Работа с большими данными

Содержание курса

01 Введение. Распределенные вычисления. MapReduce.

02 HDFS. Apache Spark. RDD.

03 Базы данных. Spark SQL. Хранение больших данных.

04 Подробнее о модели вычислений Spark. Знакомство со Scala.

05 Алгоритмы машинного обучения на больших данных. spark.ml.

06 Рекомендательные системы. Виды. Их метрики.

07 Обработка потоковых данных. Structured streaming и интеграция с spark.ml.

08 Модели в продакшен. Управление кластером.

2. HDFS. Apache Spark. RDD.

План:

1. Устройство файловой системы Linux. HDFS. Ее устройство и основные свойства.
2. Отказоустойчивость вычислений.
3. Spark - основные абстракции и объекты.
4. Действия и преобразования.
5. Data locality. Как ее реализовать. Сериализация. Пример.

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.

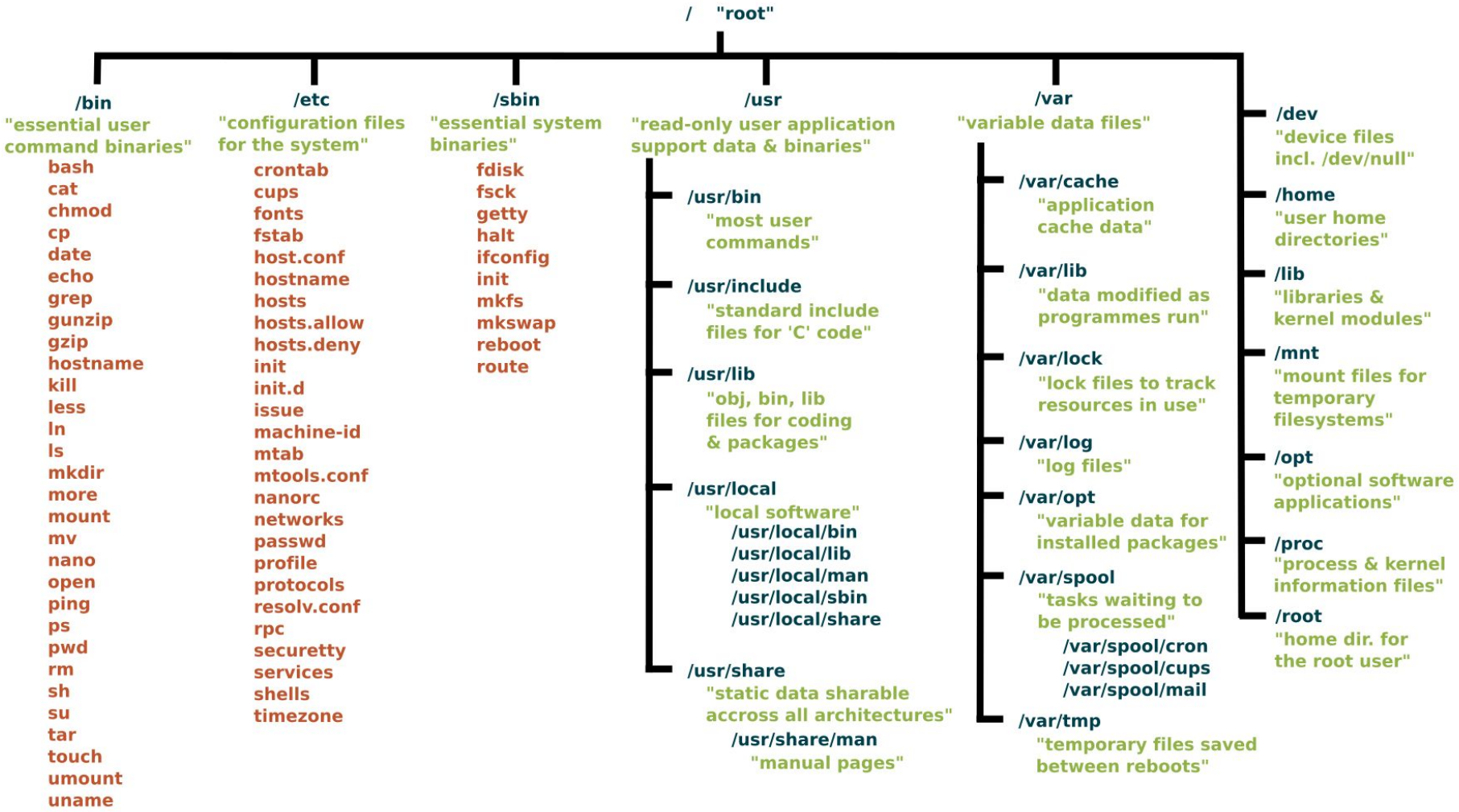
/root: The home folder for the root user.

