# Apache Hadoop

Архитектурный документ

**Команда:** Team №2

**Авторы документа:**

Хорасанджян Левон БПИ218

Черкасский Виталий БПИ216

**Преподаватель группы:**
Бегичева Антонина Константиновна

**Учебный ассистент:**
Денис Савенко Андреевич

# 0. Раздел регистрации изменений

| Версия документа | Дата изменения | Описание изменения | Автор изменения |
|---|---|---|---|
| 1.0.0 | 27.03.24 | Создание архитектурного документа | Хорасанджян Левон БПИ218 |
| 1.1.0 | 28.03.24 | Написан блок “1. Введение” | Черкасский Виталий БПИ216 |
| 1.2.0 | 28.03.24 | Написан блок “2. Архитектурные факторы” | Хорасанджян Левон БПИ218 |
| 1.3.0 | 31.03.24 | Написан блок “3. Общее архитектурное решение” | Хорасанджян Левон БПИ218 |
| 1.4.0 | 31.03.24 | Написан блок “5. Технические описания отдельных ключевых архитектурных решений ” | Черкасский Виталий БПИ216 |
| 1.4.1 | 31.03.24 | Написаны блоки:<br>● “4.1. Представление прецедентов”<br>● “4.2. Логическое представление ”<br>● “4.5. Представление развертывания”<br>● “4.7. Представление архитектуры безопасности”<br>● “4.9. Представление производительности” | Черкасский Виталий БПИ216 |
| 1.4.2 | 02.04.24 | Написаны блоки:<br>● “4.3. Представление архитектуры процессов”<br>● “4.4. Физическое представление архитектуры”<br>● “4.6. Представление архитектуры данных”<br>● “4.8. Представление реализации и разработки”<br>● “4.10. Атрибуты качества системы” | Хорасанджян Левон БПИ218 |
| 1.5.0 | 02.04.24 | Дополнен блок “6. Приложения” | Черкасский Виталий БПИ216 |
| 1.6.0 | 03.04.24 | Обновлен блок “6. Приложения” | Хорасанджян Левон БПИ218 |

# 1. Введение

## 1.1. Название проекта

Apache Hadoop

## 1.2. Задействованные архитектурные представления

## 1.3. Контекст задачи и среда функционирования системы

Разрабатываемая система на базе Apache Hadoop представляет собой распределенную вычислительную платформу, спроектированную для обработки и хранения больших объемов данных в распределенной среде. Цель разработки данного приложения - предоставление пользователям мощного инструмента для обработки и анализа большого объема данных, что является критической необходимостью для множества современных приложений.

Среда функционирования системы включает в себя ключевые компоненты:

1. **Hadoop Distributed File System (HDFS)** - для хранения данных в распределенной файловой системе;
2. **Apache YARN** - для управления ресурсами и выполнения задач;
3. **MapReduce** - для эффективного распределения задач обработки данных (Map) и слияния результатов (Reduce).

Эти компоненты обеспечивают среду для обработки данных с использованием параллельных вычислений на кластере серверов.

## 1.4. Рамки и цели проекта

Основная цель проекта - создание интегрируемого, отказоустойчивого и масштабируемого инструмента, предоставляющего функционал для анализа данных, включая обработку структурированных и неструктурированных данных, с использованием распределенных вычислений на кластере Apache Hadoop. Проект направлен на упрощение процесса обработки данных и повышение производительности аналитических задач.

Ключевые задачи проекта включают в себя:

1. Обеспечение высокой производительности и эффективного использования ресурсов кластера для обработки данных;
2. Разработка и оптимизация алгоритмов для обработки данных различной природы и форматов;
3. Предоставление пользователю гибкого интерфейса для загрузки данных, настройки задач обработки и мониторинга процесса выполнения.

# 2. Архитектурные факторы

## 2.1. Ключевые заинтересованные лица

| **Действующее лицо** | **Заинтересованность в системе** |
|---|---|
| Пользователь | Простота использования, эффективная работа с данными, надежность |
| Разработчик | Простота разработки и поддержки приложения с минимальными затратами времени и ресурсов |
| Администратор | Высокая производительность, отказоустойчивость, совместимость с различными языками программирования, удобство интеграции |

## 2.2. Ключевые требования к системе

Из ключевых требований можно особенно выделить следующие:

1. **Масштабируемость:** система должна способствовать обработке и хранению больших объемов данных с возможностью горизонтального масштабирования;
2. **Отказоустойчивость:** система должна обеспечивать автоматическое восстановление и продолжение работы в случае сбоев узлов;
3. **Производительность:** система должна обеспечивать высокую производительность при обработке данных в распределенной среде.

## 2.3. Ключевые ограничения

При разработке Apache Hadoop учитывался ряд ограничений, которые влияют на выбор архитектурных решений и функциональности. В данном контексте важно ознакомиться с ключевыми ограничениями, которые определяют рамки и параметры проектирования и разработки:

1. **Масштабируемость кластера:** размер кластера Apache Hadoop ограничен физическими ресурсами, доступными для построения и поддержки кластера. Это ограничение определяет максимальное количество узлов в кластере и его общую производительность.
2. **Технические ограничения:** возможные ограничения по производительности и масштабируемости, обусловленные аппаратными характеристиками узлов кластера, сетевой инфраструктурой и ограничениями операционной системы.
3. **Ограничения на безопасность:** необходимость соблюдения политик безопасности и конфиденциальности данных может ограничивать возможности доступа к данным и настройки безопасности кластера.

# 3. Общее архитектурное решение

При разработке Apache Hadoop использовался принцип модульной архитектуры, состоящей из различных подпроектов. Каждый подпроект выполняет определенные функции и предоставляет возможности для работы с данными в распределенной среде. В проекте Apache Hadoop имеются следующие ключевые модули системы:

1. **hadoop-assemblies**: модуль, отвечающий за сборку различных компонентов приложения, включая JAR файлы, необходимые для запуска кластера Hadoop;
2. **hadoop-build-tools**: модуль, предоставляющий инструменты и скрипты для автоматизации сборки и тестирования проекта Hadoop;
3. **hadoop-client-modules**: модуль, содержащий клиентские библиотеки и инструменты для взаимодействия с кластером Hadoop из клиентских приложений;
4. **hadoop-cloud-storage-project**: модуль, обеспечивающий в кластере Hadoop поддержку облачных хранилищ данных, таких как Amazon S3, Azure Blob Storage и Google Cloud Storage;
5. **hadoop-common-project**: основной модуль, содержащий общие компоненты и утилиты, используемые другими модулями Hadoop;
6. **hadoop-dist**: модуль, отвечающий за создание дистрибутива Hadoop, включая конфигурационные файлы и скрипты управления;
7. **hadoop-hdfs-project**: модуль, содержащий компоненты, отвечающие за хранение данных в Hadoop, включая Hadoop Distributed File System (HDFS) и связанные с ним утилиты;
8. **hadoop-mapreduce-project**: модуль, предоставляющий инфраструктуру и компоненты для параллельной обработки данных на кластере Hadoop с использованием модели MapReduce;
9. **hadoop-maven-plugins**: модуль, содержащий плагины Maven для автоматизации сборки проекта Hadoop;
10. **hadoop-minicluster**: модуль, предоставляющий инструменты для запуска локального тестового кластера Hadoop на одном компьютере;
11. **hadoop-project**: основной модуль, объединяющий все подмодули проекта Hadoop;
12. **hadoop-project-dist**: модуль, содержащий собранные дистрибутивы Hadoop для различных операционных систем и сред;
13. **hadoop-tools**: модуль, содержащий различные инструменты и утилиты для администрирования и управления кластером Hadoop, а также для обработки и анализа данных;
14. **hadoop-yarn-project**: модуль, предоставляющий фреймворк для управления ресурсами и планирования задач в кластере Hadoop, известный как YARN (Yet Another Resource Negotiator).

