МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ПСИХОЛОГО-ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационные технологии»
Кафедра «Прикладная математика»

КУРСОВАЯ РАБОТА

«Базы данных»

(по дисциплине «…», по модулю «….» или научно-исследовательская работа)

на тему: «Консольная утилита для децентрализованного файлообмена Peas»

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование)

Направленность программы Информационные системы и базы данных
(наименование)

Группа 21 ИТ-МО(б/о)ИСБД-1

Студент _____ _____ А.В. Царюк
(подпись) (дата)

Объем заимствованного текста не превышает 20 %.

Допустить к защите _____ И.С. Колотовкин
(подпись)

Руководитель курсовой работы _____ _____ И.С. Колотовкин
(подпись) (дата)

Москва, 2023

Оглавление

ВВЕДЕНИЕ

В современном мире существует множество информационных систем: базы данных (MySQL, MongoDB, PostgreSQL, ElasticSearch), платформы асинхронного event-driven[1] общения (Apache Kafka, RabbitMQ, Nats.IO), крупные игровые проекты (Genshin Impact, GTA 5), механизмы автоматического сохранения логов[2] и бэкапов[3] (Fluent Bit, via Rsync, AutoBackup) и множество других. Все они работают с огромными наборами чувствительных данных, поэтому эти системы обязаны обеспечивать консистентность и доступность.

1) Базы данных реализуют механизмы репликации[4], которые создают избыток данных, чтобы в случае отказа одного из серверов, другие могли подхватить входящие запросы

2) Системы резервного копирования сохраняют состояние чего-либо в огромный файл и должны отправить его на несколько серверов (желательно в разных ЦОДах[5]) чтобы в случае потери оригинала, можно было восстановить состояние системы

3) Крупные игровые проекты создают клиентские приложения порядком ~50GB и в момент выпуска обновления на еще ~20GB приходиться тратить ресурсы на увеличение объемов раздачи

Если абстрагироваться, то мы имеем некоторую систему, которая состоит из источников большого объема данных и клиентов, желающих эти данные получить. Основная проблема заключается в том, что способ обмена в таких системах - одноканальный и централизованный.

Более конкретные проблемы:

1) critical: Во время множественного скачивания тяжелого файла с одного или нескольких серверов, может произойти отказ в обслуживании из-за высокой или неравномерной нагрузки;

2) critical: В случае завершения работы дистрибьютора, entrypoint[6] для загрузки будет утерян;

3) high: Сервер может находиться в другом регионе, что сильно повышает задержки;

4) high: Протокол загрузки полностью зависит от дистрибьютора, поэтому может быть использован неоптимальный.

Централизованная раздача файла представлена на рис.1.

---

[1] Oracle Corporation. "Event Driven Architecture Overview", 2016, с. 14.
[2] J. Kreps. "The Log: What Every Software Engineer Should Know About Real-time Data's Unifying Abstraction", 2013, с. 8.
[3] W. Curtis Preston. "Managing Backup and Recovery Systems: Design and Implementation", 2003, с. 1-25.
[4] G. Coulouris, J. Dollimore, and T. Kindberg. "Distributed Systems Concepts and Design", 2005, с. 448-461.
[5] Scott E. Donaldson, Stanley Siegel, and Chris Kankiewicz. "Designing Data Centers: How to Plan and Manage a Reliable, Effective Data Center", 2010, с. 1-22.
[6] Richardson, Chris. "Microservices: Patterns and Practices", 2015, с. 88-91.

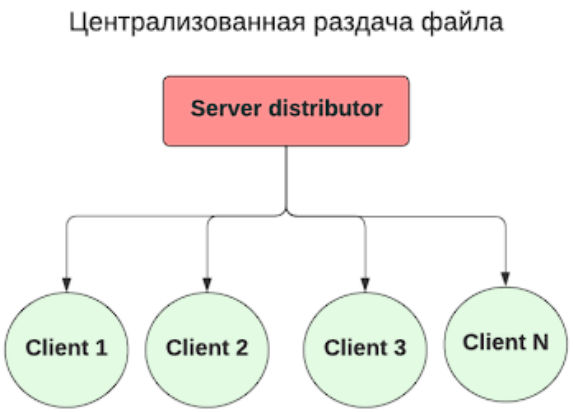

Рис.1. Централизованная раздача файла

Актуальность. Чтобы полноценно понять актуальность работы, нужно с одной стороны рассмотреть готовые решения, а с другой стороны оценить в каких местах никакое из решений не было применено.

