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

РЕФЕРАТ

Дипломный проект предоставлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата А1. Пояснительная записка: 80 страниц, 6 рисунков, 1 таблица, 11 литературных источников, 2 приложения.

Ключевые слова: регрессионный анализ, распределённая вычислительная система, кластер, виртуализация, открытый исходный код, интерактивный документ, брокер сообщений, обработка данных в режиме реального времени.

Предметная область: разработка программного продукта под платформу Linux. Объект разработки: программный продукта для проведения распределённого анализа данных из нескольких источников.

Целью проекта является разработка программного продукта для распределённого анализа данных под платформу Linux, который позволит производить анализ данных из нескольких источников на распределённой вычислительной системе.

При разработке программного продукта для распределённого анализа данных под платформу Linux использовался PyCharm, InteliJ IDEA, Apache Spark, Apache Zeppelin, Apache Kafka

С помощью распределённой вычислительной системы был разработан программный продукт, который позволяет производить параллельную обработку данных в режиме реального времени. Использование Docker позволяет использовать данный продукт на любой поддерживаемой платформе при помощи средств виртуализации. По причине использования брокера сообщений, проект является очень гибким и позволяет добавлять как неограниченное количество источников данных, так и неограниченное количество обработчиков данных.

Разработанный программный продукт, согласно приведённым расчётам является экономически эффективным, и оправдывает средства, вложенные в его разработку.

Дипломный проект является завершённым. В качестве улучшений можно рассмотреть добавление поддержки других брокеров сообщений, использование более компактного формата передачи сообщений и расширение используемых инструментов анализа. 80 страниц, 6 рисунков, 1 таблица, 11 литературный источник, 2 приложения.

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;

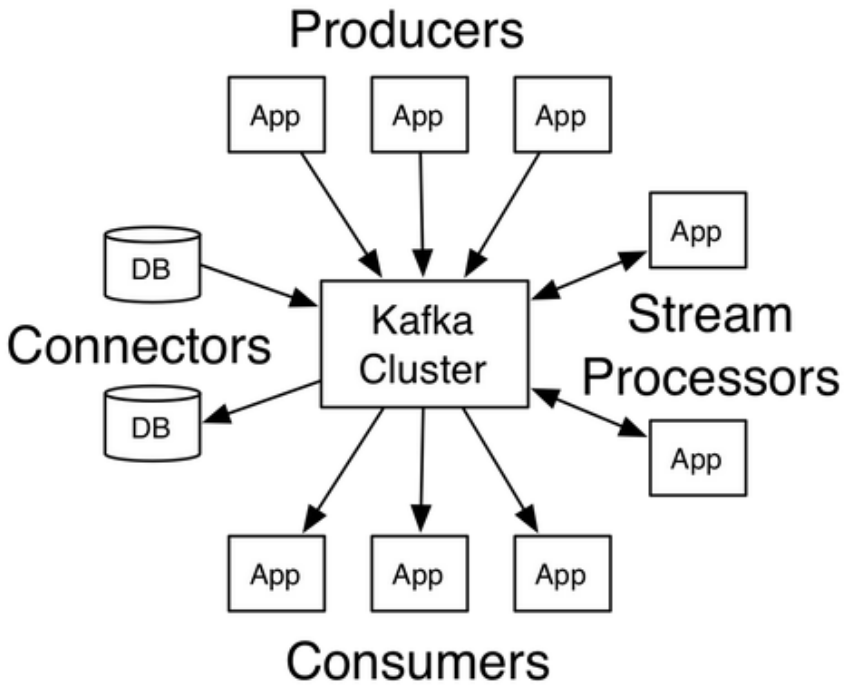

Рисунок 1.1 – Структура брокера сообщений Apache Kafka [1]

– 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 -

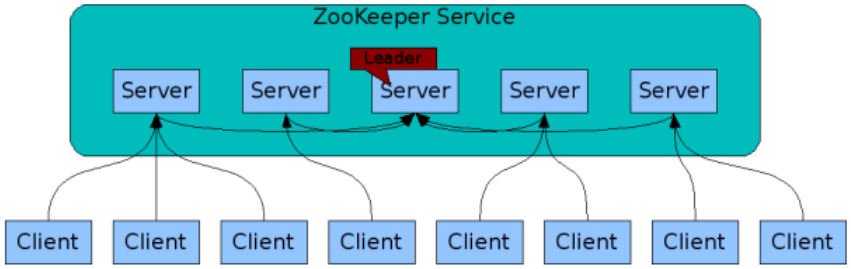

Рисунок 1.2 – Структура координирующего сервиса ZooKeeper [2]

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;

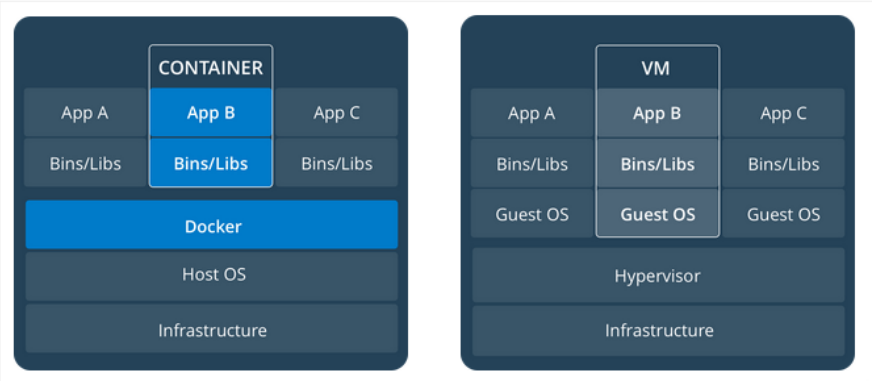

Рисунок 1.3 – Работа docker с обычной контейнеризацией и с использованием виртуальной машины [3]

– 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.

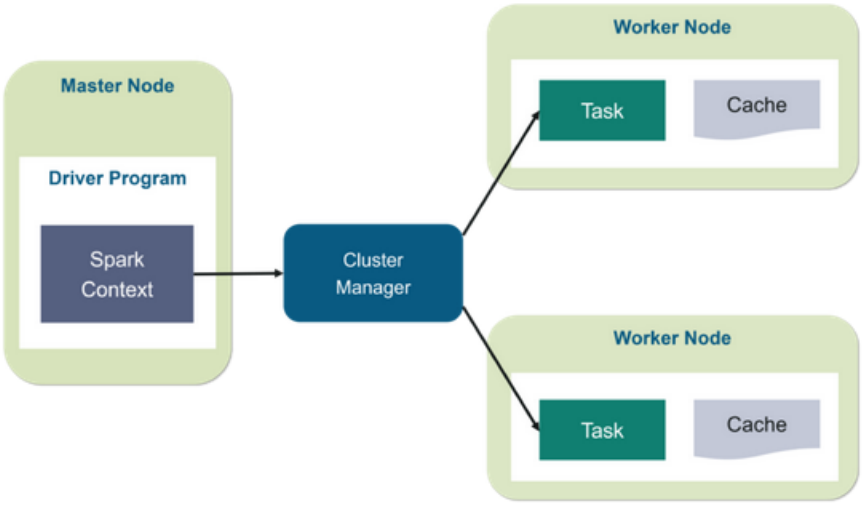

Рисунок 1.4 – Архитектура Apache Spark [5]

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 таблицей.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В прошлом разделе были описаны функции, имеющиеся в разрабатываемом программном обеспечении. В данном разделе будет описана разработка ключевых алгоритмов для данного дипломного проекта.

4.1 Алгоритм получения данных с помощью внешнего сервиса

В качестве источника данных может быть использовано как локальное хранилище, так и внешнее. Если данные не были обнаружены в локальном хранилище, то они должны быть получены из внешнего сервиса.

В качестве локального хранилища может выступать как распределённая файловая система, так и локальная файловая система. Для получения данных из подобного хранилища используется соответствующий адаптер:

– `DistributedFileSystem`;

– `DefaultFileSystem`.

Оба этих адаптера реализуют интерфейс `FileSystemAdapter`, который и используется для получения данных из локального хранилища. С помощью файла конфигурации определяется используемая файловая система, а также параметры, которые необходимы для указания директории, в которой будут расположены хранимые данные.

В хранилище данные хранятся при помощи разбиения на корзины, как это используется в Apache HIVE. Это означает, что данные за один год будут расположены в одной директории. Данный подход позволяет развернуть таблицу на этих данных с помощью Apache HIVE.

Перед обращением к сервису происходит проверка запрашиваемых значений в используемой файловой системе. Для этого запрашиваемые параметры преобразуются в строку с помощью метода `to_file_path`. Поиск осуществляется в директории с запрашиваемым годом и с названием файла, полученным после преобразования параметров в название. Если такой файл был найден, то данные из файла отправляются на дальнейшую обработку. В противном случае производится запрос к внешнему сервису.

Для получения данных из внешнего сервиса OpenWeatherMap используется HTTP протокол. Данные передаются в формате JSON. Для успешного подключения необходимы следующие параметры:

– API key. Специальный ключ, который предоставляется сервисом OpenWeatherMap при оформлении подписки;

– API Url. Адрес используемого API.