Рассмотрим более конкретно взаимодействие между модулями Apache Hadoop:

- **hadoop-assemblies** взаимодействует с **hadoop-common-project** для получения общих компонентов, которые требуются для сборки различных частей приложения. Также он может использовать компоненты из **hadoop-mapreduce-project** и **hadoop-hdfs-project** для сборки необходимых библиотек и JAR файлов;
- **hadoop-build-tools** использует компоненты из **hadoop-project** для выполнения сборки и тестирования проекта Hadoop. Он также может взаимодействовать с **hadoop-maven-plugins** для автоматизации сборки с использованием Maven;
- **hadoop-client-modules** зависит от **hadoop-common-project** для получения клиентских библиотек и инструментов, а также может использовать компоненты из **hadoop-cloud-storage-project** для взаимодействия с облачными хранилищами данных;
- **hadoop-cloud-storage-project** может использовать общие компоненты из **hadoop-common-project** для работы с данными, а также может взаимодействовать с

**hadoop-client-modules** для предоставления клиентских интерфейсов к облачным хранилищам данных;
- **hadoop-common-project** является основным модулем, который предоставляет общие компоненты и утилиты, используемые другими модулями Hadoop. Он может взаимодействовать со многими другими модулями для предоставления базовой функциональности;
- **hadoop-dist** использует компоненты из различных модулей, включая **hadoop-common-project**, **hadoop-mapreduce-project** и **hadoop-hdfs-project**, для создания дистрибутива Hadoop;
- **hadoop-hdfs-project** может использовать общие компоненты из **hadoop-common-project** для работы с Hadoop Distributed File System (HDFS) и другими утилитами;
- **hadoop-mapreduce-project** зависит от **hadoop-common-project** для предоставления общих компонентов и инфраструктуры, а также может взаимодействовать с **hadoop-client-modules** для взаимодействия с кластером Hadoop;
- **hadoop-maven-plugins** может использовать компоненты из **hadoop-build-tools** и **hadoop-project** для автоматизации сборки проекта Hadoop с использованием Maven;
- **hadoop-minicluster** может использовать компоненты из **hadoop-project** для запуска локального тестового кластера Hadoop и эмуляции работы реального кластера;
- **hadoop-project-dist** использует компоненты из различных модулей для создания собранных дистрибутивов Hadoop для различных операционных систем и сред;
- **hadoop-tools** может взаимодействовать с **hadoop-client-modules** для администрирования и управления кластером Hadoop, а также для обработки и анализа данных;
- **hadoop-yarn-project** может использовать клиентские библиотеки из **hadoop-client-modules** для взаимодействия с кластером Hadoop и предоставления фреймворка для управления ресурсами и планирования задач в кластере.

Проблема совместимости данных касается передачи данных между процессами Hadoop по сети. Hadoop использует Protocol Buffers для большинства RPC-коммуникаций. Сохранение совместимости требует запрета модификации, как описано ниже. Также следует учитывать коммуникацию, не относящуюся к RPC, например, использование HTTP для передачи образа HDFS в качестве части создания снимка или передачи вывода MapTask. Потенциальные виды коммуникации могут быть категоризированы следующим образом:

- **Клиент-Сервер**: общение между клиентами Hadoop и серверами (например, протокол клиента HDFS к NameNode или протокол клиента YARN к ResourceManager);
- **Клиент-Сервер (Административный)**: следует выделить подмножество протоколов Клиент-Сервер, используемых исключительно административными командами (например, протокол HAAdmin), поскольку эти протоколы влияют только на администраторов, которые могут терпеть изменения, которые не могут сделать конечные пользователи (которые используют общие протоколы Клиент-Сервер);
- **Сервер-Сервер**: общение между серверами (например, протокол между DataNode и NameNode или между NodeManager и ResourceManager).

## 3.1. Принципы проектирования

В процессе создания приложения особое внимание уделялось **принципам SOLID**:

1. *Single Responsibility Principle* (Принцип единственной ответственности).
2. *Open Closed Principle* (Принцип открытости/закрытости).
3. *Liskov's Substitution Principle* (Принцип подстановки Барбары Лисков).
4. *Interface Segregation Principle* (Принцип разделения интерфейса).
5. *Dependency Inversion Principle* (Принцип инверсии зависимостей).

Кроме того, при разработке Apache Hadoop использовались **принципы Clean Architecture**:

1. **Разделение на уровни**: Apache Hadoop разделяется на уровни, отвечающие за обработку данных, управление кластером и другие функции.
2. **Зависимости внутрь**: компоненты внутри Hadoop не должны зависеть от конкретных деталей реализации внешних систем или компонентов.
3. **Границы уровней**: определенные интерфейсы между уровнями обеспечивают ясное взаимодействие между компонентами.
4. **Чистая зависимость**: зависимости должны быть направлены внутрь системы, от более абстрактных компонентов к менее абстрактным.
5. **Сохранение чистоты уровней**: каждый уровень должен отвечать только за свои задачи и не знать о деталях реализации других уровней.

# 4. Архитектурные представления

## 4.1. Представление прецедентов

Система будет использоваться для обработки и анализа больших объемов данных на распределенном кластере серверов. Основными актёрами являются администраторы системы, разработчики алгоритмов обработки данных и конечные пользователи, запускающие задачи анализа данных.

**Use-Case для актёра “Пользователь”:**

1. Как пользователь, я хочу загрузить данные и запустить задачу обработки данных на кластере Hadoop.
    1.1. Пользователь загружает данные в Hadoop Distributed File System (HDFS).
    1.2. Пользователь настраивает параметры задачи обработки данных.
    1.3. Пользователь запускает задачу обработки данных с использованием MapReduce или других компонентов Apache Hadoop.
    1.4. Пользователь мониторит статус выполнения задачи и получает результаты анализа данных.
2. Как пользователь, я хочу взаимодействовать с данными с помощью различных инструментов и интерфейсов для анализа данных.
    2.1. Пользователь выбирает инструмент или интерфейс для работы с данными, такой как Apache Hive, Apache Pig или Apache Spark.
    2.2. Пользователь загружает необходимые данные или выбирает уже загруженные из HDFS.
    2.3. Пользователь проводит анализ данных, выполняет запросы и получает результаты на основе выбранного инструмента или интерфейса.

