класса используется в конструкторе родительского класса `PollutionDumper`.

Также для пропуска уже полученных данных, необходим инструмент работы с файловой системой. В качестве данного инструмента, подойдёт интерфейс `FileSystemAdapter`. В зависимости от конфигурации, мы сможем работать либо с локальной файловой системой с помощью экземпляра класса `DefaultFileSystem`, либо с распределённой файловой системой (HDFS) при помощи экземпляра класса `DistributedFileSystem`. Благодаря данному объекту, мы сможем использовать один и тот же интерфейс, не завися от того, как развёрнуто приложение - в локальном режиме или с использованием кластера Hadoop.

Рассматриваемый класс имеет следующие методы:

– `dump()`;
– `to_address()`.

Метод `dump()` принимает географическую широту, географическую долготу и год. Сначала метод с помощью используемого адаптера файловой системы ищет в ней файл, в котором находятся данные с переданными параметрами. В случае, если такой файл существует, то возвращает данные из этого файла. Если такой файл не был найден, то происходит обращение к внешнему сервису OpenWeatherMap для получения данных по переданным географическим координатам и переданному году. Для их получения составляется HTTP запрос с переданными аргументами и отправляется на сервер. В случае, если такие данные есть на сервере, то полученные данные будут записаны в соответствующий файл в используемой файловой системе. Если же таких данных нет и сервер вернул ошибку, то будет выброшено исключение, которое далее будет залогировано вызываемым классом. Возвращает найденный или полученный с сервиса индекс загрязнения диоксидом серы.

Метод `to_address()` принимает географическую широту, географическую долготу и год. Формирует адрес HTTP запроса для получения индекса загрязнения диоксидом серы. Возвращает адрес для HTTP запроса.

3.3.4 Класс `OZDumper`

Данный класс используется для получения индекса загрязнения озоном (O_3) из внешнего сервиса OpenWeatherMap. Для подключения к нему, необходимы следующие параметры:

– `api_key`. Ключ для получения доступа к api сервиса;
– `host`. Адрес самого сервиса.

Все эти параметры хранятся в файлах конфигурации приложения. После загрузки конфигурации, создаётся экземпляр класса `APIConfig`,

который как раз и содержит необходимые для доступа параметры. При помощи методов `get_api_key()` и `get_host()` этого класса возможно получить необходимые параметры: ключ для api и адрес используемого сервиса соответственно. Экземпляр данного класса используется в конструкторе родительского класса `PollutionDumper`.

Также для пропуска уже полученных данных, необходим инструмент работы с файловой системой. В качестве данного инструмента, подойдёт интерфейс `FileSystemAdapter`. В зависимости от конфигурации, мы сможем работать либо с локальной файловой системой с помощью экземпляра класса `DefaultFileSystem`, либо с распределённой файловой системой (HDFS) при помощи экземпляра класса `DistributedFileSystem`. Благодаря данному объекту, мы сможем использовать один и тот же интерфейс, не завися от того, как развёрнуто приложение - в локальном режиме или с использованием кластера Hadoop.

Рассматриваемый класс имеет следующие методы:

– `dump()`;
– `to_address()`.

Метод `dump()` принимает географическую широту, географическую долготу и год. Сначала метод с помощью используемого адаптера файловой системы ищет в ней файл, в котором находятся данные с переданными параметрами. В случае, если такой файл существует, то возвращает данные из этого файла. Если такой файл не был найден, то происходит обращение к внешнему сервису OpenWeatherMap для получения данных по переданным географическим координатам и переданному году. Для их получения составляется HTTP запрос с переданными аргументами и отправляется на сервер. В случае, если такие данные есть на сервере, то полученные данные будут записаны в соответствующий файл в используемой файловой системе. Если же таких данных нет и сервер вернул ошибку, то будет выброшено исключение, которое далее будет залогировано вызываемым классом. Возвращает найденный или полученный с сервиса индекс загрязнения озоном.

Метод `to_address()` принимает географическую широту, географическую долготу и год. Формирует адрес HTTP запроса для получения индекса загрязнения озоном. Возвращает адрес для HTTP запроса.

