|  |  |
| --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-01 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Факультет |  | И | |  |  | Информационные и управляющие системы |
|  |  | шифр | |  |  | наименование |
| Кафедра |  | И9 | |  |  | Систем управления и компьютерных технологий |
|  |  | шифр | |  |  | наименование |
| Дисциплина |  |  | Научно-исследовательская работа | | | |

|  |
| --- |
| Курсовая работа  на тему:  Анализ технологий обработки неструктурированных данных |
|  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | |  | | И9М31 |
| Николаев А.А. | | | | | | |
| Фамилия И.О. | | | | | | |
| Проверил | | | | | | |
| Верхолат А.М. | |  |  | | | |
| Фамилия И.О. Подпись | | | | | | |
| Оценка |  | | | |  | |
| «\_\_\_\_\_» |  | | | | 2018 г. | |

САНКТ-ПЕТЕРБУРГ

2018 г.

Содержание

[ВВЕДЕНИЕ 4](#_Toc5014122)

[1 Неструктурированные данные 7](#_Toc5014123)

[1.1 Обработка неструктурированной информации 8](#_Toc5014124)

[2 Основные технологии обработки неструктурированной информации 11](#_Toc5014125)

[2.1 Apache Hadoop 11](#_Toc5014126)

[2.1.1 HDFS 12](#_Toc5014127)

[2.1.1.1 Узлы имен 13](#_Toc5014128)

[2.1.1.2 Узлы данных 14](#_Toc5014129)

[2.1.1.3 Клиенты HDFS 14](#_Toc5014130)

[2.1.1.4 Файловые операции и репликация 15](#_Toc5014131)

[2.1.1.5 Ограничения HDFS 15](#_Toc5014132)

[2.1.2 MapReduce 16](#_Toc5014133)

[2.1.2.1 Hadoop MapReduce 18](#_Toc5014134)

[2.1.3 Экосистема Hadoop 23](#_Toc5014135)

[2.1.3.1 Apache Hive 24](#_Toc5014136)

[2.1.3.2 Apache Pig 26](#_Toc5014137)

[2.1.3.3 Преимущества и недостатки использования Hadoop для обработки неструктурированных данных. 27](#_Toc5014138)

[2.2 Apache Spark 27](#_Toc5014139)

[3.2.1 Ядро Spark 29](#_Toc5014140)

[2.2.2 SparkSQL 31](#_Toc5014141)

[2.2.3 Spark Streaming 32](#_Toc5014142)

[2.2.4 MLlib и GraphX 33](#_Toc5014143)

[2.2.5 Spark как часть системы обработки неструктурированных данных 33](#_Toc5014144)

[2.3 Apache Tez 34](#_Toc5014145)

[ЗАКЛЮЧЕНИЕ 36](#_Toc5014146)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc5014147)

# ВВЕДЕНИЕ

В современном мире информация занимает важное место в любой отрасли человеческой деятельности. Источников различных данных великое множество. В их качестве могут выступать непрерывно поступающие данные с измерительных устройств, события от радиочастотных идентификаторов, потоки сообщений из социальных сетей, метеорологические данные, данные дистанционного зондирования земли, потоки данных о местонахождении абонентов сетей сотовой связи, устройств аудио- и видео-регистрации. Собственно, массовое распространение перечисленных выше технологий и принципиально новых моделей использования различного рода устройств и интернет-сервисов послужило отправной точкой для проникновения больших данных едва ли не во все сферы деятельности человека. В первую очередь, научно-исследовательскую деятельность, коммерческий сектор и государственное управление.

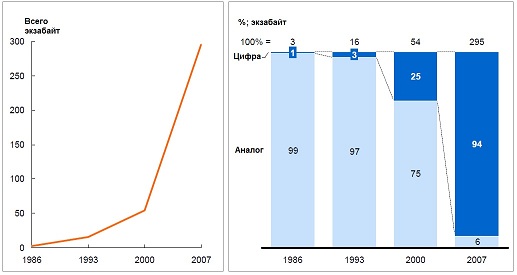
[](http://www.tadviser.ru/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:1_BigData1.jpg&filetimestamp=20120202180717)

Рис.1-Рост объемов данных (слева) на фоне вытеснения аналоговых средств хранения (справа)

Несколько занимательных и показательных фактов:

* В 2010 году корпорации мира накопили 7 экзабайтов данных, на наших домашних ПК и ноутбуках хранится 6 экзабайтов информации;
* Всю музыку мира можно разместить на диске стоимостью 600 долл;
* В 2010 году в сетях операторов мобильной связи обслуживалось 5 млрд телефонов;
* Каждый месяц в сети Facebook выкладывается в открытый доступ 30 млрд новых источников информации;
* Ежегодно объемы хранимой информации вырастают на 40%, в то время как глобальные затраты на ИТ растут всего на 5%;
* По состоянию на апрель 2011 года в библиотеке Конгресса США хранилось 235 терабайт данных;[1]

Американские компании в 15 из 17 отраслей экономики располагают большими объемами данных, чем библиотека Конгресса США. К примеру, датчики, установленные на авиадвигателе, генерируют около 10 Тб за полчаса. Примерно такие же потоки характерны для буровых установок и нефтеперерабатывающих комплексов. Только один сервис коротких сообщений Twitter, несмотря на ограничение длины сообщения в 140 символов, генерирует поток 8 Тб/сут. Так как объемы данных очень большие,не все данные можно структурировать заранее об определенную модель. Так же можно обратить внимание на то, что множество из генерируемой информации является полезной только после проведенного анализа над ним и извлечения из нее знаний. Из этого можно сделать несколько выводов, касающихся современных подходов к обработке информации:

1. Структура данных не всегда может быть определена для всей информации заранее.
2. “Сырые” данные могут быть полезны и должны быть обработаны для получения знаний в контексте какой-либо области знаний.
3. Скорость обработки таких данных важна для эффективности какого-либо процесса внутри применяемой области.[2]

В рамках данной курсовой работы будет исследованы основные инструменты хранения и обработки неструктурированной информации и выбрано наиболее эффективное для дальнейшего использования в рамках выполнения выпускной квалификационной работы.

# 1 Неструктурированные данные

Дадим определение понятию неструктурированные данные. Неструктурированные данные — информация, которая либо не имеет заранее определенной структуры данных, либо не организована в установленном порядке. Термин «неструктурированные данные» может считаться неточным по нескольким причинам:

* Структура, даже если она не определена формально, может подразумеваться.
* Данные, обладающие структурой некоторой формы, могут по-прежнему характеризоваться как неструктурированные, если их структура не предназначена для машинной обработки.
* Неструктурированная информация может иметь некоторую структуру (такая информация называется полу-структурированной) или даже быть хорошо структурированной, но теми способами, которые являются неочевидными без предварительного согласования.[3]

Важным в определении неструктурированных данных является то, что даже если структура и определена или информация является полу-структурированной, она не всегда может быть сохранена базой данных и в дальнейшем обработана в первоначальном состоянии. То есть прежде чем работать с такими данными нужно решить задачу их первоначальной обработки.

Обработка и анализ неструктурированных данных тесно связана с задачами Big Data. Большие данные (англ. big data) — 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. То есть под понятием больших данных имеют ввиду не конкретный объем информации, а определенные методы их обработки подходящего как для огромного массива данных, так и для некой конкретной “cырой” информации.