Решения: готовое решение проблемы уже есть, протокол - BitTorrent, однако клиенты, реализующие данный протокол, чаще всего - готовый продукт с графическим пользовательским интерфейсом (GUI), что может быть неудобно для автоматизированных систем обмена, так как это накладные расходы (overhead). Поэтому есть смысл создать собственное решение под интерфейс командной строки (CLI), которое будет мало весить и легко запускаться на большом количестве удаленных хостов.

Потребители: рассмотрим уже описанные выше системы и оценим, в каких опыт децентрализованной раздачи не применен.

1) Оптимизация механизма репликации баз данных (например в Cassandra[7], есть только режимы single, ring)
2) Ускорение отправки snapshot версий на несколько backup серверов (например результаты аналитики big data)
3) Выпуск большого обновления клиентской части какой-либо игры
4) Быстрая загрузка первичного ПО на только что купленные хосты с ansible

[7] Datastax. "Datastax Documentation: Replication and Consistency", 2021, с. 1-5.

Цели и задачи

Цель работы

Целью работы является решить проблемы одиночного (standalone[8]) дистрибьютора путем создания информационного продукта Peas CLI.

Задачи

1. Сравнить производительность централизованного и децентрализованного архитектур файлообмена
2. Сформировать требования для нового технологического продукта
3. Обозначить основные объекты и их связи внутри продукта
4. Конкретизировать внутреннюю архитектуру продукта
5. Выбрать подходящие инструменты для реализации
6. Решить или обозначить вопросы связанные с безопасностью
7. Обезопасить работу приложения анализом работы в конкурентной среде
8. Обозначить формат разработки и согласованности изменений с течением процесса разработки
9. Собрать продукт в автоматизированной системе и описать ручную версию процесса
10. Написать Unit-тесты и провести ручное тестирование
11. Сделать простые команды для пользователя и описать их

Более детально: необходимо создать несколько источников раздачи файла, однако тогда получатель будет знать все хосты, которые уже получили фрагменты файла (далее будет использовано слово - партиция). Это необходимый компромисс (trade-off) для достижения параллельного получения фрагментов. По сути, каждый клиент становится сервером раздачи (трекером). Так же нужно разработать механизмы балансировки нагрузки, например: выбор оптимального количество соединений на каждом трекере, выбор трекеров с наибольшей пропускной способностью сети. Нужно описать и создать модель общения клиента peas с продуктом через CLI.

[8] Jain, Gaurav. "Standalone Application vs Web Application", 2017, с. 1-5.

Глава 1. Проектирование

1.1 Проверка проблемы

Чтобы оценить централизованную и децентрализованные архитектуры файлообмена и получить приблизительные теоретические показатели производительности было проведено моделирование этих вариантов в Jupyter Notebook на языке программирования Python.

По оси абсцисс - количество участников файлообмена, изначально только один пользователь обладает всеми партициями, по ординатам - условные единицы времени (атомарных операций обмена).

Иллюстрация сравнения скорости двух архитектур файлообмена в зависимости от числа участников в сети представлена на рис.2.

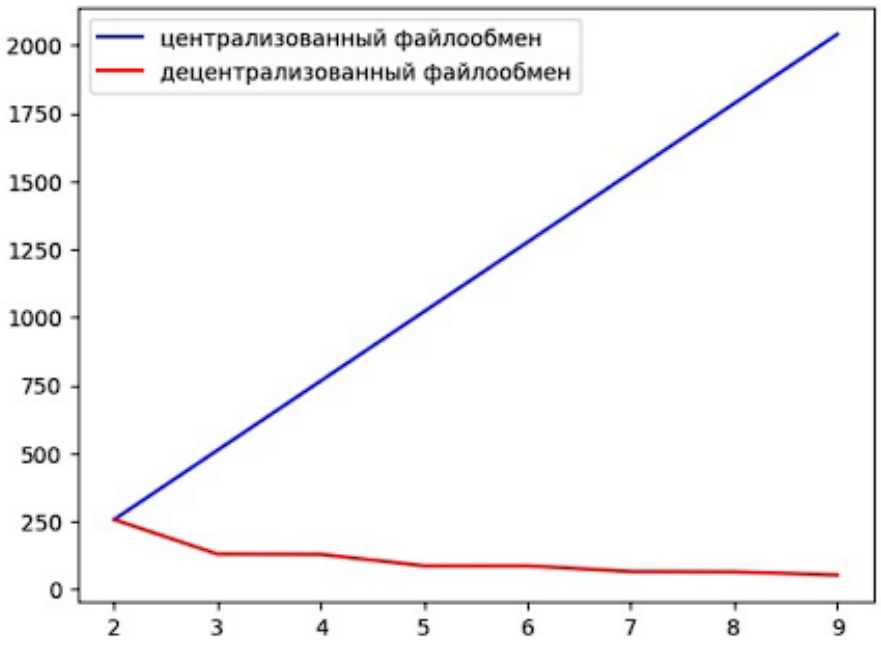

Рис.2. Иллюстрация сравнения скорости двух архитектур файлообмена в зависимости от числа участников в сети

Оказалось, что централизованная раздача файла - линейно возрастающая, а децентрализованная - гипербола.

