Промышленное машинное обучение на Spark

Лекция 1: Docker

21 октября 2023 г.

Содержание курса

План лекции

Что такое большие данные и откуда они берутся?

- Сферы производящие большие данные. Data explosion.
- Большие данные - где начало. 4 основных принципа.

Как хранить большие данные?

- Устройство файловой системы Linux.
- Отказоустойчивость.
- HDFS. Ее устройство и основные свойства.

Как обрабатывать большие данные?

- Предпосылки к созданию MapReduce.
- Компоненты системы
- Задача подсчета слов. Map. Shuffle. Reduce.

Какие есть технологии для работы с большими данными?

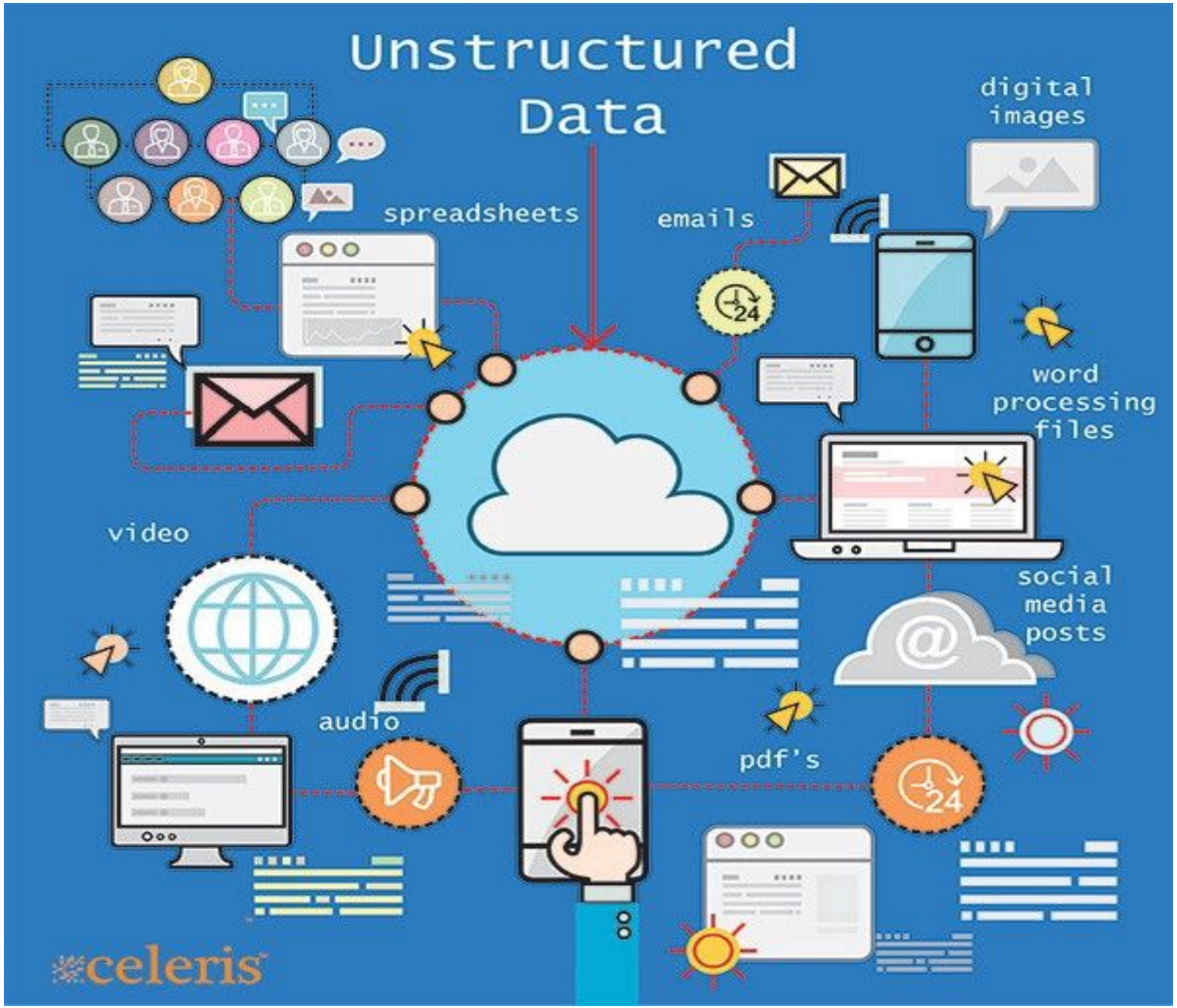
- В чём преимущество технологии Spark.
- Основные компоненты системы, RDD. DAG.
- Spark DataFrame. Поддерживаемые методы.

Сферы производящие большие данные.
Data explosion.

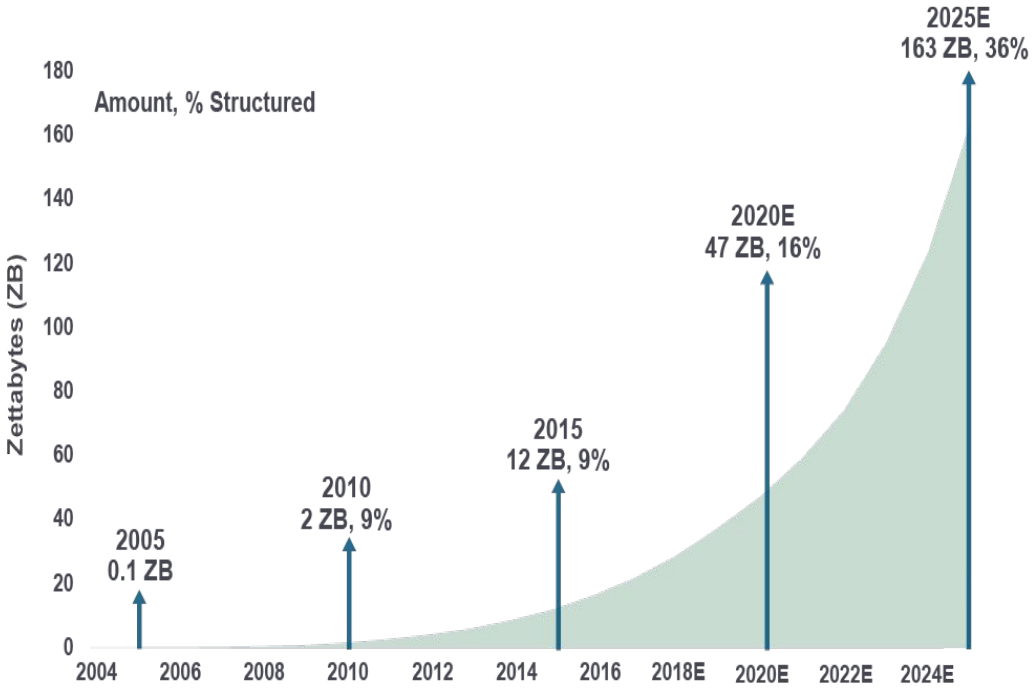
Откуда пришла Big Data

Сферы:

- Телеком
- Банки
- Социальные сети
- Медиа
- Промышленность
- Биоинформатика
- Интернет вещей



Взрыв данных



Чему равен зеттабайт? - триллиону гигабайт

1 kilobyte	**1 000**
1 megabyte	**1 000 000**
1 gigabyte	**1 000 000 000**
1 terabyte	**1 000 000 000 000**
1 petabyte	**1 000 000 000 000 000**
1 exabyte	**1 000 000 000 000 000 000**
1 zettabyte	**1 000 000 000 000 000 000 000**

Количество обрабатываемых данных

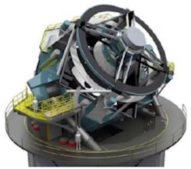
Обрабатывает 40 000PB в день

Содержит 300Pb пользовательских данных

Хранит 160Zb

Wayback Machine: 240B веб-страниц в архиве, 5 PB (1/2013)

LHC: ~15 PB в год



LSST: 6-10 PB в год (~2015)

Большие данные, где начало. 4 основных принципа.