**Use-Case для актёра “Разработчик”:**

1. Как разработчик, я хочу загружать и тестировать свои алгоритмы обработки данных на кластере Hadoop.
    1.1. Разработчик загружает тестовые данные в HDFS.
    1.2. Разработчик разрабатывает и настраивает алгоритмы обработки данных с использованием MapReduce или других компонентов Apache Hadoop.
    1.3. Разработчик запускает тестирование алгоритмов на кластере Hadoop.
    1.4. Разработчик мониторит процесс выполнения задач и анализирует полученные результаты.
2. Как разработчик, я хочу настраивать и оптимизировать кластер Hadoop для обеспечения эффективной обработки данных.
    2.1. Разработчик настраивает конфигурацию кластера, включая параметры управления ресурсами, репликации данных и другие параметры.
    2.2. Разработчик мониторит использование ресурсов кластера и производительность выполнения задач.
    2.3. Разработчик анализирует результаты мониторинга и вносит изменения в конфигурацию кластера для оптимизации его работы.
3. Как разработчик, я хочу создавать резервные копии данных и обеспечить возможность восстановления в случае сбоев или потери данных.
    3.1. Администратор создает и настраивает регулярное резервное копирование данных в HDFS с использованием инструментов резервного копирования.
    3.2. Администратор тестирует процедуры восстановления данных, чтобы убедиться в их эффективности и правильности.

3.3. При необходимости администратор восстанавливает данные из резервных копий в случае сбоев или потери данных, следуя установленным процедурам восстановления.

4. Как разработчик, я хочу обеспечить безопасность кластера Hadoop и защитить данные от несанкционированного доступа и вредоносных атак.
   4.1. Администратор настраивает аутентификацию и авторизацию пользователей и групп, используя инструменты аутентификации и управления доступом.
   4.2. Администратор настраивает механизмы шифрования данных в HDFS и других компонентах для защиты конфиденциальности данных.
   4.3. Администратор устанавливает и настраивает механизмы мониторинга безопасности и реагирования на инциденты для обнаружения и предотвращения угроз безопасности.

**Use-Case для актёра “Администратор”:**

1. Как администратор, я хочу иметь возможность управлять кластером Hadoop, включая его настройку, масштабирование и мониторинг.
   1.1. Администратор устанавливает и настраивает кластер Hadoop.
   1.2. Администратор управляет ресурсами кластера, включая добавление или удаление узлов, настройку репликации данных и балансировку нагрузки.
   1.3. Администратор мониторит состояние и производительность кластера с помощью инструментов мониторинга и анализирует результаты для оптимизации работы кластера.
   1.4. При необходимости администратор вносит изменения в конфигурацию кластера или задач для оптимизации его производительности и улучшения работы системы.

**Тест-кейсы для пользователей:**

1. Загрузка данных в HDFS:
   a. Шаги:
      i. Пользователь загружает тестовые данные в Hadoop Distributed File System (HDFS).
   b. Ожидаемый результат:
      i. Данные успешно загружены в HDFS без ошибок.
      ii. Данные доступны для последующей обработки и анализа.
1. Запуск задачи обработки данных:
   a. Шаги:
      i. Пользователь настраивает параметры задачи обработки данных.
      ii. Пользователь запускает задачу обработки данных.
   b. Ожидаемый результат:
      i. Задача успешно запускается и начинает выполнение.
      ii. Пользователь получает уведомление о статусе выполнения задачи.

**Тест-кейсы для разработчиков:**

1. Масштабируемость кластера:
   a. Шаги:
      i. Администратор добавляет новые узлы в кластер Hadoop.
      ii. Администратор перезапускает задачу обработки данных на расширенном кластере.
   b. Ожидаемый результат:
      i. Система успешно масштабируется и обрабатывает данные на новых узлах без увеличения времени выполнения задачи.
2. Резервное копирование и восстановление:
   a. Шаги:

i. Администратор создает резервную копию данных в HDFS.
ii. Администратор удаляет некоторые данные из HDFS.
iii. Администратор восстанавливает удаленные данные из резервной копии.

b. Ожидаемый результат:
i. Удаленные данные успешно восстанавливаются из резервной копии, и они совпадают с оригинальными данными.