Децентрализованное отношение скорости к числу участников сети представлено на рис.3.

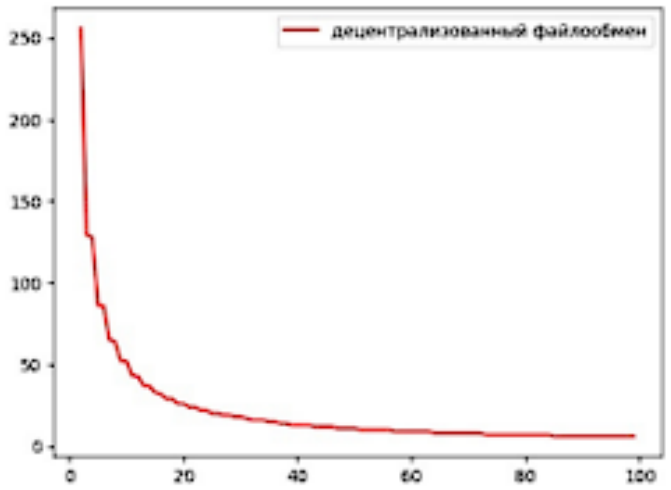

Рис.3. Децентрализованное отношение скорости к числу участников сети

Централизованное отношение скорости к числу участников сети представлено на рис.4.

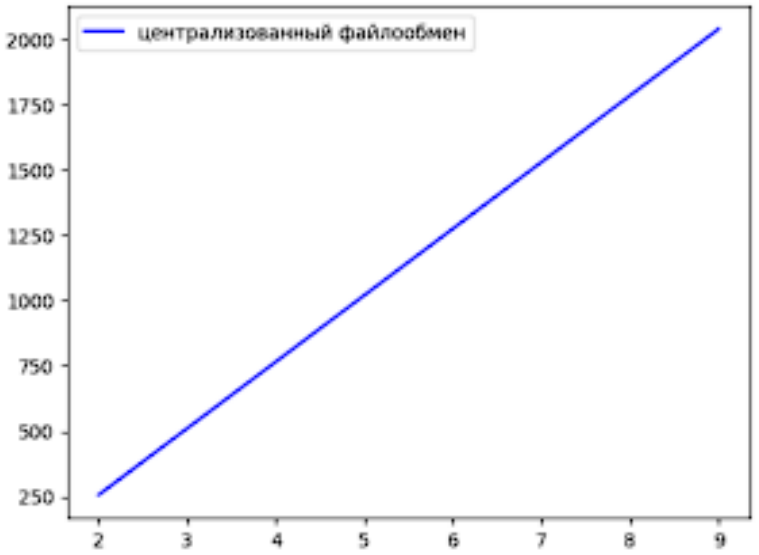

Рис.4. Централизованное отношение скорости к числу участников сети

Именно данные результаты, отражают главный плюс децентрализованного файлообмена - скорость раздачи. Однако, данное моделирование упускает множество важных параметров реальной среды, а именно: разница в скоростях каналов передачи, пропажа пакетов, изменяемость параметров сети, количества участников и много другое, но для грубого вывода этого достаточно.

1.2 Формирование требований

Чтобы продукт был полезен и удобен, он должен выполнять следующие требования:

1) продукт должен быть доступен на абсолютном большинстве операционных систем, а именно Unix (Linux, MacOS), Windows;

2) быть легковесным для моментальной загрузки, BitTorrent весит 2-3 мегабайта, поэтому наш дистрибутив должен не уступать и весить менее мегабайта;

3) так как мы реализовываем одноранговую[9] (P2P) сеть, то любой может подменить партицию на свою, поэтому необходимо реализовать механизм защиты через проверку контрольных хэш-сумм партиций;

4) в некоторых ситуациях не выгодно подключаться ко всем серверам, так как их скорость соединения неравномерна, намного лучше будет выбирать самые оптимальные трекеры;

