Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев

«7» сентября 2023 г.

# Разработка систем анализа больших данных

Учебно-методические материалы по направлению подготовки
**09.03.03 Прикладная информатика**
Образовательная программа «Прикладной искусственный интеллект»

**Екатеринбург**

**РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ**

Доцент кафедры информационных технологий и систем управления _____ А.О. Коломыцева

Ассистент кафедры информационных технологий и систем управления _____ М.В. Павлов

**СОДЕРЖАНИЕ**

1. Введение в обработку больших данных ..... 4
2. Введение в Hadoop ..... 13
3. Основы MapReduce в Hadoop ..... 19
4. Применение MapReduce для решения практических задач ..... 25
5. Работа с графами в Hadoop ..... 29
6. Введение в Pig и Hive ..... 38

Перечень литературы: ..... 45

# 1. ВВЕДЕНИЕ В ОБРАБОТКУ БОЛЬШИХ ДАННЫХ

*Данные – это новая нефть*

Брайан Кржанич, бывший гендиректор Intel

В настоящее время тот факт, что каждую секунду человечество производит колоссальное количество новой информации, не является чем-то выдающимся. Во многих сферах словосочетание «большие данные» распространено, и уже трудно найти человека, который не слышал бы о Big Data. С одной стороны такая узнаваемость способствует стремительному продвижению, безболезненному навязыванию и успешным продажам соответствующих технологий бизнесу, а с другой стороны, порождает неправильное понимание терминологии, вследствие того, что первоначально закладываемые определение в данное понятие при передаче информации через различные источники неизбежно претерпевает искажения. Рассмотрим для начала, что стоит, а что не следует понимать под «большими данными».

Исторически идея систематизировано собирать данные, а затем анализировать их для принятия эффективных решения не нова. Так, гипотетически даже древние люди могли вести учет своих запасов, в качестве базы данных используя кости. Вероятно, это позволяло им оценивать, насколько долго племени хватит текущих запасов, что в свою очередь могло влиять на планирование охоты и на принятие других решений. Данный процесс совершенствовался и переходил на качественно новые уровни по мере изобретений различных механизмов и приспособлений, упрощающих хранение или обработку. Знаковым для данной идеи, без сомнений, является прошлое столетие, действительно богатое прорывными достижениями в области вычислительных систем.

Имеется два мнения на счет даты начала отчета истории термина «большие данные». С одной стороны, утверждается, что это день выхода спецвыпуска журнала Nature, полностью посвященного данной теме. С другой

стороны, что еще в 1998 году ученый Джон Мэши в своей презентации о возрастающем количестве данных употребил данный термин в общепринятом на данный момент значении.

Но популярность больших данных началась в 2011 году, если основываться на частоте поисковых запросов, которые предоставлены сервисом Google Trends.

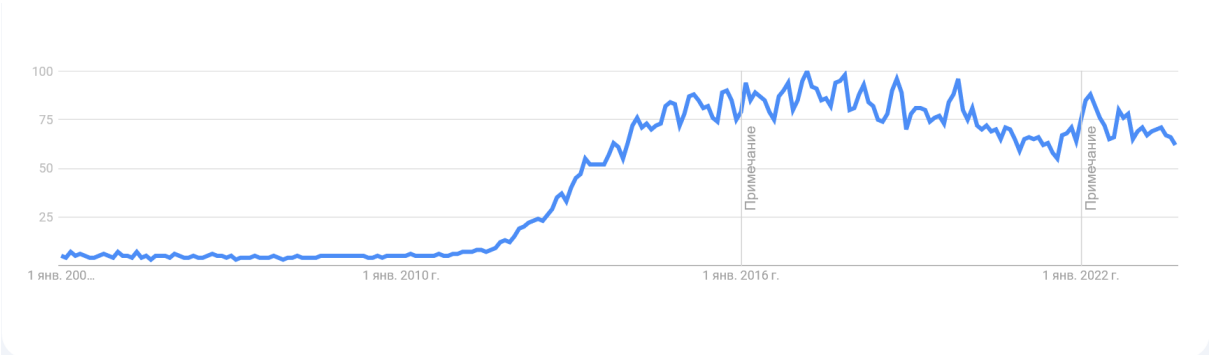


Рисунок 1 – Динамика популярности поискового запроса «Big data»

Переходя к конкретным определениям, отметим, что даже в такой фундаментальной работе как «Big Data. Concepts, Methodologies, Tools, and Applications» данному вопросу посвящено меньше одной страницы, где в качестве определения взята формулировка из Wikipedia, в русском переводе которая звучит следующим образом:

Большие данные — серия подходов, инструментов и методов обработки структурированных и неструктурированных данных огромных объёмов и значительного многообразия для получения воспринимаемых человеком результатов, эффективных в условиях непрерывного прироста, распределения по многочисленным узлам вычислительной сети, сформировавшихся в конце 2000-х годов, альтернативных традиционным системам управления базами данных и решениям класса Business Intelligence.

Рассмотрим также другое определение:

Big Data – это горизонтально масштабируемая система, использующая набор методик и технологий, позволяющих обрабатывать структурированную и неструктурированную информацию и строить связи, необходимые для

получения однозначно интерпретируемых человеком данных, не успевших потерять актуальность, и несущая ценность преследуемых им целей.

Как можно заметить, определения не дает никакой конкретной информации о том, начиная с каких объемов данные начинают считаться «большими», какие именно данные можно считать «большими». Но зато определяет инструменты, методы обработки данных. В целом прилагательное «большие» в данном определении стоит воспринимать не только в отношении данных, но и к той информационной инфраструктуре, в которой эти данные хранятся и обрабатываются.

Другой способ понять, что же из себя представляют большие данные, это изучить основные принципы. В разных источниках можно найти принципы трех, пять и восьми “V”. Рассмотрим каждый из них подробней.

Самыми важными являются следующие три характеристики, образующие три “V”: Volume, Velocity и Variety.

Volume (Объем). Подчеркивается, что объем данных, которыми приходится оперировать достаточно велик. Как критерии данную характеристику стоит понимать дополнительно в том смысле, что система хранения и обработки данных должна быть готова (способна) к масштабированию на случай, если объем данных будет увеличиваться.

Velocity (скорость обработки). У каждой конкретной задачи имеются свои требованию к скорости обработки данных, которые должны быть выполнены при разработке системы анализа больших данных. Как правило, это может быть периодическая обработка, обработка близкая к реальному времени, и обработка в режиме реального времени. Также подразумевается требование, что в будущем, как и с объемом данных, может потребоваться масштабирование.

Variety (разнообразие). Входные данные в системах анализа больших данных могут быть как структурированные, так и неструктурированные. Основное требование – это возможность находить связи между данными, несмотря на характер их структурированности.

Добавив к рассмотренной тройке еще две характеристики, получим более расширенное представление об том, что же считается Big Data.

Value (ценность информации). С одной стороны важно стремиться извлекать максимум пользы по результатам анализа, с другой стороны, важно понимать, что ценность получаемой информации в большинстве случае очень сильно зависит от времени: чем раньше получена информация, тем больше времени есть на то, чтобы принять решение. Если система анализ данных дает информацию слишком долго так, что нет возможности успеть принять оптимальное решение, то вся такая работа теряет всякий смысл.

Veracity (достоверность). Недостоверность в данных очень сильно снижает ценность получаемой из них информации, так как принимаемых на их основе решения могут быть ошибочными. Система анализа больших данных должна давать определенную гарантию, что недостоверные данные будут отфильтрованы.

Но учитывая стремительное развитие области больших данных, определение постепенно обрастает новыми характеристика. Дополним рассмотренную пятерку еще тремя “V”.

Visualization (визуализация). Одним из эффективных решений проблемы восприятия человеком больших объемов данных является их визуальное представление. Система анализа больших данных должна иметь соответствующие инструменты для решения данной задачи.

Viscosity (вязкость). Данное явление может возникать из-за различий в источниках данных, разности интенсивностей потоков интеграции и обработки данных. Здесь мы можем задать следующий вопрос: «Насколько сложны данные, чтобы работать с ними?». Решение проблемы состоит в использовании улучшенной потоковой передачи, гибких интеграционных шин и обработку сложных событий.

Таким образом, когда говорят о больших данных, стоит понимать, что речь идет не только об объеме, но как минимум еще об их разнообразии, о скорости их появления и обработки. Также напомним, что само хранение

большого объема количества данных без его обработки не несет никакой ценности.