4’V of big data

Veracity

Volume

- Петабайты
- Записи
- Транзакции
- Таблицы, файлы

Velocity

- Твиты
- Посты в facebook
- Датчики устройств

- Скорость генерации данных
- Генерируемые в реал-тайм
- Онлайн и офлайн данные
- По стримам, батчам или битам

Variety

- Видео
- Аудио
- Текстовые данные
- Временные ряды

- Структурированные
- Неструктурированные
- Полуструктурированные

Как компании справляются с большими данными?

Первый ключевой вопрос: как хранить большие данные?

Файловая система

Основными функциями файловой системы являются:

- размещение и упорядочивание на носителе данных в виде файлов;
- определение максимально поддерживаемого объема данных на носителе информации;
- создание, чтение и удаление файлов;
- назначение и изменение атрибутов файлов (размер, время создания и изменения, владелец и создатель файла, доступен только для чтения, скрытый файл, временный файл, архивный, исполняемый, максимальная длина имени файла и т. п.);
- определение структуры файла;
- поиск файлов;
- организация каталогов для логической организации файлов;
- защита файлов при системном сбое;
- защита файлов от несанкционированного доступа и изменения их содержимого

Linux File System Directories

/bin: Where Linux core commands reside like ls, mv.

/boot: Where boot loader and boot files are located.

/dev: Where all physical drives are mounted like USBs DVDs.

/etc: Contains configurations for the installed packages.

/home: Where every user will have a personal folder to put his folders with his name like /home/likegeeks.

/lib: Where the libraries of the installed packages located since libraries shared among all packages, unlike Windows, you may find duplicates in different folders.

/media: Here are the external devices like DVDs and USB sticks that are mounted, and you can access their files from here.

/mnt: Where you mount other things Network locations and some distros, you may find your mounted USB or DVD.

/opt: Some optional packages are located here and managed by the package manager.

/proc: Because everything on Linux is a file, this folder for processes running on the system, and you can access them and see much info about the current processes.

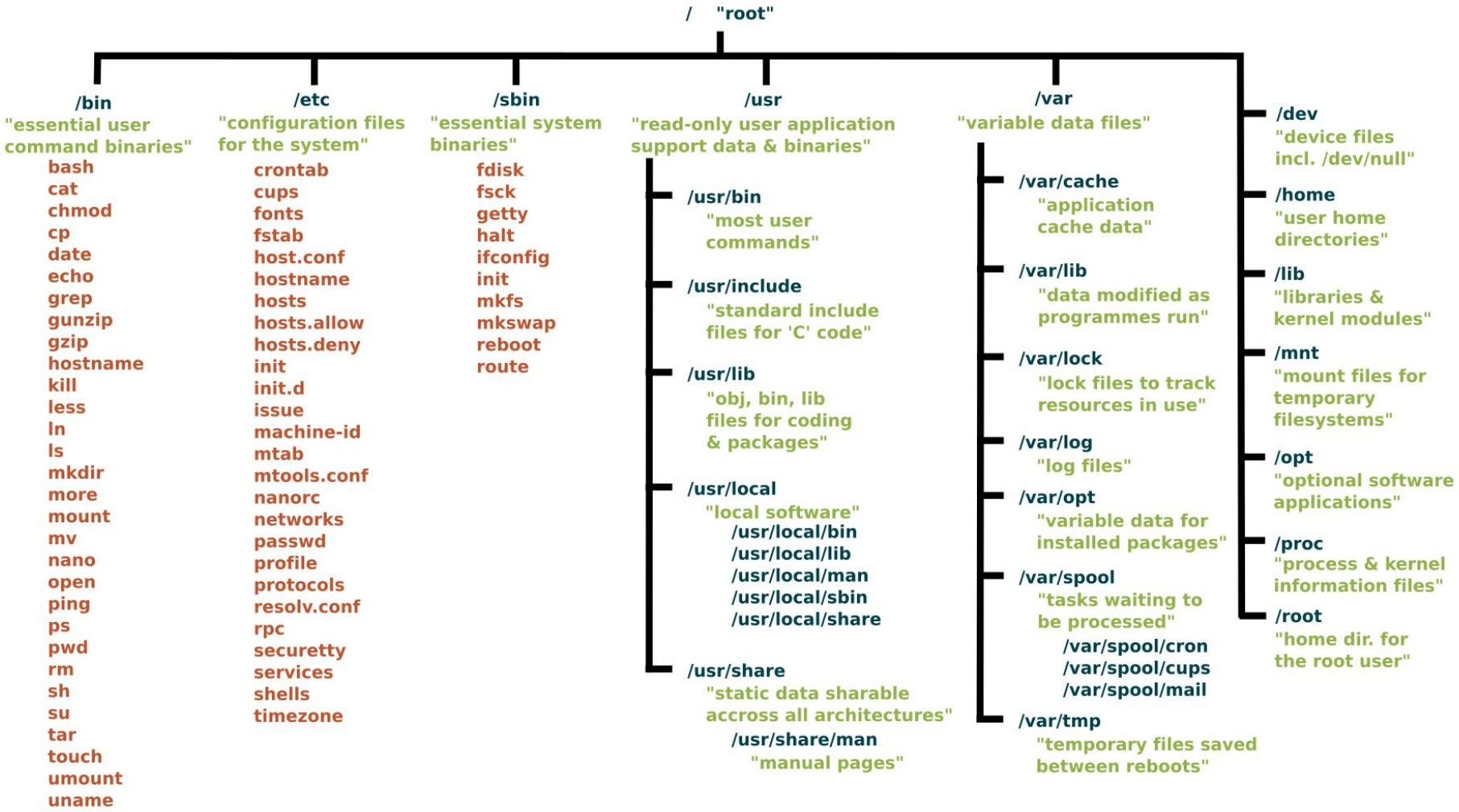
/root: The home folder for the root user.

/sbin: Like /bin, but binaries here are for root user only.

/tmp: Contains the temporary files.