В качестве определяющих характеристик для больших данных традиционно выделяют «три V»: объём (англ. *volume*, в смысле величины физического объёма), скорость (*velocity* в смыслах как скорости прироста, так и необходимости высокоскоростной обработки и получения результатов), многообразие (*variety*, в смысле возможности одновременной обработки различных типов структурированных и полуструктурированных данных).Характеристики определяют , что одной из задач в области Big data является определение методов обработки слабоструктурированной информации и ее анализ.[4]

# 1.1 Обработка неструктурированной информации

Задача обработки неструктурированной информации в науке о данных входит в задачу ETL. **ETL** (от англ. *Extract, Transform, Load* — дословно «извлечение, преобразование, загрузка») — один из основных процессов в управлении хранилищами данных, который включает в себя:

* извлечение данных из внешних источников;
* их трансформация и очистка, чтобы они соответствовали потребностям бизнес-модели;
* и загрузка их в хранилище данных.

С точки зрения процесса ETL, архитектуру хранилища данных можно представить в виде трёх компонентов:

* источник данных: содержит структурированные данные в виде таблиц, совокупности таблиц или просто файла (данные в котором разделены символами-разделителями);
* промежуточная область: содержит вспомогательные таблицы, создаваемые временно, и, исключительно для организации процесса выгрузки.
* получатель данных: хранилище данных или база данных, в которую должны быть помещены извлечённые данные.[5]

Требования к организации потока данных описываются аналитиком. ETL следует рассматривать не только как процесс переноса данных из одного приложения в другое, но и как инструмент подготовки данных к анализу.

Начальным этапом процесса ETL является процедура извлечения записи из источников данных и подготовка их к процессу преобразования. При разработке процедуры извлечения данных, в первую очередь необходимо определить частоту выгрузки данных из OLTP-систем или отдельных источников.

Процедуру извлечения данных можно реализовать двумя способами:

* извлечение данных с помощью специализированных программных средств;
* извлечение данных средствами той системы, в которой они хранятся.[6]

После извлечения данные помещаются в так называемую «промежуточную область», где для каждого источника данных создаётся своя таблица или отдельный файл, или и то и другое.

После этапа извлечения данных идет этап преобразования данных. Цель этого этапа — подготовка данных к размещению в хранилище данных и приведение их к виду более удобному для последующего анализа. При этом должны учитываться некоторые, выдвигаемые аналитиком, требования, в частности, к уровню качества данных. Поэтому в процессе преобразования может быть задействован самый разнообразный инструментарий, начиная с простейших средств ручного редактирования данных и заканчивая системами, реализующими сложные методы обработки и очистки данных. В процессе преобразования данных в рамках ETL чаще всего выполняются следующие операции:

* преобразование структуры данных;
* агрегирование данных;
* перевод значений;
* создание новых данных;
* очистка данных. [7]

И последним этапом является этап загрузки данных Процесс загрузки заключается в переносе данных из промежуточных таблиц в структуру хранилища данных. При очередной загрузке в хранилище данных переносится не вся информация из источников, а только та, которая была изменена в течение промежуточного времени, прошедшего с предыдущей загрузки. При этом выделяют два потока:

* поток добавления — в хранилище данных передается новая, ранее не существовавшая информация;
* поток обновления (дополнения) — в хранилище данных передается информация, которая существовала ранее, но была изменена или дополнена.

Для распределения загружаемых данных на потоке используются средства данных. Они фиксируют состояние данных в некоторые моменты времени и определяют, какие данные были изменены или дополнены.

Рассматривая задачу ETL для неструктурированных данных, мы говорим о переходе к читаемому машиной формату данных с возможность последующего анализа этих данных. То есть средство обработки должно включать в себя возможность хранения сырых данных, возможность преобразования таких данных в читаемый машиной формат и дальнейший анализ этой информации.

# 2 Основные технологии обработки неструктурированной информации

Задача обработки неструктурированной информации входит в задачи Big Data.Основными технологиями внутри понятия Big Data являются фреймворк Apache Hadoop и его экосистема (программы, которые входят в состав полной системы Hadoop участвуют в его работе). Рассмотрим это средство и последовательность действий для реализации задачи обработки неструктурированной информации и анализа этих данных.

# 2.1 Apache Hadoop

Apache **Hadoop** – это проект с открытым исходным кодом, находящийся под управлением Apache Software Foundation. Обычно используется различными фирмами как хранилище файлов общего назначения, способное вместить петабайты данных для исследовательских и производственных целей. Базовая инфраструктура Apache Hadoop состоит из следующих модулей[8]:

* Hadoop Common- содержит библиотеки и утилиты, необходимые для других модулей Hadoop;
* Распределенная файловая система Hadoop (HDFS)- распределенная файловая система, которая хранит данные на товарных машинах, обеспечивая очень высокую совокупную пропускную способность в кластере;
* «Hadoop YARN» - платформа, отвечающая за управление вычислительными ресурсами в кластерах и использующая их для планирования приложений пользователей
* Hadoop MapReduce- реализация модели программирования MapReduce для крупномасштабной обработки данных.

На рисунке 3 представлена схема работы Apache Hadoop:

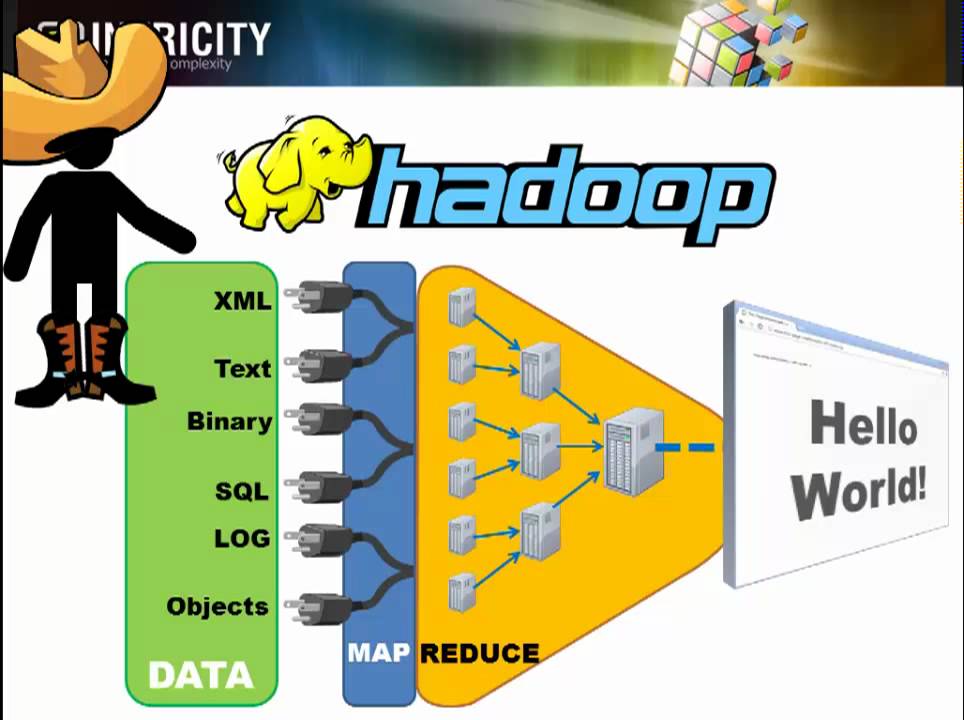


Рис.2-Схема Apache Hadoop

# 2.1.1 HDFS

HDFS (*Hadoop Distributed File System*) — файловая система, предназначенная для хранения файлов больших размеров, поблочно распределённых между узлами вычислительного кластера. HDFS является иерархической файловой системой. Таким образом, в HDFS имеется поддержка вложение каталогов. В каталоге может располагаться ноль или более файлов, а также любое количество подкаталогов.[10]

HDFS состоит из следующих обязательных компонентов:

* **Узел имен (NameNode)** – программный код, выполняющийся, в общем случае, на выделенной машине экземпляра HDFS и отвечающий за файловые операции (работу с метаданными);
* **Узел данных (DataNode)** – программный код, как правило, выполняющийся выделенной машине экземпляра HDFS и отвечающий за операции уровня файла (работа с блоками данных).