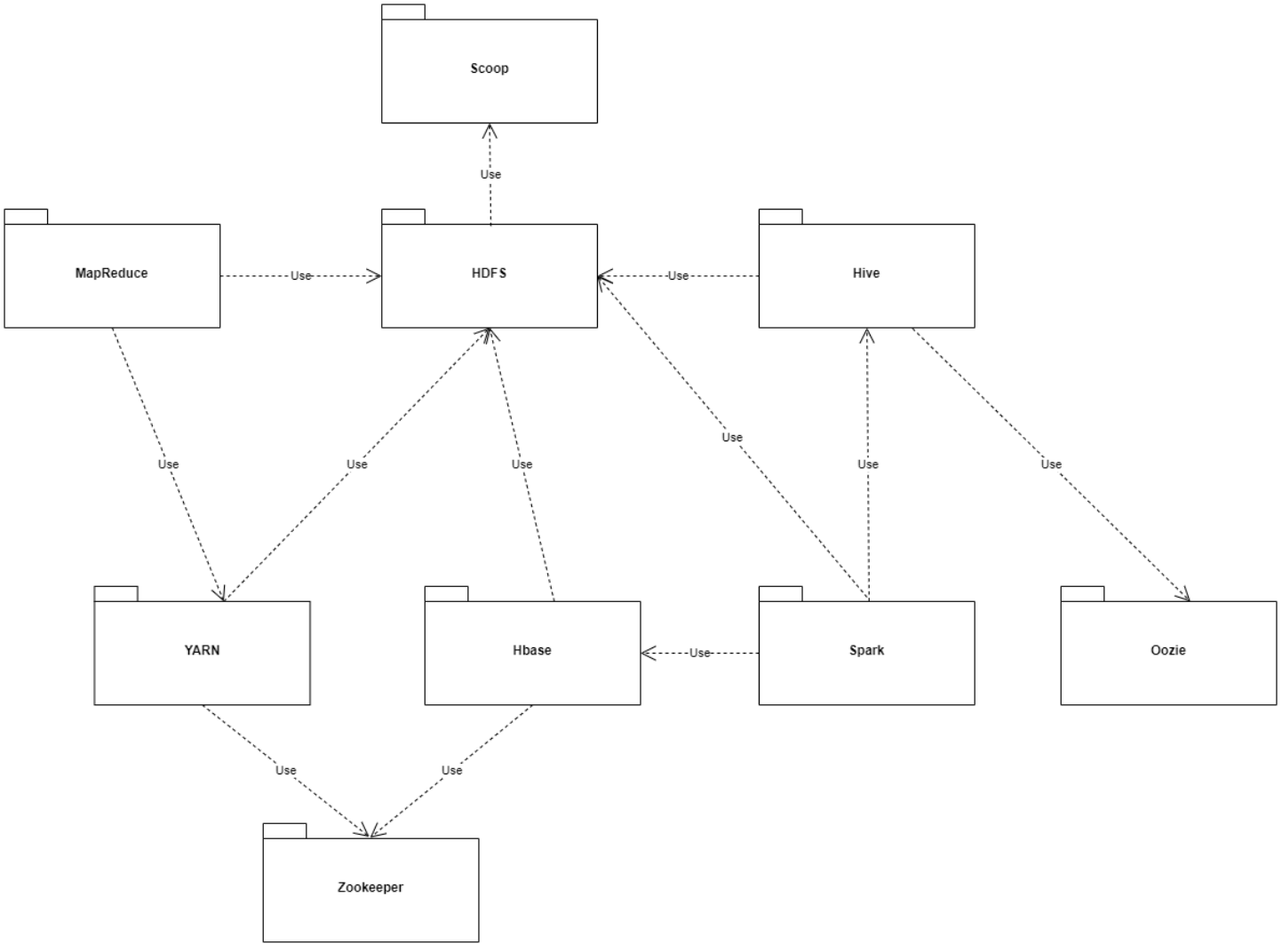
**Тест-кейсы для администраторов:**

1. Тестирование алгоритма MapReduce:
   a. Шаги:
      i. Разработчик загружает тестовые данные в HDFS.
      ii. Разработчик запускает тестирование алгоритма MapReduce на кластере Hadoop.
   b. Ожидаемый результат:
      i. Алгоритм MapReduce успешно обрабатывает тестовые данные и генерирует ожидаемые результаты.

## 4.2. Логическое представление

С **диаграммой классов** можно ознакомиться по данной ссылке. Диаграмма взята из данной статьи.

**Диаграмма пакетов** (в качестве источника использовался chat.openai.com):



## 4.3. Представление архитектуры процессов

**Диаграмма последовательностей** для MapReduce (источник находится здесь):

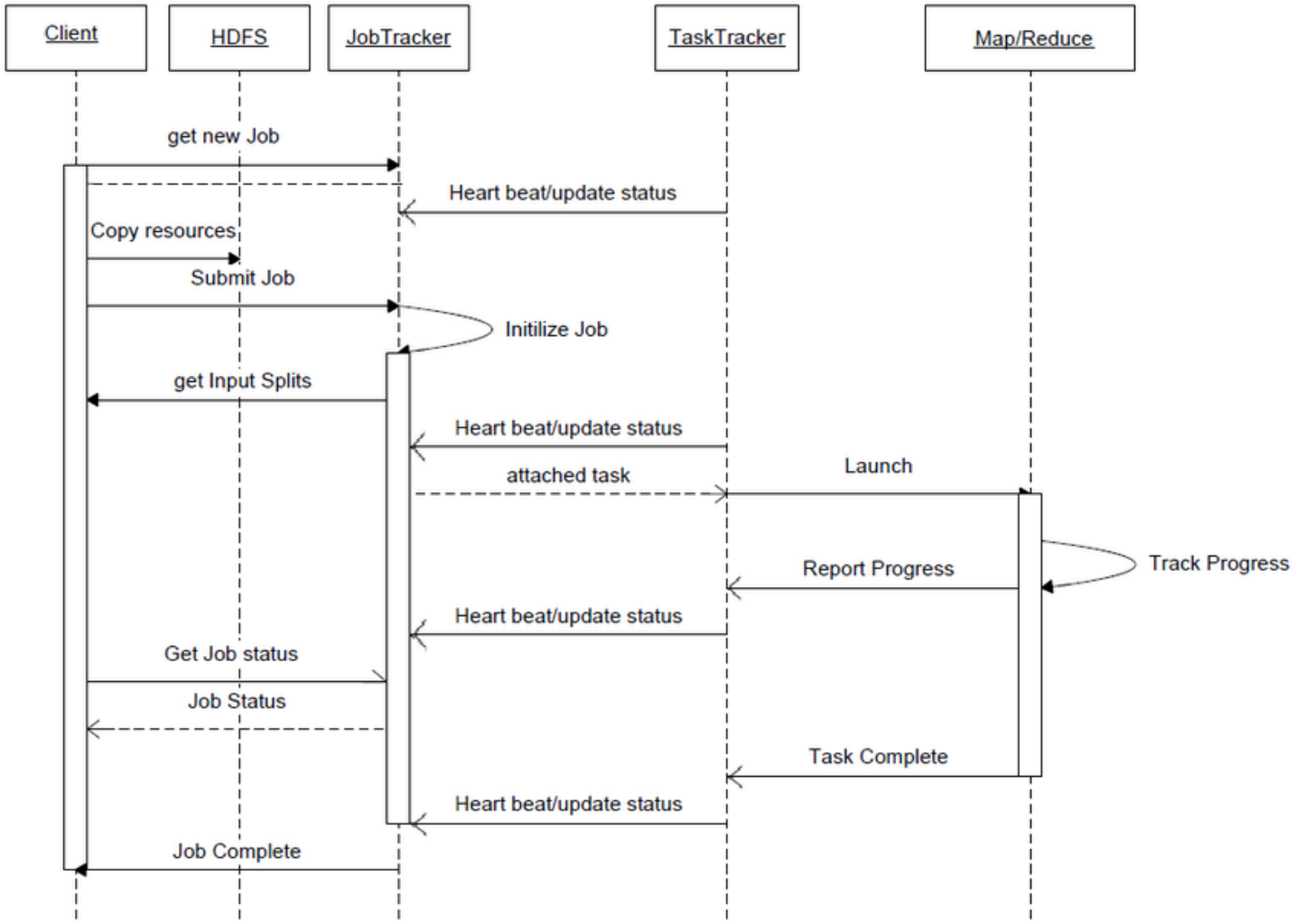
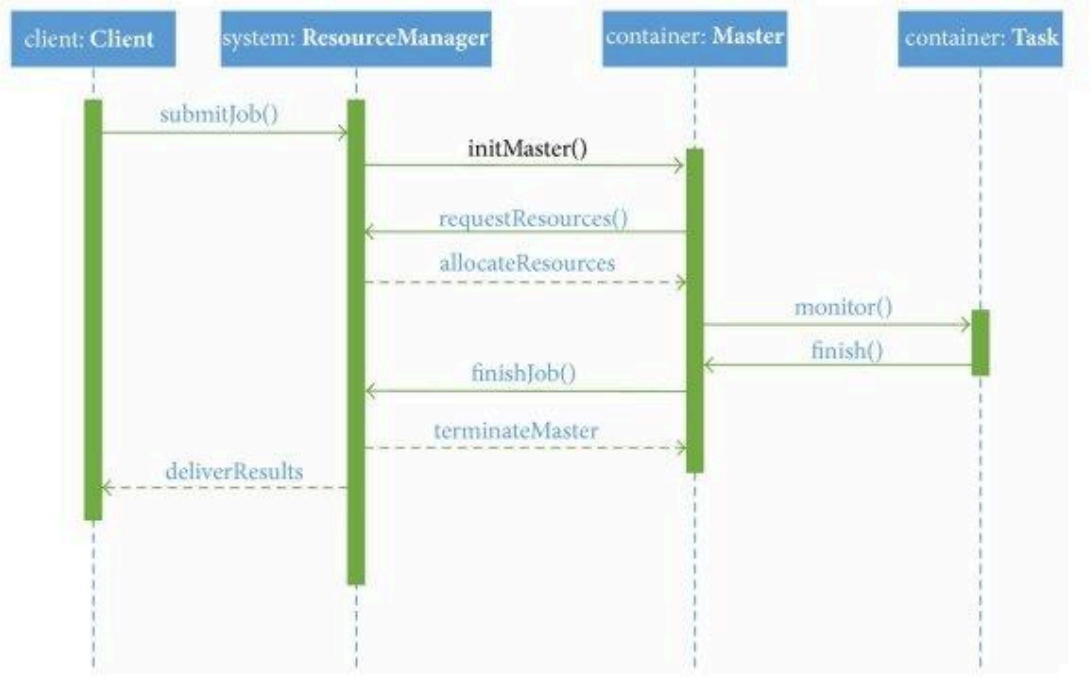