Для определения критериев выборки, на текущий момент используются следующие параметры:

– географическая широта;

– географическая долгота;
– диапазон времени.

При установке соединения, на стороне сервиса происходит проверка переданного ключа. Для используемого API на текущий момент существуют ограничения на количество запросов в промежуток времени. По этой причине сервер проверяет количество использований переданного ключа в течении установленного промежутка. Если текущий запрос укладывается в ограничение, то сервер отправляет запрошенные данные. В случае, если количество запросов было превышено, то соединение удерживается на стороне сервера до тех пор, пока не истечёт установленный промежуток времени.

В любом случае, данные будут получены, и переданы на дальнейший этап обработки. Так как данные будут получены в формате JSON, то их необходимо обработать, чтобы предоставить их в том формате, в котором они могут быть обработаны.

Полученные данные попадают в один из следующих методов в зависимости от используемого API:

– `parse_co`;
– `parse_so`;
– `parse_oz`;
– `parse_no`.

Каждый упомянутый выше метод используется для преобразования полученных данных из соответствующего JSON формата в список значений. Полученный список представляет из себя выборку значений индекса загрязнения, которые соответствуют установленному критерию. В качестве используемого критерия выступает атмосферное давление в области, где было проведено измерение индекса загрязнения. Как было указано ранее, для выборки используются значения, полученные в заданных пределах.

После преобразования и выбора необходимых значений, они сохраняются в используемой файловой системе. Путь сохраняемого файла определяется таким же способом, как это происходила при поиске. Директория определяется с помощью года, за который были запрошены данные, а название файла определяется аналогичным образом с помощью метода `to_file_path`.

Распишем данный алгоритм по шагам:

Шаг 1. Создаём объект класса `FileSystemAdapter` в зависимости от используемой файловой системы. Параметры для конструктора также определяются с помощью файла конфигурации;

Шаг 2. Преобразуем параметры для данных в строку с использованием метода `to_file_path` и сохраняем результат в переменную `file_name`;

Шаг 3. Выбирается параметр, используемый для разбиения для

разделов. Данный параметр также устанавливается с помощью файла конфигурации;

Шаг 4. С помощью вызова метода `ls` класса `FileSystemAdapter` получается список существующих директорий (разделов);

Шаг 5. Если раздел с требуемым параметром не существует, то он создаётся с помощью метода `mkdir` класса `FileSystemAdapter`;

Шаг 6. В выбранном разделе с помощью метода `ls` класса `FileSystemAdapter` получается список хранимых файлов;

Шаг 7. В полученном списке происходит поиск файла с именем `file_name`;

Шаг 8. В случае, если искомый файл не был найден - происходит переход к шагу 12;

Шаг 9. Значение вычитывается из файла с помощью метода `read_file` класса `FileSystemAdapter`;

Шаг 10. Результат в формате JSON сохраняется в переменную `value`;

Шаг 11. Происходит переход к шагу 25;

Шаг 12. Создаётся объект класса `PollutionDumper` соответствующий используемому API. Возможные варианты: `CODumper`, `NODumper`, `SODumper`, `OZDumper`. Параметры для создания этого объекта предоставляются с помощью файла конфигурации;

Шаг 13. У созданного объекта вызывается метод `dump` с текущими параметрами: географическая широта, географическая долгота, год;

Шаг 14. В вызываемом методе параметры преобразуются в адрес для GET запроса к внешнему сервису с помощью метода `to_address`;

Шаг 15. Осуществляется вызов HTTP запроса по полученному адресу;

Шаг 16. Происходит ожидание ответа;

Шаг 17. Проверяется статус ответа;

Шаг 18. В случае, если код статуса равен `200`, что соответствует корректному ответу, происходит переход к шагу 21;

Шаг 19. Выбрасывается исключение, которое обрабатывается и выводит сообщение об ошибке в лог;

Шаг 20. Происходит переход к шагу 30;

Шаг 21. Ответ в формате JSON отправляется в соответствующий обработчик. Возможные обработчики: функция `parse_co`, функция `parse_so`, функция `parse_no`, функция `parse_oz`;

Шаг 22. В обработчике происходит преобразование данных в соответствующем формате JSON в список значений;

Шаг 23. Результат обработки сохраняется в переменную `value`;

Шаг 24. Происходит запись переменной `value` в файл с именем `file_name` с помощью метода `write_file`.

Шаг 25. Используемые параметры также преобразуются в формат

JSON и сохраняется в переменную `key`;

Шаг 26. Создаётся объект класса `KafkaSender`, с параметрами, указанными в файле конфигурации;

Шаг 27. Вызывается метод `send_message` созданного объекта класса `KafkaSender`, в который передаются значения: `key`, `value`, топик и параметры. Используемый топик также определяется с помощью файла конфигурации и зависит от текущего API;

Шаг 28. При успешной отправке, вызывается метод `callback`, который отправляет лог сообщение об успешной отправке сообщения, и выводит метаинформацию отправленного сообщения.

Шаг 29. Если отправка сообщения не произошла - выбрасывается исключение;

Шаг 30. Происходит переход к следующей итерации.

Данный алгоритм также представлен на диаграмме последовательности ГУИР.400201.079 РР3.

4.2 Алгоритм отправки данных

Подготовленные данные отправляются в kafka. Для каждого типа индекса загрязнения выделен отдельный топик. Каждое сообщение из себя представляет комбинацию ключа и значения. Для ключа и значения устанавливается функция, которая будет преобразовывать данные при отправке в массив данных. Данное преобразование необходимо для поддержки пользовательского формата данных и тем самым предоставляет возможность отправлять и хранить любой объект.

В качестве ключа используются параметры, которые были использованы при получении данных:

– год, в котором было проведено измерение;

– географическая широта области, в которой было проведено измерение;

– географическая долгота области, в которой было проведено измерение.

Все эти параметры преобразуются в строку в формате JSON. Полученная строка выступает в качестве ключа в отправляемом сообщении. В качестве значения выступает список значений и соответствующего атмосферного давления, представляющий обработанные данные. Значение также передаётся в формате JSON.

Каждый тип измерений отправляется в свой топик, который определяется с помощью файла конфигурации. Также с помощью файла конфигураций устанавливаются правила поведения объекта, который отвечает за отправку сообщений. Данные параметры уже были рассмотрены в предыдущем разделе. Они влияют на алгоритм установки соединения,

возможность восстановления потерянного соединения, безопасность передачи данных и размер пакета.

Также при отправке устанавливается функция обратного вызова (англ. `callback`), что позволяет получить метаинформацию отправленного сообщения и тем самым подтвердить успешную отправку сообщения.

4.3 Алгоритм получения данных

Получение данных осуществляется в режиме реального времени с помощью технологии Spark streaming. Для осуществления подобного функционала используется пакетная обработка данных. С помощью установленных в конфигурационном файле параметров создаётся виртуальный поток данных. Такой поток данных является виртуальным, так как на самом деле потока как такового не существует, но с помощью него описываются действия, которые будут производиться с каждым полученным пакетом данных. Подобный подход используется при описании вычислений на распределённом датасете, так как он существует только во время исполнения программы.

Виртуальный поток представляет из себя поток данных, которые будут получены из kafka топиков, которые приводятся в файле конфигурации. Так как данные приходят в уже определённом формате, то их для начала необходимо разобрать на поля. Из ключа сообщения можно получить переданные поля в формате JSON:

– год, в котором было проведено измерение;

– географическая широта области, в которой было проведено измерение;

– географическая долгота области, в которой было проведено измерение.

Благодаря такой обработке, получаем распределённый датасет (RDD) уже с несколькими полями. А так как значения сообщения из себя представляет список значений индекса и соответствующего атмосферного давления, то для каждой такой записи предыдущие параметры будут одинаковые.

В результате всех преобразований получается распределённый датасет (RDD), состоящий из нескольких полей:

– год, в котором было проведено измерение;

– географическая широта области, в которой было проведено измерение;

– географическая долгота области, в которой было проведено измерение;

– атмосферное давление в области, в которой было проведено измерение;

– значение измеряемого индекса загрязнения.

Полученный датасет сохраняется в одном из следующих форматов:

– `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.

Для полученных данных лучше всего подходят форматы `parquet` и `orc` по причине неизменяемости данных. Однако формат может быть установлен на любой другой по желанию, так как данный параметр устанавливается в конфигурационном файле.

Также Apache Spark поддерживает разделение на разделы, что уже было отмечено выше. В текущей ситуации под разделение лучше всего подходит поле с годом осуществления измерения. Такое решение исходит из того, большая часть данных имеет одинаковый год, и при анализе часто будут браться данные по конкретному году. Это позволит оптимизировать операции выбора и фильтрации данных, что повысит производительность системы.

На этих данных развёртывается таблица с помощью Apache Spark. Это позволяет использовать преобразованные данные для дальнейшего анализа. Как только будет получен новый пакет, после преобразования он также добавится уже в существующую таблицу, тем самым обеспечив её обновление в режиме реального времени.