5) тоже можно сказать и про количество сессий, по сути, это параметр, в котором нужно найти золотую середину чтобы добиться максимальной скорости, в программировании очень много похожих проблем, например количество потоков в пуле;

6) утилитой будет проблематично пользоваться если в один момент не получится устанавливать N>1 файлов, поэтому должна быть поддержка параллельных загрузок маленьких партиций;

7) как и в требовании номер 3, злоумышленник может записать IP адреса в координирующий файл (peas файл), поэтому должна быть реализована защита;

8) так как это консольная утилита, то она работает в рамках процесса консоли, поэтому в случае остановки процесса пользователем, выполнение встанет. Нужно запускать процесс в фоновом режиме, то есть реализовать так называемый daemon-process;

9) чтобы администраторы могли быстро использовать данный продукт, нужно подготовить примеры заклинаний (cookbook) для системы управления конфигурациями (в нашем случае ansible), это нужно для простоты получения первичного опыта использования;

10) нужно использовать самый оптимальный транспортный и прикладной протокол для обеспечения обмена пакетами чтобы не тратить время на лишние операции. HTTP на эту роль плохо подходит из-за накладных расходов на Handshake[10] пакет.

---

[9] Van Steen, Maarten. "An Overview of Peer-to-Peer Networks", 2011, с. 1-47.

[10] Gourley, David, and Totty, Brian. "HTTP: The Definitive Guide", 2002, с. 17-20.

Таблица тестовых релизов и требований (план работ)

Таблица 1

Требование/релиз	**01.05.2023**	**15.06.2023**	**15.07.2023**
1 поддержка unix/windows	+	+	+
2 размер	+	+	+
3 защита от подмены партиции	-	Линейная	Алгоритм Меркла
4 выбор быстрых трекеров	-	+	+
5 оптимизация кол-ва сессий	-	+	+
6 поддержка нескольких загрузок	-	+	+
7 защита от спама адресами	-	-	+
8 daemon[11] процесс	-	-	+
9 готовые cookbook для ansible	-	-	+
10 сетевые протоколы	http	http/tcp	http/udp/tcp[12]

1.3 Главный примитив

Peas-файл - главный примитив данной системы, по сути, это файл, который хранит списки IP-адресов у кого можно скачать нужные

[11] Kerrisk, Michael. "The Linux Programming Interface: A Linux and UNIX System Programming Handbook", 2010, с. 1300-1303.
[12] Stevens, W. Richard. "TCP/IP Illustrated", 1994, с. 47-51, 155-163.

данные. Именно его нужно создать для начала раздачи файлов и его нужно получить чтобы эти файлы скачать.

Чтобы не путать файл раздачи и .peas файл, будем обозначать последний по его расширению .peas.

Чтобы максимально сильно уменьшить файл, нужно использовать какой-либо бинарный формат хранения, например protobuf от компании google.

Содержимое .peas файла:

1) абсолютный путь к исходному файлу;
2) хэш-сумма файла;
3) список IP-адресов трекеров;
4) размер файла;
5) список партиций [int64, локальное расположение или null, хэш-сумма];
6) дата создания горошины;
7) опционально: лимит времени на раздачу.

Пример содержимого файла в формате JSON (рис.4)

```
{
 path: "C://Users/func/Desktop/videos.zip", // фактический файл
 hashsum: [
  "7DD987F846400079F4B03C058365A4869047B4A0",
  "65A48690479F4B03C058365A4869047B4A0AHJDK"
 ],
 ip: [
  "192.162.0.1",
  "0.0.0.0",
  "127.0.0.1"
 ],
 size: 2048, // 2 килабайта
 partitions: [
  {
   id: 1,
   path: "null", // партиция не скачана
   hash: "7DD987F846400079F4B03C058365A4869047B4A0"
  },
  {
   id: 2,
   path: "null",
   hash: "65A48690479F4B03C058365A4869047B4A0AHJDK"
   Курсовая работа 8
  }
 ],
 timestamp: 1682428172,
 timelimit: 600000 // 10 минут
}
```

Рис.4. Пример содержимого файла в формате JSON

1.4 Архитектура

Так как это одно консольное приложение, то архитектура не вызывает сложностей. В финальном варианте приложения должно быть 4 модуля: сервер - для раздачи партиций, клиент - для их получения, CLI агент - для пользовательского интерфейса, модуль балансировки - для выбора оптимального количества сессий и потоков.

Связи модулей представлены на рис.5.

