Промышленное машинное обучение на Spark

Содержание курса

2. HDFS. Apache Spark. RDD.

План:

1. HDFS. Ее устройство и основные свойства.
2. Spark - основные абстракции и объекты.
3. Действия и преобразования.
4. 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

HDFS. Ее устройство и основные свойства.

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

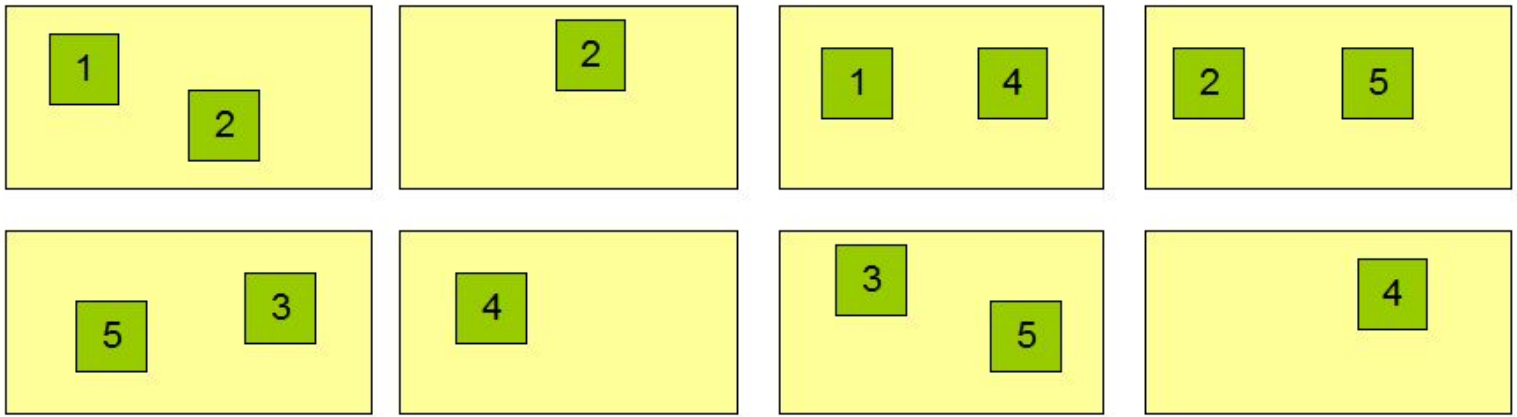
Block Replication

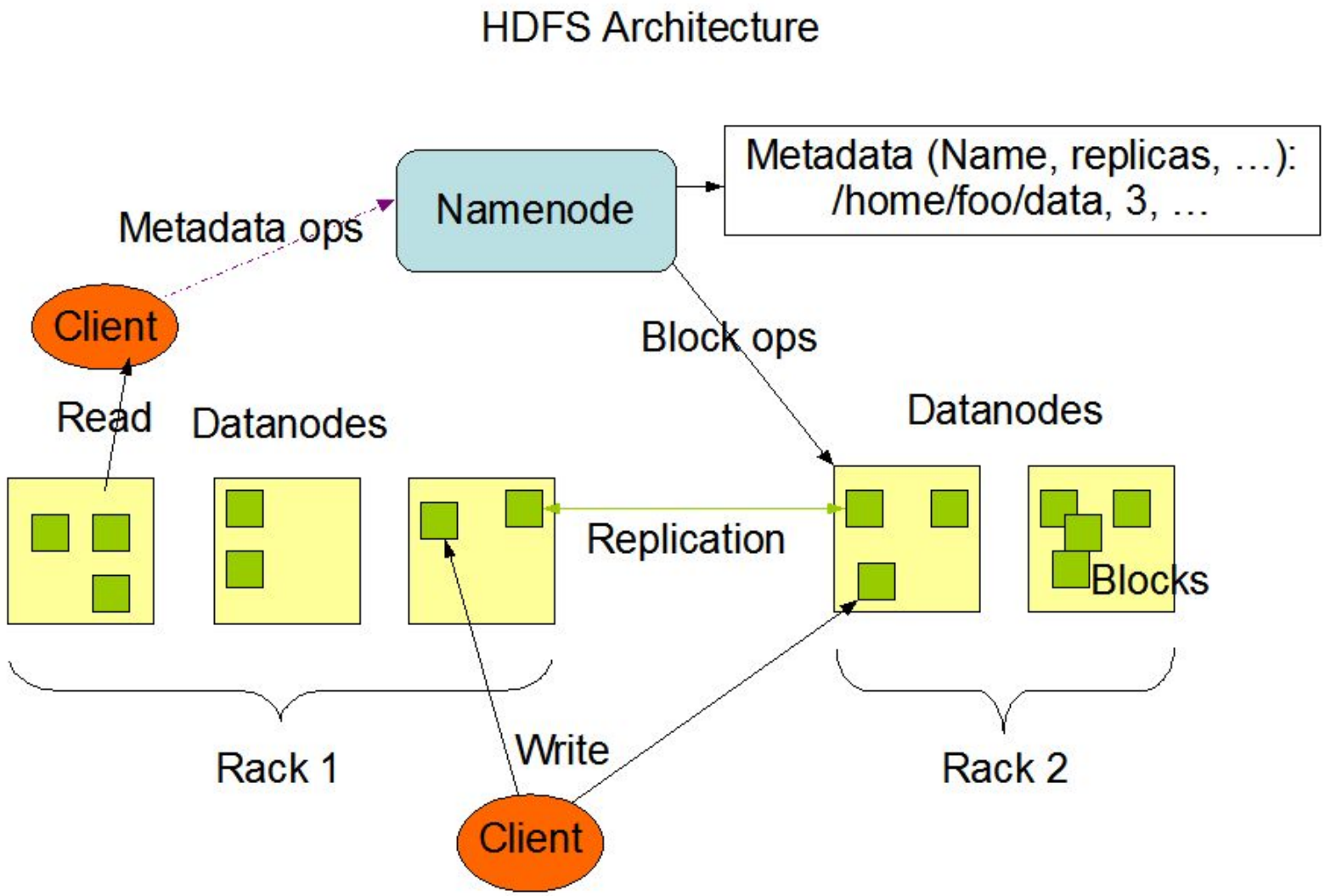
- **Управляющий узел, узел имен или сервер имен (NameNode)**

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.

Форматы хранения данных в Hadoop

Row-Oriented vs Column-Oriented

Row-oriented: rows stored sequentially in a file