Hadoop содержит единственный узел типа NameNode и произвольное количество узлов типа DataNode.

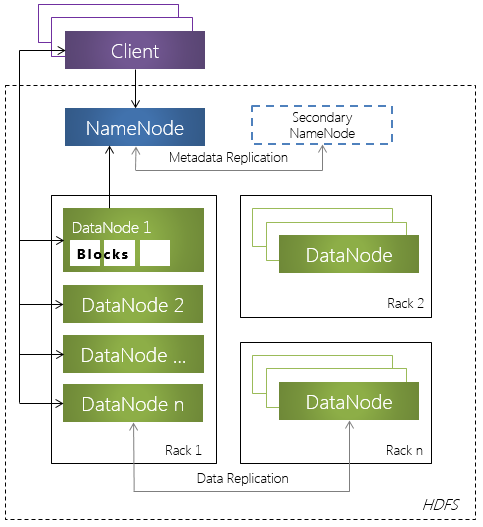


Рис.3 Архитектура HDFS

Основные концепции, заложенные при проектировании HDFS это:

* **Объем данных**
* **Отказоустойчивость**
* **Самодиагностика**
* **Производительность**

# 2.1.1.1 Узлы имен

**Узел имен (NameNode)** представляет собой программный код, выполняющийся, в общем случае, на выделенной машине экземпляра HDFS и отвечающий за файловые операции, такие как открытие и закрытие файлов, создание и удаление каталогов[11]. Кроме того, NameNode отвечает за:

* управление пространством имен файловой системы;
* управление доступом со стороны внешних клиентов;
* соответствие между файлами и реплицированными на узлах данных блоками.

Hadoop содержит единственный узел типа NameNode. Что порождает уязвимость всего кластера, вызванную выходом узел типа NameNode (единичная точка отказа). HDFS поддерживает вторичный узел имен – Secondary NameNode. Часто это факт является причиной заблуждения, что при отказе первичного узла имен, его автоматически заменит вторичный узел имен.

Вторичный узел имен выполняет следующие функции:

* копирует образ HDFS (расположенный в файле FsImage) и лог транзакций операций с файловыми блоками (EditLog) во временную папку;
* применяет изменения, накопленные в логе транзакций к образу HDFS;
* записывает новый образ FsImage на узел NameNode, после чего происходит очистка EditLog.

# 2.1.1.2 Узлы данных

Узел данных (DataNode), как и узел NameNode, также представляет собой программный код, выполняющийся, как правило, на выделенной машине экземпляра HDFS и отвечающий за операции уровня файла, такие как: запись и чтение данных, выполнение команд создания, удаления и репликации блоков, полученные от узла NameNode.

Кроме того, узел DataNode отвечает за:

* периодическую отправку сообщения о состоянии (heartbeat-сообщения);
* обработку запросов на чтение и запись, поступающие от клиентов файловой системы HDFS, т.к. данные проходят с остальных машин кластера к клиенту мимо узла NameNode.

# 2.1.1.3 Клиенты HDFS

Клиенты представляют собой программных клиентов, работающих с файловой системой. В роли клиента может выступать любое приложение или пользователь, взаимодействующий через специальный API с файловой системой HDFS. Для клиента HDFS выглядит как обычная файловая система – иерархия каталогов с вложенными в них подкаталогами и файлами. Как и в файловых системах общего назначения, клиенту, при наличии достаточных прав, разрешены следующие операции: создание, удаление, переименование, перемещение. Вышеназванные операции применимы к каталогам и файлам. Наиболее существенное отличие работы клиента с файловой системой HDFS от работы с файловой системой общего назначение – это то, что при создании файла клиент может явно указать размер блока файла (по умолчанию 64 Мб) и количество создаваемых реплик (по умолчанию значение равно 3-ем).

# 2.1.1.4 Файловые операции и репликация

Набор допустимых файловых операций в распределенной файловой системе HDFS схож с набором файловых операций в «локальных» файловых системах за исключением операции модификации файла – модификация в HDFS не поддерживается по причинам, связанным с архитектурными особенностями (в том числе и вопросами производительности и блокировок) этой файловой системы. За все файловые операции отвечает узел NameNode. Операции с конкретными файлами находятся в зоне ответственности узла DataNode, на котором эти файлы находятся.

# 2.1.1.5 Ограничения HDFS

Файловая система HDFS обладает следующими ограничениями, связанных с архитектурой системы[12]:

* узел имен NameNode является единой точкой отказа;
* отсутствие полноценной репликации Secondary NameNode;
* отсутствие возможности дописывать или оставить открытым для записи файлы в HDFS (как следствие, в plain Hadoop отсутствует поддержка обновляемых и потоковых данных);
* отсутствие поддержки реляционных моделей данных;
* отсутствие инструментов для поддержки ссылочной целостности данных;
* низкая безопасность данных.

HDFS является неотъемлемой частью проекта, однако, Hadoop поддерживает работу и с другими распределёнными файловыми системами без использования HDFS, поддержка Amazon S3 и CloudStore реализована в основном дистрибутиве. С другой стороны, HDFS может использоваться не только для запуска MapReduce-заданий, но и как распределённая файловая система общего назначения, например, поверх неё реализована распределённая NoSQL-СУБД HBase, в её среде работает масштабируемая система машинного обучения Apache Mahout.

# 2.1.2 MapReduce

MapReduce – это представленная компанией Google модель распределённых вычислений, а также её реализации, используемые для параллельной обработки больших объёмов информации. Работа MapReduce состоит из шагов Map и Reduce, названных аналогично функциям высшего порядка из многих языков программирования, применяемым на этих шагах. Программы, использующие реализацию MapReduce, автоматически распараллеливаются и исполняются на кластере, состоящем из множества связанных между собой компьютеров. Исполнительная система сама заботится о деталях разбития входных данных на части, планировании исполнения программы на наборе машин, обработке сбоев и управлении необходимым сообщением между машинами. Принцип работы заключается в том, что функция map принимает на вход список и некую функцию, затем применяет данную функцию к каждому элементу списка и возвращает новый список. Функция reduce (свёртка) преобразует список к единственному атомарному значению при помощи заданной функции, которой на каждой итерации передаются новый элемент списка и промежуточный результат. Для обработки данных пользователь библиотеки MapReduce должен только определить две эти функции, а также указать имена входных и выходных файлов и параметры обработки[13]:

* Шаг Map

Входные данные решаемой задачи представляют большой список значений, и на Map-шаге происходит его предварительная обработка. Для этого главный узел кластера (master node) получает этот список, делит его на части и передает рабочим узлам (worker node). Далее каждый из рабочих узлов преобразует элементы полученной коллекции в ноль или несколько промежуточных пар «ключ-значение».

* Шаг Reduce

На Reduce-шаге главный узел получает промежуточные ответы от рабочих узлов и передаёт их на свободные узлы для выполнения следующего шага. Система сортирует и группирует по ключу все пары «ключ-значение» и затем для каждой пары «ключ-группа значений» сворачивает значения, часто в одно или в пустой список. Получившийся результат – это решение задачи, которая изначально формулировалась.

Канонический пример приложения, написанного с помощью MapReduce – это программа на псевдокоде, подсчитывающая количество различных слов в наборе документов:

Листинг 1

// Функция, используемая рабочими узлами на Map-шаге для обработки пар ключ-значение из входного потока

void map(String name, String document):

// Входные данные:

// name – название документа

// document – содержимое документа

for each word w in document:

EmitIntermediate(w, "1");

// Функция, используемая рабочими узлами на Reduce-шаге для обработки пар ключ-значение, полученных на Map-шаге

void reduce(String word, Iterator partialCounts):

// Входные данные:

// word – слово

// partialCounts – список группированных промежуточных результатов. Количество записей в partialCounts и есть требуемое значение

int result = 0;

for each v in partialCounts:

result += parseInt(v);

Emit(AsString(result));

Модель MapReduce применима в широкой области задач, включая распределённый поиск, распределённую сортировку, обращение графа веб-ссылок, обработку статистики логов сети, построение инвертированных индексов, кластеризацию документов, машинное обучение и статистический машинный перевод. Более того, MapReduce была адаптирована под такие вычислительные среды, как многопроцессорные системы, добровольные вычислительные, динамические облачные и мобильные среды. Внутри Hadoop MapReduce помимо анализа данных реализованы алгоритмы извлечения информации из неструктурированных данных для дальнейшей работы над ними.