Сами же пакеты данных образуются с помощью использования так называемого окна (англ. window). Окно представляет из себя промежуток времени. Данные, которые были получены за этот промежуток времени и являются пакетом. Размер используемого окна также определяется с помощью конфигурационного файла.

В результате получается следующий алгоритм:

Шаг 1. Создаётся объект словаря `params`;

Шаг 2. В словарь `params` добавляются параметры получателя, которые определяются с помощью файла конфигурации;

Шаг 3. Происходит инициализация spark-контекста;

Шаг 4. Устанавливается мастер, который определяется файлом конфигурации. Используемый мастер может быть как локальный, так и удалённый;

Шаг 5. Устанавливаем окно для дальнейшей потоковой обработки;

Шаг 6. С помощью словаря `params`, созданного spark-контекста, а также топиков, которые определяются файлом конфигурации, создаётся объект виртуального потока - объект `stream` класса `DStream[String, String]`;

Шаг 7. Для виртуального потока `stream` с помощью метода `mapPartition` определяется лямбда-функция, которая будет исполняться для каждого полученного раздела;

Шаг 8. В этой функции с помощью метода `map` определяется лямбда-функция, которая будет применяться к каждой записи в разделе;

Шаг 9. В каждой записи получаем ключ и значение и записываем в переменные `key` и `value` соответственно;

Шаг 10. С помощью метода `parse_key` получаем следующие поля: географическая широта (`latitude`), географическая долгота (`longitude`), год (`year`);

Шаг 11. С помощью метода `parse_value` получаем список со следующими полями: атмосферное давление (`pressure`), значение измерения (`value`);

Шаг 12. Для каждого значения в обработанном `value` копируем соответствующие параметры (поля из обработанного `key`);

Шаг 13. В результате трансформации получается распределённый датасет (RDD) со следующими полями: `latitude`, `longitude`, `year`, `pressure`, `value`;

Шаг 14. Объединяется с другими полученными в текущем окне датасетами;

Шаг 15. Сохраняет текущее смещение (`offset`) получателя с помощью метода `checkpoint`;

Шаг 16. Ожидаем следующий пакет, пока окно не закончилось. При его появлении переходим к шагу 9;

Шаг 17. Если окно закончилось, то весь объединённый датасет записывается в переменную `resultRdd`;

Шаг 18. Полученный датасет в переменной `resultRdd` записывается в формат, установленный в файле конфигурации с помощью метода `write`;

Шаг 19. Полученный датасет добавляется в базу с помощью метода `withTempView`;

Шаг 20. Происходит переход к следующему временному окну;

Шаг 21. Переход к шагу 9.

Такой алгоритм работает бесконечно, пока не будет остановлен spark-контекст, либо при возникновении ошибки. Похожий принцип

используется при разработке веб-серверов.

Данный алгоритм также представлен на диаграмме последовательности ГУИР.400201.079 РР2.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе будет рассмотрено тестирование разработанного программного обеспечения.

Так как данный продукт представляет из себя систему компонентов, которая использует полноценный кластер, то и тестирование будет производиться в двух напрвлениях:

– юнит-тестирование;
– интеграционное тестирование.

Большое внимание будет уделяться именно интеграционному тестированию, так как оно позволяет проверить рабатоспособность всей системы. Можно сказать, что такое тестирование определяет наличие ошибки в приложении. Юнит-тестирование в свою очередь помогает определить, в каком именно модуле произошла ошибка.

Важность тестирования возрастает и с тем, что проект является открытым, и поэтому обязательно необходимо тестировать все изменения. Также для удобства тестирования были использованы инструменты для непрерывной интеграции. В частности для непрерывной интеграции использовались:

– Jenkins;
– Gitlab - pipelines.

Jenkins представляет из себя Java веб приложение, которое запускается на хосте, и позволяет запускать команды на этом хосте. Таким образом, с помощью файла `./Jenkinsfile` идёт описание инструкций, которые необходимо выполнить для репозитория на используемой машине. Более подробно с Jankins можно ознакомится на сайте с его документацией [8].

Таким образом, если на используемом хосте установлен Docker, то это позволяет полностью развернуть всю необходимую инфраструктуру для запуска всей системы. Также есть возможность настроить Jenkins на проверку удалённого репозитория по расписанию. Если с момента последнего запуска, в удалённый репозиторий были закоммичены изменения, то как только jenkins проверит удалённый репозиторий - он выполнит команды, которые указаны в репозитории. Такой режим работы называется трубой (англ. pipeline). Также можно запускать команды и вручную.

Аналогичным инструментом является Gitlab - pipelines. Благодаря возможности интеграции с сервисом Github - такой инструмент позволяет запускать

5.1 Интеграционное тестирование

Для интеграционного тестирования вызывается докер, в котором разворачивается вся необходимая инфраструктура, а также и разработанные компоненты системы. В качестве интеграционного теста происходит проверка отправки и получения сообщений, которые передаются с помощью kafka.

С помощью докера запускаются следующие компоненты:

– zookeeper;
– kafka;
– отправитель.

Отправитель использует настройки для тестирования. Однако, для взаимодействия со внешним сервисом необходим ключ, что уже было отмечено раньше.

Данное значение устанавливается в самих настройках пайплайна в jenkins. Такой подход позволяет скрыть действительно используемый API ключ от других участников проекта. В свою очередь каждый участник сможет использовать свой собственный ключ для тестирования, либо для использования проекта.

После того, как все компоненты были запущены, отправитель перебирает установленный диапазон параметров, и пытается получить для них данные. Всё действует точно также, как и при настоящей работе программы: сначала происходит проверка локального хранилища, а в случае отсутствия данных - происходит обращение к внешнему сервису. Все отправленные сообщения отправляются в топик для тестирования. Это позволяет при желании использовать тот же kafka брокер, который будет использоваться и при работе программы. В таком случае, происходит тестирование продукта в «полевых» условиях.

После того, как сообщения были отправлены в брокер, отдельный получатель проверяет количество сообщений в тестовом топике. Количество сообщений должно совпадать с диапазоном значений, который был использован при отправлении значений. Если же количество сообщений не совпадает - значит какие-то данные не были отправлены в топик. Это означает, что такая система не может полноценно работать, поэтому использование текущего окружения и настроек приведёт к ошибочному поведению программы.

Такое тестирование позволяет уловить само наличие ошибок, однако не предоставляет точной информации, в какой именно части произошёл сбой. Поэтому, при тестировании ведётся логгирование программы, что позволяет отследить полный ход программы. Возможные причины отсутствия сообщений:

– недоступность kafka-брокера;

– используемый топик не был создан;
– был указан неверный путь к локальному хранилищу;
– был указан неверный API-ключ;
– используемый API-ключ не имеет необходимой подписки для использования API для получения индекса загрязнений;
– отсутствие связи с сервером;
– неработоспособность zookeeper сервера;
– используемый порт уже используется другим приложением;
– недоступный удалённый адрес распределённой системы при использовании таковой;
– системные ошибки.

К системным ошибкам относятся недостаток свободного места на диске, нехватка оперативной памяти, принудительное завершение процесса и тому подобное. Остальные ошибки можно распознать при чтении log-файла.

5.2 Юнит тестирование

Юнит тестирование выполняет функцию локализации ошибки. Другими словами, при установке наличия ошибки, с помощью юнит тестов можно определить, в какой чатси программы она произошла. Также юнит тестирование применяется для проверки работоспособности модуля, что позволяет проверять внесённые изменения в отдельный компонент программы, без запуска всей системы.

Для юнит тестирования использовалась библиотека `pytest`. Компоненты программы, которые подвергаются тестированию:
– классы для обработки конфигурации;
– классы для работы с файловой системой;
– классы для получения данных из внешнего сервиса;
– классы для преобразовния данных;
– классы для подключения и отправки сообщений.

5.2.1 Тестирование классов конфигурации

Главная задача классов конфигурации - правильно обработать исходный файл конфигурации и на его основе создать параметры конфигурации для всего приложения. В качестве параметров используется список строк, что позволяет проводить тестирование данных классов без использования файлов конфигурации. При тестировании создаётся список строк, каждая из которых определяет параметр. В результате такой обработки, параметры, которые были переданы, и параметры, которые содержатся в созданном экземпляре класса должны быть идентичными.

Один из примеров тестирования обработки конфигурации:

```
def test_fs_parse_from_lines(self):
    lines = [
        ”Dir   :  /users/hdfs/dumper/test_out/ ”,
        ”Host : 10.0.2.5:8088”
    ]
    actual = FSConfig.parse_from_lines(lines)
    expected = FSConfig(”/users/hdfs/dumper/test_out/”,
       ”10.0.2.5:8088”)
    self.assertEqual(actual, expected, ”FS Configs are not
       equal”)
```

5.2.2 Тестирование классов работы с файловой системой

Главная задача классов для работы с файловой системой - производить простые операции с файловой системой. В частности, такие операции как:

– запись в файл;
– чтение из файла;
– создание директории;
– удаление файла;
– получение списка файлов в директории;
– проверка существования файла.