Key	Fname	Lname	State	Zip	Phone	Age	Sales
1	Bugs	Bunny	NY	11217	(123) 938-3235	34	100
2	Yosemite	Sam	CA	95389	(234) 375-6572	52	500
3	Daffy	Duck	NY	10013	(345) 227-1810	35	200
4	Elmer	Fudd	CA	04578	(456) 882-7323	43	10
5	Witch	Hazel	CA	01970	(567) 744-0991	57	250

Column-oriented: each column is stored in a separate file
Each column for a given row is at the same offset.

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
CA
CA

Zip
11217
95389
10013
04578
01970

Phone
(123) 938-3235
(234) 375-6572
(345) 227-1810
(456) 882-7323
(567) 744-0991

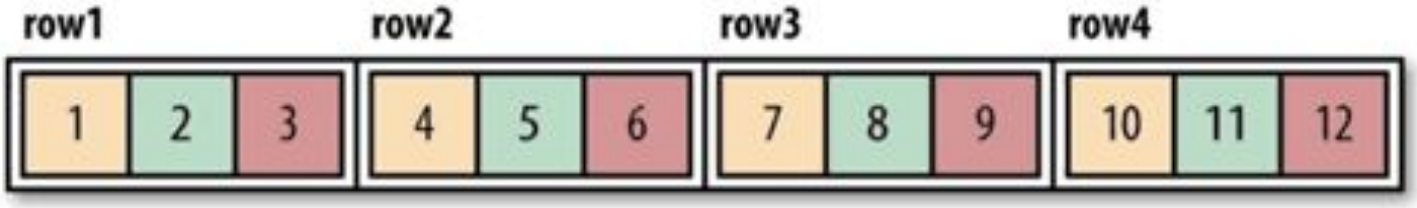
Age
34
52
35
43
57

Sales
100
500
200
10
250

[Гайд по форматам хранения данных]