3.3.5 Класс `CODumper`

Данный класс используется для получения индекса загрязнения монооксидом углерода (CO) из внешнего сервиса OpenWeatherMap. Для подключения к нему, необходимы следующие параметры:

– `api_key`. Ключ для получения доступа к api сервиса;

– `host`. Адрес самого сервиса.

Все эти параметры хранятся в файлах конфигурации приложения. После загрузки конфигурации, создаётся экземпляр класса `APIConfig`, который как раз и содержит необходимые для доступа параметры. При помощи методов `get_api_key()` и `get_host()` этого класса возможно получить необходимые параметры: ключ для api и адрес используемого сервиса соответственно. Экземпляр данного класса используется в конструкторе родительского класса `PollutionDumper`.

Также для пропуска уже полученных данных, необходим инструмент работы с файловой системой. В качестве данного инструмента, подойдёт интерфейс `FileSystemAdapter`. В зависимости от конфигурации, мы сможем работать либо с локальной файловой системой с помощью экземпляра класса `DefaultFileSystem`, либо с распределённой файловой системой (HDFS) при помощи экземпляра класса `DistributedFileSystem`. Благодаря данному объекту, мы сможем использовать один и тот же интерфейс, не завися от того, как развёрнуто приложение - в локальном режиме или с использованием кластера Hadoop.

Рассматриваемый класс имеет следующие методы:

– `dump()`;

– `to_address()`.

Метод `dump()` принимает географическую широту, географическую долготу и год. Сначала метод с помощью используемого адаптера файловой системы ищет в ней файл, в котором находятся данные с переданными параметрами. В случае, если такой файл существует, то возвращает данные из этого файла. Если такой файл не был найден, то происходит обращение к внешнему сервису OpenWeatherMap для получения данных по переданным географическим координатам и переданному году. Для их получения составляется HTTP запрос с переданными аргументами и отправляется на сервер. В случае, если такие данные есть на сервере, то полученные данные будут записаны в соответствующий файл в используемой файловой системе. Если же таких данных нет и сервер вернул ошибку, то будет выброшено исключение, которое далее будет залогировано вызываемым классом. Возвращает найденный или полученный с сервиса индекс загрязнения монооксидом углерода.

Метод `to_address()` принимает географическую широту, географическую долготу и год. Формирует адрес HTTP запроса для получения индекса загрязнения монооксидом углерода. Возвращает адрес для HTTP запроса.

3.4 Классы модуля обработки загрузчика

От сервера приходят данные в формате JavaScript Object Notation (JSON). Такие данные не подвергаются анализу, поэтому их необходимо обработать и получить значения, которые могут быть использованы в последующих операциях. Для обработки полученных данных как раз и используются рассматриваемые ниже классы.

3.4.1 Класс `DumpParser`

Данный класс используется для обработки данных, полученных от сервера OpenWeatherMap. Так как каждый API предоставляет данные в своём формате, то и для каждого API необходим свой обработчик. Обработчиками являются экземпляры класса `JsonParser`. При создании экземпляра данного класса, в конструктор передаются экземпляры, имплементирующие интерфейс `JsonParser`. Так как для каждого API устанавливается свой обработчик, то и поля соответсвуют используемым API:

– `co_parser`. Соответсвует обработчику полученных данных об индексе загрязнения монооксидом углерода (CO);

– `so_parser`. Соответсвует обработчику полученных данных об индексе загрязнения диоксидом серы (SO_2);

– `oz_parser`. Соответсвует обработчику полученных данных об индексе загрязнения озоном (O_3);

– `no_parser`. Соответсвует обработчику полученных данных об индексе загрязнения диоксидом азота (NO_2).

Метод `parse_co()` используется для преобразования полученных данных по API, предоставляющего данные о индексе загрязнения монооксидом углерода (CO) в формате JSON. Принимает данные в формате JSON. Возвращает список значений с критериями, которые были указаны в используемом обработчике данных для моонооксида углерода (CO).