Накопленный опыт практического применения Big Data уже насчитывает сотни и тысячи примеров того, как внедрение технологий обработки больших данных дает компаниям новые возможности и решает их проблемы. Рассмотрим несколько кейсов из различных сфер, демонстрируя универсальность.

В медицине большие данные зарекомендовали себя при решении следующих проблем. Так анализ постов в социальной сети Twitter позволил специалистам из Бразилии для отслеживать ход эпидемии лихорадки денге. Этап сбора данных достаточно просто – отбирались посты, где присутствовало слово «денге», а затем посты распределялись по территориальному посту автора. Далее решалась задача определения постов с описанием реального опыта человека. Как показал опыт, корреляция с официальными данными и результатами анализа была достаточно высокой. Анализ данных, получаемых с различных устройств мониторинга за здоровьем, позволяет выявлять заболевания на самых ранних стадиях. Кроме этого, Big Data часто применяется для клинических испытаний.

В банковской сфере с помощью больших данных решаются следующие проблемы:

- контроль за использованием кредитных карт;
- неправомерное использование дебетовых карт;
- оценка прозрачности бизнеса;
- оценка платёжеспособности клиентов;
- противодействие отмыванию денег и др.

В производстве системы анализа больших данные также находят свое применение. Например, оптимизация производственных процессов, что сокращает издержки, позволяет планировать расход ресурсов. Также решаются задачи повышения безопасности на опасных производствах путем мониторинга рабочей среды, предотвращения выхода из строя оборудования.

Также Big Data применяется при разработке новых продуктов: анализ дизайна и потребительских свойств.

Другой важной сферой, где повсеместно используется анализ больших данных является маркетинг. Так, например, с помощью анализа предпочтений покупателей крупные интернет-магазины могут определять, что предложить покупателям, делающим похожие покупки. Создание таргетированной рекламы позволяет доставлять ее адресно тем посетителям, которые могут быть заинтересованы в предложении вероятней всего. Для достижения этой цели также нужно обладать и проанализировать большой массив информации о потенциальных пользователях той или иной услуги/продукта. Сходно функционируют программы лояльности, где компании нужно знать как можно больше о потребностях своих клиентах, чтобы удерживать их внимание и предлагать совершать больше покупок.

Сфера государственного управления также применяет большие данные для решения ряда следующих проблем:

– прогнозирование преступности;

– контроль за выдаваемыми пособиями;

– управление государственными финансовыми ресурсами;

– персонализация государственных сервисов и услуг;

– предвидение и предотвращение угроз национальной безопасности.

Как видно, Big Data применима практически в любой области. Однако стоит понимать, что и данный инструмент не является лишенным недостатков. Рассмотрим некоторые актуальные проблемы. Во-первых, имеет место инфраструктурные проблемы. Компания, планирующая внедрение технологий анализа больших данных, должна обладать возможностями не только хранения большого объема данных, но и необходимой пропускной способностью, чтобы иметь оперативный доступ к данным. В большинстве случаем данная проблема решается с помощью облачных сервисов хранения данных. Во-вторых, нужно четкое понимание для чего внедряются технологии больших данных. Одной из частей ответа на данный вопрос является

определение объема и срока хранения данных, в случае наличия данных с разными характеристиками появляется проблема приоритетов. Все это можно объединить под проблемой определения стратегии работы с большими данными. Неправильное определение стратегии будет неизбежно приводить к необоснованным тратам и потери большого числа ресурсов. В-третьих, важную роль играет то, как компания относится к защите данных. Компрометация или утечка данных, включая персональную или финансовую может стоит компании очень дорого, в связи с чем важно наладить управление доступом к информации среди сотрудников и решить многие другие вопросы, направленные на обеспечение защиты данных.

Остановимся на одной из распространенных моделей распределенной обработки данных, получившей название MapReduce. Основная концепция данной модели состоит в распределении обработки больших данных на компьютерных кластерах. Рассмотрим три основных этапа работы MapReduce.

Первый этап имеет название **map** и заключается в обработке и фильтрации входящих данных, осуществляемых при помощи задаваемой пользователем функции. Результатом является множество пар ключ-значение. При этом особое значение имеет именно ключ, так как при последующих этапах он будет использоваться для объединения данных. При этом в один момент времени может работать несколько экземпляров данной функции, что позволяет масштабировать данный этап. Другой особенностью является то, что запуск функции часто осуществляется на той же машине, где хранятся данные, что позволяет экономить ресурсы на пересылки данных.

Второй этап получил название **shuffle.** Единственные из трех этапов, на котором пользователь не может ни на что повлиять. На самом деле это не является недостатком, так как работа данного этапа заключается в распределении результатов первого по т.н. корзинам, где каждая корзина соответствует уникальному ключу. В результате получаем распределенные по отдельным корзинам данные, которые передаются на следующий этап. Отметим, что данный этап также может быть распределенным.

Третий этап имеет название **reduce**, где происходит финальное действие, задаваемое также пользователем в виде функции, которая будет применена для каждой корзины. Соответственно, данный этап также может быть распределенным.

Как можно заметить все этапы данной модели могут быть распределенными, а значит их легко масштабировать. Более подробно данные этапы и модель MapReduce будут рассмотрены в следующих темах.

Контрольные вопросы:

1. Какую дату можно считать точкой отсчета BigData?
2. Как можно определить понятие больших данных?
3. Какие составляющие входят в описание больших данных через три “V”?
4. Какие составляющие дополняют определение трех “V”?
5. Какие сферы применения больших данных можно выделить?
6. Какие проблемы могут возникать при работе с большими данными?
7. Что из себя представляет модель MapReduce?
8. Каковы основные этапы MapReduce и их содержание?

Источники по теме:

1. Big Data: Concepts, Methodologies, Tools, and Applications (Volume I, Section 1 Chapter 1 Big Data Overview)

2. Большие данные. Подготовка волнорезов. https://habr.com/ru/articles/290714/

3. Что такое «Big Data»? https://habr.com/ru/companies/productstar/articles/503580/

4. Билл Фрэнкс. Революция в аналитике. Как в эпоху Big Data улучшить ваш бизнес с помощью операционной аналитики. – М.: Альпина Диджитал, 2020. – 430с.

5. Nitin Kumar. Big Data Using Hadoop and Hive. David Pallai. Mercury

Learning and information. Dulles, 2021. – 201c.

# 2. ВВЕДЕНИЕ В HADOOP

Что же такое Hadoop? Это фреймворк для разработки распределенных систем анализа больших данных. Он позволяет хранить большие объемы данных в среде, которая может быть распределена между несколькими вычислительными кластерами, а также эффективно их обрабатывать. За счет особенностей свой архитектуры позволяет легко масштабировать систему, адаптируя ее к текущим требованиям компании. Реализован на языке программирования Java. Распространяется бесплатно с открытым исходным кодом.

Рассмотрим возможности Hadoop:

– обработка массивных наборов данных;

– низкая стоимость по сравнению с предоставляемой производительностью;

– возможность простого гибкого масштабирования;

– распределенная обработка;

– надежность обработки (обнаружение ошибок и попытка повторао обработки);

– поддержка принципа распределения ответственности;

– быстрая обработка за счет параллелизма;

– обеспечение отказоустойчивости и высокой доступности;

– широкие возможности конфигурирования;

– широкие возможности мониторинга.

Рассмотрим основные компоненты, которые включаются в Hadoop.

Центральным компонентом является распределенная файловая система Hadoop (Hadoop Distributed File System). Она обеспечивает хранения порций данных на различных машина кластера, тем самым создавая распределенную среду хранения, которая позволяет хранить очень большие файлы. Такой подход стал возможен благодаря наличию сервера имен, который знает, где какой файл находиться.

Появившийся во второй версии Hadoop фреймворк YARN отвечает за функции управления ресурсами, планирования и мониторинга задач.

Ключевым компонентом для распределенной обработки больших данных является MapReduce, концептуальная модель которого уже была представлена ранее.

Для обеспечения координации распределенной работы Hadoop используется компонент Zookeeper.

Для анализа данных используется высокоуровневый язык Pig (Pig Latin). С его помощью возможно легко создавать последовательности MapReduce программ.

Apache Hive представляет SQL подобный язык для извлечения данных, хранимых в Hadoop. По сути, это позволяет использовать декларативные конструкции языка SQL, которые будут транслированы в MapReduce программы, выполняющий требуемый запрос.

HBase является базой данных, которая поддерживает возможность создания схем, подобно реляционным базам данных, но при этом работает на распределенной файловой системой Hadoop.