Для тестирования этих операций, создаётся директория, которая предназначена для тестирования. В этой директории как раз и будут создаваться и удаляться файлы в ходе тестирования. Для тестирования записи в файл - происходит запись в файл определённых данных, а после происходит вычитывание данных из созданного файла. При идентичности записанных и считанных данных тест считается пройденным. Для проверки чтения всё происходит аналогично, только в этот раз чтение осуществляется с помощью тестируемого класса, а запись - при помощи стандартных средств.

При проверке удаления файла, сначала происходит получение файлов в тестовой директории. После этого происходит удаление одного файла. Далее снова происходит получение списка файлов в тестовой директории. После происходит сравнение списка до удаления и после.

5.2.3 Тестирование классов для получения данных

Главная задача классов для получения данных является обращение к внешнему сервису и отправка запроса к нему. Тестированию подвергается два аспекта данных классов:

– построение запроса к внешнему сервису;
– сохранение и выдача результата при успешном запросе.

Рисунок 5.1 – Результаты юнит тестирования

Для тестирования построения запроса к внешнему сервису происходит конфигурирование класса для запроса и вызов метода построения URL адреса. Полученный адрес сравнивается с ожидаемым адресом для используемых аргументов. Пример такого тестирования для одного из запросов:

```
def test_co_address(self):
    co_dumper = CODumper(”some_host”, ”atztpcnk”, ”test”,
       self.__adapter)
    actual = co_dumper.to_address(25, 45, 2017)
    expected = ”some_host/co/25,45/2017.json?appid=atztpcnk”
    self.assertEqual(actual, expected)
```

Для тестирования сохранения и выдачи данных происходит подмена функции запроса. При нормальной работе класс использует стандартную функцию для отправки запроса. Однако если использовать такой вариант при тестировании - появляется зависимость от внешних факторов, таких как подключение устройства к внешней сети, работоспособность внешнего сервиса и корректность используемого ключа. Так как тестирование должно быть максимально изолированным и не должно зависеть от внешних факторов, то добавляется возможность подменить используемую функцию для отправки запроса на любую другую. Такой подход позволяет протестировать саму логику приложения.

В итоге происходит подмена функции отправки запроса на ложную. Например, одна из таких функций:

```
def mock_oz_request(address):
    if address == ”some_host/o3/24,92/2018.json?appid=
       atztpcnk”:
        return ResponseMock(200, ”Yes, it’s ok”)
    return ResponseMock(400, ”Something wrong”)
```

5.2.4 Тестирование классов обработки данных

Главная цель классов обработки данных является преобразование полученных данных в формате JSON в вид, который позволяет легко их обрабатывать. При тестировании таких классов происходит обработка заготовленного JSON объекта и проверяется результат обработки. При идентичности полученного и ожидаемого результата тест считается успешным.

Результаты тестирования представлены на рисунке 5.1

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данном разделе будет описана последовательность действий, которую необходимо произвести пользователям для использования данного программного продукта.

Сам продукт распространяется в качестве проекта с открытым исходным кодом по лицензии Apache License. Это означает, что каждый желающий может присоединиться к разработке и дальнейшему развитию проекта.

По этой причине, важную часть проекта также занимает документация и инструменты непрерывной интеграции.

Данный проект может использоваться как отдельная программа в локальной системе, либо в качестве компонента более масштабной системы. По этой причине продукт можно сконфигурировать под разные окружения.

Основные режимы работы:

– Локальный режим;
– Режим кластера.

6.1 Руководство для локального режима

Данное руководство предназначено для локальной развёртки приложения и работы на одной машине.

6.1.1 Требования к программному обеспечению

Для корректной сборки и работы данного дипломного проекта требуется следующее программное обеспечение:

– Docker-compose версии выше 1.10.0+ (поддерживающий API версии 2.0);
– любой веб-браузер.

Для работы продукта в локальном режиме будет достаточно вышеперечисленного программного обеспечения. Всё окружение, которое необходимо для работы компонентов системы обеспечивается с помощью Docker образов, а все компоненты системы работают в контейнере.

6.1.2 Настройка рабочего окружения

В первую очередь, необходимо установить docker-compose. Инструкцию по установке данного инструмента для своей операционной системы можно найти на официальном сайте [11]

Ниже приведён пример установки docker-compose для операционной системы Centos 7:

```
$ export DOCKER_COMPOSE_VERSION='1.24.0'
$ sudo yum remove docker \
    docker-client \
    docker-client-latest \
    docker-common \
    docker-latest \
    docker-latest-logrotate \
    docker-logrotate \
    docker-engine
$ sudo yum install -y yum-utils \
    device-mapper-persistent-data \
    lvm2
$ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
$ sudo yum install docker-ce docker-ce-cli containerd.io
$ sudo curl -L "https://github.com/docker/compose/releases/
   download/$DOCKER_COMPOSE_VERSION/docker-compose-$(uname -
   s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
$ sudo systemctl start docker
$ sudo usermod -aG docker $(whoami)
```

Для проверки корректности установки следует использовать следующие команды:

```
$ docker run hello-world
$ docker-compose --version
```

Браузер необходим для взаимодействия с интерактивными документами. Для примера возьмём установку браузера Mozilla Firefox для Linux:

```
$ sudo yum install -y firefox
```

После установки всех необходимых компонентов, можно приступить к запуску самого приложения. Для получения приложения можно использовать DVD-диск, который идёт в комплекте к дипломному проекту, либо получить последнюю версию из GitHub:

```
$ wget https://github.com/FyodorovAleksej/
   OpenAirPollutionMonitor/archive/master.zip
$ unzip ./OpenAirPollutionMonitor-master.zip
```

После получения проекта для начала необходимо установить необходимые настройки, либо использовать настройки по-умолчанию. Для локального запуска подойдут настройки по-умолчанию. Единственный параметр, который необходимо поменять - это указать свой API ключ для сервиса OpenWeatherMap. Для его получения необходимо зарегистрироваться на этом сервисе и оформить подписку на используемый API (").После оформления подписки, на используемую почту придёт

письмо с ключом. Этот ключ необходимо вписать в файл конфигурации. Эту операцию на Linux можно сделать следующим образом:

```
$ export OPEN_WEATHER_MAP_API_KEY=”xxxxxxxxx”
$ sed –i –e ”s/#YOUR_API_KEY#/$OPEN_WEATHER_MAP_API_KEY/g”
   ./src/OAPM/dumper/config/dumper_config.ini
```

После установки ключа, можно полноценно использовать приложение в локальном режиме. Для этого можно использовать скрипт `startup.sh`. Данный скрипт развернёт всё необходимое окружение для работы с помощью docker-compose. Все компоненты приложения собираются в контейнеры, и запускаются в подготовленной среде.

В качестве готовых компонентов выступают следующие контейнеры:

– Zookeeper;
– Apache Kafka;
– Apache Spark;
– Apache Zeppelin.

Zookeeper необходим для контроля kafka брокеров. По-умолчанию он разворачивается в контейнере, и публикует порт 2181 наружу. Это означает, что он доступен не только внутри созданной локальной подсети, но и с хоста. Это позволяет отслеживать работу Zookeeper, и тем самым следить за работой kafka брокеров.

Apache kafka запускается в контейнере. Так как kafka использует zookeeper для контроля брокеров, то необходимо задать адрес zookeeper сервера. Docker-compose создаёт локальную подсеть, для связи всех контейнеров в собираемом файле, поэтому kafka контейнер будет находиться в одной сети с zookeeper. Таким образом обеспечивается взаимосвязь между брокерами и zookeeper.

Для работы с kafka - публикуется порт 9092, с помощью которого можно подключиться и использовать топики. Также сразу при создании контейнера запускается команда на создание топиков, которые необходимы приложению:

– co-topic;
– so-topic;
– no-topic;
– oz-topic.

Apache Spark состоит из нескольких контейнеров:

– master;
– worker 1;
– worker 2;

В разворачиваемом кластере должен быть один мастер узел, и сколько угодно рабочих узлов. Как уже было упомянуто раньше, задача будет отправляться на мастер узел, а он в свою очередь распределит её на рабочие

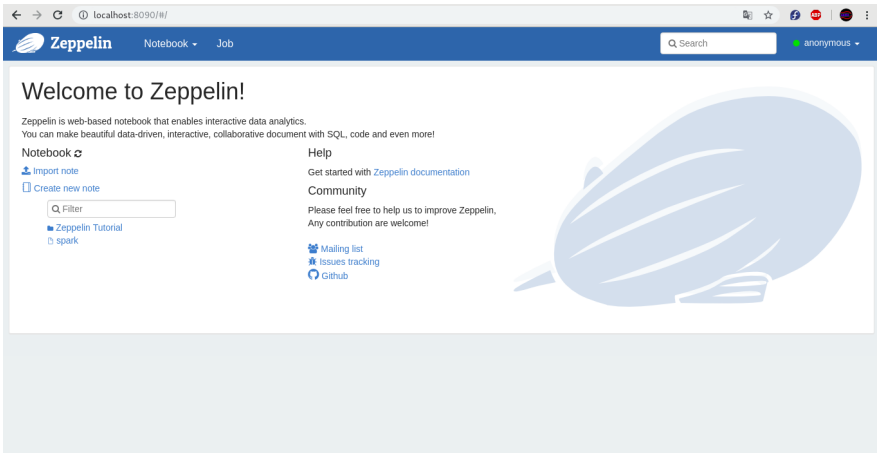

Рисунок 6.1 – Приветственная страница Apache Zeppelin

узлы, на которых задача будет выполняться параллельно. Количество рабочих узлов при необходимости можно увеличить.