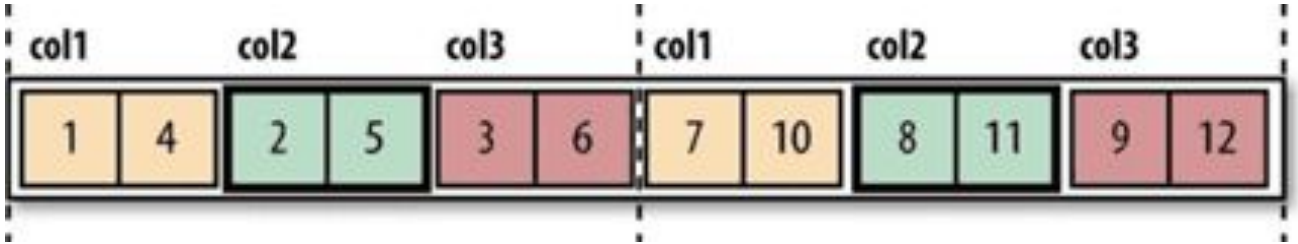
Линейные форматы

В линейных форматах (AVRO, Sequence) строки данных одного типа хранятся вместе, образуя непрерывное хранилище. Даже если необходимо получить лишь некоторые значения из строки, все равно вся строка будет считана с диска в память. Линейный способ хранения данных обусловливает пониженную скорость операций чтения и выполнении избирательных запросов, а также больший расход дискового пространства.



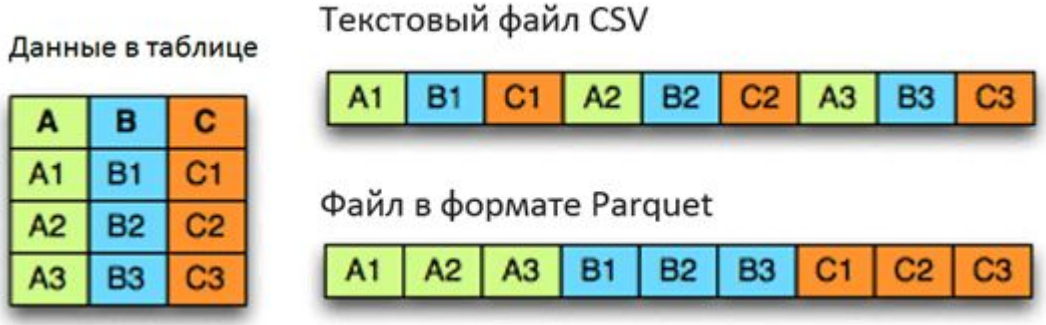
Колоночные форматы

В колоночно-ориентированных форматах (Parquet, RCFile, ORCFile) файл разрезается на несколько столбцов данных, которые хранятся вместе, но могут быть обработаны независимо друг от друга. Такой метод хранения информации позволяет пропускать ненужные столбцы при чтении данных, что существенно ускоряет чтение данных и отлично подходит в случае, когда необходим небольшой объем строк или выполняются избирательные запросы, как, например, в СУБД Apache Hive.