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

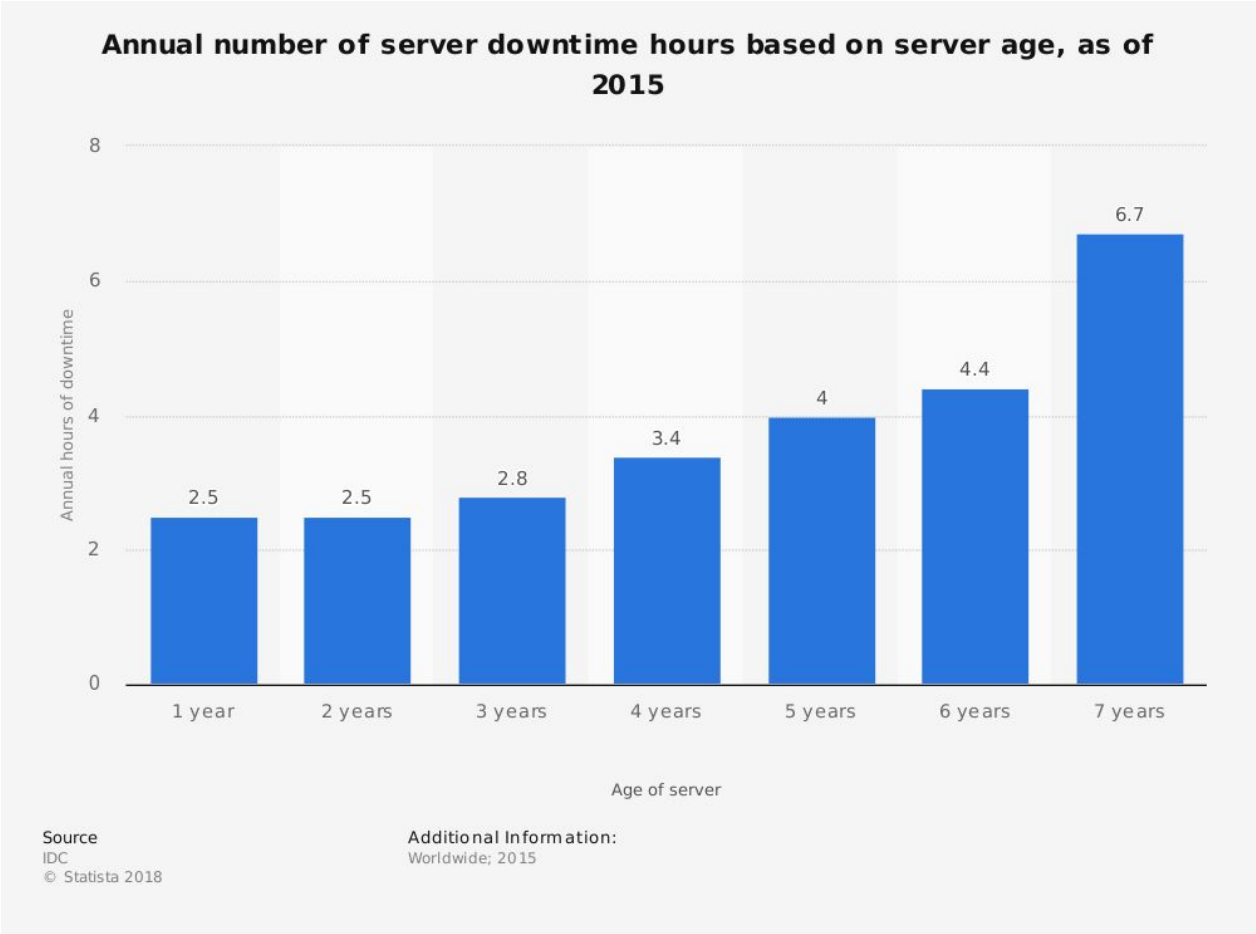
/tmp: Contains the temporary files.

/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 вычислений.



Вероятность, что в следующий час случится поломка

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

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

1000 машин в кластере

Вероятность, что какой-нибудь сломается ближайший час

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

HDFS. Ее устройство и основные свойства.

Полезные определения

Кластер - совокупность компьютеров объединенных сетью и выполняющих задачи, посылаемые клиентами.

Нода (Node) - один из компьютеров подключенных к кластеру.