Apache Zeppelin представляет из себя аналог Jupyter. При развёртке данного контейнера используется монтирование локальной директории, что позволяет сразу использовать заготовленные интерактивные документы. Также для Zepelin указывается адрес spark-мастера. Так как все контейнеры находятся в одной подсети - то указывается адрес созданного контейнера со spark-мастером. Для подключения со стороны хоста - пробрасывается порт 8090.

Кроме вышеупомянутых сторонних контейнеров, также используется контейнер с самим приложением. Он также монтируется к локальной системе, что позволяет использовать файлы хоста. Такой принцип используется, чтобы можно было перезапускать данный контейнер и при этом не терять файлы, которые были получены с внешнего сервиса OpenWeatherMap.

После запуска всех необходимых контейнеров, для дальнейшей работы понадобится браузер. Для этого необходимо перейти по адресу хоста, на котором развёрнуты все контейнеры через порт, который был опубликован для Apache Zeppelin. По умолчанию - для Apache Zeppelin публикуется порт 8090. Так как все эти контейнеры были развёрнуты на локальной машине, то необходимо перейти по адресу `localhost:8090`. В случае, если Apache Zeppelin был успешно запущен, то при обращении по указанному адресу должна открыться страница, подобно той, что приведена на рисунке 6.1.

При открытии данной страницы, необходимо выбрать из предложенных интерактивных документов в списке `Notebooks`, документ с названием `spark`. Когда откроется документ, остаётся только запустить его. В случае, если Zeppelin попросит выбрать используемый интерпретатор, то необходимо выбрать `Spark`.

6.2 Работа в режиме кластера

Отличие в данном режиме заключается в том, что уже существует кластер, в котором есть все используемые компоненты. Также в качестве подобного кластера может использоваться специальный образ (`sandbox`), который представляет из себя платформу Hadoop.

Например компании Cloudera и Hortonworks предоставляют подобный продукт. В предоставляемом продукте уже существуют используемые компоненты. Также в платформе Hadoop используется распределённая файловая система. Разрабатываемый продукт как раз поддерживает работу с таким типом файловой системой. Также из-за специфики распределения файлов, есть возможность создания Hive таблицы на скопированных данных. Тем самым можно отслеживать используемые приложением файлы в системе.

В состав Hadoop платформы входят необходимые для разрабатываемого продукта компоненты:

– Apache Zookeeper;
– Apache Kafka;
– Apache Spark;
– Apache Zeppelin.

В связи с этим отпадает потребность в локальном развёртывании вышеперечисленных компонентов. В таком случае, необходимо установить необходимые адреса в файле конфигурации `./src/OAPM/dumper/config/dumper_config.ini`.

Используемые топики должны быть созданы в используемом kafka брокере.

Для этого, необходимо выполнить следующую команду:

```
$ kafka–topics.sh ––create–topic ––partitions 1 ––
   replication 1 ––bootstrapserver localhost:6667 co–topic
```

Количество разделов и количество реплик необходимо устанавливать в соответствии с кластером. Их количество обычно зависит от количества узлов данных в кластере.

Для проверки создания топиков необходимо использовать следующую команду:

```
$ kafka–topics.sh ––zookeeper zookeeper:2181 ––list
```

В выводе команды должны быть указаны создаваемые топики.

Так как существует доступ ко всем необходимым компонентам системы, то можно запустить разрабатываемый продукт локально. Для локального запуска потребуются следующие компоненты:

– Python и pip версии 3.0 и выше;

Для запуска необходимо сначала установить все необходимые зависимости, которые приведены в файле `./src/OAPM/requirements.txt`. Это можно сделать с помощью команды:

```
$ sudo pip install –r ./src/OAPM/requirements.txt
```

Также вместо использования системного интерпретатора можно создать виртуальное окружение, и использовать его.

Для установки самого пакета необходимо использовать команду:

```
$ sudo python ./src/OAPM/setup.py install
```

После утановки, достаточно запустить Python модуль `./src/OAPM/dumper/cli.py`.

```
$ sudo air_pollution_dumper
```

В качестве параметра данный скрипт принимает путь к файлу конфигурации. Использует предоставляемый файл конфигурации `./src/OAPM/config/dumper_config.ini` в качестве файла по-умолчанию.

После того, как приложение будет запущено, то можно использовать Apache Zeppelin. Подключившись к нему, необходимо экспортировать интерактивный документ из файла `./src/docker/zeppelin/notebooks/spark/note.json`. После экспортирования документа, необходимо открыть его и запустить.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ АНАЛИЗА ЗАГРЯЗНЕНИЯ ОКРУЖАЮЩЕЙ СРЕДЫ НА ОСНОВЕ НЕЙРОННОЙ СЕТИ

7.1 Краткая характеристика программного продукта

В результате дипломного проектирования был разработан проект, который позволяет осуществлять анализ данных о загрязнении окружающей среды из внешнего источника в режиме реального времени. Из-за абстрагирования от источника данных, такой проект может использовать абсолютно любой источник данных. Данная особенность предоставляет проекту высокую гибкость. Также из-за возможности произведения вычислений на полноценном кластере появляется возможность обрабатывать огромные массивы данных, что позволит увеличить нагрузку на систему. При использовании распределённой системы хранения обеспечивается надёжность хранения полученных данных.

Текущий источник данных в лице сервиса OpenWeatherMap не позволяет обеспечить высокую нагрузку и скорость данных. Так как используемый API для этого сервиса на текущий момент ещё развивается, то он имеет некоторые ограничения. Из-за этих ограничений, для полноценной работы сервиса пришлось эмулировать источник данных с помощью локальных данных. Такой подход позволил оценить производительность системы при высокой частоте отправки сообщений и высокой нагрузке.

Для проверки работы на полноценном кластере был использован инструмент контейнеризации, который позволил на одной машине запустить несколько контейнеров, представляющих из себя компоненты кластера. Также он позволил организовать связь между запущенными контейнерами, что позволило сэмулировать полноценный кластер, в котором все машины связаны между собой.

В результате полученный продукт обладает следующими качествами:

– отказоустойчивость при использовании нескольких машин.

– высокая пропускная способность из-за использования параллельных вычислений.

– гибкий источник данных. Так как в его роли может выступать любой процесс, который будет отправлять данные в брокер сообщений.

– анализ в режиме реального времени из-за использования пакетной обработки данных.

7.2 Определение трудоемкости разработки программного продукта

Оценка трудоемкости разработки может быть определена с помощью укрупненного метода. Для этого требуется использовать следующую формулу:

$$T_{рз} = T_{оа} + T_{бс} + T_{п} + T_{отл} + T_{др} + T_{до}, \quad (7.1)$$

где $T_{оа}$ – трудоемкость подготовки описания задачи и исследования алгоритма решения;

$T_{бс}$ – трудоемкость разработки блок-схемы алгоритма;

$T_{п}$ – трудоемкость программирования по готовой блок-схеме;

$T_{отл}$ – трудоемкость отладки программы на ПК;

$T_{др}$ – трудоемкость подготовки документации по задаче в рукописи;

$T_{до}$ – трудоемкость редактирования, печати и оформления документации по задаче.

Трудоемкость описания задачи и исследования алгоритма решения может быть определена по следующей формуле:

$$T_{оа} = \frac{Q \cdot W \cdot K}{80}, \quad (7.2)$$

где Q – условное число операторов в разрабатываемом программном продукте;

W – коэффициент увеличения затрат труда вследствие недостаточного или некачественного описания задачи;

K – коэффициент квалификации разработчика алгоритмов и программ.

Аналогично рассчитываются и оставшиеся неизвестные величины в формуле (7.1):

$$T_{бс} = \frac{Q \cdot K}{22} \quad (7.3)$$

$$T_{п} = \frac{Q \cdot K}{22} \quad (7.4)$$

$$T_{отл} = \frac{Q \cdot K}{4{,}5} \quad (7.5)$$

$$T_{др} = \frac{Q \cdot K}{18} \quad (7.6)$$

$$T_{до} = 0{,}75 \cdot T_{др} \quad (7.7)$$

Условное число операторов Q можно рассчитать по формуле:

$$Q = q \cdot C \cdot (1 + p), \tag{7.8}$$

где q – число операторов в программе;

C – коэффициент сложности программы;

p – коэффициент коррекции программы в ходе ее разработки.