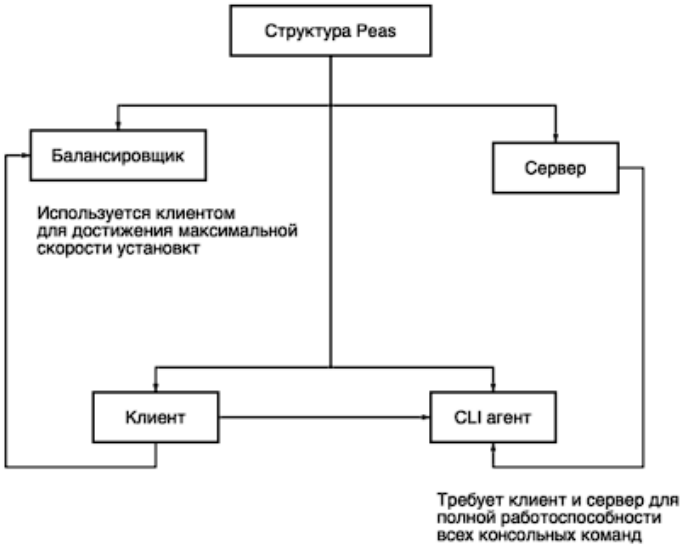

Рис.5. Связи модулей

Глава 2. Конструирование

2.1 Реализация

2.1.1 Подбор инструментов

Чтобы начать реализовывать продукт, отвечающий всем требованиям к архитектуре, безопасности и удобству, нужно выбрать надлежащие инструменты.

1) Java - это язык программирования, который я хорошо знаю и который имеет широкую поддержку в различных операционных системах (требование 1). Мною был выбран язык Java, потому что это надежный и стабильный язык программирования, который позволяет создавать кроссплатформенные приложения.

2) Kryo - это библиотека сериализации объектов, которая позволяет быстро и компактно преобразовывать объекты Java в бинарный формат (требование 6). Я выбрал Kryo, потому что она обеспечивает высокую производительность и эффективность при передаче данных по сети.

3) Netty[13] - это фреймворк для разработки сетевых приложений на Java. Был выбран Netty, потому что он обеспечивает высокую производительность и масштабируемость (требование 6, 10) при работе с сетевыми протоколами.

4) Gradle - это система автоматической сборки проектов, которая позволяет управлять зависимостями и конфигурацией проекта. Ее заслуги в том, что он обеспечивает удобную и гибкую настройку проекта.

5) Picocli - это библиотека для создания командной строки в Java (требование 8). Она обеспечивает удобство и гибкость в создании интерфейса командной строки для приложения.

2.1.2 Выбор сетевого протокола

Был выбран протокол TCP для децентрализованной раздачи файлов, потому что он обеспечивает надежную и гарантированную доставку данных. TCP использует механизм управления потоком, который позволяет контролировать скорость передачи данных и гарантировать, что все данные будут доставлены в нужном порядке. Это особенно важно при передаче больших файлов, где потеря данных может привести к необходимости повторной передачи всего файла.

UDP, в свою очередь, обеспечивает быструю передачу данных без гарантии доставки и без контроля порядка доставки. UDP используется

[13] N. Timo. "Netty in Action", 2015, с. 1-15.

для передачи потоковых данных, таких как аудио и видео, где потеря нескольких пакетов не является критической.

Таким образом, выбор протокола TCP для децентрализованной раздачи файлов является лучшим вариантом из-за его надежности и гарантированной доставки данных.

Однако, выбор протокола следует реализовать и уже на деле убедиться в правильности решения, тем более Netty (выбранная библиотека для сетевого взаимодействия) позволяет это сделать.

2.1.3 Безопасность

Если с централизованным вариантом все понятно, а именно использование в обмене TLS (Transport Layer Protocol) v1.2+, то в нашем случае открывается множество новых возможностей для компрометации. То что лежит на поверхности, это обман клиента и передача ему вредного .peas файла, это проблему не устранить наверняка. Подобная проблема близка к доверию, надежности и всему тому, что не гарантированно. Однако, ее можно свести к минимуму проработанными способами, а именно: централизация мест первичного получения .peas файлов, что-то наподобие банка-глоссария. Это в свою очередь, выходит за рамки поставленных задач.

1) Один из участников сети может прислать нам неправильную партицию, это решается профицитом доверия к поставщику .peas файла, в нем содержится контрольная сумма всех файлов, а значит, каждая полученная патриция подлежит сверке на месте вычисленной контрольной суммой с ожидаемой. Контрольную сумму будем считать через hash-функцию XXH3[14], она является криптографически стойкой и не выделяет памяти в процессе вычисления, поэтому она полностью подходит под нашу задачу.