Метод `parse_so()` используется для преобразования полученных данных по API, предоставляющего данные о индексе загрязнения диоксидом серы (SO_2) в формате JSON. Принимает данные в формате JSON. Возвращает список значений с критериями, которые были указаны в используемом обработчике данных для диоксида серы (SO_2).

Метод `parse_oz()` используется для преобразования полученных данных по API, предоставляющего данные о индексе загрязнения озоном (O_3) в формате JSON. Принимает данные в формате JSON. Возвращает список значений с критериями, которые были указаны в используемом обработчике данных для озона (O_3).

Метод `parse_no()` используется для преобразования полученных

данных по API, предоставляющего данные о индексе загрязнения диоксидом азота (NO_2) в формате JSON. Принимает данные в формате JSON. Возвращает список значений с критериями, которые были указаны в используемом обработчике данных для диоксида азота (NO_2).

3.4.2 Интерфейс `JsonParser`

Для обработки данных в формате JSON используется отдельный интерфейс `JsonParser`. Так как структура предоставляемых данных может измениться в будущем, то необходимо предоставить возможность заменить используемые обработчики на другие, для поддержания нового формата данных. Благодаря такому подходу, можно использовать различные источники данных. Сам интерфейс содержит только один метод `parse()`.

Метод `parse()` используется для преобразования полученных данных в значения, которые можно проанализировать. На вход принимает данные в формает JSON. Возвращает список полученных значений.

3.4.3 Класс `NOParser`

Рассматриваемый класс используется для обработки данных об индексе загрязнения диоксидом азота в формате JSON. Так как структура предоставляемых данных может измениться в будущем, то необходимо предоставить возможность заменить данный обработчик на другой, для поддержания нового формата данных. Благодаря такому подходу, можно использовать различные источники данных.

Так как используемый формат предоставляет множество значений для разного давления, то необходимо ограничить значения давления, для которого и будут отбираться данные. Для этих целей используются поля, которые передаются в конструктор:

– `min_pressure`;

– `max_pressure`.

Поле `min_pressure` задаёт минимальную границу давления, для которого будут отобраны значения. Поле `max_pressure` задаёт максимальную границу давления, для которого будут отобраны значения. Оба этих параметра передаются в контруктор класса `NOParser`.

Метод `parse()` используется для преобразования полученных данных в значения, которые можно проанализировать. На вход принимает данные об индексе загрязнения диоксидом азота в формате JSON. Возвращает список полученных значений индекса загрязнения диоксидом азота.

3.4.4 Класс `SOParser`

Рассматриваемый класс используется для обработки данных об индексе загрязнения диоксидом серы в формате JSON. Так как структура предоставляемых данных может измениться в будущем, то необходимо предоставить возможность заменить данный обработчик на другой, для поддержания нового формата данных. Благодаря такому подходу, можно использовать различные источники данных.

Так как используемый формат предоставляет множество значений для разного давления, то необходимо ограничить значения давления, для которого и будут отбираться данные. Для этих целей используются поля, которые передаются в конструктор:

– `min_pressure`;
– `max_pressure`.

Поле `min_pressure` задаёт минимальную границу давления, для которого будут отобраны значения. Поле `max_pressure` задаёт максимальную границу давления, для которого будут отобраны значения. Оба этих параметра передаются в контруктор класса `SOParser`.

Метод `parse()` используется для преобразования полученных данных в значения, которые можно проанализировать. На вход принимает данные об индексе загрязнения диоксидом серы в формате JSON. Возвращает список полученных значений индекса загрязнения диоксидом серы.

3.4.5 Класс `OZParser`

Рассматриваемый класс используется для обработки данных об индексе загрязнения озоном в формате JSON. Так как структура предоставляемых данных может измениться в будущем, то необходимо предоставить возможность заменить данный обработчик на другой, для поддержания нового формата данных. Благодаря такому подходу, можно использовать различные источники данных.

Так как используемый формат предоставляет множество значений для разного давления, то необходимо ограничить значения давления, для которого и будут отбираться данные. Для этих целей используются поля, которые передаются в конструктор:

– `min_pressure`;
– `max_pressure`.