**Диаграмма последовательностей** для YARN (источник находится здесь):



## 4.4. Физическое представление архитектуры

В целом, физическое представление архитектуры Apache Hadoop зависит от конфигурации кластера, на котором будет работать система. Обычно кластер Hadoop состоит из следующих основных компонентов:

1. **Узлы данных (Data Nodes)** — это серверы, на которых хранятся данные. Узлы данных обычно оборудованы жесткими дисками большой емкости для хранения данных Hadoop Distributed File System (HDFS);
2. **Узлы имен (Name Nodes)** — это серверы, управляющие метаданными HDFS. Они хранят информацию о том, где располагаются блоки данных на узлах данных;
3. **Ресурсные менеджеры (Resource Managers)** — это серверы, отвечающие за управление ресурсами кластера. Они принимают запросы на выполнение задач и распределяют ресурсы между различными приложениями;
4. **Задачные трекеры (Task Trackers)** — это процессы, запущенные на узлах данных, которые управляют выполнением задач на кластере. Они получают задачи от ресурсного менеджера и координируют их выполнение;
5. **Журналы (Journals)** — это важные элементы системы, которые используются для сохранения списков операций и транзакций, а также играют ключевую роль в обеспечении надежности и восстановления в случае сбоев.

Эти компоненты обычно связаны через сеть, используя высокоскоростные сетевые соединения, такие как Ethernet. Каждый узел кластера обычно имеет собственный IP-адрес для обеспечения коммуникации между узлами. Кроме того, для обеспечения отказоустойчивости и распределенности, данные обычно реплицируются на несколько узлов данных в кластере.

## 4.5. Представление развертывания

**Разворачивание системы:**

1. Hadoop Distributed File System (HDFS): для хранения и управления данными.
2. YARN (Yet Another Resource Negotiator): для управления ресурсами кластера и планирования выполнения задач.
3. MapReduce или Apache Spark: для обработки и анализа данных.
4. Дополнительные инструменты и библиотеки, такие как Apache Hive, Apache Pig, Apache HBase и др., в зависимости от требований проекта.

**Отдельно устанавливаемые компоненты и модули:**

1. Hadoop Common: общие библиотеки и утилиты.
2. HDFS DataNode: служба, ответственная за хранение и управление данными в HDFS.
3. HDFS NameNode: служба, отвечающая за управление метаданными и их распределение по узлам кластера.
4. YARN ResourceManager: служба, отвечающая за управление ресурсами кластера и планирование выполнения задач.
5. YARN NodeManager: служба, управляющая ресурсами и выполнением задач на отдельном узле кластера.

**Артефакты для заказчика:**

1. Клиентские приложения и конфигурационные файлы, необходимые для взаимодействия с кластером Hadoop.
2. Пакеты с дополнительными инструментами и библиотеками, если они используются в проекте.

3. Документация по установке, настройке и использованию системы Apache Hadoop.

**Обновление системы:**

1. Загрузка новых версий компонентов и модулей.
2. Остановка и перезапуск служб кластера на каждом узле с обновленными компонентами.
3. Проверка совместимости новых версий с имеющимися данными и приложениями.
4. Тестирование обновленной системы на тестовых или отдельных кластерах перед внедрением в рабочую среду.
5. Резервное копирование данных перед началом процесса обновления, чтобы обеспечить возможность восстановления в случае сбоев или проблем.

## 4.6. Представление архитектуры данных

1. **Типы данных**: в системе обрабатываются различные типы данных, включая *структурированные*, *полуструктурированные и неструктурированные*;
2. **Хранение данных**: данные в системе хранятся в распределенной файловой системе HDFS. HDFS разбивает данные на блоки определенного размера и реплицирует их на различных узлах кластера для обеспечения отказоустойчивости и параллельной обработки;
3. **Интеграция данных**: для интеграции данных в систему используются различные инструменты и технологии. Например, Apache Sqoop используется для передачи данных между реляционными базами данных и HDFS, Apache Kafka — для обработки потоковых данных, а Apache Flume — для сбора и передачи данных из различных источников в HDFS;
4. **Репликация данных**: для обеспечения отказоустойчивости и доступности данных в системе Apache Hadoop применяется механизм репликации данных. Каждый блок данных в HDFS реплицируется на несколько узлов кластера. По умолчанию, каждый блок реплицируется три раза, что, в свою очередь, обеспечивает надежность данных;
5. **Потоки данных**: в системе Apache Hadoop для обработки потоков данных используются различные фреймворки и инструменты, такие как Apache Spark Streaming, Apache Flink и Apache Storm. Благодаря ним можно обрабатывать данные в режиме реального времени и осуществлять аналитику на основе поступающих данных;
6. **Гонки данных**: в системе могут возникать гонки данных, особенно при параллельном доступе к общим ресурсам несколькими процессами. Для предотвращения гонок используются механизмы синхронизации, блокировки и координации, а также алгоритмы распределенной блокировки и управления доступом.