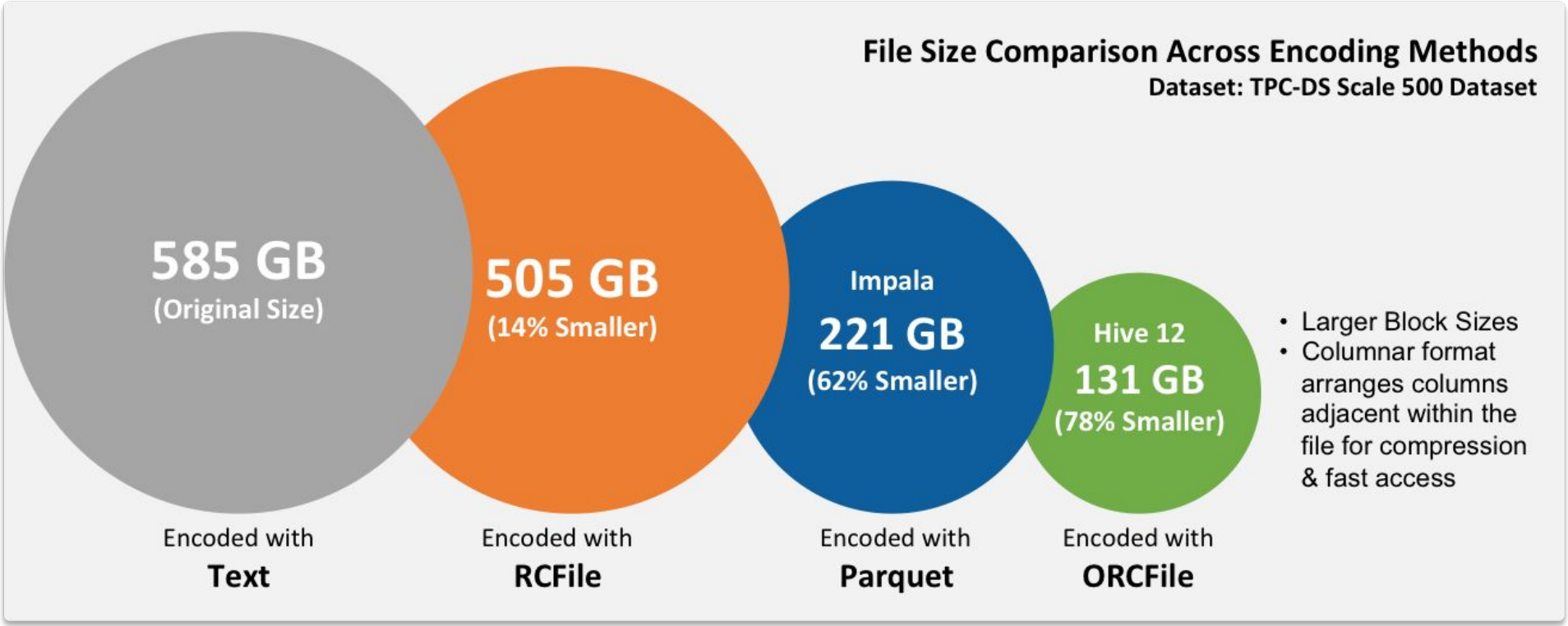
Parquet

Parquet - это столбчатый формат хранения данных. Это помогает повысить производительность, иногда значительно, разрешая хранение и доступ к данным для каждого столбца. Интуитивно понятно, что если бы вы работали с файлом размером 1 ГБ со 100 столбцами и 1 миллионом строк и хотели запрашивать данные только из одного из 100 столбцов, возможность доступа только к отдельному столбцу была бы более эффективной, чем доступ ко всему файл.



ORC

ORC расшифровывается как Optimized Row-Columnar. В некотором смысле это дополнительный уровень оптимизации по сравнению с чистыми столбцовыми форматами, такими как Parquet. ORCFiles хранит данные не только по столбцам, но и по строкам, также известным как полосы. Таким образом, файл с данными в табличном формате можно разделить на несколько более мелких полос, каждая из которых состоит из подмножества строк из исходного файла. При таком разделении данных, если пользовательской задаче требуется доступ только к небольшому фрагменту данных, процесс может опросить конкретную полосу, содержащую данные.


File Size Comparison Across Encoding Methods
Dataset: TPC-DS Scale 500 Dataset
585 GB
(Original Size)
505 GB
(14% Smaller)
Impala
221 GB
(62% Smaller)
Hive 12
131 GB
(78% Smaller)
• Larger Block Sizes
• Columnar format arranges columns adjacent within the file for compression & fast access
Encoded with
Text
Encoded with
RCFile
Encoded with
Parquet
Encoded with
ORCFile

Spark - основные абстракции и объекты.

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.