Поле `min_pressure` задаёт минимальную границу давления, для которого будут отобраны значения. Поле `max_pressure` задаёт максимальную границу давления, для которого будут отобраны значения. Оба этих параметра передаются в контруктор класса `OZParser`.

Метод `parse()` используется для преобразования полученных данных

в значения, которые можно проанализировать. На вход принимает данные об индексе загрязнения озоном в формате JSON. Возвращает список полученных значений индекса загрязнения озоном.

3.4.6 Класс `COParser`

Рассматриваемый класс используется для обработки данных об индексе загрязнения диоксидом углерода в формате JSON. Так как структура предоставляемых данных может измениться в будущем, то необходимо предоставить возможность заменить данный обработчик на другой, для поддержания нового формата данных. Благодаря такому подходу, можно использовать различные источники данных.

Так как используемый формат предоставляет множество значений для разного давления, то необходимо ограничить значения давления, для которого и будут отбираться данные. Для этих целей используются поля, которые передаются в конструктор:

– `min_pressure`;
– `max_pressure`.

Поле `min_pressure` задаёт минимальную границу давления, для которого будут отобраны значения. Поле `max_pressure` задаёт максимальную границу давления, для которого будут отобраны значения. Оба этих параметра передаются в контруктор класса `COParser`.

Метод `parse()` используется для преобразования полученных данных в значения, которые можно проанализировать. На вход принимает данные об индексе загрязнения монооксидом углерода в формает JSON. Возвращает список полученных значений индекса загрязнения монооксидом углерода.

3.5 Классы модуля отправки загрузчика

После обработки данных, они отправляются в соответсвующий топик в платформе kafka. В качестве ключа сообщения используется преобразованная строка, которая содержит информацию о типе api, географических координатах и годе. Для отправлении сообщений используется класс `KafkaAdapter`.

3.5.1 Класс `KafkaAdapter`

Данный класс используется для взаимодействия с платформой kafka. В частности, используется для отправки сообщений. Для каждого типа даных используется свой топик. Такой способ позволяет производить обработку конкретного типа данных.

Для обеспечения работы с платформой, необходимо произвести конфгурацию отправителя. За решение этой задачи ответственнен

экземпляр класса `KafkaProducerConfig`. Этот класс как раз хранит все необходимые параметры конфигурации, которые необходимы для осуществления взаимодействия с упомянутой платформой. Используемые при конфигурации параметры:

– `bootstrap_servers`. Указывает адреса kafka серверов в формате «host[:port]»;

– `client_id`. Указывает уникальный идентификатор клиента в платформе kafka;

– `key_serializer`. Указывает имя класса сериализатора для ключа в сообщении;

– `value_serializer`. Указывает имя класса сериализатора для значения в сообщении;

– `acks`. Указывает требуемое количество подтверждений от kafka серверов;

– `compression_type`. Указывет тип используемого сжатия;

– `retries`. Указывает количество попыток отправить данные заново при неудаче;

– `batch_size`. Указывает максимальный размер пакета для отправки;

– `max_request_size`. Указывает максимальный размер запроса;

– `request_timeout_ms`. Указывает максимальное время ожидания ответа при отправке пакета в миллисекундах;

– `security_protocol`. Указывает используемый протокол для связи с брокерами kafka.

После настройки можно отправлять сообщения. Для этого используется метод `send_message()`. Метод принимает топик, в который будет отправляться сообщение, само сообщение, и ключ. Ключ из себя представляет преобразованную строку, которая содержит информацию о типе api, географических координатах и годе. Такой способ используется для того, чтобы при получении сообщения, можно было узнать, от куда были получены данные. Если указанные брокеры недоступны, либо отправка не осуществилась по другой причине, то будет выброшено исключение, которое будет обработано вызывающим функционалом.

3.6 Классы модуля приёма обработчика

Для анализа данных необходимо их сначала получить, и перенаправить в анализирующую систему. Для этого как раз используется kafka приёмник. Его роль исполняет класс `KafkaConsumerAdapter`.