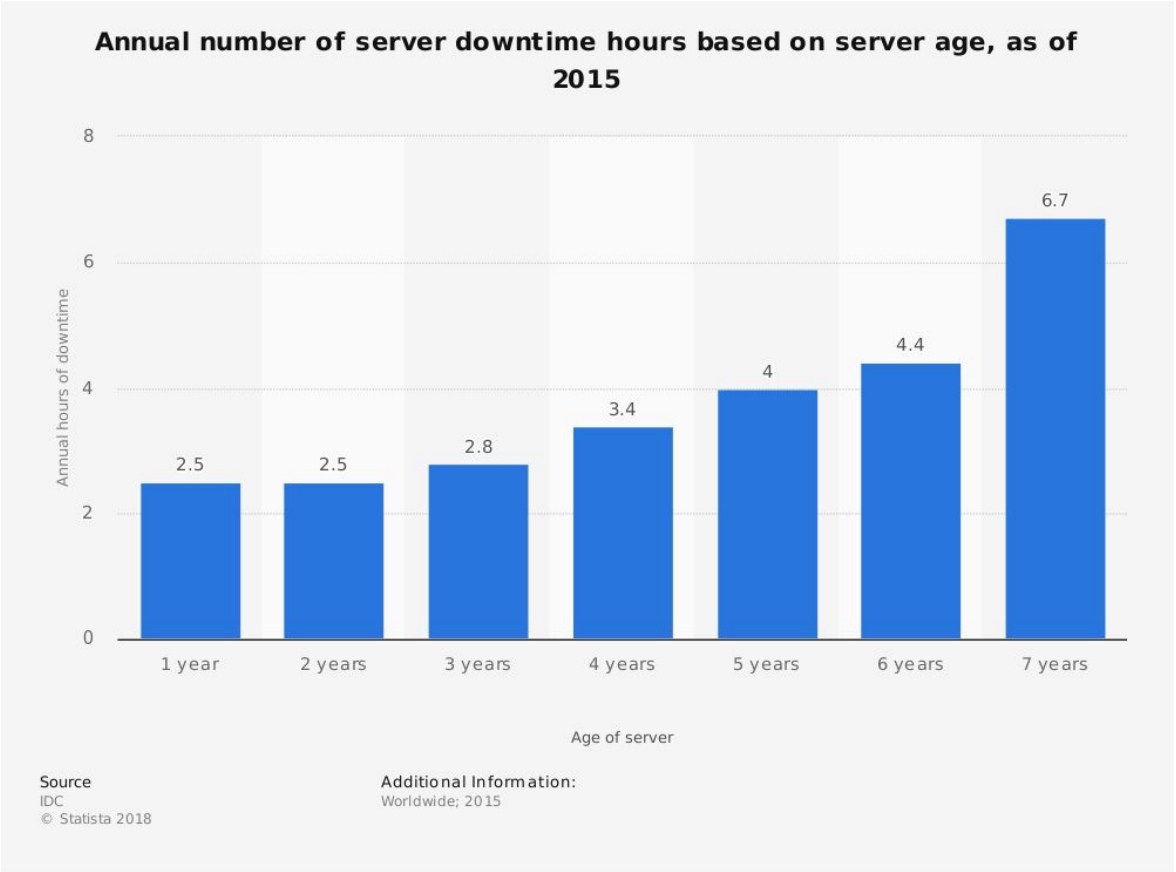
/usr: Where the utilities and files shared between users on Linux.

/var: Contains system logs and other variable data.


/ "root"
/bin
"essential user command binaries"
bash
cat
chmod
cp
date
echo
grep
gunzip
gzip
hostname
kill
less
ln
ls
mkdir
more
mount
mv
nano
open
ping
ps
pwd
rm
sh
su
tar
touch
umount
uname
/etc
"configuration files for the system"
crontab
cups
fonts
fstab
host.conf
hostname
hosts
hosts.allow
hosts.deny
init
init.d
issue
machine-id
mtab
mtools.conf
nanorc
networks
passwd
profile
protocols
resolv.conf
rpc
securetty
services
shells
timezone
/sbin
"essential system binaries"
fdisk
fsck
getty
halt
ifconfig
init
mkfs
mkswap
reboot
route
/usr
"read-only user application support data & binaries"
/usr/bin
"most user commands"
/usr/include
"standard include files for 'C' code"
/usr/lib
"obj, bin, lib files for coding & packages"
/usr/local
"local software"
/usr/local/bin
/usr/local/lib
/usr/local/man
/usr/local/sbin
/usr/local/share
/usr/share
"static data sharable accross all architectures"
/usr/share/man
"manual pages"
/var
"variable data files"
/var/cache
"application cache data"
/var/lib
"data modified as programmes run"
/var/lock
"lock files to track resources in use"
/var/log
"log files"
/var/opt
"variable data for installed packages"
/var/spool
"tasks waiting to be processed"
/var/spool/cron
/var/spool/cups
/var/spool/mail
/var/tmp
"temporary files saved between reboots"
/dev
"device files incl. /dev/null"
/home
"user home directories"
/lib
"libraries & kernel modules"
/mnt
"mount files for temporary filesystems"
/opt
"optional software applications"
/proc
"process & kernel information files"
/root
"home dir. for the root user"

Отказоустойчивость хранения данных и вычислений.

Насколько надёжно хранить данные на сервере



Вероятность, что в следующий час случится поломка:

$$P = 2.5 / (24 * 365) = 0.00028$$

$$P(\text{не выйдет из строя}) = (1 - P) = 0.9997$$

Если в кластере 1000 машин, вероятность, что один из серверов будет недоступен в ближайший час:

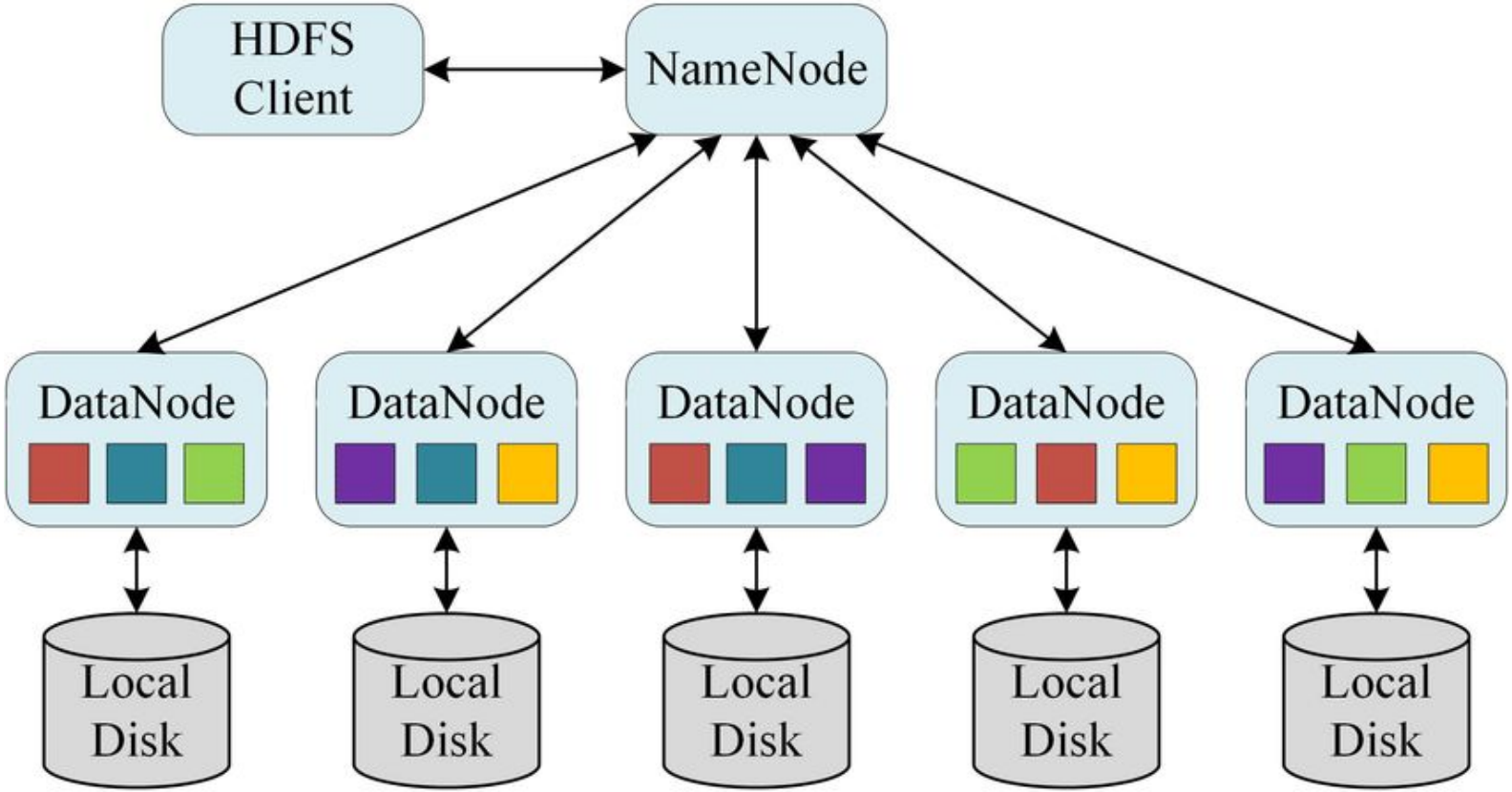
$$1 - 0.9997^{1000} = 0.25$$

HDFS - Hadoop Distributed File System

Основные свойства:

- Доступность данных, за счёт механизмов отказоустойчивость
- Приложения, работающие в HDFS, имеют большие наборы данных. Типичный файл в HDFS имеет размер от гигабайтов до терабайт.
- Эффективно работает в парадигме Write once Read many
- Не требует дорогостоящего оборудования и может быть запущена на маломощных машинах

Как устроена HDFS

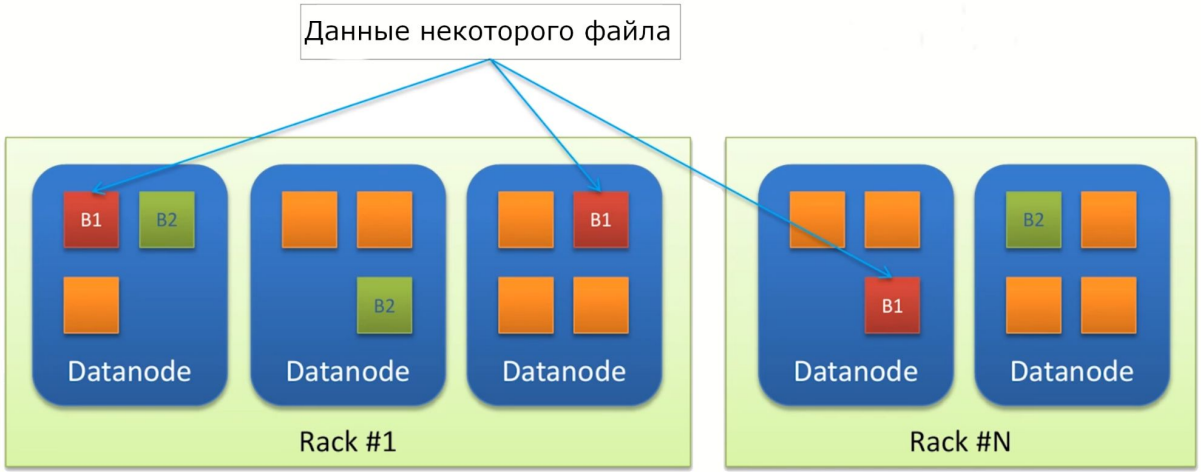


Из чего состоит HDFS

- ***Кластер -*** набор вычислительных узлов (серверов), соединённых сетью;
- ***HDFS client*** - аппаратный или программный компонент системы, откуда поступают команды на произведение некоторых действий в распределённом хранилище данных;
- ***NameNode*** - специально выделенных узел, на котором содержится мета-информация о том, на каком узле хранятся данные для того или иного файла
- ***DataNode*** - один из множество серверов, на котором непосредственно располагаются данные

Отказоустойчивость в HDFS

Основной принцип, благодаря которому достигается надёжное хранение данных в HDFS заключается в том, что копии одних и тех же данных располагаются на нескольких отдельных друг от друга серверах.



Фактор репликации

Число N равное количеству копий одних и тех же данных в системе называется ***фактором репликации***

- При выборе N необходимо подбирать баланс между объёмом занимаемой памяти и уровнем надёжности. Чем больше фактор репликации тем на бóльшем количестве узлов будут содержаться копии данных, но при этом возрастает объём требуемой памяти для хранения

- Как правило на практике зачастую используют N = 3

Второй ключевой вопрос: как обрабатывать большие данные?

Модель вычислений Map-Reduce

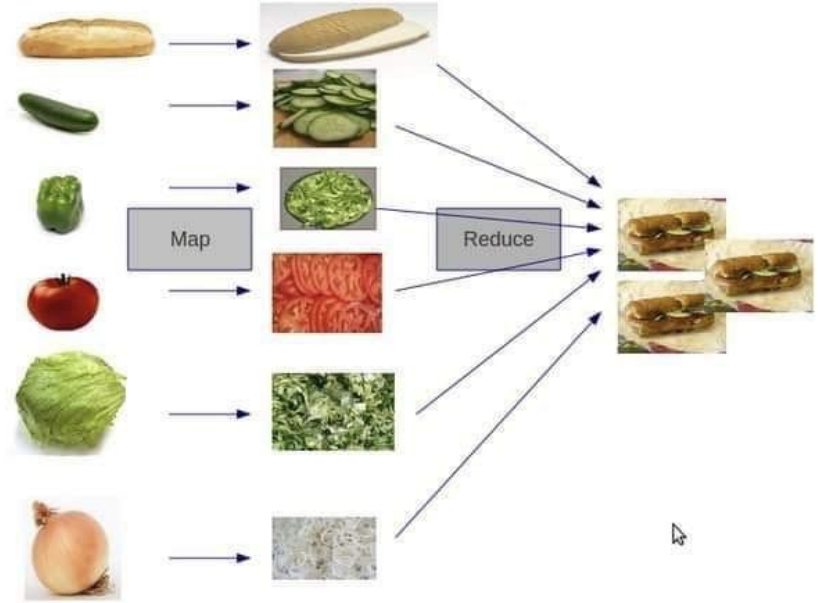
Для эффективной обработки больших объёмов данных применяется двухшаговая модель вычислений Map-Reduce.

- Шаг Map: на данном шаге к каждой порции данных применяется некоторая функция F.

$$map(F, [x_1, \ldots, x_N]) \rightarrow [F(x_1), \ldots, F(x_N)]$$

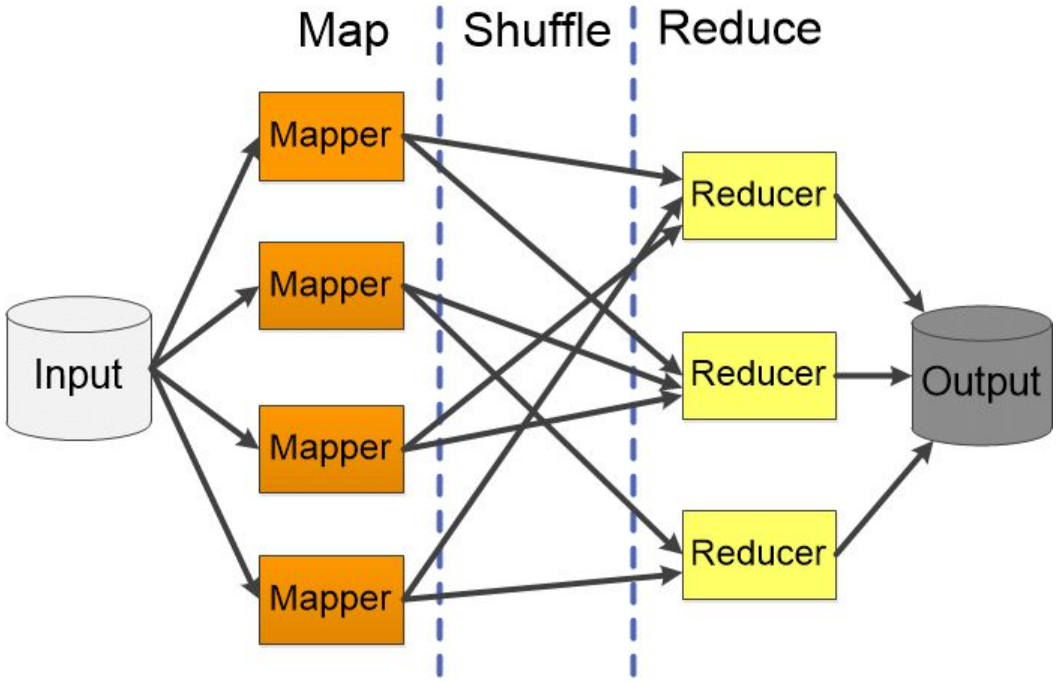
- Шаг Reduce: на данном шаге происходит объединение результатов из предыдущего шага Map и получение итогового результата

$$Reduce([F(x_1), \ldots, F(x_N)]) \rightarrow Y$$



Архитектура Map-Reduce

- Вычислительные узел, на котором происходит вычисление шага Map называются Mapper
- А вычислительный узел, на которым происходит шаг Reduce, называется Reducer