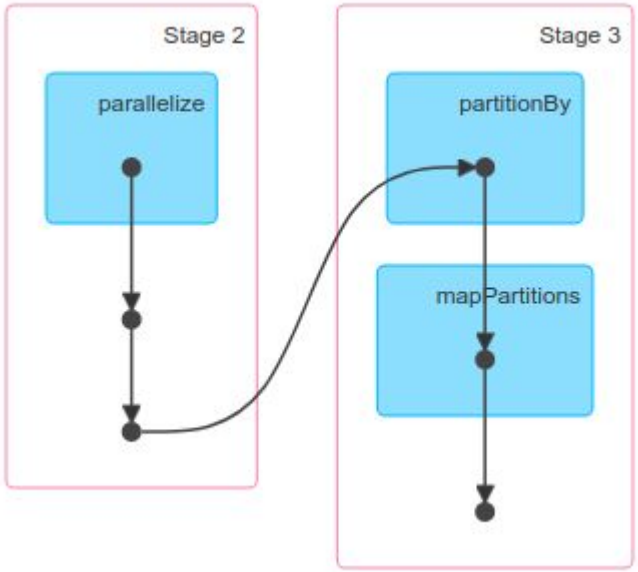
SparkContext - это точка входа для всех операций Spark и средство, с помощью которого приложение подключается к ресурсам кластера Spark. Он инициализирует экземпляр Spark и впоследствии может использоваться для создания RDD, выполнения действий и преобразований в RDD, а также извлечения данных и других 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 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.

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.

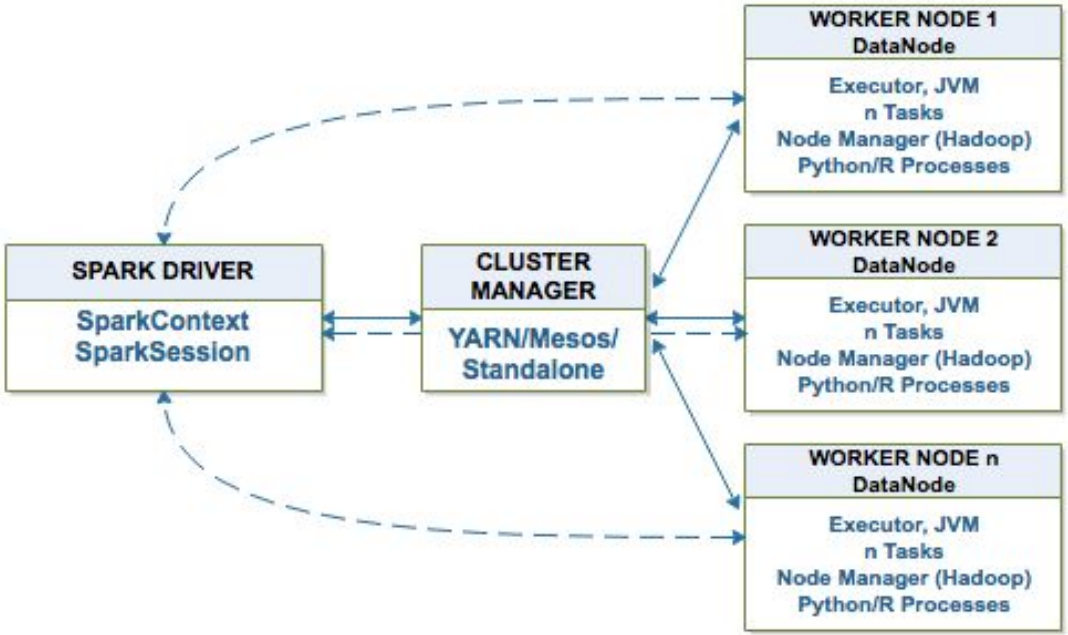
Transformations.