Так как для каждого типа даных используется свой топик, то можно осуществить подписку только на те топики, где располагаются интересующие данные. Для потоковой обработки данных, в kafka применяется метод подписки. Получатель может подписаться на

интересующие его топики, и тем самым, получать актуальную информацию, как только данные поступят в платформу. Такая обработка осуществляется пакетами, которые берутся из значений, которые поступили в систему в определённое временное окно. Так как существует задержка между тем моментом, когда сообщение поступило в систему и тем, когда это сообщение будет получено, то применяется понятие водяного знака (англ. watermark). В kafka каждое сообщение имеет также временной штамп (англ. timestamp), который соответствует времени создания сообщения. Водяной знак позволяет отсеивать сообщения, которые пришли позже установленного времени. При чём, если временной штамп сообщения походит под текущее окно, то оно попадает в текущий пакет. Apache Spark позволяет описать процесс обработки всего потока. Все описанные операции будут применяться к каждому полученному пакету за установленное окно.

3.6.1 Класс `KafkaConsumerAdapter`

Данный класс используется для взаимодействия с платформой kafka для считывания из неё данных. Для обеспечения работы с платформой, необходимо произвести конфгурацию приёмника. За решение этой задачи ответственнен экземпляр класса `KafkaConsumerConfig`. Этот класс как раз хранит все необходимые параметры конфигурации, которые необходимы для осуществления взаимодействия с упомянутой платформой. Используемые при конфигурации параметры:

– `bootstrap_servers`. Указывает адреса kafka серверов в формате «host[:port]»;

– `client_id`. Указывает уникальный идентификатор клиента в платформе kafka;

– `group_id`. Указывает уникальный идентификатор группы получателей в платформе kafka;

– `key_deserializer`. Указывает имя класса десериализатора для ключа в сообщении;

– `value_deserializer`. Указывает имя класса десериализатора для значения в сообщении;

– `auto_offset_reset`. Указывает правило для сбрасывания смещения при переподключении;

– `enable_auto_commit`. Указывает возможность использования автоматического сохранения текущего смещения в разделе;

– `auto_commit_interval_ms`. Указывает интервал, с которым происходит автоматическое сохранение смещения;

– `session_timeout_ms`. Указывает максимальное время ожидания специального сигнала, отсылаемого брокером получателю;

– `security_protocol`. Указывает используемый протокол для связи с брокерами kafka.

Все эти настройки предоставляются с помощью экземпляра класса `KafkaConsumerConfig`. Если указанные брокеры недоступны, либо получение не осуществилась по другой причине, то будет выброшено исключение, которое будет обработано вызывающим функционалом. Метод `create_steam()` используется для создания потока, который будет обрабатываься дальше с помощью Apache Spark.

3.7 Классы модуля анализа обработчика

Модуль анализа используеся для обработки значений, которые поступают из системы kafka. Сама обработка производится с помощью потоков в Apache Spark.

Поток представляет из себя постоянно появляющиеся пакеты данных, которые представлены в виде Resilent Distributed Dataset (RDD). Данные объекты из себя представляют абстракции распределённых данных. Сами данные создаются и обрабатываются непосредственно на этапе исполнения.

3.7.1 Класс `StreamPerformer`

Данный класс используется для обработки потока RDD. Также этот класс обязательно должен быть сериализуемым, так как этот класс используется для обработки. Это значит, что объект этого класса будет сериализован и отправлен на используемые рабочие машины, которые использует Apache Spark. На самих машинах этот объект десериализуется и может быть использован для обработки данных.

Метод `processStream()` используется для обработки всего потока. Операции будут применены к каждому RDD в потоке. И так это будет продолжаться, пока процесс не будет завершён извне. На полученных данных разворачивается внутренняя таблица. Полученная таблица в дальнейшем будет использована для предоставления результатов.

Метод `process()` используется для обработки одного RDD.

3.7.2 Класс `COStreamPerformer`

Данный класс используется для обработки потока RDD с данными об индексе загрязнения монооксидом углерода. Также этот класс обязательно должен быть сериализуемым, так как этот класс используется для обработки.