Посчитав число операторов в программе, получим $q = 1900$. Приняв $C = 2{,}2$, а $p = 0{,}35$, рассчитаем условное число операторов Q по формуле (7.8):

$$Q = 1900 \cdot 2{,}2 \cdot (1 + 0{,}35) = 5643 \tag{7.9}$$

Из-за стажа работы разработчика алгоритмов менее двух лет, получаем $K = 0{,}8$ и $W = 1{,}3$. Эти величины понадобятся для расчета по формулам выше.

Подставив результат (7.9) в выражения (7.2) – (7.6), получим следующие результаты:

$$\mathrm{T_{оа}} = \frac{5643 \cdot 1{,}3 \cdot 0{,}8}{80} \approx 73{,}36 \tag{7.10}$$

$$\mathrm{T_{бс}} = \frac{5643 \cdot 0{,}8}{22} \approx 205{,}20 \tag{7.11}$$

$$\mathrm{T_{п}} = \frac{5643 \cdot 0{,}8}{22} \approx 205{,}20 \tag{7.12}$$

$$\mathrm{T_{отл}} = \frac{5643 \cdot 0{,}8}{4{,}5} \approx 1003{,}20 \tag{7.13}$$

$$\mathrm{T_{др}} = \frac{5643 \cdot 0{,}8}{18} \approx 250{,}80 \tag{7.14}$$

На основании расчетов выше и формулы (7.7) получим значение трудоемкости редактирования, печати и оформления документации по задаче:

$$\mathrm{T_{до}} = 0{,}75 \cdot 250{,}80 \approx 188{,}10 \tag{7.15}$$

Тогда на основании формулы (7.1) и расчетов ранее, трудоемкость разработки составит:

$$\mathrm{T_{рз}} = 73{,}36 + 205{,}20 + 205{,}20 + 1003{,}20 + 250{,}80 + 188{,}10 = 1925{,}86 \tag{7.16}$$

7.3 Определение стоимости машино-часа работы ПК

Произведем расчет стоимости машино-часа работы ПК. Она определяется по формуле:

$$S_{\text{мч}} = \text{С}_{\text{э}} + \frac{\text{А}_{\text{пк}} + \text{Р}_{\text{пк}} + \text{А}_{\text{пл}} + \text{Р}_{\text{пл}} + \text{Р}_{\text{ар}}}{\Фпк}, \quad (7.17)$$

где $\text{С}_{\text{э}}$ – расходы на электроэнергию за час работы ПК, руб;

$\text{А}_{\text{пк}}$ – годовая величина амортизационных отчислений на реновацию ПК, руб;

$\text{Р}_{\text{пк}}$ – годовые затраты на ремонт и техническое обслуживание ПК, руб;

$\text{А}_{\text{пл}}$ – годовая величина амортизационных отчислений на реновацию производственных площадей, занимаемых ПК, руб;

$\text{Р}_{\text{пл}}$ – годовые затраты на ремонт и содержание производственных площадей, руб;

$\text{Р}_{\text{ар}}$ – годовая величина арендных платежей за помещение, занимаемое ПК, руб;

$\Фпк$ – годовой фонд времени работы ПК, ч.

Расходы на электроэнергию за час работы ПК определяются по формуле:

$$\text{С}_{\text{э}} = N_{\text{э}} \cdot k_{\text{ис}} \cdot \text{Ц}_{\text{э}}, \quad (7.18)$$

где $N_{\text{э}}$ – установленная мощность блока питания ПК, кВт;

$k_{\text{ис}}$ – коэффициент использования энергоустановок по мощности;

$\text{Ц}_{\text{э}}$ – стоимость киловатт-часа электроэнергии, руб.

Примем установленную мощность блока питания ПК равной 0,085кВт и коэффициент использования энергоустановок по мощности 0,9. Стоимость киловатт-часа электроэнергии на момент расчетов составила 0,25 руб. Тогда расходы на электроэнергию за час работы ПК составляют:

$$\text{С}_{\text{э}} = 0{,}085 \cdot 0{,}9 \cdot 0{,}25 \approx 0{,}02 \text{ руб} \quad (7.19)$$

Годовая величина амортизационных отчислений на реновацию ПК определяется по формуле:

$$\text{А}_{\text{пк}} = \text{Ц}_{\text{пк}}^{\text{б}} \cdot \frac{\text{Н}_{\text{пк}}^{\text{а}}}{100}, \quad (7.20)$$

где $\text{Ц}_{\text{пк}}^{\text{б}}$ – балансовая стоимость ПК, руб;

$\text{Н}_{\text{пк}}^{\text{а}}$ – норма амортизационных отчислений на ПК, %.

Балансовая стоимость ПК определяется по формуле:

$$\text{Ц}_{\text{пк}}^{\text{б}} = \text{Ц}_{\text{пк}} \cdot k_{\text{у}} \cdot k_{\text{м}}, \quad (7.21)$$

где $Ц_{пк}$ – цена ПК на момент его выпуска, руб;

$k_у$ – коэффициент удорожания ПК;

$k_м$ – коэффициент, учитывающий затраты на монтаж и транспортировку ПК.

Приняв стоимость нового ПК равной 1400 руб, коэффициент удорожания ПК равным 1, а коэффициент $k_м$ равным 1,05, получим балансовую стоимость ПК:

$$Ц^{б}_{пк} = 1400 \cdot 1 \cdot 1{,}05 \approx 1470{,}00 \text{ руб} \tag{7.22}$$

Если взять норму амортизационных отчислений на ПК, равную 10%, то годовая величина амортизационных отчислений на реновацию ПК составит:

$$А_{пк} = 1470{,}00 \cdot \frac{10}{100} = 147{,}00 \text{ руб} \tag{7.23}$$

Годовые затраты на ремонт и техническое обслуживание ПК укрупненно могут быть определены по формуле:

$$Р_{пк} = Ц^{б}_{пк} \cdot k_{ро}, \tag{7.24}$$

где $k_{ро}$ – коэффициент, учитывающий затраты на ремонт и техническое обслуживание ПК.

Подставив $k_{ро} = 0{,}13$, найдем затраты на ремонт и техническое обслуживание ПК:

$$Р_{пк} = 1470{,}00 \cdot 0{,}13 \approx 191{,}10 \text{ руб} \tag{7.25}$$

Годовая величина амортизационных отчислений на реновацию производственных площадей, занятых ПК, определяется по формуле:

$$А_{пл} = Ц^{б}_{пл} \cdot \frac{Н^{а}_{пл}}{100}, \tag{7.26}$$

где $Ц^{б}_{пл}$ – балансовая стоимость площадей, руб;

$Н^{а}_{пл}$ – норма амортизационных отчислений на производственные площади, %.

Балансовая стоимость площадей определяется по формуле:

$$Ц^{б}_{пл} = S_{пк} \cdot k_{д} \cdot Ц_{пл}, \tag{7.27}$$

где $S_{пк}$ – площадь, занимаемая ПК, м^2;

$k_д$ – коэффициент, учитывающий дополнительную площадь;

$Ц_{пл}$ – цена квадратного метра производственной площади, руб.

Если принять, что один ПК занимает 1 м^2, $k_д = 3$, а цена одного

квадратного метра производственной площади равна 300 руб, тогда балансовая стоимость площадей составит:

$$Ц^{б}_{пл} = 1 \cdot 3 \cdot 300 \approx 900{,}00 \text{ руб} \quad (7.28)$$

Подставив $Н^{а}_{пл} = 1{,}2\%$, получим годовую величину амортизационных отчислений на реновацию производственных площадей, занятых ПК:

$$А_{пл} = 900{,}00 \cdot \frac{1{,}2}{100} \approx 10{,}80 \text{ руб} \quad (7.29)$$

Годовые затраты на ремонт и содержание производственных площадей укрупненно могут быть определены по формуле:

$$Р_{пл} = Ц^{б}_{пл} \cdot k_{рэ}, \quad (7.30)$$

где $k_{рэ}$ – коэффициент, учитывающий затраты на ремонт и эксплуатацию производственных площадей.

Подставив $k_{рэ} = 0{,}05$, получим:

$$Р_{пл} = 900{,}00 \cdot 0{,}05 \approx 45{,}00 \text{ руб} \quad (7.31)$$

Годовая величина арендных платежей за помещение, занимаемое ПК, рассчитывается по формуле:

$$Р_{ар} = S_{пк} \cdot k_{д} \cdot k_{ар} \cdot k_{комф} \cdot k_{пов} \cdot 12, \quad (7.32)$$

где $k_{ар}$ – ставка арендных платежей за помещение;

$k_{комф}$ – коэффициент комфортности помещения;

$k_{пов}$ – повышающий коэффициент, учитывающий географическое размещение площади.

Предположим, что ставка арендных платежей за помещение составляют 13,6 руб/м^2. Коэффициент комфортности возьмем равным 0,9, поскольку он соответствует помещениям, расположенным в цокольном этаже. Из-за расположения арендуемых зданий в городе Минске, $k_{пов} = 0{,}85$. Тогда годовая величина арендных платежей за помещение, занимаемое ПК, составит:

$$Р_{ар} = 1 \cdot 3 \cdot 13{,}6 \cdot 0{,}9 \cdot 0{,}85 \cdot 12 \approx 374{,}54 \text{ руб} \quad (7.33)$$

Годовой фонд времени работы ПК определяется исходя из режима ее работы и может быть рассчитан по формуле:

$$Ф_{пк} = t_{сс} \cdot T_{сг}, \quad (7.34)$$

где $t_{сс}$ – среднесуточная фактическая загрузка ПК, ч;

$T_{сг}$ – среднее количество дней работы ПК в год.