## 4.7. Представление архитектуры безопасности

**Обеспечение безопасности и сохранности данных:**

1. Шифрование данных: для защиты конфиденциальности данных в HDFS и во время их передачи между узлами кластера используется шифрование данных.
2. Аутентификация и авторизация: для проверки подлинности пользователей и управления их доступом к данным и ресурсам кластера используются механизмы аутентификации и авторизации.
3. Механизмы безопасности файловой системы: Apache Hadoop предоставляет различные механизмы безопасности файловой системы для ограничения доступа к данным на уровне файлов и каталогов.

**Аутентификация и авторизация:**

1. Kerberos: Kerberos является основным механизмом аутентификации в системе Hadoop. Он обеспечивает безопасное и эффективное аутентифицированное взаимодействие между узлами кластера и пользователями.
2. ACL (Access Control Lists): ACL используются для управления доступом к файлам и каталогам в HDFS. Они определяют, какие пользователи или группы имеют доступ на чтение, запись или выполнение операций над данными.
3. Ranger или Sentry: дополнительные инструменты, такие как Apache Ranger или Apache Sentry, могут использоваться для централизованного управления доступом и политиками безопасности в системе Hadoop.

**Прочие меры безопасности:**

1. Логирование и мониторинг: ведение журналов и мониторинг активности пользователей и системы для обнаружения подозрительной активности и инцидентов безопасности.
2. Файрволы и сетевые политики: управление доступом к узлам кластера и сетевым ресурсам с помощью фаерволов и сетевых политик.

## 4.8. Представление реализации и разработки

Организация среды для написания приложения и используемые инструменты:

1. **Языки программирования**: основным языком программирования при разработке системы является Java. Использовались следующие языки:
   1.1. C/C++ (для разработки нативных клиентов и инструментов для взаимодействия с HDFS [можно найти здесь]);
   1.2. JavaScript (для веб-приложений, которые могут использоваться для различных модулей системы [для hadoop-hdfs можно найти здесь & для hadoop-yarn можно найти здесь]).
2. **Язык скрипта**: в качестве языка скрипта для интеграции с другими системами используется Shell;
3. **Среда разработки**: в качестве среды разработки можно использовать IntelliJ IDEA или Eclipse;
4. **Управление версиями и контроля изменений в исходном коде**: Git;
5. **Хранение и управление репозиторием с исходным кодом**: GitHub.
6. **Тестирование**: в качестве инструментов тестирования использовались следующие фреймворки:
   6.1. JUnit — модульное тестирование;
   6.2. TestNG — интеграционное тестирование;
   6.3. Mockito — тестирование через заглушки реальных объектов.

Проект организован по принципу модульной архитектуры, включая каталоги для исходного кода, ресурсов, тестов и зависимостей. Весь код разделен на модули, соответствующие различным компонентам приложения (“*hadoop-assemblies*”, “*hadoop-build-tools*”, “*hadoop-client-modules*”, “*hadoop-cloud-storage-project*”, “*hadoop-common-project*”, “*hadoop-dist*”, “*hadoop-hdfs-project*”, “*hadoop-mapreduce-project*”, “*hadoop-maven-plugins*”, “*hadoop-minicluster, hadoop-project*”, “*hadoop-project-dist*”, “*hadoop-tools*”, “*hadoop-yarn-project*”).

## 4.9. Представление производительности

**Используемые алгоритмы и структуры данных:**

1. Алгоритмы обработки данных: как для распределенной обработки данных (например, MapReduce или Apache Spark), так и для работы с данными в HDFS (например, алгоритмы распределенного чтения/записи).

2. Алгоритмы оптимизации и параллелизации: для эффективного использования ресурсов кластера и распределения задач между узлами используются алгоритмы оптимизации и параллелизации, например, планировщики ресурсов и планировщики задач.
3. Структуры данных для обработки: в зависимости от типа данных и операций, выполняемых над ними, могут использоваться различные структуры данных, такие как хэш-таблицы, деревья, списки и т. д.

**Обеспечение производительности системы:**

1. Оптимизация запросов и задач: анализ и оптимизация запросов и задач перед их выполнением для минимизации времени выполнения и использования ресурсов.
2. Распараллеливание и распределение задач: распределение задач и операций между узлами кластера для балансировки нагрузки и ускорения выполнения задач.
3. Кэширование и предварительная загрузка данных: использование кэширования и предварительной загрузки данных для уменьшения времени доступа к данным и улучшения производительности запросов.
4. Масштабирование и оптимизация аппаратного обеспечения: увеличение количества узлов в кластере, улучшение характеристик узлов (например, объема памяти, процессорной мощности) и оптимизация сетевой инфраструктуры для увеличения пропускной способности и производительности системы.
5. Мониторинг и анализ производительности: регулярный мониторинг производительности системы с использованием специализированных инструментов и анализ полученных данных для выявления узких мест и определения возможных улучшений.

## 4.10. Атрибуты качества системы

**Нефункциональные аспекты проекта системы**: масштабируемость, отказоустойчивость, производительность.

### 4.10.1. Объем данных и производительность системы

Система Apache Hadoop предназначена для работы с огромными **объемами данных**, характерными для Big Data. Она способна эффективно обрабатывать данные в масштабах от терабайт до петабайт и более.

Выделим следующие важные **критерии производительности** для системы Apache Hadoop:

1. **Пропускная способность**: система должна обеспечивать высокую скорость обработки и передачи данных между узлами кластера;
2. **Время отклика**: время, необходимое для выполнения запросов и задач, должно быть минимальным, чтобы обеспечить быстрый доступ к данным и результатам обработки;
3. **Масштабируемость**: система должна эффективно масштабироваться при добавлении новых узлов кластера или увеличении объема данных;
4. **Устойчивость**: система должна демонстрировать стабильную работу даже при больших нагрузках и в условиях возможных отказов оборудования.

Для системы был создан класс-бенчмарк **NNThroughputBenchmark**, с реализацией которого можно ознакомиться здесь. В нем измеряется количество операций, выполняемых именованным узлом (name-node) в секунду. Для каждой протестированной операции засечено общее время выполнения в секундах (Elapsed Time), пропускная способность операций (Ops per sec) и среднее время выполнения операций (Average Time). Очевидно, что чем выше значение пропускной способности и чем меньше среднее время выполнения, тем лучше производительность. Рассмотрим пример отчета после выполнения команды, которая открывает 100 000 файлов с использованием 1000 потоков против удаленного именованного узла (name-node):

```
$ hadoop org.apache.hadoop.hdfs.server.namenode.NNThroughputBenchmark -fs
hdfs://nameservice:9000 -op open -threads 1000 -files 100000

--- open inputs ---
nrFiles = 100000
nrThreads = 1000
nrFilesPerDir = 4
--- open stats  ---
# operations: 100000
Elapsed Time: 9510
 Ops per sec: 10515.247108307045
Average Time: 90
```

Более подробно ознакомиться с документацией класса **NNThroughputBenchmark** можно здесь.