Пример реализации в коде представлен на рис.6.

```
long resHash = xx3().hashBytes(
    res.array(),
    res.readerIndex(),
    (int) partSize
);
var expectedHash = file.partitions().get(part);

assert resHash == expectedHash;
```

Рис.6. Пример реализации в коде

[14] Collet, Yann. "XXH3: Faster Hashing at Almost Zero Cost", 2019, с. 1-15.

2) .peas файл модифицируется новыми владельцами, это нужно чтобы повысить эффективность сети, однако участник может записать себя сколько угодно раз с разных IP-адресов и забить файл мишурой. Данную проблему пока оставим без решения, однако приходят на ум валидация нового IP-адреса через аналога из мира криптовалюты Proof Of Stake[15] - проверка через доверие крупным серверам раздачи.

Также, важно отметить о потере такого свойства файлообмена как анонимность передачи, это обусловлено архитектурой и не подлежит радикальному решению.

2.1.4 Параллелизм

В программировании часто бывают проблемы с различными проявлениями параллелизма: гонка данных (race condition[16]), API race. Для того чтобы недопустимо подобного, рассмотрим все возможные моменты в соответствии со Java Memory Model[17]:

1) словари активных каналов загрузки и очереди партиций на получение реализованы благодаря классу ConcurrentMap, поэтому при работе с данными, они блокируются, а не в целом весь словарь;

2) так как партиции устанавливаются параллельно, то чтобы сверять количество уже полученных - используется счетчик AtomicInteger [18]основанный на CMS (атомарная операция Compare And Set[19]), поэтому все “вклинивания” между чтением и записью будут откинуты;

3) для блокирования потока до момента полной установки файла используется Phaser, однако на самом деле он тут не нужен, достаточно будет вызвать событие завершение и протянуть его каким-нибудь eventbus (библиотека для событийного программирования);

4) для вывода будет выделен отдельный поток через инструмент Executors.newSingleThreadScheduledExecutor();

5) для поддержания сессий будет использован инструмент Executors.newCachedThreadPool(), он возьмет на себя задачу определения оптимального количества активных потоков ОС по требованию 5.

[15] Buterin, Vitalik. "Proof of Stake versus Proof of Work", 2014, с. 1-7.
[16] M. Ben-Ari. "Principles of Concurrent and Distributed Programming", 2006, с. 93-105.
[17] J. Bloch. "Effective Java: Programming Language Guide", 2008, с. 267-289.
[18] B. Goetz, et al. "Java Concurrency in Practice", 2006, с. 1-15.
[19] Intel Corporation. "Overview of Lock-Free Design: The Compare and Swap Instruction", 2003, с. 1-5.

2.1.5 Git flow

В разработке этого продукта участвует два человека, поэтому большого смысла в разделении веток и просмотру коду с комментариями со работника (code review) не требуется через механизм запроса вливания изменений (merge request), на эту роль лучше подойдет прямое общение или через мессенджер.

Цикл разработки первого рабочего варианта представлен на рис.7.

Рис.7. Цикл разработки первого рабочего варианта

2.2 Сборка продукта

Сборка продукта Автоматизированная Так проект размещен на ресурсе GitHub, то при каких-либо изменениях он собирается автоматически для проверки успешности компиляции благодаря GitHub Workflow[20].

Конфигурация описывающая сборку с комментариями представлена на рис.8.

```
name: Java CI with Gradle

on:
  push:
    branches: [ "main" ] # при новом изменении в ветке main выполнять сборку

permissions:
  contents: read # указываем доступные права нашему процессу сборки (чтение)

jobs:
  build:
    runs-on: ubuntu-latest # будем собирать в операционной системе Ubuntu
    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with: # используем эту версию Java Development Kit
        java-version: '17'
        distribution: 'temurin'
    - name: Build with Gradle
      uses: gradle/gradle-build-action@67421db6bd0bf253fb4bd25b31ebb98943c375e1
      with:
        arguments: build # вызываем систему сборки с аргументом на сборку компонента
```

Рис.8. Конфигурация описывающая сборку с комментариями

[20] Daigle, Kyle. "Understanding the GitHub Workflow", 2018, с. 1-6.