Рассмотрев введение Hadoop, можно прийти к выводу, что данный фреймворк по сути позволяет хранить данные и выполнять над ними определенные операции. Данная функциональность напоминать ту, которую предлагают классические реляционные СУБД. Сравнив, в чем заключаются различия между ними и Hadoop.

Во-первых, стоит отметить, что SQL предназначен для оперирования только структурированными данными, в то время как Hadoop в ряде случае имеет дело с данными неструктурированными. В данном случае Hadoop выступает универсальным инструментов, чем реляционные базы данных. С другой стороны, как уже было отмечено имеется возможности использования SQL поверх Hadoop для работы со структурированными данными.

Во-вторых, стоит отметить разницу в подходах к пониманию того, что является единицей данных. В реляционных базах все данные хранятся в

таблицах, где отдельная запись будет представлена в виде кортежа. Это позволяет опираться на серьезно развитую формальную теорию. Однако, когда дело доходит для хранения определенных видов данных, таких как изображения, текстовые файлы, начинают возникать проблемы. В Hadoop же применяется подход, когда единицей данных является пара ключ-значение. Такой подход обеспечивает возможность хранить любые виды данных, а также универсально осуществлять обработку.

В-третьих, подобно пункту выше, имеются различия в том, как строится обработка данных. В реляционных базах данных используется декларативный язык SQL, с помощью которого задается желаемый результат, но как именно он будет достигнут и выполнен – задача СУБД. В Hadoop применяется подход функционального программирования, когда пользователь самостоятельно задает, что необходимо выполнить с данными. Такой выбор продиктован необходимостью обработки не только структурированных данных, но и с целью дать возможность выполнять продвинутые операции над различными типа данных, включая обработку изображений и других сложных форматов данных.

В-четвертых, имеется разница в том, какие режимы работы с данными закладывались при проектировании сравниваемых инструментов. Так, реляционные базы данных хорошо решают задачи, когда необходимо проводить манипуляции над конкретными единицами хранения: строками, столбцами, при этом возможно применение механизмов транзакций. В это же время Hadoop проектировался под задачу работы с данными, которые записываются один раз и многократно считаются. Так как при необходимости повторения запроса, Hadoop скорее-всего будет выполнять обработку снова для всех данных сразу.

Теперь подробно рассмотрим распределенную файловую систему Hadoop. Основная задача такого распределенного хранилища данных состоит в том, что абстрагироваться от сложности, которая возникает при работе с сетью. Пользователь Hadoop не должен думать, когда хочет получить тот или

иной файл, на какой конкретной машине он находится, особенно когда эти данные представляют собой колоссальные объемы. Именно это задача решается с помощью HDFS. Основной принцип хранения заключается в разделении больших файлов на мелкие блоки фиксированного размера, хранимые в рамках кластеров Hadoop. Стоит отметить, что частые обновления файлов поддерживаются очень плохо, это следствие заложенной модели «write-once-read-many», но которая дает очень высокую пропускную способность.

Рассмотрен возможности, предоставляемые HDFS:

– обеспечение автоматической обработки отказа оборудования;

– поддержка больших (и очень больших) объемов данных;

– высокая пропускная способность за счет использования модели, когда данные записываются один раз, а затем многократно считываются;

– принцип локальности данных, который обеспечивает обработку данных по месту их хранения;

– параллельная обработка, обеспечиваемая за счет разбиения больших файлов на мелкие фрагменты, которые мог быть обработаны параллельно;

– отказоустойчивость, обеспечиваемая за счет репликации блоков на разных нодах, с целью минимизации риска потери данных;

– различные возможности доступа к данным, предоставляемые посредством разнообразных API: командная строка, WebHDFS, RESTful API, HDFS Java API.

Рассмотрим, как хранятся данные в распределенной файловой системе. Для начала введем несколько понятий, которые составляют структурные блоки HDFS.

FileSystem Namespace – пространство имен в виде древовидной файловой системы, подобно традиционной файловой системе с поддержкой стандартного набора операций: создание, переименование и удаление файла, перемещение между директориями.

NameNode – это один сервер, в котором хранятся метаданные о всех файлах, на каком сервере они хранятся. Фактически данный сервер и поддерживает пространство имен. Клиент системы обращается именно к данному серверу с запросов.

DataNode – это сервер, на котором хранятся данные в виде фиксированных блоков. Стандартный размер блока составляет 128 MB. Сервер отвечает за создание, удаление и репликацию блоков, за счет получаемых инструкций от NameNode.

Важная особенность обработки данных, заложенная в Hadoop, состоит в том, что блоки данных могут обрабатываться независимо друг от друга. Кроме того, обработка блоков по возможности выполняется именно на том сервере, на котором они расположены. Такой подход позволяет уменьшить количество операций ввода/вывода, которые могли бы затрачиваться при перемещении данных с одной машины на другую. Если по каким-то причинам обработка не может полностью быть выполнена на том же сервере, на котором хранятся данные, тогда Hadoop попробует выполнить на ней часть задач, которые еще возможно выполнить, и только потом начнет прибегать к транспортировке данных. Данный процесс опирается на параметры, заложенные конфигурации.

Контрольные вопросы

1. Что такое Hadoop?

2. Какие возможности предоставляет Hadoop?

3. Какие основные компоненты входят в Hadoop?

4. Какой компонет Hadoop непосредственно отвечает за обработку данных?

5. Какие два компонента предоставляют возможность для доступа к данным, хранимых в Hadoop?

6. Каковы основные отличия хранения и обработки данных в Hadoop и в реляционной базе данных?

7. Каковы принципы работы заложены в распределенной файловой системе Hadoop?

8. Какие возможности предоставляет HDFS?

9. Каковы основные компоненты HDFS?

# 3. ОСНОВЫ MAPREDUCE В HADOOP

Как уже было отмечено выше MapReduce представляет собой модель обработки данных. Благодаря возможности масштабирования за счет разбиения вычислений на множество узлов кластера именно данная модель наиболее применима для обработки больших данных.

Определим, что примитивами обработки данных в MapReduce являются два вида функций, от названия которых и образовано название модели данных. Это распределители (map) и редукторы (reduce). Далеко не всегда просто разложить приложение таким образом, чтобы осуществлять обработку в данной модели, однако это цена за то, что мы получаем после: возможность масштабирования обработки на множество узлов.

Отметим также и то, что задачи, где требуется быстрый ответ или обработка данных, не допускающая разделения данных на множество узлов, не могут эффективно решаться с помощью MapReduce.

Процесс обработки в модели MapReduce более формально можно представить следующим образом. На этапе map происходит распределение значений:

$$(K_1,V_1) \rightarrow \text{list}(K_2,V_2)$$

Затем происходит скрытый этап перемешивания (shuffle), который можно описать следующим образом:

$$\text{list}(K_2,V_2) \rightarrow (K_2,\text{list}(V_2))$$

На этапе reduce происходит следующее преобразование:

$$(K_2,\text{list}(V_2)) \rightarrow \text{list}(K_3,V_3)$$

Более детально процесс обработки данных представлен на рис. 2.