Стойка (Rack) - совокупность нескольких нод, объединенных сетью

Демон - компьютерная программа в системах класса UNIX, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем

Клиент - это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

Цели:

- Отказ оборудования - это скорее норма, чем исключение.
- Приложения, работающие в HDFS, имеют большие наборы данных. Типичный файл в HDFS имеет размер от гигабайтов до терабайт.
- Write once Read many
- «Перемещение вычислений дешевле, чем перемещение данных» (Data locality)
- HDFS был разработан таким образом, чтобы его можно было легко переносить с одной платформы на другую.

Основные задачи HDFS

1. Отказоустойчивость в условиях частых сбоев и поломок
2. Параллельная обработка частей данных

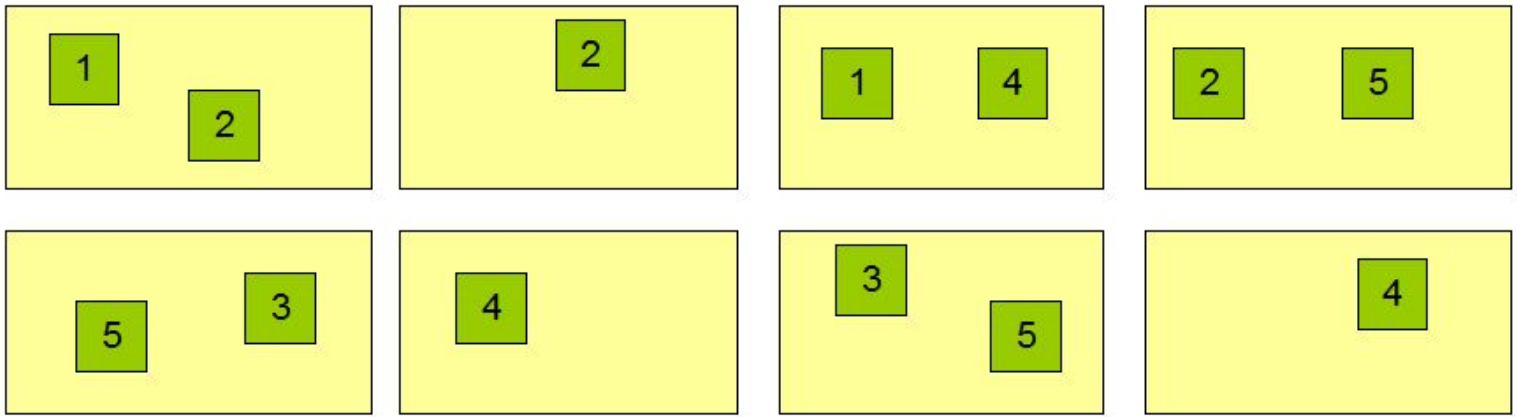
Block Replication

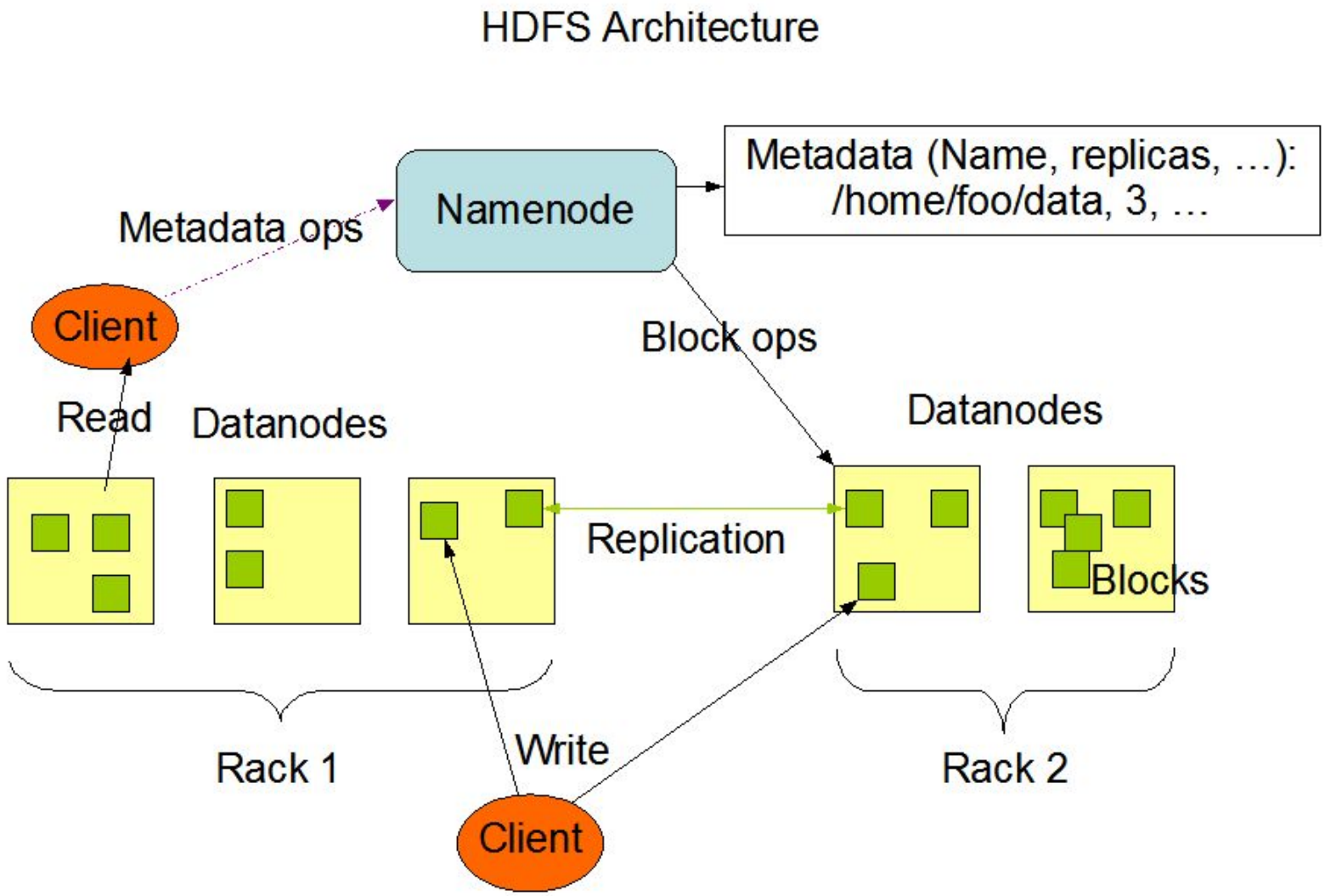
- **Управляющий узел, узел имен или сервер имен (NameNode)**

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Datanodes

- **Узел или сервер данных (DataNode, Node)**




HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Особенности:

- большой размер блока