# 2.1.2.1 Hadoop MapReduce

**Hadoop MapReduce** – программная модель выполнения распределенных вычислений для больших объемов данных в рамках парадигмы map/reduce, представляющая собой набор Java-классов и исполняемых утилит для создания и обработки заданий на параллельную обработку.[14]

Основные концепции Hadoop MapReduce можно сформулировать как:

* обработка/вычисление больших объемов данных;
* масштабируемость;
* автоматическое распараллеливание заданий;
* работа на ненадежном оборудовании;
* автоматическая обработка отказов выполнения заданий.

Работу Hadoop MapReduce можно условно поделить на следующие этапы:

**Input read:**

Входные данные делятся на блоки данных предопределенного размера (от 16 Мб до 128 Мб) – сплиты (от англ. split). MapReduce Framework закрепляет за каждой функцией Map определенный сплит.

**Map:**

Каждая функция Map получает на вход список пар «ключ/значение» <k,v>, обрабатывает их и на выходе получает ноль или более пар <k',v'>, являющихся промежуточным результатом: map(k, v) -> [(k', v')]где k' - в общем случае, произвольный ключ, не совпадающий с k. Все операции map() выполняются параллельно и не зависят от результатов работы друг друга. Каждая функция map () получает на вход свой уникальный набор данных, не повторяющийся ни для какой другой функции map ().

**Partition / Combine:**

Целью этапа partition (разделение) является распределение промежуточных результатов, полученных на этапе map, по reduce-заданиям:

(k', reducers\_count) -> reducer\_id где reducers\_count - количество узлов, на которых запускается операция свертки; reducer\_id - идентификатор целевого узла. В простейшем случае, reducer\_id = hash(k') mod reducers\_count. Основная цель этапа partition – это балансировка нагрузки. Некорректно реализованная функция partition может привести к неравномерному распределению данных между reduce-узлами. Функция combine запускается после map-фазы. В ней происходит промежуточная свертка, локальных по отношению к функции map, значений - [(k', v')] -> (k', [v']). Основное значение функции combine – комбинирование промежуточных данных, что в свою очередь ведет, к уменьшению объема передаваемой между узлами информации.

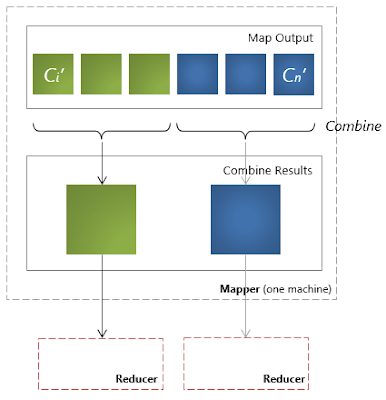
[](https://2.bp.blogspot.com/-ZzLy4OVNaCI/UDDGmTkaJtI/AAAAAAAAAgc/7pQZRbNaiAc/s1600/combine.PNG)

Рис.4-Схема Map/Reduce для одного Data Node

**Copy/Сompare/Merge:**

* Copy: копирование результатов, полученных в результате работы функций map и combine (если такая была определена), с map-узлов на reduce-узлы.
* Сompare (или Sort): сортировка, группировка по ключу k полученных в результате операции copy промежуточных значений на reduce-узле. compare(k'n, k'n+1) -> {-1, 0, +1}
* Merge: «слияние» данных, полученных от разных узлов, для операции свёртки.

**Reduce:**

Framework вызывает функцию reduce для каждого уникального ключа k' в отсортированном списке значений. reduce(k', [v']) -> [v''] Все операции reduce() выполняются параллельно и не зависят от результатов работы друг друга. Таким образом, результаты работы каждой функции reduce() пишутся в отдельный выходной поток.

**Output wri**te:

Результаты, полученные на этапе reduce, записываются в выходной поток (в общем случае, файловые блоки в HDFS). Каждый reduce-узел пишет в собственный выходной поток.

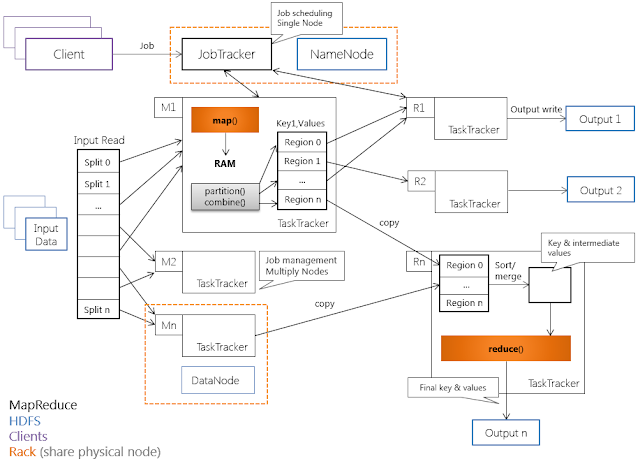
[](https://1.bp.blogspot.com/-aRTkMr8y8Yw/UDDGzFIm7ZI/AAAAAAAAAgo/tawDNpaH6Rk/s1600/a.PNG)

Рис.5-Полная схема Map/Reduce процесса на кластере Hadoop

Разработчику приложения для Hadoop MapReduce необходимо реализовать базовый обработчик, который на каждом вычислительном узле кластера обеспечит преобразование исходных пар «ключ/значение» в промежуточный набор пар «ключ/значение» (класс, реализующий интерфейс Mapper), и обработчик, сводящий промежуточный набор пар в окончательный, сокращённый набор (класс, реализующий интерфейс Reducer).

Все остальные фазы выполняются программной моделью MapReduce без дополнительного кодирования со стороны разработчика. Кроме того, среда выполнения Hadoop MapReduce выполняет следующие функции:

* планирование заданий;
* распараллеливание заданий;
* перенос заданий к данным;
* синхронизация выполнения заданий;
* перехват «проваленных» заданий;
* обработка отказов выполнения заданий и перезапуск проваленных заданий;
* оптимизация сетевых взаимодействий.

При создании архитектуры/разработке программных систем с использованием Hadoop MapReduce следует учитывать следующие аспекты использования MapReduce:

Преимущества:

* Эффективная работа с большим (от 100 Гб) объемом данных;
* Масштабируемость;
* Отказоустойчивость;
* Унифицированность подхода;
* Снижение требований к квалификации разработчика, в том числе его знаний и опыта по написанию многопоточного кода;
* Дешевизна лицензирования (Open Source).

Ограничения:

* Смешение ответственности для Reducer (сортировка и агрегация данных). Таким образом, Reducer – это все, что «не map»;
* Отсутствие контроля над потоком данных у разработчика (поток данных управляется фреймворком Hadoop MapReduce автоматически);
* Как следствие предыдущего пункта, невозможность простыми средствами организовать взаимодействие между параллельно выполняющимися потоками.

Недостатки:

* Эффективность применение MapReduce снижается при малом количество машин в кластере (высоки издержки на взаимодействие, а степень распараллеливания невелика);
* Невозможно предсказать окончание стадии map;
* Этап свертки не начинается до окончания стадии map;
* Как следствие предыдущего пункта, задержки в исполнении любого запущенного map-задания ведут к задержке выполнения задачи целиком;
* Низкая утилизация ресурсов вследствие жесткого деления ресурсов кластера на map- и reduce-слоты.
* Сбой узла JobTracker приводит к простою всего кластера.