Шаг 1: для сборки проекта на новом устройстве или не подготовленной среде, нужно для начала установить проект командой (для выполнения нужно скачать Git Bash[21])

Команда установки представлена на рис.9.

```
git clone git@github.com:funcid/peas-cli.git
```

Рис.9. Команда установки

Шаг 2*:* После установки, нужно открыть директорию проекта и выполнить команду сборки и запуска Unit тестирования.

Команда сборки и запуска Unit тестирования представлена на рис.10.

```
cd peas-cli
gradlew build -Dorg.gradle.java.home="C:\Program Files\Java\jdk-17.0.2"
```

Рис.10. Команда сборки и запуска Unit тестирования

Шаг 3: Переходим в нужную директорию и запустим наше приложение

Команда перехода в директорию и запуска приложения представлена на рис.11.

```
cd build/libs
java -jar peas-parent-all.jar
```

Рис.11. Команда перехода в директорию и запуска приложения

2.3 Тестирование

Это очень разносторонняя тема, различных видов тестирования много, конкретно в данном случае, для проверки минимального рабочего варианта хватит unit и ручного тестирования.

В программе присутствует класс для больших чисел вне стандартного хранилища объектов в Java (heap, куча), его использование сильно повышает производительность, однако, может повлечь экстренное завершение приложения с фатальной ошибкой, которую нельзя обработать. Так как, в данном случае, роль сборщика мусора (механизм очистки памяти - garbage collector[22]) лежит на нас

[21] J. Loeliger and M. McCullough. "Version Control with Git", 2012, с. 1-8.
[22] Oracle Corporation. "Garbage Collection Basics", 2021, с. 1-8.

из-за off heap[23] аллокации памяти, то нужно проводить unit-тестирование[24] перед каждой новой версией приложения.

Для ручного тестирования создается специальная среда, с несколькими виртуальными машинами (или отдельными серверами если есть средства), на которых размещается и запускается данное приложение с разными начальными параметрами, некоторые выступают в роли раздатчиков, а другие в роле приемников.

Создание пары из клиента и сервера представлено на рис.12.

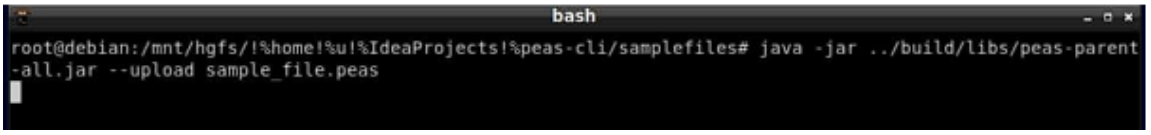

Рис.12. Создание пары из клиента и сервера

2.4 Функции для командной строки (CLI[25])

Функции для командной строки (CLI) Были сделаны следующие команды для пользовательского интерфейса:

- peas [-hu] [<файлы>...]
- Загрузить/выгрузить файл
- [<файлы>...] Файлы, которые нужно загрузить/выгрузить
- -h, --help Показать это сообщение
- -u, --upload Режим выгрузки
- peas create [-h] [-s=SIZE] [-w=OWNERS]... <файл>
- Создать .peas файл
- <файл> Файл, для которого должен быть создан .peas файл
- -h, --help Показать это сообщение
- -s, --part-size=SIZE Размер блока
- -w, --owners=OWNERS Владельцы файла (трекеры по умолчанию

[23] Oaks, Scott. "Java Performance: The Definitive Guide", 2018, с. 505-509.
[24] G. Meszaros. "xUnit Test Patterns: Refactoring Test Code", 2007, с. 1-15.
[25] J. Turnbull. "The Art of Command Line", 2017, с. 1-215.

Заключение

Проблема файлообмена с одним источником данных и распространением их всем остальным самым оптимальным и удобным способом - была решена. Смоделировано сравнение двух подходов к файлообмену, в котором было доказано преимущество децентрализованного по скорости раздачи. Было создано минимальное работоспособное консольное приложение с открытым исходным кодом (open source), с помощью которого можно создать описательный файл (.peas) и раздавать/принимать кусочки этого файла параллельно. Были реализованы базовые защитные механизмы, которые не дадут навредить клиентам. Приложение может быть использовано для задач массовой раздачи, например репликация таблиц, обновление компьютерных игр, первичная установка ПО на выделенные сервера и многое другое.

Сравним разработанный продукт с аналогами, BitTorrent решает похожие задачи, однако клиенты не консольные. Rsync синхронизирует данные на разных машинах, однако делает это не оптимальным способом и через SSH протокол, который предназначен для администрирования сервером, поэтому при активной работе, будут проблемы с скоростью взаимодействия администратора и сервера. Остальные аналоги рассматривать нет смысла, так как ответные аргументы схожи.