- ориентация на недорогие и, поэтому не самые надежные сервера
- репликация на уровне кластера
- репликация происходит в асинхронном режиме
- клиенты могут считывать и писать файлы HDFS напрямую через программный интерфейс Java;
- файлы пишутся однократно,
- принцип WORM (Write-once and read-many)
- сжатие данных и рациональное использование дискового пространства
- самодиагностика
- все метаданные сервера имен хранятся в оперативной памяти.

Spark - основные абстракции и объекты.

Основные концепции

- Resilient distributed datasets (RDD)
- Directed acyclic graphs (DAG)
- SparkContext (sc)
- Spark DataFrames
- Actions and transformations
- Spark deployment options

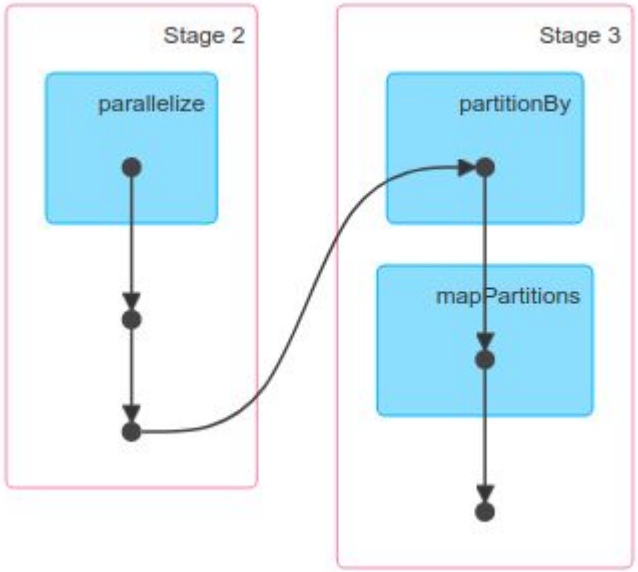
Resilient distributed datasets (RDD).

RDD (Resilient Distributed Dataset) - это фундаментальная структура данных Apache Spark, которая представляет собой неизменяемую коллекцию объектов, которые вычисляются на разных узлах кластера. Каждый набор данных в Spark RDD логически разделен на множество серверов, чтобы их можно было вычислить на разных узлах кластера.

Directed Acyclic Graph (DAG).