# 2.1.3 Экосистема Hadoop

Центральное место экосистемы Hadoop занимает хранилище данных (Data Storage). Hadoop поддерживает хранение как неструктурированных данных с распределенной файловой системе HDFS, так и структурированных данных в нереляционной базе данных HBase. Фреймворк MapReduce отвечает за планирование задач и выполнение распределенных вычислений. Рассмотрим основные компоненты экосистемы Hadoop в контексте обработки неструктурированных данных[15]:

* Apache Hive или Cloudera Impala – data warehouse инструменты для SQL-like обработки больших данных.
* Pig – язык программирования для анализа данных на высоком уровне.
* Apache Mahout – библиотека и движок машинного обучения на больших данных;
* Apache Sqoop – Технология передачи большого количества данных, хранящихся на кластере Hadoop, и хранилищами структурированных данных, такие как, реляционные базы данных
* Apache Flume – Технология передачи большого количества данных, хранящихся на кластере Hadoop, и нереляционными базами данных.
* Apache Avro – Сериализация данных
* Apache Oozie – Управление задачами внутри Hadoop

Так же стоит заметить, что для хранения метаданных используются специальные NoSQL СУБД разных типов, тип которой выбирается при определении задачи.

На рисунке 5 представлена схема Hadoop c основными технологиями экосистемы:

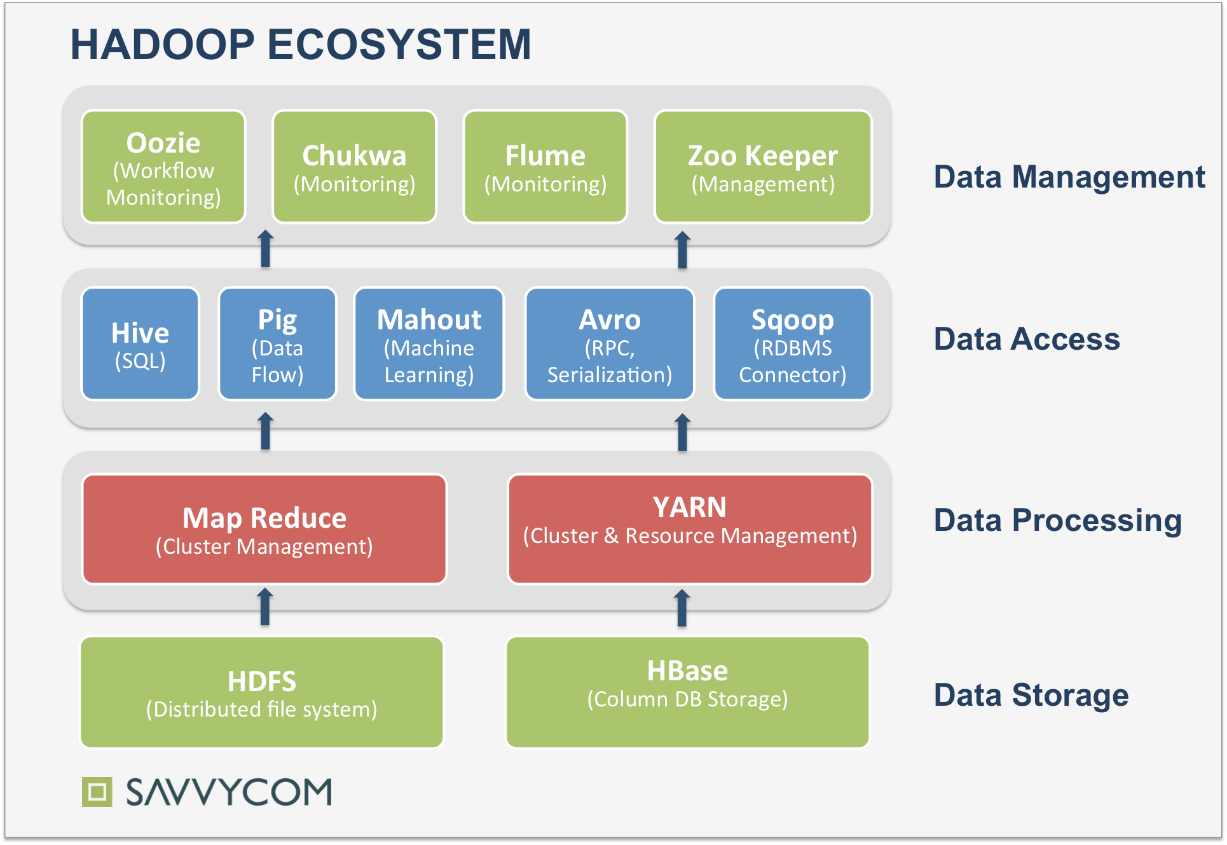


Рис.5-Схема Apache Hadoop и её экосистемы

# 2.1.3.1 Apache Hive

Классические СУБД, такие как Postgres, MySQL или Oracle не имеют такой гибкости в масштабировании при обработке больших массивов данных Apache Hive был придуман для того чтобы объединить два этих достоинства[16]:

* Масштабируемость MapReduce
* Удобство использования SQL для выборок из данных.

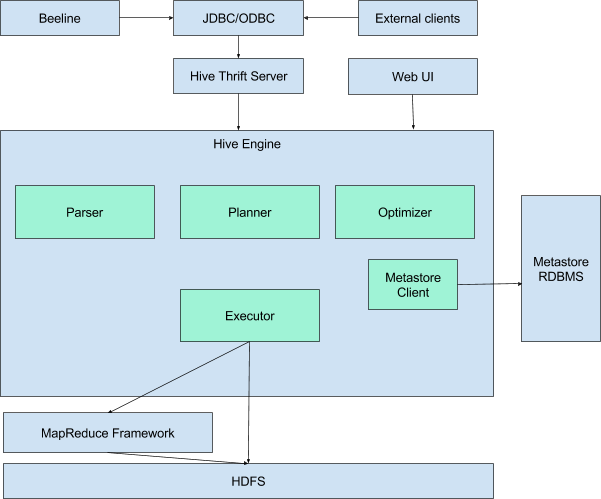


Рис.6-Архитектура Apache Hive

Apache Hive представляет из себя движок, который превращает SQL-Like-запросы в цепочки Map/Reduce задач. Движок включает в себя такие компоненты, как Parser (разбирает входящие SQL-запросы), Optimimer (оптимизирует запрос для достижения большей эффективности), Planner (планирует задачи на выполнение), Executor (запускает задачи на фреймворке MapReduce. Для работы Hive также необходимо хранилище метаданных. Дело в том, что SQL предполагает работу с такими объектами как база данных, таблица, колонки, строчки, ячейки и поскольку сами данные, которые использует Hive хранятся просто в виде файлов на HDFS — необходимо где-то хранить соответствие между объектами Hive и реальными файлами. В качестве хранилища метаданных используется обычная реляционная СУБД, такая как MySQL, PostgreSQL, Oracle.