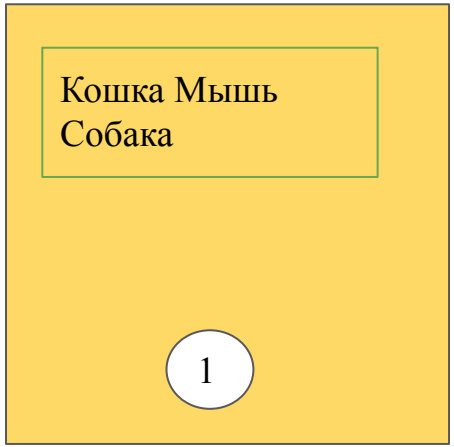
Вычислительный кластер

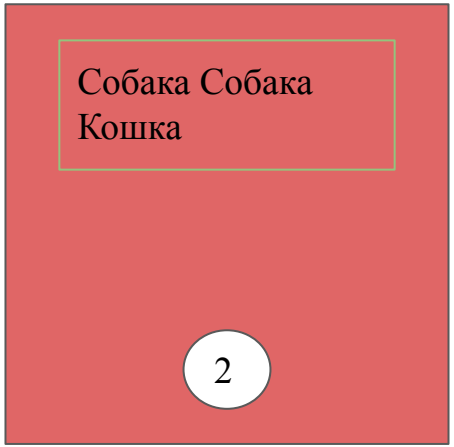
Задача подсчета слов

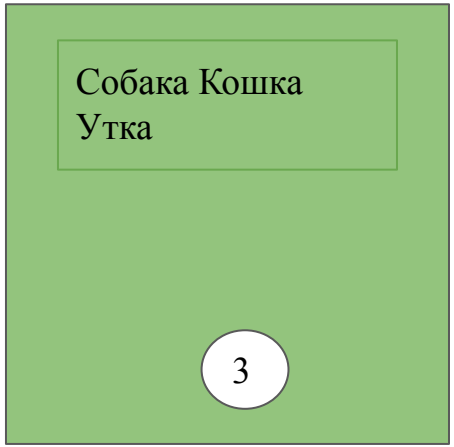
Кошка Мышь Собака

Собака Собака Кошка

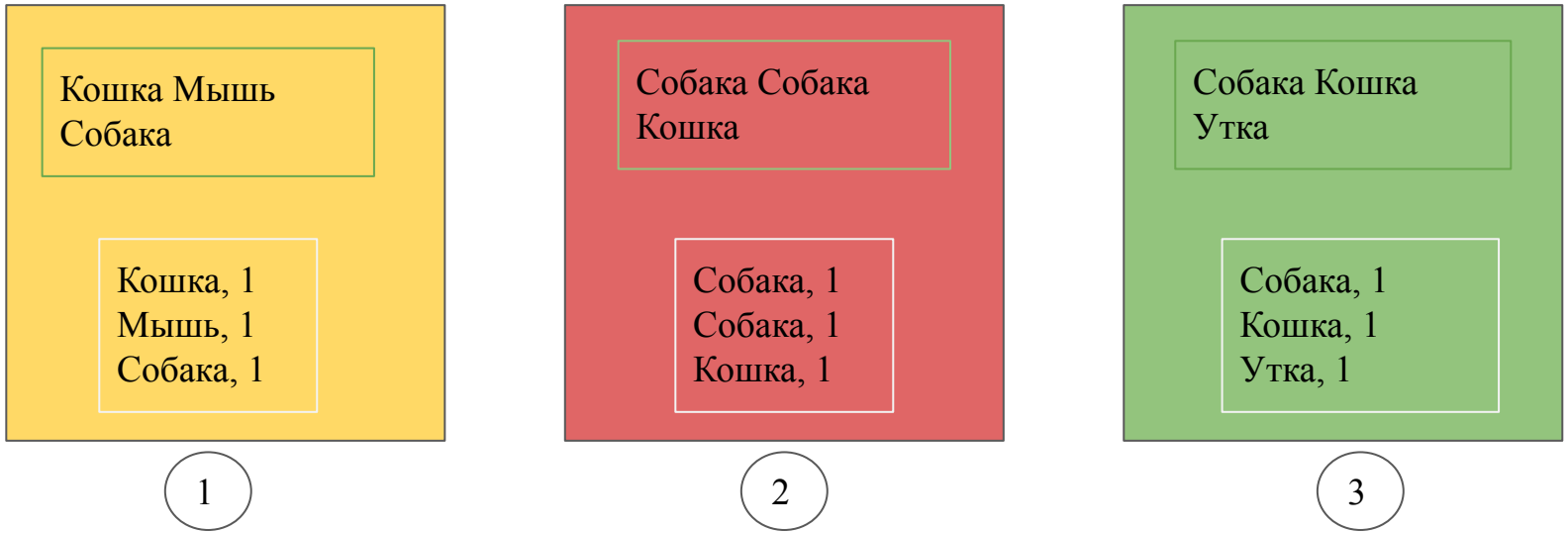
Собака Кошка Утка





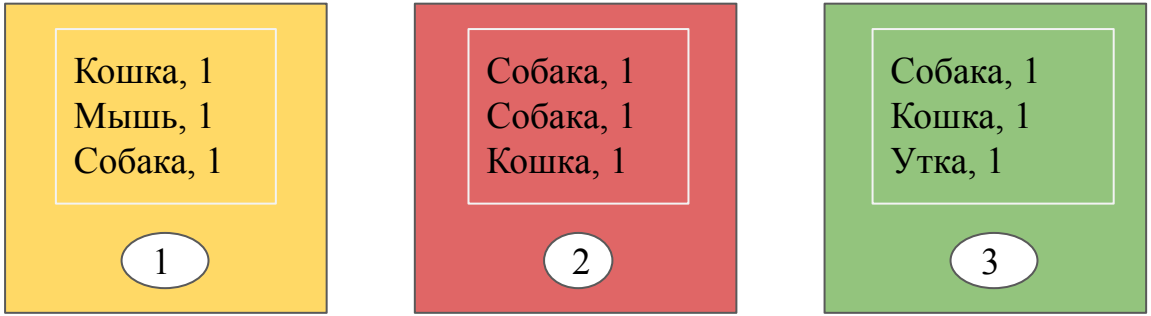


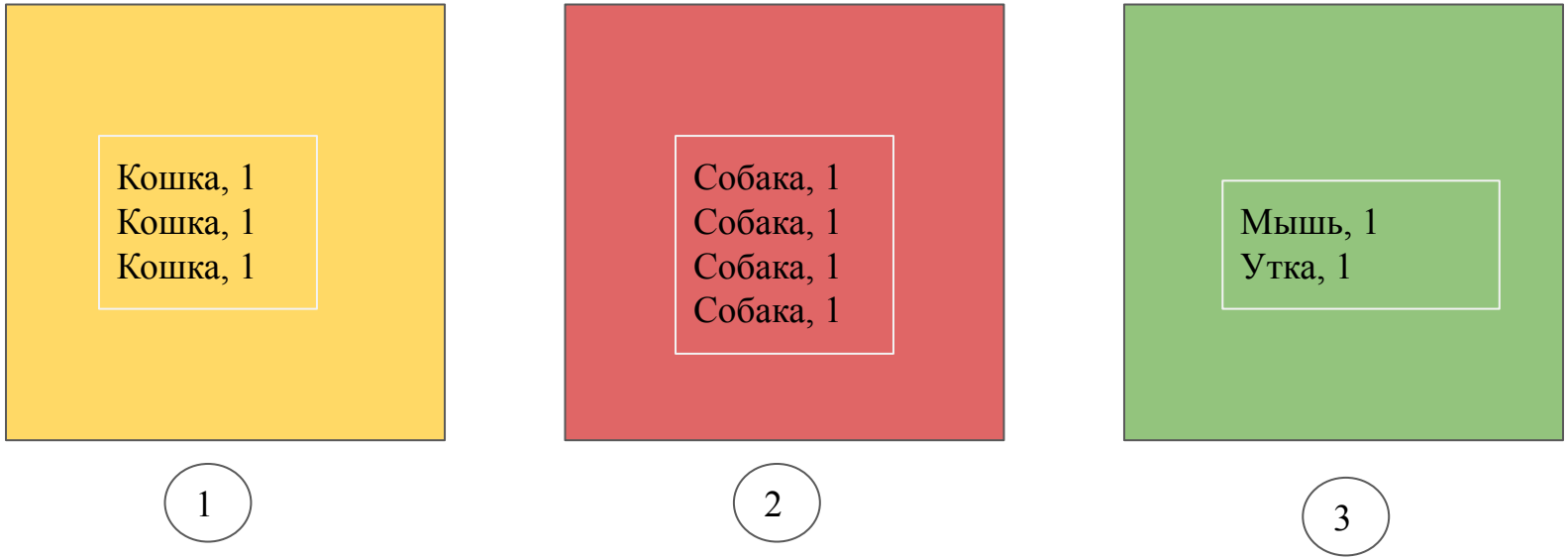
Шаг Map


Кошка Мышь
Собака
Кошка, 1
Мышь, 1
Собака, 1
1
Собака Собака
Кошка
Собака, 1
Собака, 1
Кошка, 1
2
Собака Кошка
Утка
Собака, 1
Кошка, 1
Утка, 1
3

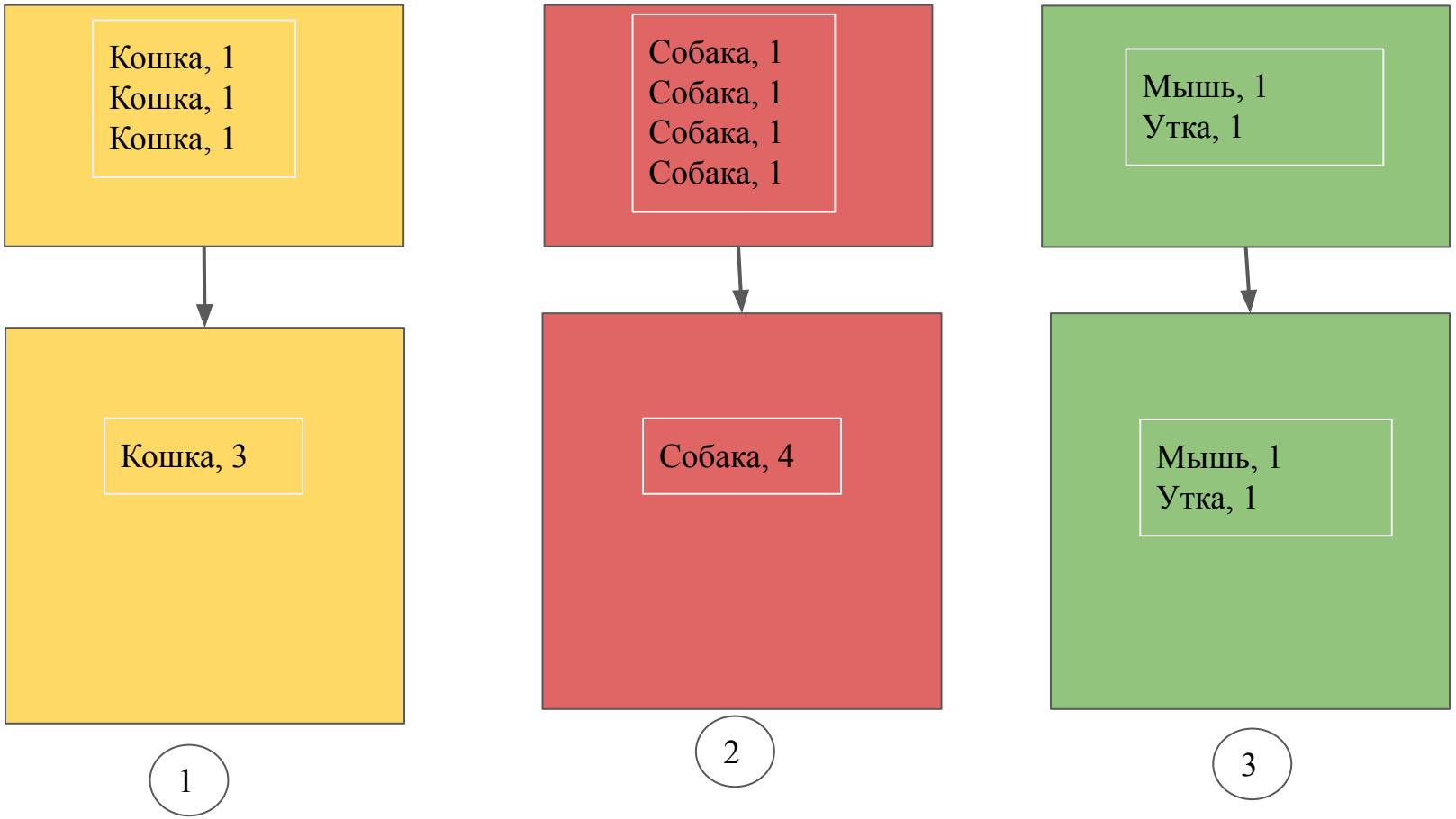
Шаг Shuffle

Цель данной процедуры отправить правильным образом данные на каждый Reducer





Шаг Reduce


Кошка, 1
Кошка, 1
Кошка, 1
Кошка, 3
1
Собака, 1
Собака, 1
Собака, 1
Собака, 1
Собака, 4
2
Мышь, 1
Утка, 1
Мышь, 1
Утка, 1
3

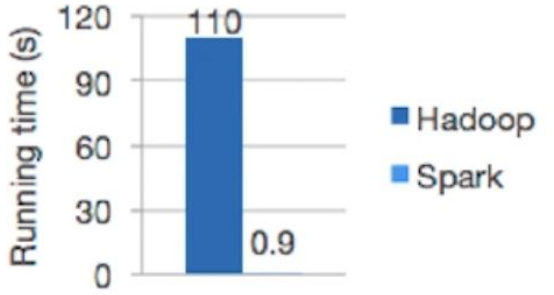
Как со всем этим работают на практике?

Инструменты для работы с большими данными

Запускать вычисления на распределённом вычислительном кластере крайне непростая задача. Необходимо решить множество аспектов, связанных с как с настройкой самого кластера так и с написанием корректных и эффективных программ на нём.