Кроме того, имеется бенчмарк для тестирования операции чтения файлов в Apache Hadoop. Он реализован с помощью фреймворка OpenJDK JMH (Java Microbenchmark Harness — инструмент, предназначенный для проведения микробенчмарков в Java) для проведения измерений производительности. Ознакомится с исходным кодом можно здесь.

### 4.10.2. Гарантии качества работы системы

**Гарантии качества работы системы будут обеспечиваться следующими мероприятиями:**

1. **Тестирование:** вся функциональность системы будет подвергнута тщательному тестированию, включая модульное тестирование, интеграционное тестирование и системное тестирование. Это позволит выявить и исправить ошибки и недочеты до выпуска системы в продакшн.
2. **Мониторинг:** в системе будет реализован мониторинг состояния и производительности, позволяющий оперативно выявлять и реагировать на любые проблемы или сбои в работе системы.
3. **Отладка и регистрация ошибок:** для отслеживания и регистрации ошибок в работе системы будут использоваться соответствующие инструменты и механизмы. Обнаруженные ошибки будут быстро исправляться, чтобы обеспечить непрерывную работу системы.
4. **Восстановление после сбоев:** в случае сбоев или отказов в работе системы будут предусмотрены механизмы восстановления, позволяющие быстро восстановить работоспособность и непрерывную доступность сервисов.
5. **Бэкапы и резервное копирование:** для обеспечения сохранности данных система будет регулярно создавать резервные копии данных и хранить их в безопасном месте. Это позволит восстановить данные в случае их потери или повреждения.
6. **Следование стандартам и руководствам:** при разработке и сопровождении системы будет соблюдаться согласованный набор стандартов и руководств, что позволит обеспечить согласованность и качество работы системы.

**Проверка работоспособности системы будет осуществляться путем выполнения ряда мероприятий:**

1. **Тестирование:** запуск тестовых сценариев для проверки работоспособности различных компонентов системы.
2. **Мониторинг:** анализ метрик производительности и состояния системы с использованием инструментов мониторинга.
3. **Автоматические проверки:** запуск автоматизированных проверок и скриптов для обнаружения проблем и сбоев в работе системы.

4. **Анализ журналов и отчетов об ошибках:** анализ журналов событий и отчетов об ошибках для выявления и устранения проблем и недочетов в работе системы.

## 4.11. Другие представления

Не предоставляются.

# 5. Технические описания отдельных ключевых архитектурных решений

## 5.1. Техническое описание решения №1: Хранение информации

### 5.1.1. Проблема

Хранение информации в системе должно быть обеспечено таким образом, чтобы данные сохранялись надежно и были доступны для обработки и анализа.

### 5.1.2. Идея решения

Для обеспечения хранения информации в системе Apache Hadoop следует использовать Hadoop Distributed File System (HDFS), который является стандартным средством для хранения данных в кластере Hadoop.

### 5.1.3. Факторы

1. Требования ТЗ позволяют использовать HDFS, так как не требуется особенной производительности для хранения и доступа к данным.
2. Использование HDFS обеспечивает надежное хранение данных и предотвращает их потерю при сбоях или отказах узлов кластера.

### 5.1.4. Решение

Hadoop Distributed File System (HDFS) представляет собой распределенную файловую систему, специально разработанную для хранения больших объемов данных на кластере серверов. В HDFS данные автоматически реплицируются на несколько узлов кластера, что обеспечивает отказоустойчивость и защиту от потери информации. Доступ к данным в HDFS осуществляется через стандартные интерфейсы, такие как команды командной строки или API Hadoop, что обеспечивает удобство использования и интеграции.

### 5.1.5. Мотивировка

Выбор разработки и последующего использования Hadoop Distributed File System (HDFS) обусловлен его отличной масштабируемостью, отказоустойчивостью и эффективностью в хранении больших объемов данных. Это стандартное средство в экосистеме Apache Hadoop, которое широко используется и имеет поддержку со стороны сообщества.

### 5.1.6. Неразрешенные вопросы

Отсутствуют.

### 5.1.7. Альтернативы

В качестве альтернативы рассматривалось использование других распределенных файловых систем, таких как Amazon S3 или Google Cloud Storage. Однако, использование этих систем может потребовать дополнительных затрат и настройки интеграции с Apache Hadoop, что может повлечь за собой увеличение сложности и стоимости системы.

## 5.2. Техническое описание решения №2: Производительность

### 5.2.1. Проблема

Необходимо выбрать подход к обработке и анализу больших объемов данных в системе, учитывая требования к производительности и возможность масштабирования.

### 5.2.2. Идея решения

Для решения проблемы следует использовать архитектурный шаблон MapReduce.

### 5.2.3. Факторы

1. Требования к обработке больших объемов данных.
2. Возможность параллельного выполнения задач для ускорения обработки данных.
3. Масштабируемость для работы с растущими объемами данных.

### 5.2.4. Решение

Архитектурный шаблон MapReduce предполагает разделение обработки данных на два основных этапа:

1. **Map:** на этом этапе данные разбиваются на части и обрабатываются параллельно на различных узлах кластера.
2. **Reduce:** результаты, полученные на этапе Map, объединяются и обрабатываются для получения конечного результата.

Пример реализации MapReduce может включать в себя использование Apache Hadoop MapReduce Framework или Apache Spark, которые предоставляют удобные средства для разработки и выполнения задач обработки данных.

### 5.2.5. Мотивировка

Выбор архитектурного шаблона MapReduce обусловлен его способностью обрабатывать большие объемы данных параллельно на распределенном кластере серверов, что позволяет обеспечить высокую производительность и масштабируемость системы. Этот подход также хорошо подходит для решения задач обработки данных, таких как агрегация, фильтрация, сортировка и т. д.

### 5.2.6. Неразрешенные вопросы

Отсутствуют.

### 5.2.7. Альтернативы

В качестве альтернативы рассматривался использование инструментов и фреймворков для обработки потоков данных, таких как Apache Kafka и Apache Flink. Однако, MapReduce остается популярным выбором для обработки пакетных данных из-за своей простоты и распространенности.

## 5.3. Техническое описание решения №3: Эффективное использование ресурсов

### 5.3.1. Проблема

Необходимо решить проблему эффективного использования ресурсов кластера Apache Hadoop при выполнении задач обработки данных.

### 5.3.2. Идея решения

Введение механизмов динамического распределения ресурсов в кластере с использованием Apache YARN (Yet Another Resource Negotiator).

### 5.3.3. Факторы

1. Требования к эффективному использованию вычислительных и сетевых ресурсов кластера.
2. Необходимость балансировки нагрузки и оптимального распределения задач между узлами.
3. Управление ресурсами и приоритезация задач в зависимости от их важности и приоритета.

### 5.3.4. Решение

Для решения проблемы эффективного использования ресурсов кластера Apache Hadoop предлагается использовать Apache YARN, который является фреймворком для управления ресурсами и планирования выполнения задач на кластере.