Использует специальный язык Apache Hive Hive Query Language, который основан на языке SQL, но не имеет полной поддержки стандарта SQL-92. HiveQL имеет функции для работы с форматами XML и JSON, поддержку нескалярных типов данных, таких как массивы, структуры, ассоциативные массивы, поддерживает широкий набор агрегирующих функций, определяемые пользователем функции (User Defined Functions), блокировки

# 2.1.3.2 Apache Pig

**Apache Pig** – это платформа для анализа наборов данных больших объёмов, которая состоит из высокоуровневого языка для написания программ анализа данных, а также инфраструктуры выполняющей эти программы. Характерным свойством программ Pig является то, что их структура поддается существенному распараллеливанию, что в свою очередь позволяет с его помощью обрабатывать очень большие наборы данных. В Apache Pig используется язык Pig Latin, который характеризуется следующими ключевыми свойствами[17]:

* Простота программирования. Задачи анализа данных легко распределяются по кластеру. Сложные задачи, состоящие из нескольких взаимосвязанных преобразований данных, явно кодируются в виде последовательности потока данных, что позволяет их легко писать и делает их понимаемыми и поддерживаемыми.
* Возможности оптимизации. Способ, в котором закодированы задачи позволяет системе автоматически оптимизировать их выполнение, позволяя пользователю сосредоточиться на семантике, а не эффективности.

Пользователи могут создавать свои собственные функции для специальных целей. Pig Latin может быть расширен с помощью функции, определённых пользователями, написанными на Java, Python, JavaScript, Ruby или Groovy. Пользовательские функции могут вызываться Pig Latin.Также как и Hive, Pig облегчает задачу написания Map/Reduce заданий и увеличивает возможности использования и обработки данных.

# 2.1.3.3 Преимущества и недостатки использования Hadoop для обработки неструктурированных данных.

Hadoop кластер хорошо подходит для обработки неструктурированной информации и дальнейшего его анализа с помощью различных компонентов внутри экосистемы. Однако существуют некоторые ограничения, связанные с архитектурой данной системы при которой применение этого средства будет не эффективно и излишним.

1. Так как Hadoop обрабатывает данные разных типов с помощью распределенных вычислений то, исходя из опыта разработчиков, его не эффективно использовать его при небольших объёмах данных, однако все равно задачу обработки можно решить при потерях скорости обработки
2. Использовать Hadoop нужно тогда, когда стоит задача не только обработки неструктурированных данных, но и их хранение данных.
3. Hadoop сложно использовать при важности скорость обработки данных или аналитика данных в реальном времени
4. Данные внутри HDFS не часто обновляются (это так же связанно с принципом построения HDFS «один раз записать – много раз прочитать»).

В общем, после исследования данного программного средства можно прийти к выводу, что его эффективное использование Hadoop – это хранение большого количества различных типов данных (как структурированных, так и неструктурированных) и дальнейшая их обработка в рамках поставленной задачи, при не частом обновлении этих данных и незначительной важности скорости обработки данных.

# 2.2 Apache Spark

Как можно было заметить, технология Apache Hadoop не всегда может быть эффективно использована для обработки и анализа неструктурированных данных. К примеру, её если нам нужно получать обработанные данные быстро или неструктурированные данные часто обновляются внутри хранилища, то использование Apache Hadoop будет неэффективно и лучше уйти в сторону к другим решениям в области решения этой задачи. Так же не стоит использовать Apache Hadoop как обработчик неструктурированных данных если объём данных не превышает объем одной машины.

Если пользователя интересует скорость обработки и частота обновления неструктурированных данных, при отсутствии задачи эффективного хранения, то стоит использовать технологию Apache Spark.

Apache Spark — программный каркас с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию MapReduce с дисковым хранилищем, использует специализированные примитивы для рекуррентной обработки в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач, в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения. По сравнению с Hadoop Spark ускоряет работу программ на одном кластере более чем в 100 раз, а на диске – более чем в 10 раз. Проект предоставляет программные интерфейсы для языков Java, Scala, Python, R. Состоит из ядра и нескольких расширений, таких как Spark SQL (позволяет выполнять SQL-запросы над данными), Spark Streaming (надстройка для обработки потоковых данных), Spark MLlib (набор библиотек машинного обучения), GraphX (предназначено для распределённой обработки графов).[18]

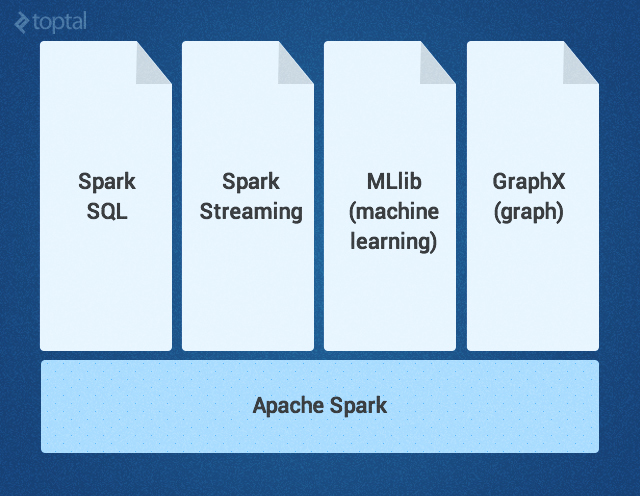


Рис.7-Архитектура Apache Spark

# 3.2.1 Ядро Spark

Ядро Spark – это базовый движок для крупномасштабной параллельной и распределенной обработки данных. Ядро отвечает за:

* управление памятью и восстановление после отказов
* планирование, распределение и отслеживание заданий кластере
* взаимодействие с системами хранения данных

В Spark вводится концепция RDD (Resilient Distributed Dataset), (устойчивый распределенный набор данных) – неизменяемая отказоустойчивая распределенная коллекция объектов, которые можно обрабатывать параллельно. В RDD могут содержаться объекты любых типов; RDD создается путем загрузки внешнего набора данных или распределения коллекции из основной программы (driver program). В RDD поддерживаются операции двух типов:

* Трансформации – это операции (например, отображение, фильтрация, объединение и т.д.), совершаемые над RDD; результатом трансформации становится новый RDD, содержащий ее результат.
* Действия – это операции (например, редукция, подсчет и т.д.), возвращающие значение, получаемое в результате некоторых вычислений в RDD.

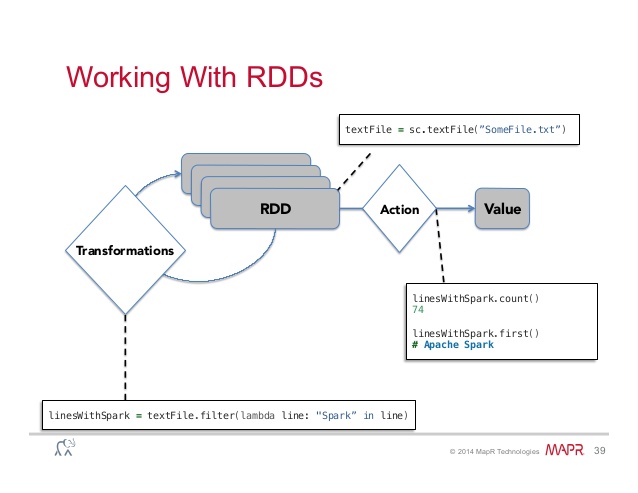


Рис.8-Архитектура ядра Spark

Трансформации в Spark осуществляются в «ленивом» режиме — то есть, результат не вычисляется сразу после трансформации. Вместо этого они просто «запоминают» операцию, которую следует произвести, и набор данных (напр., файл), над которым нужно совершить операцию. Вычисление трансформаций происходит только тогда, когда вызывается действие, и его результат возвращается основной программе. Благодаря такому дизайну повышается эффективность Spark. Например, если большой файл был преобразован различными способами и передан первому действию, то Spark обработает и вернет результат лишь для первой строки, а не станет прорабатывать таким образом весь файл.

По умолчанию каждый трансформированный RDD может перевычисляться всякий раз, когда вы выполняете над ним новое действие. Однако RDD также можно долговременно хранить в памяти, используя для этого метод хранения или кэширования; в таком случае Spark будет держать нужные элементы на кластере, и вы сможете запрашивать их гораздо быстрее.