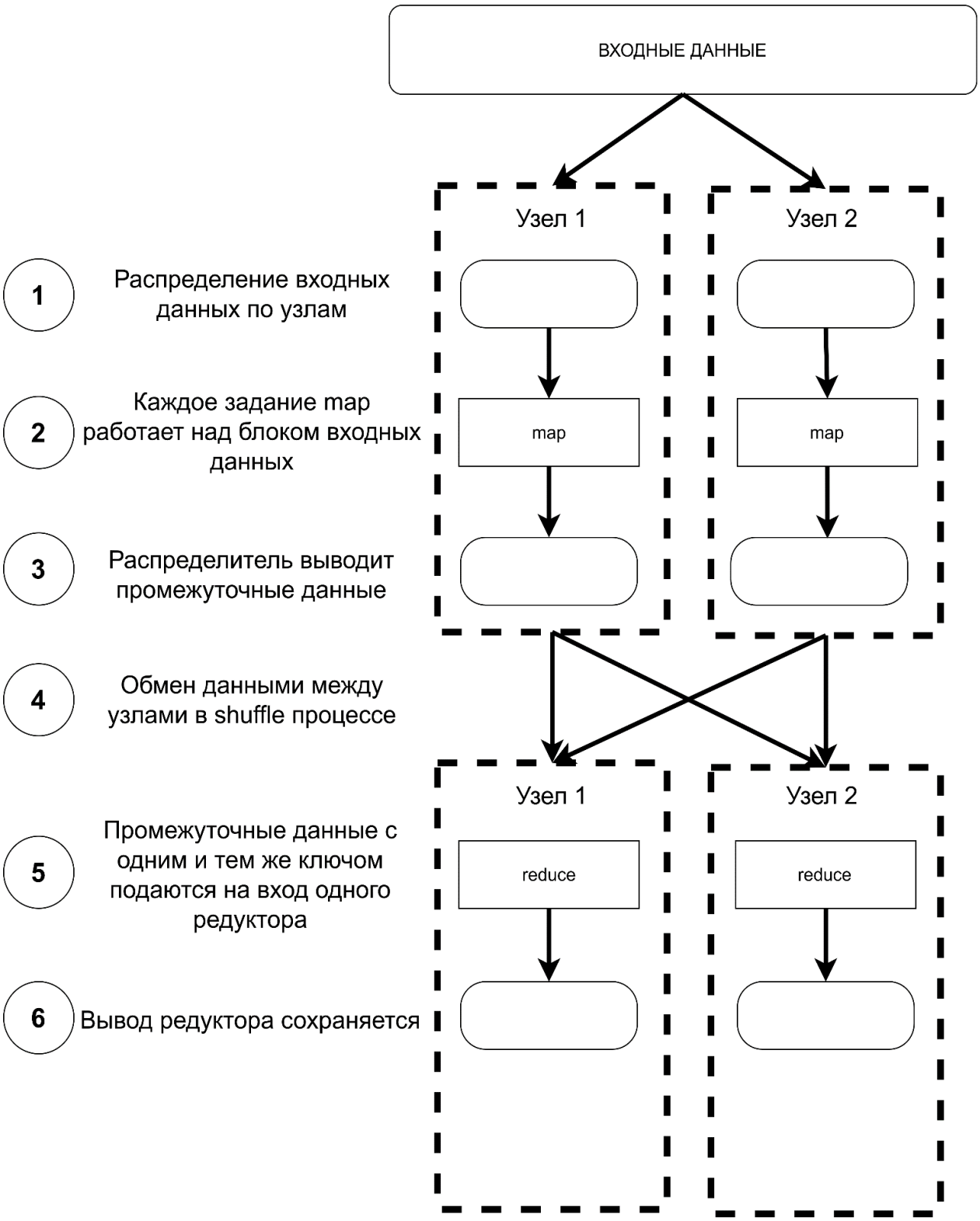


Рисунок 2 – Общий поток данных в MapReduce

Рассмотрим, какие типы данных поддерживает Hadoop. Важно отметить, что Hadoop не поддерживает в качестве ключей и значений пар стандартные классы Java, но предлагает свои собственные. Это связано в

первую очередь с тем, что Hadoop выполняет много работы внутри, включая сериализацию данных для передачи между узлами.

Все типы данных, которые используются в Hadoop, так или иначе связаны с двумя определенными интерфейсами: Writable и WritableComparable<T>, где WritableComparable<T> на самом деле расширяет интерфейсы Writable и Comparable<T> из стандартной библиотеки Java. Ключи пар должны реализовывать интерфейс WritableComparable<T>, так как на этапе reduce будет происходить их сортировка, поэтому они должны быть сравнимыми. А вот значения пар могут реализовывать либо WritableComparable<T>, либо просто Writable.

Hadoop предлагает ряд готовых реализаций, включая обертки для стандартных типов данных. Перечень таких классов представлен в таблице 1.

Таблица 1

Типы данных в Hadoop

| № | Класс | Описание |
|---|---|---|
| 1 | BooleanWritable | Обертка стандартного типа Boolean |
| 2 | ByteWritable | Обертка одного байта |
| 3 | DoubleWritable | Обертка типа Double |
| 4 | FloatWritable | Обертка типа Float |
| 5 | IntWritable | Обертка типа Integer |
| 6 | LongWritable | Обертка типа Long |
| 7 | Text | Обертка для хранения текста в кодировке UTF8 |
| 8 | NullWritable | Заглушка для случая, когда ключ или значение не нужны |

Кроме этого, можно создавать и пользовательские типы данных. Для этого достаточно реализовать тот или иной интерфейс.

Рассмотрим, как реализовываются распределители и редукторы. Все они являются наследниками класса MapReduceBase, который содержит два метода, играющих роль конструктора и деструктора.

Для создания распределителя также требуется реализовать интерфейс Mapper, содержащий единственный метод, в котором и происходит обработка. Ниже представлена сигнатура данного метода:

```
void map(K1 key, V1 value, OutputCollector<K2,V2> output,
Reporter reporter) throws IOException
```

Таблица 2

Параметры метода map

| № | Параметр | Описание |
|---|---|---|
| 1 | key | Ключ пары входных данных |
| 2 | value | Значение пары входных данных |
| 3 | output | Объект, получающий выходные данные распределителя |
| 4 | reporter | Объект предоставляет дополнительную информацию о ходе работы |

Hadoop также предлагает несколько готовых реализаций интерфейса Mapper.

IdentityMapper<K,V> – реализация, которая отображает вход на выход без преобразований.

InverseMapper<K,V> – реализация, которая меняет места ключ и значение.

RegexMapper<K> – реализация, порождающая пары для каждой входной пары, удовлетворяющей регулярному выражению.

TokenCountMapper<K> – реализация, порождающая пары, соответствующая лексемам, выделенным из входного значения.

Для создания редуктора, как и распределителя необходимо унаследовать класс MapReduceBase, а также реализовать интерфейс Reducer. Данный интерфейс содержит единственный метод, сигнатура которого рассмотрена далее.

void reduce(K2 key, Iterator<V2> values, OutputCollector<K3,V3> output, Reporter reporter) throws IOException

Таблица 3

Параметры метода reduce

| № | Параметр | Описание |
|---|---|---|
| 1 | key | Ключ пары входных данных |
| 2 | values | Итератор на значения входных данных |
| 3 | output | Объект, получающий выходные данные редуктора |
| 4 | reporter | Объект предоставляет дополнительную информацию о ходе работы |

Hadoop предлагает несколько готовых реализаций интерфейса Reducer.

IdentityReducer<K,V> – реализация, в которой не происходит редукция, а входные данные записываются на выход.

LongSumReducer<K,V> – реализация, в которой происходит суммирование по ключам.

Контрольные вопросы

1. Что представляет собой MapReduce?

2. Какие особенности MapReduce обуславливают ее применение для обработки больших данных?

3. Какие основные этапы выделяются в модели MapReduce?

4. Какие типы данных используются в Hadoop?

5. Что необходимо сделать, что реализовать пользовательский тип данных в Hadoop?

6. Что необходимо сделать, для определения собственного распределителя? редуктора?

# 4. ПРИМЕНЕНИЕ MAPREDUCE ДЛЯ РЕШЕНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

Рассмотрим простой пример решения задачи подсчета слов, используя модель обработки данных MapReduce.

Определим задачу следующим образом. Пусть дан произвольный текст. Необходимо произвести подсчет слов, входящих в него, то есть определить уникальные слова, входящие в его состав, и количество вхождений по каждому слову. Визуально, как будет осуществляться обработка с помощью MapReduce, можно увидеть на рис. 3.

На первом этапе происходит разделение данных из исходного текста. В случае примера разделение произведено по одной строке на узел обработке. В итоге получили первую вид пары ключ-значение, которые будут теперь обрабатываться параллельно.

На этапе распределения (map) происходит разделение исходной строки на отдельные слова, которые будут выступать в роли ключа для формируемых пар. Значение будет выступать «1», которая в данном случае интерпретируется как факт того, что некоторое слово встретилось один раз.

На этапе перемешивания (shuffle) полученные результаты предыдущего этапа распределяются таким образом, что пары с одинаковым ключом должны попасть в один редуктор.

На этапе редукции на вход подаются данные, где ключ – слово, а список значений состоит из единиц, обозначающих вхождение данного слово в тексте. Выходит, что остается просто просуммировать единицы, чтобы получить количество вхождений слова в тексте.

В результате получаем список пар, где ключ – уникальное слово, а значением выступаем количество его вхождений в тексте. Результаты также должны быть записаны в файл.