Для упрощения работы с большими данными была разработана технология Spark, позволяющая разработчикам:

1. Абстрагироваться от технических деталей
2. Писать более эффективные программы за счёт оптимизации вычислений



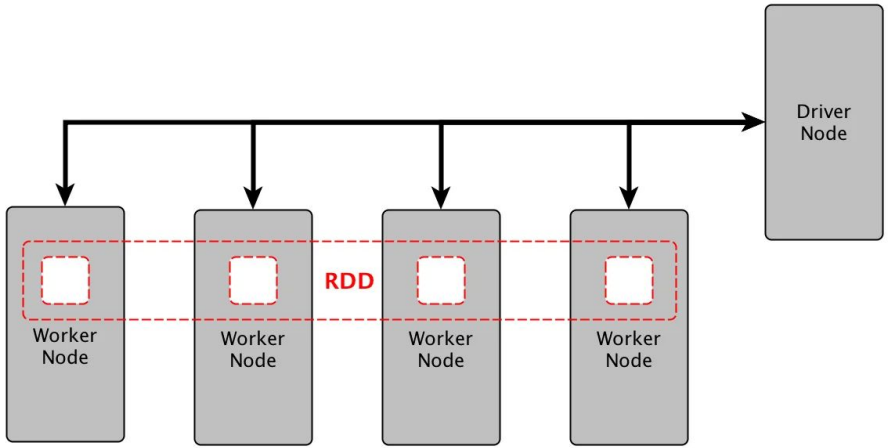
Logistic regression in Hadoop and Spark

Основные преимущества Spark

1. Высокая скорость работы
2. Удобные абстракции над данными
3. Поддержка нескольких языков программирования: Java, Python, Scala
4. Поддержка разных источников данных
    3.1. Файловые системы: HDFS, Linux File System
    3.2. Базы данных: HBase, Cassandra, S3
5. Является Open-Source проектом
6. Имеет большое сообщество специалистов вокруг себя

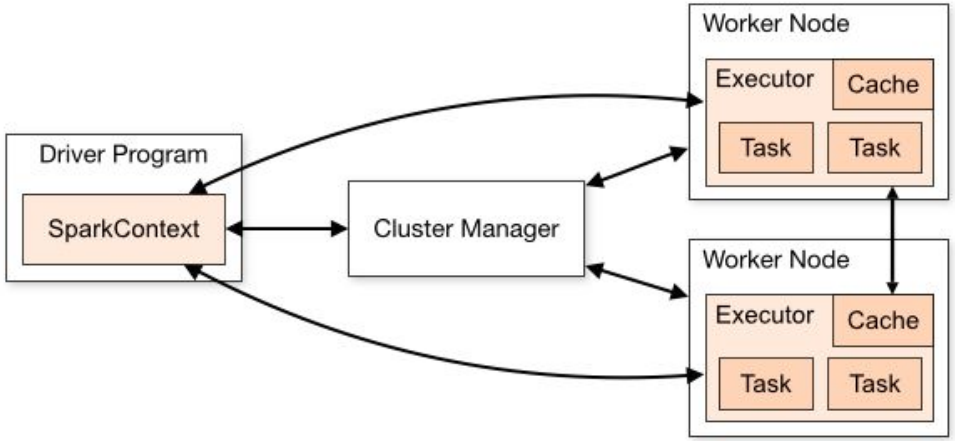
Основные компоненты. RDD.

RDD (Resilient Distributed Dataset) — это фундаментальная структура данных Spark, которая представляет собой неизменяемый набор данных, который вычисляются и располагается на разных узлах кластера. Каждый набор данных в Spark RDD логически разделен на множество серверов, чтобы их можно было вычислить на разных узлах кластера.



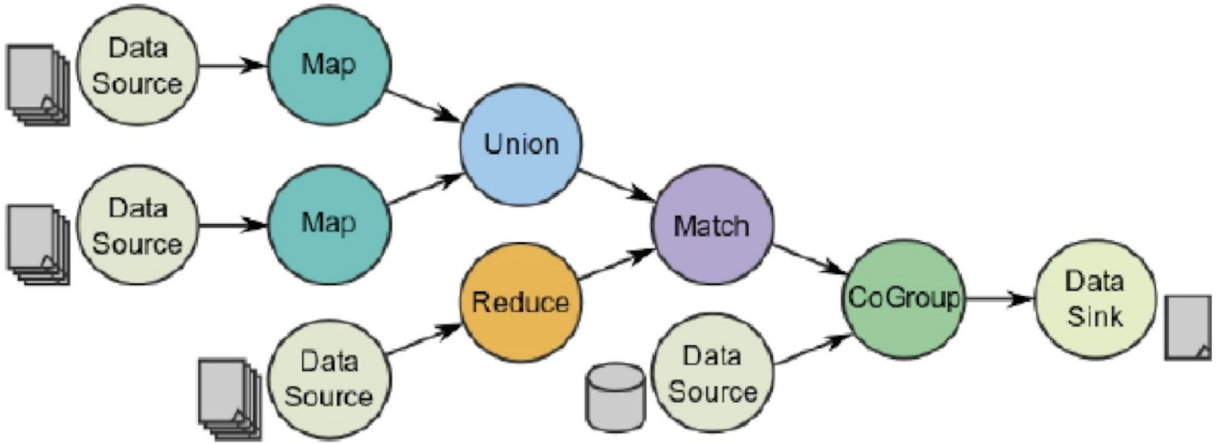
Основные компоненты. Spark Context

SparkContext — это точка входа для всех операций Spark и средство, с помощью которого приложение подключается к ресурсам кластера Spark. Через этот специальный объект Spark происходит конфигурирование ресурсов кластера, обращение к данным и постановка различных задач на исполнение.



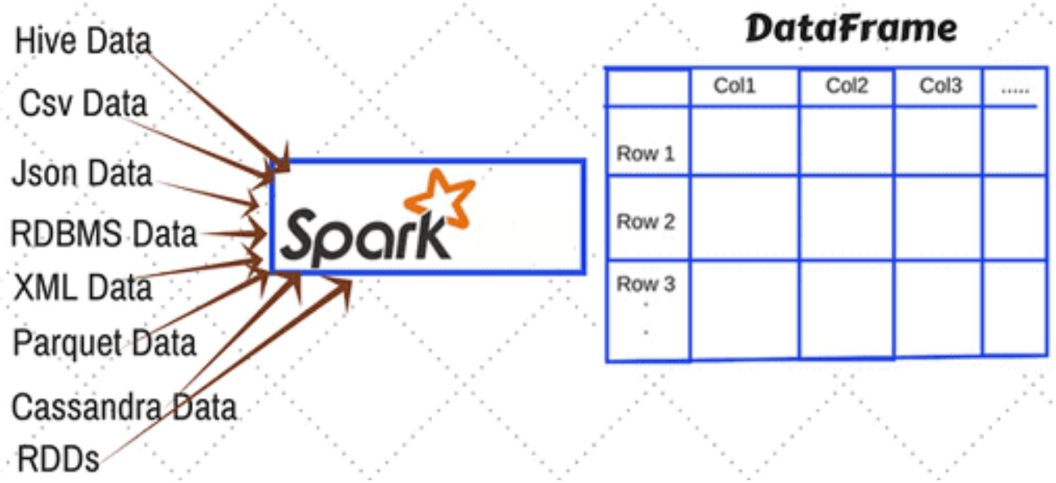
Основные компоненты. DAG.

DAG (Направленный ациклический граф) — это набор вершин и ребер, где вершины представляют RDD, а ребра представляют операцию, которая будет применяться к RDD. В Spark DAG каждое ребро направляет от более раннего к более позднему в последовательности.



Основные компоненты. Spark DataFrame.

Spark DataFrame — это набор данных, организованный в виде таблицы. Концептуально он эквивалентен таблице базе данных или фрейму данных в Python, но с более обширной внутренней оптимизацией. Spark DataFrames могут быть созданы из различных источников, таких как файлы структурированных данных, таблицы в Hive, внешние базы данных или существующие RDD.