Механизм работы Apache YARN состоит из следующих основных компонентов:

1. **ResourceManager (RM):** центральный компонент, отвечающий за управление ресурсами кластера и планирование выполнения задач.
2. **NodeManager (NM):** компонент, запускаемый на каждом узле кластера, отвечающий за управление ресурсами узла и выполнение задач.
3. **ApplicationMaster (AM):** компонент, создаваемый для каждой выполняемой задачи, который отвечает за управление жизненным циклом задачи и взаимодействие с ResourceManager для запроса ресурсов и мониторинга выполнения задачи.

Apache YARN позволяет динамически распределять ресурсы в кластере в зависимости от текущей нагрузки и требований задач, что обеспечивает эффективное использование ресурсов и оптимальную производительность системы.

### 5.3.5. Мотивировка

Выбор использования Apache YARN обусловлен его способностью эффективно управлять ресурсами кластера и планировать выполнение задач, что позволяет обеспечить оптимальное использование вычислительных и сетевых ресурсов и улучшить производительность системы в целом.

### 5.3.6. Неразрешенные вопросы

Отсутствуют.

### 5.3.7. Альтернативы

В качестве альтернативы можно рассмотреть использование других фреймворков для управления ресурсами и выполнения задач на кластере, таких как Apache Mesos или Kubernetes. Однако было принято решение о разработке Apache YARN для более удобного, гибкого и понятного управления ресурсами в экосистеме Apache Hadoop и широкого применения в индустрии.

# 6. Приложения

## 6.1. Словарь терминов

1. **JDBC (Java Database Connectivity)** — стандартный интерфейс в языке программирования Java для взаимодействия с реляционными базами данных.
2. **HDFS (Hadoop Distributed File System)** — распределенная файловая система, используемая в Apache Hadoop для хранения данных на кластере серверов.
3. **MapReduce** — модель программирования и соответствующий фреймворк для обработки и анализа больших объемов данных в распределенной среде.
4. **Apache Spark** — фреймворк для обработки и анализа данных, предоставляющий более высокую производительность и удобный API по сравнению с MapReduce.
5. **YARN (Yet Another Resource Negotiator)** — фреймворк в Apache Hadoop для управления ресурсами кластера и планирования выполнения задач.
6. **NoSQL (Not Only SQL)** — тип баз данных, который предлагает механизмы хранения и извлечения данных, отличные от традиционных реляционных баз данных.
7. **ACL (Access Control Lists)** — списки управления доступом, используемые для определения прав доступа пользователей к ресурсам, таким как файлы и каталоги.
8. **Kerberos** — протокол аутентификации с открытым исходным кодом, который обеспечивает безопасное взаимодействие между узлами кластера и пользователями в системе Hadoop.

## 6.2. Ссылки на используемые документы

1. Папка с изображениями упоминаемых в документе диаграмм;
2. Документация Apache Hadoop;
3. Нативный код on C/C++;
4. Веб-приложение на JavaScript для HDFS;
5. Веб-приложение на JavaScript для YARN;
6. Исходный код NNThroughputBenchmark;
7. Страница с информацией о NNThroughputBenchmark;
8. Исходный код VectoredReadBenchmark;
9. Сведения об Apache Hadoop и MapReduce — Azure HDInsight | Microsoft Learn;
10. Извлечение, преобразование и загрузка в большом масштабе (Azure HDInsight) | Microsoft Learn;
11. Управление кластерами Apache Hadoop с помощью PowerShell — Azure HDInsight | Microsoft Learn.

## 6.3. Определения

1. **Аутентификация** — процесс проверки подлинности пользователя или системы. В контексте информационной безопасности, аутентификация обычно осуществляется с помощью логина и пароля или других средств идентификации, таких как биометрические данные.
2. **Авторизация** — процесс определения прав доступа пользователя к определенным ресурсам или функциям системы после успешной аутентификации. Например, различные уровни доступа могут быть назначены пользователям в зависимости от их роли или полномочий.
3. **Масштабируемость** — способность системы или приложения эффективно обрабатывать увеличение нагрузки или объема данных путем добавления дополнительных ресурсов или узлов в кластер. Масштабируемость может быть вертикальной (добавление ресурсов на одном уровне) или горизонтальной (добавление новых узлов).

4. **Отказоустойчивость** — свойство системы, которое позволяет ей продолжать работу даже в случае возникновения сбоев или отказов в отдельных компонентах. Отказоустойчивость обычно достигается за счет резервирования ресурсов, репликации данных и механизмов восстановления после сбоев.
5. **Балансировка нагрузки** — процесс распределения равномерной нагрузки между различными узлами или ресурсами в системе с целью оптимизации использования ресурсов и обеспечения высокой производительности. Балансировка нагрузки может осуществляться на уровне сети, вычислений, хранения данных и т. д.
6. **Шифрование** — процесс преобразования данных с использованием специального ключа, чтобы предотвратить несанкционированный доступ к ним. Зашифрованные данные могут быть прочитаны или восстановлены только с использованием правильного ключа шифрования.
7. **Журналирование** — процесс записи событий, операций или изменений, происходящих в системе или приложении, в специальный журнал. Журналирование используется для отслеживания действий пользователей, мониторинга работы системы и обнаружения проблем.
8. **Логирование** — процесс записи сообщений или событий, происходящих в системе или приложении, в специальный файл или базу данных с целью анализа, мониторинга и отладки. Логи могут содержать информацию о выполненных операциях, ошибочных ситуациях, предупреждениях и других событиях, которые могут быть полезными для диагностики проблем и выявления недостатков.
9. **Отладка** — процесс исследования и устранения ошибок или неисправностей в программном обеспечении. Отладка может включать в себя анализ логов, отслеживание выполнения кода, тестирование и другие методы для выявления и исправления проблем.
10. **Мониторинг** — процесс отслеживания состояния и производительности системы или приложения с целью выявления аномалий, проблем и трендов. Мониторинг может включать в себя сбор и анализ данных о нагрузке, использовании ресурсов, общем состоянии системы и других параметрах для обеспечения стабильной работы и быстрого реагирования на проблемы.
11. **Репликация** — процесс создания копий данных или ресурсов и распределения их на различные узлы или серверы в системе. Репликация используется для обеспечения отказоустойчивости, увеличения доступности данных и улучшения производительности путем распределения нагрузки.
12. **Сжатие данных** — процесс уменьшения размера данных путем удаления избыточной информации или применения специальных алгоритмов сжатия. Сжатие данных позволяет экономить место на диске, уменьшать время передачи данных по сети и улучшать производительность приложений.
13. **Интеграция данных** — процесс объединения данных из различных источников и форматов в единое хранилище или модель данных. Интеграция данных может включать в себя преобразование форматов, очистку данных, сопоставление схем и другие операции для обеспечения единого представления данных.

## 6.4. Другое

***Репозиторий** с проектом можно найти здесь: [apache/hadoop: Apache Hadoop (github.com)](apache/hadoop: Apache Hadoop (github.com)).*