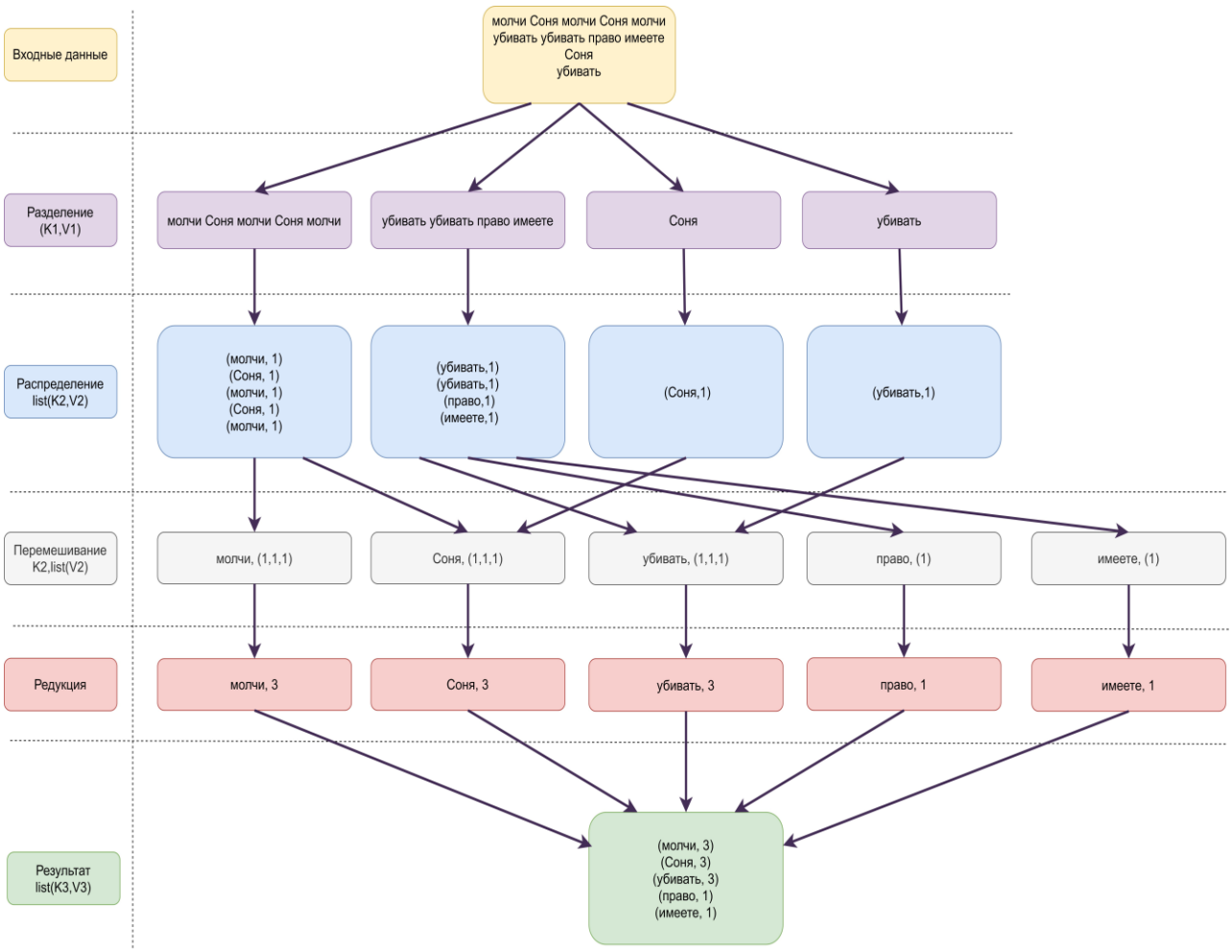


Рисунок 3 – MapReduce программа для подсчета слов

Теперь перейдем к обзору исходного кода, с помощью которого выполняется данная обработка. Начнем с операции распределения данных, то есть реализации класса Map. Данный класс наследует определенный в фреймворке MapReduce класс Mapper, который принимает четыре параметра типа: по два на входные и выходные типы пар ключ-значение. Так, принимать мы будем в качестве ключа некоторый идентификатор, которые никак не будет использовать в обработке, а в качестве значения непосредственно текст. На выходе будет отдавать пары, где ключом будет выступать уникальный токен (слово), а значение – константа – единица.

Для разделения строки на отдельные слова будет использовать класс StringTokenizer из стандартной библиотеки Java.

```
public static class Map
     extends Mapper<LongWritable,Text,Text,IntWritable> {

     public void map(LongWritable key, Text value, Context
context ) throws IOException,InterruptedException {
          String line = value.toString();
          StringTokenizer tokenizer = new
     StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
               value.set(tokenizer.nextToken());
               context.write(value, new IntWritable(1));
          }
     }
}
```

Определим код для редуктора. Для этого создаем класс Reduce, который наследует от класса Reducer из фреймворка MapReduce. Он, как и класс Mapper, принимает четыре параметра типа, определяющие типы для входных и выходных значений. Логика обработки сводиться к получению входных значений и суммированию их количества, так как мы точно знаем, что в значении может содержаться только одно значение – единица.

```
public static class Reduce
     extends Reducer<Text,IntWritable,Text,IntWritable> {

     public void reduce(Text key, Iterable<IntWritable>
values,
Context context) throws IOException,InterruptedException {
          int sum=0;
          for(IntWritable x: values) {
               sum++;
          }
          context.write(key, new IntWritable(sum));
     }
}
```

Кроме определения непосредственно обработчиков для двух основных операций MapReduce, требуется также определить класс, называемый драйвером. В нем конфигурируется задача, которая будет выполнена Hadoop. Так, здесь будет указано название задачи, переданы ссылки на созданные ранее классы обработчиков Map и Reduce, чтобы они вызывались на соответствующих стадия процесс обработки, а также другие параметры,

которые будут получены из командной строки при запуске задачи: пути, где будут браться исходные данные, а также куда будут записаны результаты обработки. Как правило следующий код может размещается в главном методе main().

```
Configuration configuration= new Configuration();
Job job = new Job(configuration,"Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

Запуск подобной программы осуществить достаточно просто. После сборки данной программы в JAR, необходимо выполнить следующую команду в командной строке:

```
$hadoop jar wordcount.jar input output
```

где `wordcount.jar` – название архива

`input` – директория в HDFS, где находятся входные данные

`output` – директория в HDFS, куда будут записываться выходные данные.

# 5. РАБОТА С ГРАФАМИ В HADOOP

Рассмотрим решение задачи анализа графов в Hadoop.

Пусть стоит задача выявить в заданном графе компоненты связности, а также количество вершин в каждой компоненте. Компонент связности – подграф графа, где существует путь между любыми двумя вершинами. На рис. 4 представлен граф, имеющий три компоненты связности, выделенные штриховкой.

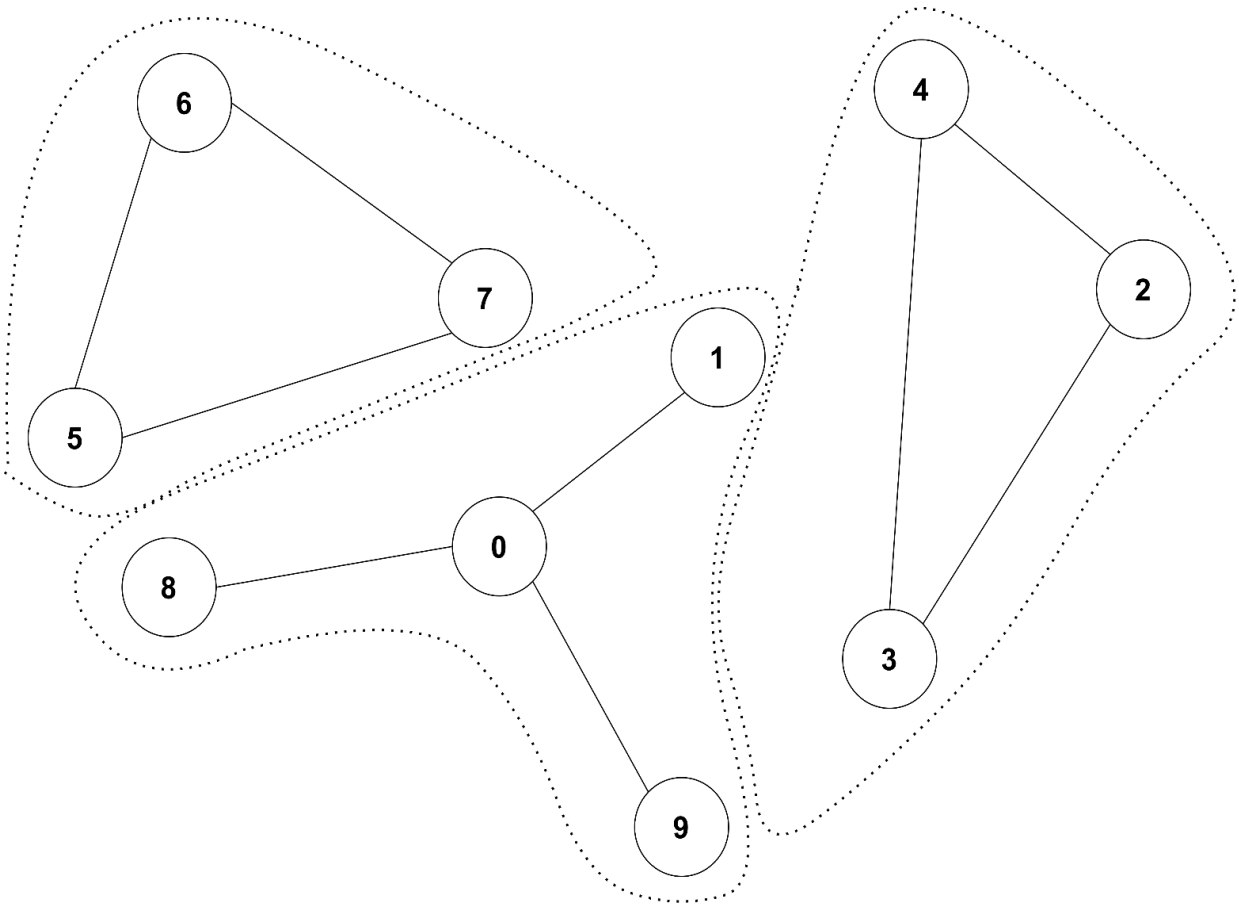


Рисунок 4 – Пример анализируемого графа

Входные данные, представляющие граф, будут в виде многострочного текст, где первое число строки будет обозначать номер вершины, а последующие числа – номера вершин, с которыми связана данная. Для графа, представленного на рис.4 текстовое представление:

```
0,1,8,9
```

```
1,0
2,3,4
3,2,4
4,2,3
5,6,7
6,5,7
7,5,6
8,0
9,0
```

Для решения данной задачи уже потребуется более, чем одна MapReduce программа, а именно три, выполняющие различные действия. Опишем, что будет происходить на каждом этапе.

Первый этап состоит в обработке исходных данных в нужный формат. Так как среди поставляемых типов данных Hadoop нет подходящего для хранения графа, создадим пользовательский, реализуя интерфейс Writable, имеющий два метода. Метод `void readFields(DataInput in)` необходим для десериализации объекта, а метод `void write(DataOutput out)` — для сериализации. Класс Vertex будет представлять вершину графа, храня ее номер, а также список смежных вершин. Среди дополнительных полей выделим номер группы. Данное поле будет использоваться в ходе обработке, для хранения номера компонента связности, которому принадлежит данная вершина.

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.RequiredArgsConstructor;
import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

@Data
@AllArgsConstructor
@RequiredArgsConstructor
public class Vertex implements Writable {
    private int tag;  // 0 - вершина графа, 1 - номер группы
```

```
        private long group; // номер группы, к которой
принадлежит вершина
        private long id; // номер вершины
        private List<Long> adjacent = new ArrayList<>();  //
смежные вершины

        public Vertex(int tag, long group) {
            this.tag = tag;
            this.group = group;
        }

        public String toString() {
            return "Vertex [tag" + tag + ", group=" + group + ",
id =" + id + ", adjacent=" + adjacent + "]";
        }

        public void readFields(DataInput in) throws IOException
{
            tag = in.readShort();
            group = in.readLong();
            id = in.readLong();

            adjacent = new ArrayList<>();
            int n = in.readInt();
            for (long i = 0; i < n; i++) {
                adjacent.add(in.readLong());
            }
        }

        public void write(DataOutput out) throws IOException {
            out.writeShort(tag);
            out.writeLong(group);
            out.writeLong(id);
            int nsize = adjacent.size();
            out.writeInt(nsize);

            for (int i = 0; i < nsize; i++) {
                out.writeLong(adjacent.get(i));
            }
        }

    }
```

Рассмотрим реализацию функции map на первом этапе обработке данных. В ее задачи входит считывание исходной сроки, и конвертация ее в экземпляр класса Vertex. Функция reduce на данном этапе фактически ничего не делает, а просто отображает данные для дальнейших действий. Данное преобразование можно описать следующим образом:

$$map: (Object,Text) \rightarrow list(LongWritable,Vertex),$$

$$reduce: (LongWritable,list(Vertex)) \rightarrow list(LongWritable,Vertex),$$

где Object – произвольный ключ, так как на данном этапе он не используется;

Text – входная строка, содержащая информацию об одной вершине;

LongWritable – во всех случаях представляет номер вершины;

Vertex – экземпляр класса, содержащий информацию о вершине.

```
     public static class InputDataReadMapper extends Mapper
<Object,Text,LongWritable,Vertex> {

    @Override
    public void map (Object Key,Text value,Context context)
     throws IOException, InterruptedException {

        Scanner sc = new
Scanner(value.toString()).useDelimiter(",");
        long vid= sc.nextLong();
        Vertex vertex= new Vertex();
        vertex.setTag(0);
        vertex.setId(vid);
        vertex.setGroup(vid);

        while(sc.hasNext()){
            vertex.getAdjacent().add(sc.nextLong());
        }
        context.write(new LongWritable(vertex.getId()), vertex);
        sc.close();
    }
}

public static class InputDataReadReducer extends
     Reducer<LongWritable,Vertex,LongWritable,Vertex>{

    public void reduce(LongWritable key, Iterable<Vertex>
values, Context context) throws IOException,
InterruptedException {
        for(Vertex v : values) {
            Vertex vertex = new Vertex(v.getTag(),
                                       v.getGroup(),
                                       v.getId(),
                                       v.getAdjacent());
            context.write(key,vertex);
        }
    }
}
```

Ниже также представлен вывод после обработки рассматриваемого графа.

```
0 -> Vertex [tag0, group=0, id =0, adjacent=[1, 8, 9]]
1 -> Vertex [tag0, group=1, id =1, adjacent=[0]]
2 -> Vertex [tag0, group=2, id =2, adjacent=[3, 4]]
3 -> Vertex [tag0, group=3, id =3, adjacent=[2, 4]]
4 -> Vertex [tag0, group=4, id =4, adjacent=[2, 3]]
5 -> Vertex [tag0, group=5, id =5, adjacent=[6, 7]]
6 -> Vertex [tag0, group=6, id =6, adjacent=[5, 7]]
7 -> Vertex [tag0, group=7, id =7, adjacent=[5, 6]]
8 -> Vertex [tag0, group=8, id =8, adjacent=[0]]
9 -> Vertex [tag0, group=9, id =9, adjacent=[0]]
======================
0 -> Vertex [tag0, group=0, id =0, adjacent=[1, 8, 9]]
1 -> Vertex [tag0, group=1, id =1, adjacent=[0]]
2 -> Vertex [tag0, group=2, id =2, adjacent=[3, 4]]
3 -> Vertex [tag0, group=3, id =3, adjacent=[2, 4]]
4 -> Vertex [tag0, group=4, id =4, adjacent=[2, 3]]
5 -> Vertex [tag0, group=5, id =5, adjacent=[6, 7]]
6 -> Vertex [tag0, group=6, id =6, adjacent=[5, 7]]
7 -> Vertex [tag0, group=7, id =7, adjacent=[5, 6]]
8 -> Vertex [tag0, group=8, id =8, adjacent=[0]]
9 -> Vertex [tag0, group=9, id =9, adjacent=[0]]
```

На втором этапе происходит следующая обработка данных.

Функция map получает исходные вершины графа, и создает новые пары, где ключ – это номер вершины, а значением выступает номер вершины, к которой можно перейти из ключа. Таким образом для каждой смежной с данной вершины будет создана пара. При этом в таких парах значение переменной tag будет установлено в 1, что указывает на то, что данная вершина не является начальной. Также данная функция запишет пары, где ключ и значение будут указывать на одну и ту же вершину. В таких пара поле tag объекта Vertex будет иметь значение 0, указывающее на то, что данная вершина является стартовой точкой.

Функция reduce производит следующее. Для всех вершин, которые приходят в редуктор с одинаковым ключом, выбирается такое число m, которое будет минимальным среди всех номеров вершин. Таким образом, определяется номер группы, к которой принадлежат все вершины. При этом вершина со значением переменной tag, равной нулю, сохраняет в себе

информацию о своих смежных вершинах. В результате получаем пары, где ключом является номер группы (минимальный номер вершины в данной компоненте связности). Важно отметить, что данная этап необходимо повторять определенное количество раз, чтобы получить финальный результат.