В дальнейшем нужно применить данный инструмент в автоматизированных системах на практике. Повысить безопасность, устранив известные проблемы, например: доверие ко всем новым владельцам файла. Нужно написать подробную документацию на команды и показать реальные замеры производительности по сравнению с web конкурентами. Также, не помешает пример использования продукта с системой управления конфигурациями - Ansible. И на самом деле, было написано мало тестов, лучше использовать библиотеку Test Containers для имитирования реальных ситуаций и проверки их правильности исполнения.

Список литературы

1. Oracle Corporation. "Event Driven Architecture Overview", 2016, с. 1-2.
2. Тузовский, А. Ф. Проектирование и разработка web-приложений: учебное пособие для среднего профессионального образования / А. Ф. Тузовский. - Москва, Издательство Юрайт, 2022. - 218 с.
3. J. Kreps. "The Log: What Every Software Engineer Should Know About Real-time Data's Unifying Abstraction", 2013, с. 1-8.
4. W. Curtis Preston. "Managing Backup and Recovery Systems: Design and Implementation", 2003, с. 1-25.
5. G. Coulouris, J. Dollimore, and T. Kindberg. "Distributed Systems Concepts and Design", 2005, с. 448-461.
6. Scott E. Donaldson, Stanley Siegel, and Chris Kankiewicz. "Designing Data Centers: How to Plan and Manage a Reliable, Effective Data Center", 2010, с. 1-22.
7. Richardson, Chris. "Microservices: Patterns and Practices", 2015, с. 88-91.
8. Schneier, Bruce. "The Overhead of Security", 2009, с. 1-5.
9. Datastax. "Datastax Documentation: Replication and Consistency", 2021, с. 1-5.
10. Введение в СУБД MySQL: курс лекций / - Москва, Интуит НОУ, 2016. - 228 с.
11. Свистунов, А.Н. Построение распределенных систем на Java: курс лекций / Свистунов А.Н. - Москва, Интуит НОУ, 2016. — 317 с.
12. Вязовик, Н.А. Программирование на Java: курс лекций / Вязовик Н.А. - Москва, Интуит НОУ, 2016. - 603 с.
13. Гудсон, Джон Практическое руководство по доступу к данным / Джон Гудсон, Роб Стюард. - М.: БХВ-Петербург, 2018. - 304 с.
14. Винкоп, Стефан Использование Microsoft SQL Server 7.0. Специальное издание / Стефан Винкоп. - М.: Вильямс, 2017. - 816 с.
15. Кригель, А. SQL. Библия пользователя / А. Кригель. - М.: Диалектика / Вильямс, 2019. - 318 с.
16. Jain, Gaurav. "Standalone Application vs Web Application", 2017, с. 1-5.
17. Van Steen, Maarten. "An Overview of Peer-to-Peer Networks", 2011,

c. 1-47.
18. Gourley, David, and Totty, Brian. "HTTP: The Definitive Guide", 2002, c. 17-20.
19. Kerrisk, Michael. "The Linux Programming Interface: A Linux and UNIX System Programming Handbook", 2010, c. 1300-1303.
20. Stevens, W. Richard. "TCP/IP Illustrated", 1994, c. 47-51, 155-163.
21. Collet, Yann. "XXH3: Faster Hashing at Almost Zero Cost", 2019, c. 1-15.
22. Buterin, Vitalik. "Proof of Stake versus Proof of Work", 2014, c. 1-7.
23. Daigle, Kyle. "Understanding the GitHub Workflow", 2018, c. 1-6.
24. Oaks, Scott. "Java Performance: The Definitive Guide", 2018, c. 505-509.
25. Oracle Corporation. "Garbage Collection Basics", 2021, c. 1-8.
26. G. Meszaros. "xUnit Test Patterns: Refactoring Test Code", 2007, c. 1-15.
27. J. Loeliger and M. McCullough. "Version Control with Git", 2012, c. 1-8.
28. Intel Corporation. "Overview of Lock-Free Design: The Compare and Swap Instruction", 2003, c. 1-5.
29. B. Goetz, et al. "Java Concurrency in Practice", 2006, c. 1-15.
30. M. Ben-Ari. "Principles of Concurrent and Distributed Programming", 2006, c. 93-105.
31. J. Bloch. "Effective Java: Programming Language Guide", 2008, c. 267-289.
32. N. Timo. "Netty in Action", 2015, c. 1-15.
33. J. Turnbull. "The Art of Command Line", 2017, c. 1-215.