Метод `processStream` используется для обработки всего потока. Операции будут применены к каждому RDD в потоке. И так это будет продолжаться, пока процесс не будет завершён извне. На полученных

данных разворачивается внутренняя таблица. Полученная таблица в дальнейшем будет использована для предоставления результатов.

Метод `process()` используется для обработки одного RDD.

3.7.3 Класс `SOStreamPerformer`

Данный класс используется для обработки потока RDD с данными об индексе загрязнения диоксидом серы. Также этот класс обязательно должен быть сериализуемым, так как этот класс используется для обработки.

Метод `processStream()` используется для обработки всего потока. Операции будут применены к каждому RDD в потоке. И так это будет продолжаться, пока процесс не будет завершён извне. На полученных данных разворачивается внутренняя таблица. Полученная таблица в дальнейшем будет использована для предоставления результатов.

Метод `process()` используется для обработки одного RDD.

3.7.4 Класс `NOStreamPerformer`

Данный класс используется для обработки потока RDD с данными об индексе загрязнения диоксидом азота. Также этот класс обязательно должен быть сериализуемым, так как этот класс используется для обработки.

Метод `processStream()` используется для обработки всего потока. Операции будут применены к каждому RDD в потоке. И так это будет продолжаться, пока процесс не будет завершён извне. На полученных данных разворачивается внутренняя таблица. Полученная таблица в дальнейшем будет использована для предоставления результатов.

Метод `process()` используется для обработки одного RDD.

3.7.5 Класс `OZStreamPerformer`

Данный класс используется для обработки потока RDD с данными об индексе загрязнения озоном. Также этот класс обязательно должен быть сериализуемым, так как этот класс используется для обработки.

Метод `processStream()` используется для обработки всего потока. Операции будут применены к каждому RDD в потоке. И так это будет продолжаться, пока процесс не будет завершён извне. На полученных данных разворачивается внутренняя таблица. Полученная таблица в дальнейшем будет использована для предоставления результатов.

Метод `process()` используется для обработки одного RDD.

3.8 Классы модуля сохранения обработчика

После обработки данные могут быть сохранены в разных форматах, которые поддерживает Apache Spark. Сохранение обработанных данных позволит развернуть полноценную таблицу на файле, и использовать уже готовые данные для последующего анализа.

Для сохранения используются несколько форматов, поддерживаемых Apache Spark:

– `orc`. Данный формат эффективен для хранения больших объёмов данных, так как использует сжатие. Также данный формат структурированный, что позволяет использовать пропуск полей (англ. file prunning). Это значит, что при чтении такого файла для определённого запроса, Apache Spark может пропускать не используемые в запросе поля, и не тратить время на их распаковку;

– `csv`. Данный формат эффективен для хранения данных в строгой структуре;

– `json`. Данный формат эффективен для хранения сложной структуры данных;

– `parquet`. Данный формат эффективен для файлов, которые часто подвергаются чтению, и редко подвергаются записи;

– `avro`. Данный формат эффективен для сложных данных с изменяемой структурой.

Также помимо простого сохранения в файл, Apache Spark позволяет использовать разбиение на разделы (англ. partition) и корзины (англ. bucketing). Разбиение по разделам означает то, что каждое уникальное значение для указаного поля будет представлять из себя отдельную директорию, в которой будут храняться данные с этим значением. Такой способ крайне удобен при сохранении данных с каким-то типом. В итоге, каждый тип буде размещён в отдельную директорию. Использование корзин, в свою очередь, является аналогом хэш-таблицы. Все значения разиваются на определённоё количество корзин с помощью нахождения хэш-функции. Тем самым, при попытке найти запись с определённым значением поля, которое было разбито на корзины - необходимо вычислить хэш сравниваемого значения, определить корзину, уоторая соответствует полученному значению, и искать нужную запись только в этой корзине.

3.8.1 Класс `DataSaver`

Данный класс используется для сохранения потока RDD. Для конфигурации используются следующие поля:

– `format`. Определяет формат, в котором будут сохранены данные;

– `path`. Указывает путь к файлу, в который будут сохранены данные;

– `mode`. Определяет режим записи;

– `partition`. Определяет поле, по которому будет производиться разделение;

– `bucketing`. Определяет поле, по которому будет производится разбиение на корзины;

– `bucket_count`. Определяет количество корзин, на которые будут разбиваться данные.