Данные преобразования можно описать следующим образом:

$$\text{map: (LongWritable,Vertex)} \rightarrow \text{list(LongWritable,Vertex)},$$

$$\text{reduce: (LongWritable,list(Vertex))} \rightarrow \text{list(LongWritable,Vertex)}.$$

Ниже представлен код для рассмотренных функций.

```
public static class VertexHandleMapper extends

Mapper<LongWritable,Vertex,LongWritable,Vertex> {

    public void map(LongWritable Key,Vertex value, Context
context )
                        throws IOException,
                    InterruptedException {
        context.write(new LongWritable(value.getId()),value);

        for(Long v : value.getAdjacent()) {
            context.write(
                    new LongWritable(v),
                    new Vertex(1,value.getGroup())
            );
        }
    }
}

public static class Reducer2 extends

Reducer<LongWritable,Vertex,LongWritable,Vertex> {

    public void reduce(LongWritable vid,Iterable<Vertex> values,
Context context) throws IOException, InterruptedException {
        List<Long> adjacent = new ArrayList<>();
        long m=Long.MAX_VALUE;
        for(Vertex vertex : values) {
            if(vertex.getTag() == 0) {
                adjacent = new
ArrayList<>(vertex.getAdjacent());
            }
            m = Math.min(m,vertex.getGroup());
        }
        context.write(
```

```
            new LongWritable(m),
            new Vertex((short)0,m,vid.get(),adjacent)
        );
    }
}
```

Ниже также представлен вывод после обработки рассматриваемого графа.

```
    0 -> [Vertex [tag0, group=0, id=0, adjacent=[1,8,9]], Vertex
[tag1, group=1, id=0, adjacent=[]], Vertex [tag1, group=8, id=0,
adjacent=[]], Vertex [tag1, group=9, id =0, adjacent=[]]]
    1 -> [Vertex [tag1, group=0, id=0, adjacent=[]], Vertex
[tag0, group=1, id=1, adjacent=[0]]]
    2 -> [Vertex [tag0, group=2, id =2, adjacent=[3, 4]], Vertex
[tag1, group=3, id =0, adjacent=[]], Vertex [tag1, group=4, id =0,
adjacent=[]]]
    3 -> [Vertex [tag1, group=2, id =0, adjacent=[]], Vertex
[tag0, group=3, id =3, adjacent=[2, 4]], Vertex [tag1, group=4, id
=0, adjacent=[]]]
    4 -> [Vertex [tag1,group=2,id=0,adjacent=[]], Vertex
[tag1,group=3, id =0, adjacent=[]], Vertex [tag0, group=4, id=4,
adjacent=[2,3]]]
    5 -> [Vertex [tag0, group=5, id =5, adjacent=[6, 7]], Vertex
[tag1, group=6, id =0, adjacent=[]], Vertex [tag1, group=7, id =0,
adjacent=[]]]
    6 -> [Vertex [tag1, group=5, id =0, adjacent=[]], Vertex
[tag0, group=6, id =6, adjacent=[5, 7]], Vertex [tag1, group=7, id
=0, adjacent=[]]]
    7 -> [Vertex [tag1, group=5, id =0, adjacent=[]], Vertex
[tag1, group=6, id =0, adjacent=[]], Vertex [tag0, group=7, id =7,
adjacent=[5, 6]]]
    8 -> [Vertex [tag1, group=0, id =0, adjacent=[]], Vertex
[tag0, group=8, id =8, adjacent=[0]]]
    9 -> [Vertex [tag1, group=0, id =0, adjacent=[]], Vertex
[tag0, group=9, id =9, adjacent=[0]]]
    ======================
    0 -> Vertex [tag0, group=0, id =0, adjacent=[1, 8, 9]]
    0 -> Vertex [tag0, group=0, id =1, adjacent=[0]]
    2 -> Vertex [tag0, group=2, id =2, adjacent=[3, 4]]
    2 -> Vertex [tag0, group=2, id =3, adjacent=[2, 4]]
    2 -> Vertex [tag0, group=2, id =4, adjacent=[2, 3]]
    5 -> Vertex [tag0, group=5, id =5, adjacent=[6, 7]]
    5 -> Vertex [tag0, group=5, id =6, adjacent=[5, 7]]
    5 -> Vertex [tag0, group=5, id =7, adjacent=[5, 6]]
    0 -> Vertex [tag0, group=0, id =8, adjacent=[0]]
```

```
0 -> Vertex [tag0, group=0, id =9, adjacent=[0]]
```

На третьем этапе происходят действия, подобные тем, что выполнялись при работе программы, осуществлявшей подсчет слов в тексте.

Функция map производит пары, где ключом является номер группы, а значением является единица. Количество таких пар будет соответствовать количеству вершины в группе.

Функции reduce остается только просуммировать такие пары. Преобразование можно записать следующим образом:

$$\text{map: (LongWritable,Vertex)} \rightarrow \text{list(LongWritable,IntWritable)},$$

$$\text{reduce: (LongWritable,list(IntWritable))} \rightarrow \text{list(LongWritable,LongWritable)}$$

.

```
public static class SizeCountMapper extends

Mapper<LongWritable,Vertex,LongWritable,IntWritable>{

    public void map(LongWritable group, Vertex values,Context
context)
                            throws IOException,
InterruptedException {
        context.write(group,new IntWritable(1));
    }
}

public static class SizeCountReducer extends
        Reducer<LongWritable,
IntWritable,LongWritable,LongWritable> {

    public void reduce(LongWritable key, Iterable<IntWritable>
values, Context context)throws IOException, InterruptedException
{
        long sum = 0;
        for(IntWritable v: values) {
            sum+=v.get();
```

```
        }
        context.write(key,new LongWritable(sum));
    }
}
```

Ниже также представлен вывод после обработки рассматриваемого графа.

```
0 -> [1, 1, 1, 1]
2 -> [1, 1, 1]
5 -> [1, 1, 1]
======================
0 -> 4
2 -> 3
5 -> 3
```

В результате получили искомый результат: группы и их размер.

# 6. ВВЕДЕНИЕ В PIG И HIVE

Apache Pig — это инструмент, который позволяет абстрагироваться от написания программ MapReduce. Он позволяет писать программы для анализа больших наборов данных, представляя их в виде потоков данных. Для этих целей Pig предоставляет язык высокого уровня под названием Pig Latin. Написанные на этом языке скрипты с помощью специального компонента Pig Engine транслируются в специальные функции map и reduce, которые затем выполняются в Hadoop. Таким образом, с помощью Pig можно писать более высокоуровневый код, избегая реализации функций анализа данных в Java. Более того, код на Pig будет выглядеть значительно короче, чем то же описание в терминах функций map и reduce на Java. В свою очередь это позволяет сократить и время разработки.

Такой подход, когда написанный на языке Pig Latin скрипт затем транслируется в задачи MapReduce. Позволяет производить оптимизации, которые будут происходить автоматически.

Скриптовый язык Pig Latin во многом заимствует идеи SQL, что облегчает его изучение тем, кто знаком с язык запросов к реляционным базам данных. Pig Latin предлагает множество встроенных операторов для поддержки операций с данными, таких как соединения, фильтры, упорядочивание и т. д. Кроме того, он также предоставляет вложенные типы данных, такие как кортежи, пакеты и карты, которые отсутствуют в MapReduce.

В случае, если какой-то функциональности требовательному пользователю не будет хватать, то Pig Latin позволяет разрабатывать свои собственные функции для чтения, обработки и записи данных, таким образом расширяя возможности стандартных средств языка. Часто для обозначения этой возможности используется аббревиатура UDF (user defined functions).

Работа с типами данных в Pig состоит в неформальном правиле: «Свинья ест все», что обозначает возможность обработки как структурированных, так

и неструктурированных данных. Ниже в таблице представлены стандартные поставляемые типы данных, используемые в PigLatin.