Приняв, что среднесуточная фактическая нагрузка составляет 8 часов, а в году в среднем 305 рабочих дней, рассчитаем годовой фонд времени работы ПК:

$$\Phi_{\text{пк}} = 8 \cdot 305 = 2440{,}00 \tag{7.35}$$

Тогда стоимость машино-часа работы ПК составит:

$$S_{\text{мч}} = 0{,}02 + \frac{147{,}00 + 191{,}10 + 10{,}80 + 45{,}00 + 374{,}54}{2440{,}00} \approx 0{,}33 \text{ руб} \tag{7.36}$$

7.4 Определение себестоимости создания программного продукта

Для определения себестоимости создания программного продукта необходимо определить затраты на заработную плату разработчика по формуле:

$$З_{\text{рз}} = Т_{\text{рз}} \cdot t_{\text{чр}} \cdot К_{\text{пр}} \cdot \left(1 + \frac{Н_{\text{д}}}{100}\right) \cdot \left(1 + \frac{Н_{\text{соц}}}{100}\right), \tag{7.37}$$

где $Т_{\text{рз}}$ – трудоемкость разработки программного продукта, чел-ч;

$t_{\text{чр}}$ – среднечасовая ставка работника, осуществлявшего разработку программного продукта, руб;

$К_{\text{пр}}$ – коэффициент, учитывающий процент премий в организации-разработчике;

$Н_{\text{д}}$ – норматив дополнительной заработной платы;

$Н_{\text{соц}}$ – норматив отчислений от фонда оплаты труда.

Среднечасовая ставка работника определяется исходя из единой тарифной системы оплаты труда в Республике Беларусь по следующей формуле:

$$t_{\text{чр}} = \frac{ЗП_{1р} \cdot k_{\text{т}}}{t_{\text{мес}}}, \tag{7.38}$$

где $ЗП_{1р}$ – среднемесячная заработная плата работника первого разряда;

$k_{\text{т}}$ – тарифный коэффициент работника соответствующего разряда;

$t_{\text{мес}}$ – среднее нормативное количество рабочих часов в месяце.

Среднемесячную заработную плату работника первого разряда возьмем в размере 490 руб. Из-за использования работников первой категории 11 разряда, тарифный коэффициент работника $k_{\text{т}}$ составит 3,54. Учитывая, что в среднем в месяце 170 рабочих часов, на основе формулы (7.38) рассчитаем среднечасовую ставку работника, который осуществляет разработку программного продукта:

$$t_{\text{чр}} = \frac{490 \cdot 3{,}54}{170} \approx 10{,}20 \text{ руб} \quad (7.39)$$

На сегодняшний день норматив отчислений от фонда оплаты труда составляет 34,6%. Предположим, что $К_{\text{пр}} = 1{,}7$ и $Н_{\text{д}} = 13\%$. На основании расчетов выше и формулы (7.37) найдем затраты на заработную плату разработчика:

$$З_{\text{рз}} = 1925{,}86 \cdot 10{,}20 \cdot 1{,}7 \cdot \left(1 + \frac{13}{100}\right) \cdot \left(1 + \frac{34{,}6}{100}\right) \approx 50\,792{,}23 \text{ руб} \quad (7.40)$$

Также в себестоимость разработки программного продукта включаются также затраты на его отладку в процессе создания. Для определения их величины необходимо рассчитать стоимость машино-часа работы ПК, на котором осуществлялась отладка. Данная величина соответствует величине арендной платы за час работы ПК и будет определена ниже. Затраты на отладку программы определяются по формуле:

$$З_{\text{от}} = Т_{\text{отл}} \cdot S_{\text{мч}}, \quad (7.41)$$

где $Т_{\text{отл}}$ – трудоемкость отладки программы, ч;

$S_{\text{мч}}$ – стоимость машино-часа работы ПК, руб/ч.

Подставив все известные данные в формулу (7.41), получим:

$$З_{\text{от}} = 1003{,}20 \cdot 0{,}33 \approx 331{,}06 \text{ руб}, \quad (7.42)$$

Себестоимость разработки программного продукта определяется по формуле:

$$С_{\text{пр}} = З_{\text{рз}} \cdot F + З_{\text{от}}, \quad (7.43)$$

где F – коэффициент накладных расходов проектной организации без учета эксплуатации ПК.

Предположив, что $F = 1{,}18$, рассчитаем себестоимость разработки по формуле (7.43):

$$С_{\text{пр}} = 50\,792{,}23 \cdot 1{,}18 + 331{,}06 \approx 60\,265{,}89 \text{ руб} \quad (7.44)$$

7.5 Определение оптовой и отпускной цены программного продукта

Оптовая цена складывается из себестоимости создания ПП и плановой прибыли на программу. Оптовая цена определяется по формуле:

$$Ц_{\text{о}} = С_{\text{пр}} + П_{\text{р}}, \quad (7.45)$$

где $П_р$ – плановая прибыль на программу, руб.

Плановая прибыль на программу определяется по формуле:

$$П_р = С_{пр} \cdot Н_п, \tag{7.46}$$

где $Н_п$ – норма прибыли проектной организации.

Предположим, что норма прибыли организации составляет 0,27. Тогда плановая прибыль на программу составит:

$$П_р = 60\,265{,}89 \cdot 0{,}27 \approx 16\,271{,}79 \text{ руб} \tag{7.47}$$

Оптовая цена будет составлять:

$$Ц_о = 60\,265{,}89 + 16\,271{,}79 = 76\,537{,}68 \text{ руб} \tag{7.48}$$

Отпускная цена программы определяется по формуле:

$$Ц_{пр} = Ц_о + (З_{рз} + П_р) \cdot НДС, \tag{7.49}$$

где НДС – ставка налога на добавленную стоимость.

На сегодняшний день в Республике Беларусь НДС $= 34{,}6\%$, поэтому отпускная цена программы составит:

$$Ц_{пр} = 76\,537{,}68 + (50\,792{,}23 + 16\,271{,}79) \cdot 34{,}6\% \approx 99\,741{,}83 \text{ руб} \tag{7.50}$$

7.6 Определение годовых текущих затрат, связанных с эксплуатацией задачи

Для расчета годовых текущих затрат, связанных с эксплуатацией ПП, необходимо определить время решения данной задачи на ПК. Время решения задачи на ПК определяется по формуле:

$$Т_з = (T_{вв} + T_р + T_{выв}) \cdot \frac{1 + d_{пз}}{60}, \tag{7.51}$$

где $T_{вв}$ – время ввода в ПК исходных данных, необходимых для решения задачи, мин;

$T_р$ – время вычислений, мин;

$T_{выв}$ – время вывода результатов решения задачи, мин;

$d_{пз}$ – коэффициент, учитывающий подготовительно-заключительное время.

Время ввода в ПК исходных данных может быть определено по формуле:

$$T_{\text{вв}} = \frac{K_z \cdot H_z}{100}, \tag{7.52}$$

где K_z – среднее количество знаков, набираемых с клавиатуры при вводе исходных данных;

H_z – норматив набора 100 знаков, мин.

Ввод исходных данных в ПК подразумевает ввод парольного слова, которое будет в дальнейшем анализироваться. Для определенности возьмем среднее количество знаков, набираемых с клавиатуры, в размере 70 символов. Предположив, что норматив набора 100 знаков составляет 1 минут, получим $T_{\text{вв}}$:

$$T_{\text{вв}} = \frac{70 \cdot 1}{100} \approx 0{,}70 \tag{7.53}$$

Времена вычислений и вывода результатов составляют доли секунд, поэтому в сравнении с $T_{\text{вв}}$ равны нулю. Взяв величину $d_{\text{пз}} = 0{,}17$, рассчитаем время решения задачи:

$$\text{Т}_{\text{з}} = (0{,}70 + 0 + 0) \cdot \frac{1 + 0{,}17}{60} \approx 0{,}01 \text{ ч} \tag{7.54}$$

На основе рассчитанного времени решения задачи может быть определена заработная плата пользователя данного программного продукта. Затраты на заработную плату пользователя программного продукта определяются по формуле:

$$\text{З}_{\text{п}} = \text{Т}_{\text{з}} \cdot k \cdot t_{\text{чп}} \cdot \text{К}_{\text{пр}} \cdot \left(1 + \frac{\text{Н}_{\text{д}}}{100}\right) \cdot \left(1 + \frac{\text{Н}_{\text{соц}}}{100}\right), \tag{7.55}$$

где $\text{Т}_{\text{з}}$ – время решения задачи на ПК, ч;

k – периодичность решения задачи в течение года, раз/год;

$t_{\text{чп}}$ – среднечасовая ставка пользователя программы, руб.