# 2.2.2 SparkSQL

SparkSQL – это компонент Spark, поддерживающий запрашивание данных либо при помощи SQL, либо посредством Hive Query Language. Библиотека возникла как порт Apache Hive для работы поверх Spark (вместо MapReduce), а сейчас уже интегрирована со стеком Spark. Она не только обеспечивает поддержку различных источников данных, но и позволяет переплетать SQL-запросы с трансформациями кода; получается очень мощный инструмент.

Благодаря представлению информации в виде так называемых упругих распределенных наборов данных (RDD), осуществляется быстрый и простой доступ к данным через прикладные интерфейсы на Python, Scala и Java. Кроме того, такая технология позволяет запрашивать данные и одновременно запускать сложные алгоритмы их анализа.

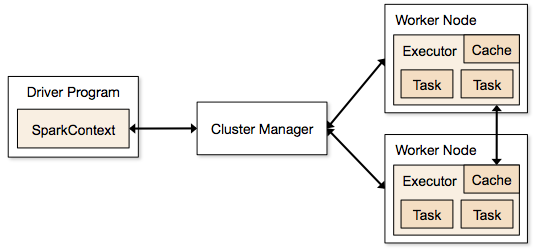
[](http://datareview.info/wp-content/uploads/2014/08/cluster-overview.png)

Рис.9-Архитектура ядра Spark

Ниже приведен пример Hive-совместимого запроса:

Листинг 2

// sc – это существующий SparkContext.

val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")

sqlContext.sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")

// Запросы формулируются на HiveQL

sqlContext.sql("FROM src SELECT key, value").collect().foreach(println)

# **2.2.3 Spark Streaming**

Spark Streaming поддерживает обработку потоковых данных в реальном времени; такими данными могут быть файлы логов рабочего веб-сервера (напр. Apache Flume и HDFS/S3), информация из соцсетей, например, Twitter, а также различные очереди сообщений вроде Kafka. «Под капотом» Spark Streaming получает входные потоки данных и разбивает данные на пакеты. Далее они обрабатываются движком Spark, после чего генерируется конечный поток данных (также в пакетной форме) как показано ниже.

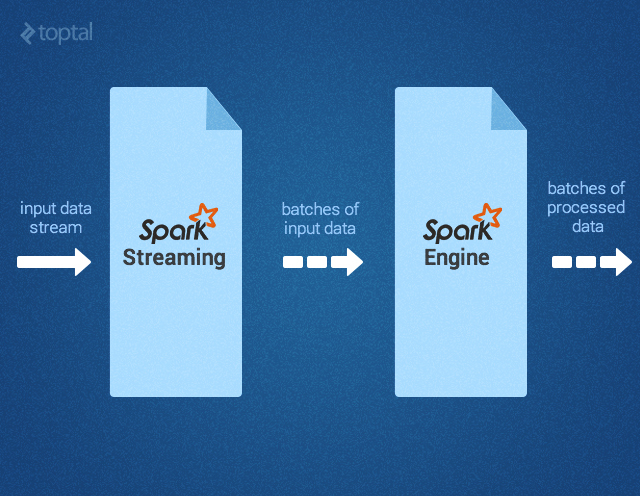


Рис.10 - Архитектура ядра Spark

API Spark Streaming точно соответствует API Spark Core, поэтому программисты без труда могут одновременно работать и с пакетными, и с потоковыми данными.

# 2.2.4 MLlib и GraphX

MLlib – это библиотека для машинного обучения, предоставляющая различные алгоритмы, разработанные для горизонтального масштабирования на кластере в целях классификации, регрессии, кластеризации, совместной фильтрации и т.д. Некоторые из этих алгоритмов работают и с потоковыми данными — например, линейная регрессия с использованием обычного метода наименьших квадратов или кластеризация по методу k-средних (список вскоре расширится). Apache Mahout (библиотека машинного обучения для Hadoop) уже ушла от MapReduce, теперь ее разработка ведется совместно с Spark MLlib

GraphX – это библиотека для манипуляций над графами и выполнения с ними параллельных операций. Библиотека предоставляет универсальный инструмент для ETL, исследовательского анализа и итерационных вычислений на основе графов. Кроме встроенных операций для манипуляций над графами здесь также предоставляется библиотека обычных алгоритмов для работы с графами, например, PageRank.

# 2.2.5 Spark как часть системы обработки неструктурированных данных

Так как Spark является платформой обработки стоит заметить, что она не решает задачу хранения неструктурированных данных в полной мере. Обычно этот фреймворк используется как часть “экосистемы” для обработки различных типов данных, как структурированных, так и неструктурированных. Например, его можно использовать как часть Hadoop-кластера, который будет отвечать за обработку данных вместо Map/Reduce, что ускорит задачу агрегации в десятки раз и позволяет работать с данными в реальном времени.

Главная причина недостаточности MapReduce при работе с аналитическими приложениями, в машинном обучении и т. п. заключена в применении ациклической модели потоков данных (Directed Acyclic Graph, DAG), не пригодной для реализации какой-либо возможности вложения циклов на всем протяжении процесса от Map до Reduce, что противоречит основной идее подобных приложений. В данный момент, чтобы адаптировать модель MapReduce для целей анализа, необходимо многократно повторять основной цикл, а это приводит к заметному снижению производительности. Apache Spark позволяет в десятки раз увеличить скорость обработки — благодаря переходу на принципиально иную алгоритмическую основу. Spark можно рассматривать как дальнейшее развитие идеи DAG с возможностью циклической обработки потоков данных, размещенных в памяти. Отход от пакетного режима поддержан новым, не имеющим аналогов способом представления данных, получившим название Resilient Distributed Dataset (RDD). Организация данных в виде RDD позволяет распределять и перераспределять данные на различных стадиях параллельных вычислений, открывая путь к итерационным процессам, невозможным в случае DAG. Вот почему RDD оказывается эффективным с точки зрения, как скорости обработки, так и обеспечения высокой надежности, требуемой при распределенной работе на кластерах.[19]

# 2.3 Apache Tez

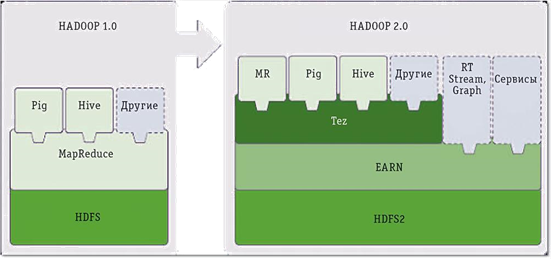
Другим альтернативным движком обработки внутри “экосистемы” Hadoop является Apache Tez. Он также увеличивает эффективность обработки данных в сравнении с классическим Map/Reduce подходом, использующемся в Hadoop.Однако в сравнении с Apache Spark не с помощью перехода на принципиально иную алгоритмическую основу, а усовершенствовании старого алгоритма. Apache Tez выполняет DAG как набор задач, интерпретируя обработку данных как граф потоков, где вершины — процессы, а ребра — перемещения данных между процессами. В вершинах размещается аналитическая логика, а ребра отражают процесс передачи данных и связи между их источниками и потребителями.[20] В Tez единый этап Reduce разделяется на отдельные,связанные между собой стадии Reduce. Такое деление стало возможным с появлением конструкции YARN (см рис. 1.3.1), поддерживающей связь с файловой системой HDFS2 и и кластером.