Все эти поля предоставляются с помощью экземпляра класса `SaverConfig`.

3.9 Модуль визуализации обработчика

Для представления результатов используется Apache Zeppelin. Этот модуль представляет из себя набор интерактивных документов, которые используются для предоставления в удобном виде полученных результатов. Данные получаются из таблиц, созданных с помощью Apache Spark на предыдущем этапе. Каждый API предоставляется отедльной таблицей, так что все интерактивные документы могут работать с определённой таблицой.

ЗАКЛЮЧЕНИЕ

В рамках данного дипломного проекта был разработан программный продукт для анализа данных с помощью произведения распределённых вычислений. Данный продукт может использоваться в двух режимах работы:

– локальный;
– распределённый.

При работе в локальном режиме, все компоненты системы разворачиваются на используемой машине, и позволяют производить вычисления посредством контейнеризации. Для получения данных использовался внешний сервис OpenWeatherMap, который предоставляет доступ к индексам загрязнений окружающей среды. Собранные данные послу обработки кэшируются и отправляются в используемый брокер сообщений.

Одновременно с этим, запускается интерактивный документ Zeppelin, который обрабатывает сообщения, принятые за конкретный промежуток времени. После обработки происходит регрессионный анализ обработанных данных и в итоге получаются предполагаемые индексы загрязнения окружающей среды на несколько лет вперёд для определённого географического положения.

Данный продукт может быть использован в качестве получения и обработки других данных в режиме реального времени. Наиболее яркий пример, это анализ произведённых транзакций в режиме реального времени.

Дипломный проект был разработан в полном объёме и выполняет возложенные на него функции. В дальнейшем он может быть улучшен следующим образом:

– поддержка других брокеров сообщений;
– поддержка Microsoft Azure Blob Storage;
– увеличение количества источников данных;
– увеличение количества обработчиков данных.

Таким образом, разработанный продукт может применяться как самостоятельный проект, либо в качестве фреймворка для распределённого анализа данных. Гибкая архитектура позволяет использовать данный проект для любого формата данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Apache Kafka Documentation [Электронный ресурс]. — Режим доступа : https://kafka.apache.org/intro. — Дата доступа : 18.03.2019.

[2] Apache ZooKeeper Documentation [Электронный ресурс]. — Режим доступа : https://zookeeper.apache.org/doc/current/zookeeperOver. — Дата доступа : 18.03.2019.

[3] Docker Documentation [Электронный ресурс]. — Режим доступа : https://docs.docker.com/get-started. — Дата доступа : 18.03.2019.

[4] Spark Documentation [Электронный ресурс]. — Режим доступа : https://spark.apache.org/docs/latest/. — Дата доступа : 18.03.2019.

[5] Apache spark architecture. Distributed system architecture explained [Электронный ресурс]. — Режим доступа : https://www.edureka.co/blog/spark-architecture/. — Дата доступа : 18.03.2019.

[6] White, Tom. Hadoop: The Definitive Guide / Tom White. — O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009. — Pp. 45 – 50.

[7] Data visualization with using Apache Zeppelin [Электронный ресурс]. — Режим доступа : https://scalegrid.io/blog/data-visualization-using-apache-zeppelin/. — Дата доступа : 18.03.2019.

[8] Jenkins user documentation [Электронный ресурс]. — Режим доступа : https://jenkins.io/doc/. — Дата доступа : 18.03.2019.

[9] Current weather and forecast — OpenWeatherMap [Электронный ресурс]. — Режим доступа : https://openweathermap.org/. — Дата доступа : 18.03.2019.

[10] kafka python 1.4.6 documentation [Электронный ресурс]. — Режим доступа : https://kafka-python.readthedocs.io/en/master/apidoc/. — Дата доступа : 18.03.2019.

ПРИЛОЖЕНИЕ А
(*обязательное)*

Схема структурная

ПРИЛОЖЕНИЕ Б
(*обязательное)*

Диаграмма классов