Таблица 4

Типы данных PigLatin

| № | Тип данных | Описание |
|---|---|---|
| 1 | int | Представляет 32-bit integer.<br>**Пример**: 8 |
| 2 | long | Представляет 64-bit integer.<br>**Пример**: 5L |
| 3 | float | Представляет 32-bit число с плавающей точкой.<br>**Пример**: 5.5F |
| 4 | double | Представляет а 64-bit число с плавающей точкой.<br>**Пример**: 10.5 |
| 5 | chararray | Представляет массив символов (UTF-8).<br>**Пример**: "tutorials point" |
| 6 | Bytearray | Представляет массив байт (blob). |
| 7 | Boolean | Представляет логическое значение.<br>**Пример**: true/ false. |
| 8 | Datetime | Представляет время.<br>**Пример**: 1970-01-01T00:00:00.000+00:00 |
| 9 | Biginteger | Представляет класс Java BigInteger.<br>**Пример**: 60708090709 |
| 10 | Bigdecimal | Представляет класс Java BigDecimal<br>**Пример**: 185.98376256272893883 |
| **Сложные типы данных** | | |
| 11 | Tuple | Представляет упорядоченный набор.<br>Пример: (raja, 30) |
| 12 | Bag | Представляет собой неупорядоченное множество.<br>Пример: {(raju,30),(Mohhammad,45)} |
| 13 | Map | Представляет собой карту.<br>Пример: ["name":"Raju", "age":"30"] |

Ниже в таблице представлены также наиболее употребляемые операторы языка Pig Latin, разделенные по группам.

Таблица 5

Операторы Pig Latin

| Оператор | Описание |
|---|---|
| **Загрузка и сохранение данных** | |
| LOAD | Загружает данные из файловой системы (local/HDFS) |
| STORE | Сохраняет результат в файловую систему (local/HDFS). |
| **Фильтрация** | |
| FILTER | Фильтрует строки. |
| DISTINCT | Фильтр, очищающий входные данные от одинаковых строк |
| FOREACH, GENERATE | Производит трансформацию данных на основе столбцов |
| STREAM | Производит трансформацию, используя внешнюю программу |
| **Объединение и группировка** | |
| JOIN | Производит объединение входных данных |
| COGROUP | Производит группировку входных данных |
| GROUP | Производит группировку входных данных в одну таблицу |
| CROSS | Создает перекрестное произведение входных данных |
| **Сортировка** | |
| ORDER | Производит сортировку таблицы в заданном порядке на основе одного или нескольких полей |
| LIMIT | Ограничивает вывод таблицы заданным числом строк |
| **Слияние и разделение** | |
| UNION | Соединяет две или несколько таблиц в одну. |

| | |
|---|---|
| SPLIT | Разделяет таблицу на две или более новых таблиц |
| **Операции для откладки** | |
| DUMP | Выводит содержимое таблицы в консоль |
| DESCRIBE | Выводит описание схемы таблицы. |
| EXPLAIN | Выводит логический, физический или MapReduce планы вычислений для данной таблицы. |
| ILLUSTRATE | Выводит пошаговое выполнение серии операторов |

Перейдем к рассмотрению Hive. До создания данного инструмента организации использовали коммерческое хранилище данных на базе РСУБД для анализа и обработки больших наборов данных. Но когда порядок ежедневных объемов данных вырос с гигабайт до терабайт, а затем и до петабайт, этого оказалось недостаточно для обработки таких больших растущих наборов данных. Идея была похожая с тем, что и у Pig: с MapReduce нелегко работать, сложно добиться возможности повторного использования ранее созданного решения. Разработчик может часами создавать программы MapReduce, чтобы анализировать простые наборы данных, что отрицательно влияет на производительность.

Решения, закладываемые при разработке Hive, позволяют использовать его в качестве системы управления и выполнения запросов к структурированным данных. Такое ограничение с одной стороны не позволяет работать с неструктурированными данными, но с другой позволяет производить определенные оптимизации, которые выполнить, используя только MapReduce значительно сложней. Тот факт, что в Hive используется SQL подобный язык позволяет значительно быстрее освоить его начинающим, а также в свое время открыло дорогу в Hadoop пользователям, которые не являются разработчиками.

Стоит отметить, что Hive не является реляционной базой данных, не подходит для обработки транзакций и выполнения запрос на уровне обновления строк.

Основные возможности, которые предоставляет Hive:

- хранение схемы в базе данных и обработка данных в HDFS;
- заложенная поддержка OLAP;
- предоставляет SQL-подобный язык запросов (HiveQL или HQL);
- легко осваиваемый, при знакомстве с SQL;
- быстрый;
- масштабирование;
- расширяемый.

Язык Hive (HiveQL) ведет себя как SQL. Он поддерживает SQL-запросы, такие как SELECT, JOIN (например, INNER JOIN, LEFT OUTER JOIN и RIGHT OUTER JOIN), декартово группировать и UNION ALL. Он также поддерживает различные команды, подобные СУБД, такие как как SHOW TABLES, CREATE TABLES и DESCRIBE TABLES.

Принцип хранения данных в Hive можно описать следующим образом. Хранение данных таблиц в Hive осуществляется в HDFS. Каждая таблица сопоставляется с данными каталога. Разделы таблицы представляют собой отдельные подкаталоги. Сегменты – отдельные файлы в подкаталогах разделов.

База данных Hive работает как каталог для хранения пространства имен таблиц и помогает определять группу таблиц для пользователя и избежать конфликта имен. Hive инициализируется с базой данных по умолчанию (default); однако пользователь также может создать собственную базу данных. Hive создает папки/каталоги для каждой базы данных и подкаталоги для таблицы состоят из этой конкретной базы данных; однако мы можем явно определить другое место при создании таблицы HiveQL — это язык запросов Hive, аналогичный языку SQL реляционных баз данных. но он предназначен для обработки больших данных. Рассмотрим основные отличия:

- отсутствие поддержки таких запросов как INSERT, UPDATE и DELETE для записей;

– отсутствие поддержки транзакций для получения еще большей производительности при обработки больших массивов данных, хотя в некоторых случаях Hive предоставляет данную возможность.

Ниже представлена таблица с подробным сравнением.

Таблица 6

Сравнение реляционных баз данных с Hive

| № | Признак сравнения | SQL в реляционной базе данных | Hive |
|---|---|---|---|
| 1 | Язык | PL/SQL | Открытие и чтение файлов |
| 2 | CRUD | INSERT, UPDATE, DELETE | INSERT OVERWRITE, |
| 3 | Транзакции | Да | Нет (только в Hadoop 3) |
| 4 | Скорость выполнения запросов | ms | минуты |
| 5 | Поддержка индексов | Да | Нет, данные всегда полностью пересматриваются |
| | Порядок размера оперируемых данных | терабайты | петабайты |

Контрольные вопросы:

1. Что такое Pig в Hadoop?
2. Чем отличается MapReduce программа от скрипта Pig Latin?
3. Какие основные типы данных существуют в Pig Latin?
4. С какими типами данных можно работать, используя Pig?
5. Какие основные операторы имеются в Pig Latin?
6. Что такое Hive в Hadoop?

7. Какие особенности имеет Hive, относительно типов данных, с которыми можно работать?

8. Что позволяет, а что нет Hive по сравнению с использованием SQL в реляционных базах данных?

## ПЕРЕЧЕНЬ ЛИТЕРАТУРЫ:


1. Чак Лэм. Графы и алгоритмы Hadoop в действии. – М.: ДМК Пресс, 2012. – 424с.: ил.

2. Nitin Kumar. Big Data Using Hadoop and Hive. David Pallai. Mercury Learning and information. Dulles, 2021. – 201c.

3. Deepak Vohra. Kubernetes Microservices with Docker. Apress Media. White Rock, British Columbia. 2016. – 440c.

4. Билл Фрэнкс. Революция в аналитике. Как в эпоху Big Data улучшить ваш бизнес с помощью операционной аналитики. – М.: Альпина Диджитал, 2020. – 430с.

5. Aravind Shenoy. Hadoop Explained. – Birmingham: Packt Publishing, 2014. – 426c.

6. Tom White. Hadoop: The Definitive Guide, Fourth Edition. – Sebastopol: O'Reilly Media, 2015. – 756c.

7. Гудов, А. М.; Базы данных и системы управления базами данных. Программирование на языке PL/SQL : учебное пособие.; Кемеровский государственный университет, Кемерово; 2010; https://biblioclub.ru/index.php?page=book&id=232497 (Электронное издание)

8. Дьяков, И. А.; Базы данных. Язык SQL : учебное пособие.; Тамбовский государственный технический университет (ТГТУ), Тамбов; 2012; https://biblioclub.ru/index.php?page=book&id=277628 (Электронное издание)

9. Kevin Sitto and Marshall Presser. Field Guide to Hadoop. – Sebastopol: O'Reilly Media, 2015. – 132c.

10. Nick Dimiduk and Amandeep Khurana. HBase in Action. –Shelter Island, Manning Publications, 2013. – 362c.