Пользователями разрабатываемого продукта также будут являться разработчиками аналогичного уровня, поэтому $t_{\text{чп}} = t_{\text{чр}} = 10{,}20$ руб. Число использования фреймворка зависит от пользовательской базы, а также от особенностей приложения, в котором используется. Если принять, что ежедневно приложением пользуются от 700 до 1400 пользователей, а фреймворк используется каждый раз при использовании приложения, то $k = 402875$. Тогда затраты на заработную плату пользователя программного продукта составят:

$$\text{З}_{\text{п}} = 0{,}01 \cdot 402\,875 \cdot 10{,}20 \cdot 1{,}7 \cdot \left(1 + \frac{13}{100}\right) \cdot \left(1 + \frac{34{,}6}{100}\right) \approx 106\,253{,}42 \text{ руб} \tag{7.56}$$

В состав затрат, связанных с решением задачи включаются также затраты, связанные с эксплуатацией ПК. Затраты на оплату аренды ПК для решения задачи определяются по следующей формуле:

$$З_а = Т_з \cdot k \cdot S_{мч} \tag{7.57}$$

Стоимость одного машино-часа для пользователей и для разработчиков также будет совпадать по вышеупомянутой причине. Поэтому $З_а$ составит:

$$З_а = 0{,}01 \cdot 402\,875 \cdot 0{,}33 \approx 1329{,}49 \text{ руб} \tag{7.58}$$

Годовые текущие затраты, связанные с эксплуатацией задачи, определяются по формуле:

$$З_т = З_п + З_а = 106\,253{,}42 + 1329{,}49 = 107\,582{,}91 \text{ руб} \tag{7.59}$$

7.7 Определение годовых затрат при решении задачи с помощью аналога

Для расчета годовых затрат, связанных с эксплуатацией аналога, необходимо определить время решения данной задачи на ПК. Время решения задачи на ПК $Т_{за}$ определяется по формуле, аналогичной (7.51):

$$Т_{за} = (T_{ввa} + T_{ра} + T_{выва}) \cdot \frac{1 + d_{пза}}{60} \tag{7.60}$$

Ввод исходных данных в ПК в этом случае также подразумевает ввод парольного слова. Для определенности, возьмем эту величину K_{za} в размере 90 символов. Предположив, что норматив набора 100 знаков также составляет 1 минут, в соответствии с формулой (7.52) получим время ввода исходных данных для аналога $T_{ввa}$:

$$T_{ввa} = \frac{K_{za} \cdot H_z}{100} = \frac{90 \cdot 1}{100} \approx 0{,}90 \tag{7.61}$$

Время вычислений при использовании аналога $T_{ра}$ будет пренебрежительно малым. Время вывода результатов $T_{выва}$ зависит от скорости интернет соединения, поэтому будет колебаться в пределах от 1 секунды до 6 секунд, то есть составит приблизительно 0.05 минут.

Коэффициент $d_{пза}$, учитывающий подготовительно-заключительное время, составит 0.18. Рассчитаем время решения задачи для аналога:

$$Т_{за} = (0{,}90 + 0 + 0{,}05) \cdot \frac{1 + 0{,}18}{60} \approx 0{,}02 \text{ ч} \tag{7.62}$$

На основе рассчитанного времени решения задачи может быть определена заработная плата пользователя аналога программного продукта по формуле, аналогичной (7.55). Она будет отличаться от оригинальной только временем решения задачи, найденного в формуле (7.62):

$$З_{па} = Т_{за} \cdot k \cdot t_{чп} \cdot К_{пр} \cdot \left(1 + \frac{Н_{д}}{100}\right) \cdot \left(1 + \frac{Н_{соц}}{100}\right) \quad (7.63)$$

Затраты на заработную плату пользователя аналога программного продукта составят:

$$З_{па} = 0{,}02 \cdot 402\,875 \cdot 10{,}20 \cdot 1{,}7 \cdot \left(1 + \frac{13}{100}\right) \cdot \left(1 + \frac{34{,}6}{100}\right) \approx 212\,506{,}84 \text{ руб} \quad (7.64)$$

В состав затрат, связанных с решением задачи включаются также затраты, связанные с эксплуатацией ПК. Затраты на оплату аренды ПК для решения задачи определяются формуле, аналогичной (7.57). Тогда $З_{аа}$ составит:

$$З_{аа} = 0{,}02 \cdot 402\,875 \cdot 0{,}33 \approx 2658{,}98 \text{ руб} \quad (7.65)$$

Годовые текущие затраты, связанные с эксплуатацией аналога, определяются по формуле, аналогичной (7.59):

$$З_{та} = З_{па} + З_{аа} = 212\,506{,}84 + 2658{,}98 = 215\,165{,}82 \text{ руб} \quad (7.66)$$

7.8 Определение ожидаемого прироста прибыли в результате внедрения программного продукта

Ожидаемый прирост прибыли в результате внедрения задачи вместо ее аналога укрупненно может быть определен по формуле:

$$П_{у} = (З_{та} - З_{т}) \cdot \left(1 - \frac{С_{нп}}{100}\right), \quad (7.67)$$

где $С_{нп}$ – ставка налога на прибыль.

На момент расчетов ставка налога на прибыль составляла 18%. Прирост прибыли составит:

$$П_{у} = (215\,165{,}82 - 107\,582{,}91) \cdot \left(1 - \frac{18}{100}\right) \approx 88\,217{,}99 \quad (7.68)$$

7.9 Расчет показателей эффективности использования программного продукта

Для определения годового экономического эффекта от разработанной программы необходимо определить суммарные капитальные затраты на разработку и внедрения программы по формуле:

$$К_о = К_з + Ц_{пр}, \tag{7.69}$$

где $К_з$ – капитальные и приравненные к ним затраты;

$Ц_{пр}$ – отпускная цена программного продукта.

Капитальные и приравненные к ним затраты определяются по формуле:

$$К_з = Ц^б_{пк} \cdot \left(1 - x \cdot \frac{Н^а_{пк}}{100}\right) \cdot Т_з \cdot \frac{k}{Ф_{пк}}, \tag{7.70}$$

где $Ц^б_{пк}$ – балансовая стоимость комплекта вычислительной техники, необходимой для решения задачи, руб;

x – возраст используемого ПК, лет.

Предположив, что для решения задачи будет использован ПК с возрастом 1 год, рассчитаем капитальные затраты:

$$К_з = 1470{,}00 \cdot \left(1 - 1 \cdot \frac{10}{100}\right) \cdot 0{,}01 \cdot \frac{402\,875}{2440{,}00} \approx 70 \tag{7.71}$$

Суммарные капитальные затраты составят:

$$К_о = 70 + 99\,741{,}83 = 99\,811{,}83 \tag{7.72}$$

Годовой экономический эффект от замены одного программного продукта на аналогичный при обработке информации определяется по формуле:

$$ЭФ = П_у - E \cdot К_о, \tag{7.73}$$

где E – коэффициент эффективности, равный ставке по кредитам на рынке долгосрочных кредитов.

Приняв $E = 0{,}1$, получим значение годового экономического эффекта:

$$ЭФ = 88\,217{,}99 - 0{,}1 \cdot 99\,811{,}83 \approx 78\,236{,}81 \tag{7.74}$$

Срок возврата инвестиций определяется по формуле:

$$Т_в = \frac{К_о}{П_у} = \frac{99\,811{,}83}{88\,217{,}99} \approx 1{,}13 \tag{7.75}$$

Результаты расчета сводятся в таблице 7.1.

Таблица 7.1 – Технико-экономические показатели проекта

Наименование показателя	Величина показателя
1. Трудоемкость решения задачи, ч	1925,86
2. Периодичность решения задачи, раз/год	402 875
3. Годовые текущие затраты, связанные с решением задачи, руб	107 582,91
4. Отпускная цена программы, руб	99 741,83
5. Степень новизны программы	В
6. Группа сложности алгоритма	2
7. Прирост условной прибыли, руб	88 217,99
8. Годовой экономический эффект, руб	78 236,81
9. Срок возврата инвестиций, лет	1,13

В результате технико-экономического обоснования инвестиций в разработку программного продукта были получены следующие значения показателей их эффективности:

1. Прирост условной прибыли составит 88 217,99 руб.
2. Годовой экономический эффект составит 78 236,81 руб.
3. Все инвестиции окупаются за 1.13 года.

Таким образом, разработка данного программного продукта является эффективной с учетом того, что он разрабатывается как для нужд самого предприятия, так и для продажи сторонним компаниям. Инвестиции в его реализацию целесообразны.

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. Благодаря системному подходу к проектированию возможно дальнейшее ее улучшение и расширение функциональности в целом.

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.

[11] Install Docker Compose. Docker documentation [Электронный ресурс]. — Режим доступа : https://docs.docker.com/compose/install/. — Дата доступа : 15.05.2019.

ПРИЛОЖЕНИЕ А
(*обязательное)*

Спецификация

ПРИЛОЖЕНИЕ Б
(*обязательное)*

Ведомость документов