Рис.11 – Apache Tez в среде Hadoop

Более высокая производительность алгоритмов Tez по сравнению с Hadoop MapReduce обеспечивается двумя факторами. Во-первых, динамическим распознаванием графа, отражающим тот факт, что обычно распределенные данные оказываются динамичными по своей природе и многое из того, что с ними нужно делать, определяется уже в процессе исполнения. Динамика находится в противоречии с пакетным режимом, где все следует определить заранее, а Tez включает в себя модули модификации вершин графа, позволяющие найти лучшее решение. Во-вторых, Tez использует оптимальное управление ресурсами. Эта функция реализуется им совместно с YARN, где есть элемент динамики — потребность в ресурсах меняется, и на разных фазах решения задачи они могут быть разными. Система Tez не предназначена для конечных пользователей и ориентирована на разработчиков приложений, поддерживая хранилище данных Apache Hive и аналитическую платформу с языком высокого уровня Apache Pig.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был проведен анализ самых распространенных технологий в области больших данных для обработки слабоструктурированных и неструктурированных данных. Результаты, полученные в ходе анализа, позволяют легко сделать выбор в пользу комбинации из слабоструктурированного формата данных и одной, наиболее оптимальной в рассматриваемом случае, технологии - Apache Spark, как части Hadoop-кластера. Кроме технических достоинств немаловажны и социально-коммерческие. Поэтому стоит обратить внимание и на то, что проект Apache Spark привлекает к себе огромное внимание, про него написано большое количество маленьких практических статей, он стал частью Hadoop 2.0. Плюс он быстро оброс дополнительными фреймворками, такими, как Spark Streaming, SparkML, Spark SQL, GraphX, а кроме этих «официальных» фреймворков появилось множество проектов — различные коннекторы, алгоритмы, библиотеки и так далее. Что говорит о его заслуженной оценённости в мире IT и соответствии ожиданиям в контексте рассматриваемого проекта. Так же было обнаружено что для полноты анализа необходима вторичная обработка данных, полученных после обработки на Hadoop-кластере, с помощью специализированных СУБД как классических реляционных, так и NOSQL решений. В зависимости от задачи и самих данных выбор такой СУБД будет меняться.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Tom White. Hadoop: The Definitive Guide: Storage and Analysis at internet Scale. –: "O'Reilly Media Inc. ", 2015. – 756 с.
2. Чак Лэм. Hadoop в действии. –: "Litres", 2017.
3. [Srinath Perera.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Srinath+Perera%22&source=gbs_metadata_r&cad=7) Hadoop Mapreduce Cookbook. –: "Packt Publishing Ltd", 2013. – 300 c.
4. [VK Jain.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22VK+Jain%22&source=gbs_metadata_r&cad=5) Big Data and Hadoop. –: "Khanna Publishing", 2017. – 600 c.
5. [Jason Venner,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Jason+Venner%22&source=gbs_metadata_r&cad=6) [Sameer Wadkar,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Sameer+Wadkar%22&source=gbs_metadata_r&cad=6) [Madhu Siddalingaiah.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Madhu+Siddalingaiah%22&source=gbs_metadata_r&cad=6) Pro Apache Hadoop. –: "Apress", 2014. – 444 c.
6. [Arun Murthy,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Arun+Murthy%22&source=gbs_metadata_r&cad=7) [Vinod Vavilapalli,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Vinod+Vavilapalli%22&source=gbs_metadata_r&cad=7) [Douglas Eadline,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Douglas+Eadline%22&source=gbs_metadata_r&cad=7) [Joseph Niemiec,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Joseph+Niemiec%22&source=gbs_metadata_r&cad=7) [Jeff](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Jeff+Markham%22&source=gbs_metadata_r&cad=7) [Markham.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Jeff+Markham%22&source=gbs_metadata_r&cad=7) Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2. –: "Addison–Wesley Professional", 2014. – 400 c.
7. [Dirk deRoos.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Dirk+deRoos%22&source=gbs_metadata_r&cad=7) Hadoop For Dummies. –: "John Wiley & Sons", 2014. – 408 c.
8. [Vignesh Prajapati.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Vignesh+Prajapati%22&source=gbs_metadata_r&cad=7) Big Data Analytics with R and Hadoop. –: "Packt Publishing Ltd", 2013. – 238 c.
9. [Boris Lublinsky,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Boris+Lublinsky%22&source=gbs_metadata_r&cad=7) [Kevin T. Smith,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Kevin+T.+Smith%22&source=gbs_metadata_r&cad=7) [Alexey Yakubovich.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Alexey+Yakubovich%22&source=gbs_metadata_r&cad=7) Professional Hadoop Solutions. –: "John Wiley & Sons", 2013. – 504 c.
10. [Kevin Sitto,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Kevin+Sitto%22&source=gbs_metadata_r&cad=7) [Marshall Presser.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Marshall+Presser%22&source=gbs_metadata_r&cad=7) Field Guide to Hadoop: An Introduction to Hadoop, Its Ecosystem, and Aligned Technologies. –: "O'Reilly Media, Inc.", 2015. – 132 c.
11. [Garry Turkington.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Garry+Turkington%22&source=gbs_metadata_r&cad=6) Hadoop Beginner's Guide. –: "Packt Publishing Ltd", 2013.-398 c.
12. [Danil Zburivsky.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Danil+Zburivsky%22&source=gbs_metadata_r&cad=7) Hadoop Cluster Deployment. –: "Packt Publishing Ltd", 2013. – 126 c.

1. [Kevin Roebuck.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Kevin+Roebuck%22&source=gbs_metadata_r&cad=2) MapReduce: High–impact Strategies – What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. –: "Lightning Source", 2011. – 170 c.
2. [Donald Miner,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Donald+Miner%22&source=gbs_metadata_r&cad=7) [Adam Shook.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Adam+Shook%22&source=gbs_metadata_r&cad=7) MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems. "O'Reilly Media, Inc.", 2012. – 250 c.
3. [Thilina Gunarathne.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Thilina+Gunarathne%22&source=gbs_metadata_r&cad=7) Hadoop MapReduce v2 Cookbook – Second Edition. –: "Packt Publishing Ltd", 2015. – 322 c.
4. [Билл Фрэнкс.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22%D0%91%D0%B8%D0%BB%D0%BB+%D0%A4%D1%80%D1%8D%D0%BD%D0%BA%D1%81%22&source=gbs_metadata_r&cad=6) Укрощение больших данных: Как извлекать знания из массивов информации с помощью глубокой аналитики. –: "Манн, Иванов Фербер", 2014.
5. [Виктор Майер–Шенбергер](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22%D0%92%D0%B8%D0%BA%D1%82%D0%BE%D1%80+%D0%9C%D0%B0%D0%B9%D0%B5%D1%80-%D0%A8%D0%B5%D0%BD%D0%B1%D0%B5%D1%80%D0%B3%D0%B5%D1%80%22&source=gbs_metadata_r&cad=7), [Кеннет Кукьер](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22%D0%9A%D0%B5%D0%BD%D0%BD%D0%B5%D1%82+%D0%9A%D1%83%D0%BA%D1%8C%D0%B5%D1%80%22&source=gbs_metadata_r&cad=7). Большие данные: Революция,которая изменит то, как мы живем, работаем и мыслим. –: "Манн, Иванов, Фербер", 2013. – 240 c.
6. [Peter Bühlmann,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Peter+B%C3%BChlmann%22&source=gbs_metadata_r&cad=7) [Petros Drineas,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Petros+Drineas%22&source=gbs_metadata_r&cad=7) [Michael Kane,](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Michael+Kane%22&source=gbs_metadata_r&cad=7) [Mark van der Laan.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Mark+van+der+Laan%22&source=gbs_metadata_r&cad=7) Handbook of Big Data. –: "CRC Press", 2016. – 464 c.
7. [Frank J. Ohlhorst.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Frank+J.+Ohlhorst%22&source=gbs_metadata_r&cad=7) Big Data Analytics: Turning Big Data into Big Money. –: "John Wiley & Sons", 2012. – 176 c.
8. [Y. Lakshmi Prasad.](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Y.+Lakshmi+Prasad%22&source=gbs_metadata_r&cad=7) Big Data Analytics Made Easy. –: "Notion Press", 2016. -192 c.