Пример с прошлой лекции подсчет слов

```
wordCountsCollected = (sparkdata
 .map(mapword)
 .reduceByKey(lambda a,b: a+b)
 .collect())
```



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. При вызове действия созданный DAG отправляется планировщику DAG, который далее разбивает граф на этапы и задачи.

SparkContext (sc).

SparkContext - это точка входа для всех операций Spark и средство, с помощью которого приложение подключается к ресурсам кластера Spark. Он инициализирует экземпляр Spark и впоследствии может использоваться для создания RDD, выполнения действий и преобразований в RDD, а также извлечения данных и других функций Spark. SparkContext также инициализирует различные свойства процесса, такие как имя приложения, количество ядер, параметры использования памяти и другие характеристики. В совокупности эти свойства содержатся в объекте SparkConf, который передается в SparkContext в качестве параметра.

Spark Dataframes.

Spark DataFrame – это набор данных, организованный в именованные столбцы. Концептуально он эквивалентен таблице в реляционной базе данных или фрейму данных в R / Python, но с более обширной внутренней оптимизацией. DataFrames могут быть созданы из широкого спектра источников, таких как файлы структурированных данных, таблицы в Hive, внешние базы данных или существующие RDD.

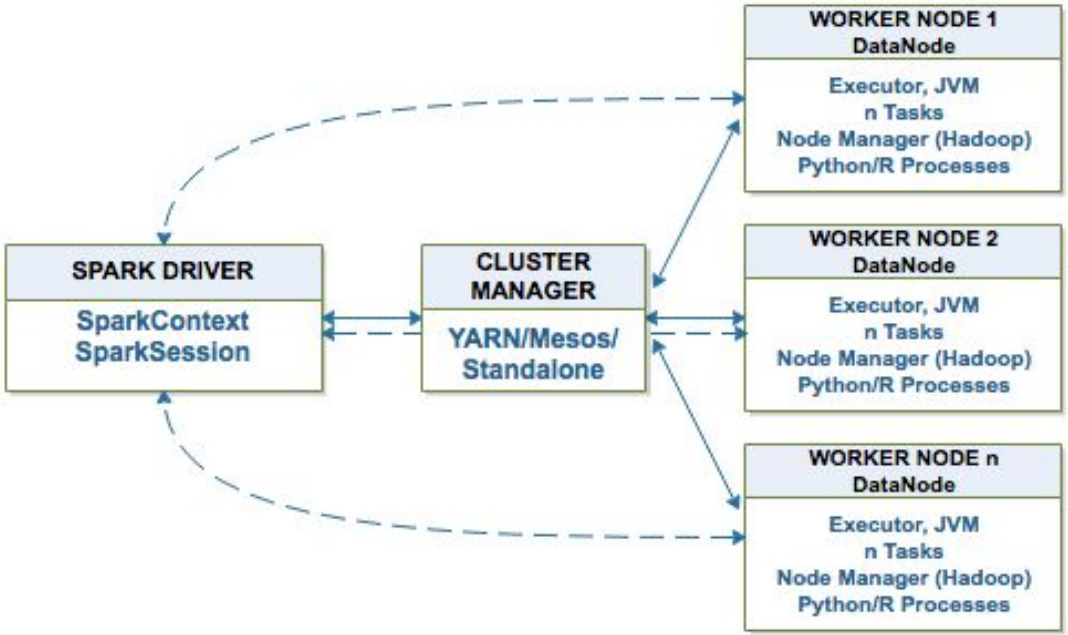
Transformations.

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.



WorkerNode - это серверы, на которых размещены приложения Spark. Каждое приложение получает свой собственный уникальный процесс-исполнитель, а именно процессы, которые выполняют фактическое действие и задачи преобразования.

SparkDriver отправляет инструкции рабочим узлам для планирования задач.

Менеджеры кластеров координируют обмен данными между рабочими узлами, управляют узлами (такими как запуск, остановка)

Data locality. Как ее реализовать. Сериализация. Пример.

Локальность данных - это процесс перемещения вычисления ближе к месту, где находятся фактические данные на узле, вместо перемещения больших данных в вычисления. Это минимизирует перегрузку сети и увеличивает общую пропускную способность системы.

1) Данные лежат на том же узле, что и mapper -good
2) Данные лежат на соседнем узле в стойке - not so good
3) Данные лежат в ноде на другой стойке - bad

Локальность данных обеспечивается сериализацией кода

Сериализация — процесс перевода какой-либо структуры данных в последовательность байтов. Обратной к операции сериализации является операция десериализации (структуризации) — восстановление начального состояния структуры данных из битовой последовательности.

Но не все функции сериализуются - например

```
def func(x):
    f = open('file.txt')
    return x in f.readlines
```