Операции над данными

В Spark поддерживается два вида операторов, которые работают над RDD:

- Transformations - операторы, которые создают новые RDD на основе имеющихся, при этом самого исполнения операции не происходит

- Actions - операторы, которые запускают вычисления и возвращают результат работы в программу или сохраняют его на диск

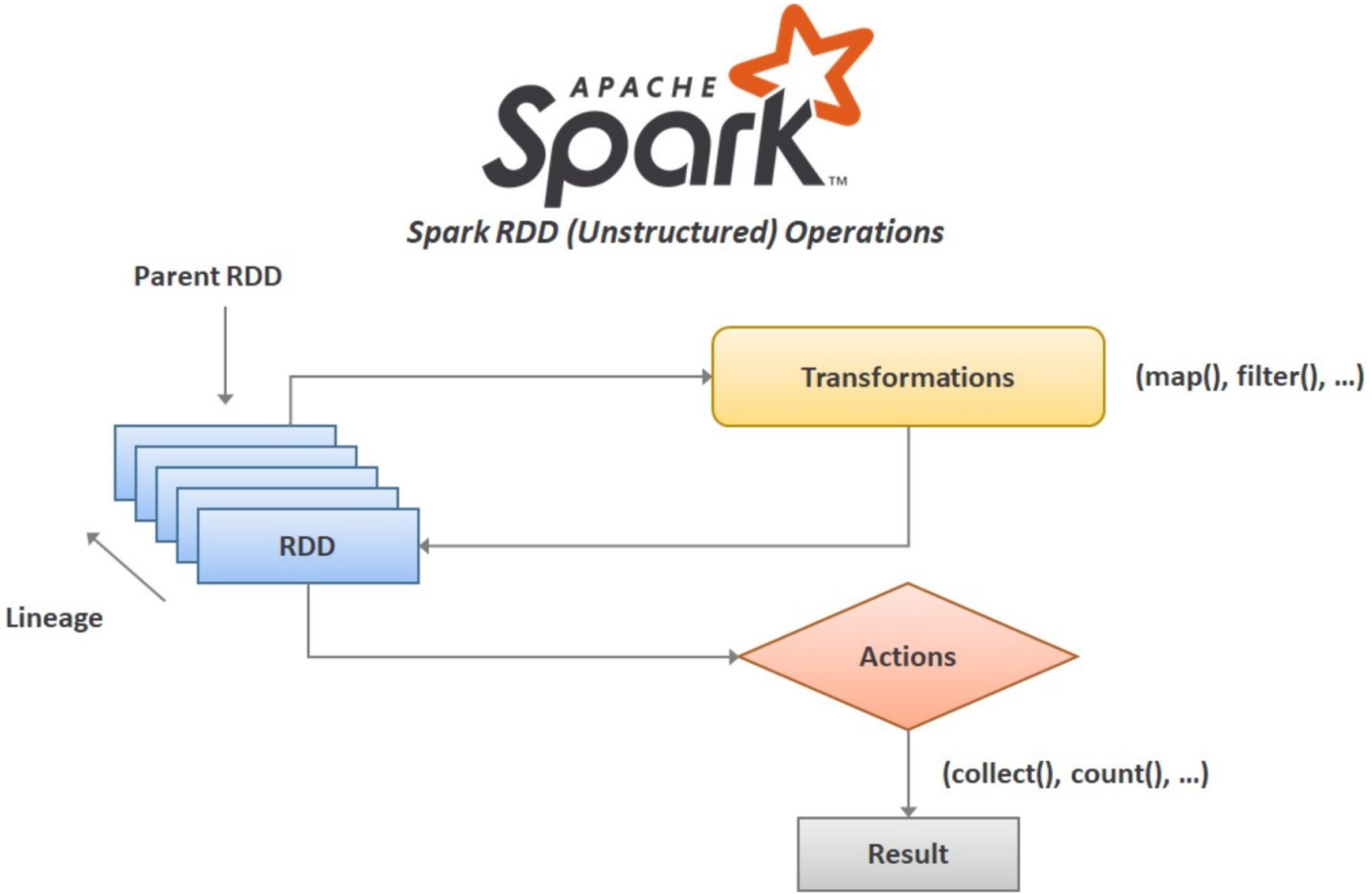
Transformations

map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
sortByKey([*ascending*], [*numPartitions*])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.
repartition(*numPartitions*)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
groupByKey([*numPartitions*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. **Note:** If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance. **Note:** By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numPartitions argument to set a different number of tasks.
reduceByKey(*func*, [*numPartitions*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.

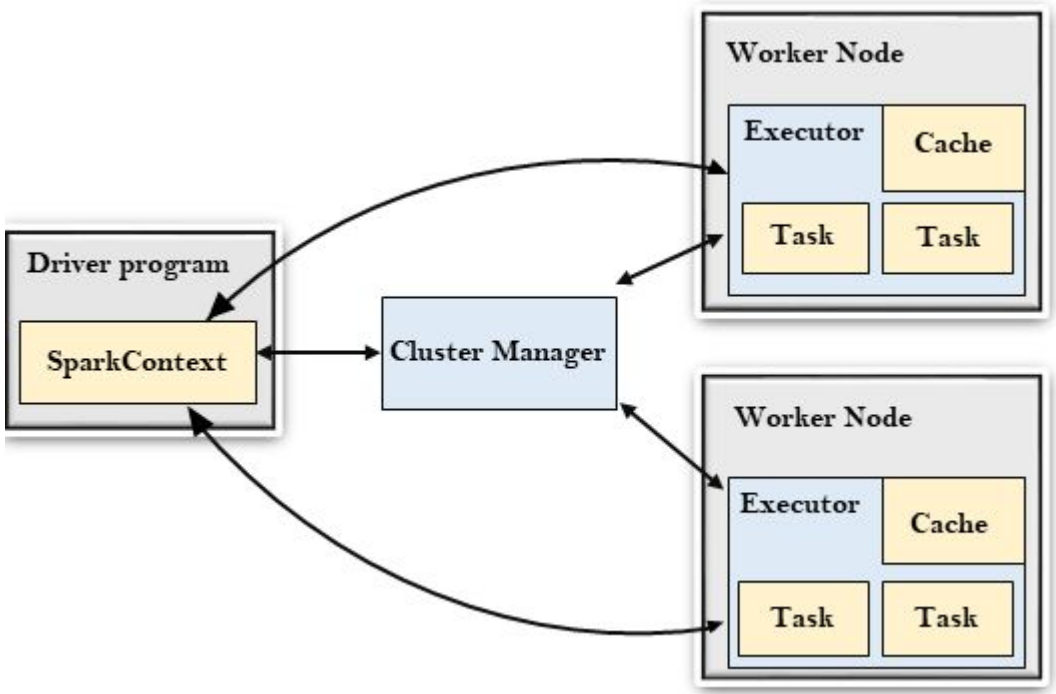
Actions

reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

Жизненный цикл RDD



Устройство Spark кластера



- ***Worker Node*** - компоненты системы Spark, которые выполняют фактическое действие над данными;
- ***Spark Driver*** - компонент, ответственный за планирование вычислений и подготовку задач для Worker'ов;
- ***Cluster Manager*** - компонент, который координирует обмен данными между узлами;

Data locality

В Spark реализованы механизмы, поддерживающие локальность данных. Под локальностью данных понимается то, что вычисления производятся на том узле, где располагаются необходимые данные. Это позволяет уменьшить нагрузку на сеть и увеличить производительность системы.

Достигается это через передачу Spark-кода на узлы, на которых располагаются данные, а не наоборот.