map(*func*)	Возвращает новый распределенный набор данных, сформированный путем передачи каждого элемента источника через функцию func.
filter(*func*)	Возвращает новый набор данных, сформированный путем выбора тех элементов источника, для которых функция func возвращает true.
mapPartitions(*func*)	Аналогично map, но выполняется отдельно на каждом разделе (блоке) RDD, поэтому функция должна иметь тип Iterator<T> => Iterator<U> при запуске на RDD типа T.
sortByKey([*ascending*], [*numPartitions*])	При вызове набора данных пар (K, V), где K упорядочено, возвращает набор данных пар (K, V), отсортированных по ключам в порядке возрастания или убывания, как указано в логическом аргументе по возрастанию.
repartition(*numPartitions*)	Произвольно перетасуйте данные в RDD, чтобы создать больше или меньше разделов и сбалансировать их между ними. Это всегда перетасовывает все данные по сети.

groupByKey([*numPartitions*])	При вызове набора данных из пар (K, V) возвращает набор данных из пар (K, Iterable<V>). **Примечание:** Если вы группируетесь для выполнения агрегации (например, суммы или среднего значения) по каждому ключу, использование reduceByKey или aggregateByKey даст гораздо лучшую производительность. **Примечание:** По умолчанию уровень параллелизма в выходных данных зависит от количества разделов родительского RDD. Вы можете передать необязательный аргумент numPartitions, чтобы задать другое количество задач.
reduceByKey(*func*, [*numPartitions*])	При вызове набора данных из пар (K, V) возвращает набор данных из пар (K, V), где значения для каждого ключа агрегируются с помощью заданной функции reduce func, которая должна иметь тип (V,V) => V. Как и в groupByKey, количество задач reduce настраивается с помощью необязательного второго аргумента.

Actions.

reduce(*func*)	Агрегируйте элементы набора данных с помощью функции func (которая принимает два аргумента и возвращает один). Функция должна быть коммутативной и ассоциативной, чтобы ее можно было правильно вычислять параллельно.
collect()	Верните все элементы набора данных в виде массива в программе драйвера. Это обычно полезно после фильтрации или другой операции, которая возвращает достаточно небольшое подмножество данных.
count()	Возвращает количество элементов в наборе данных.
first()	Возвращает первый элемент набора данных (аналогично take(1)).
take(*n*)	Возвращает массив с первыми n элементами набора данных.
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
```

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

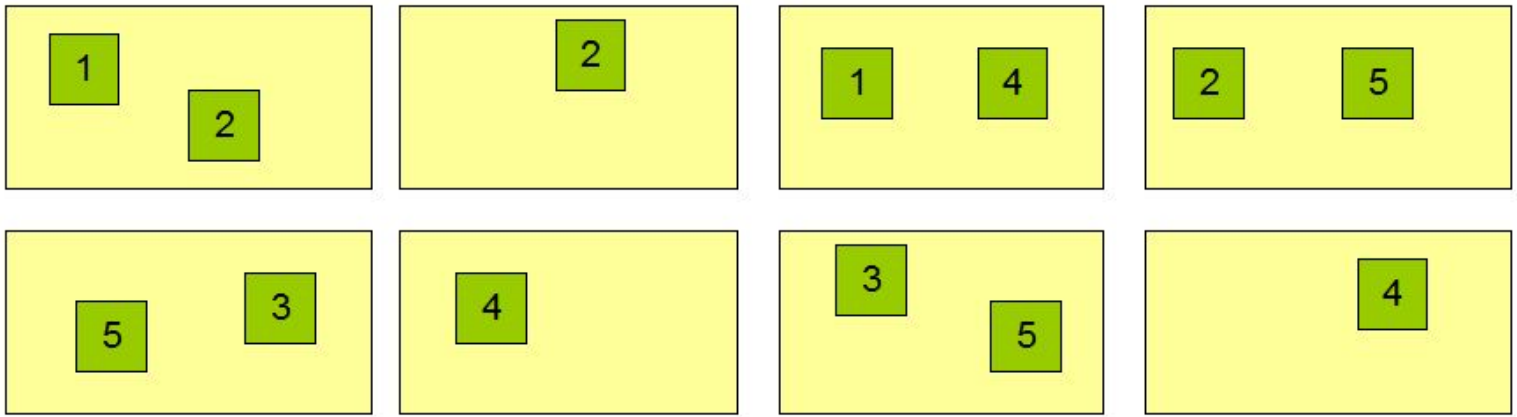
Block Replication

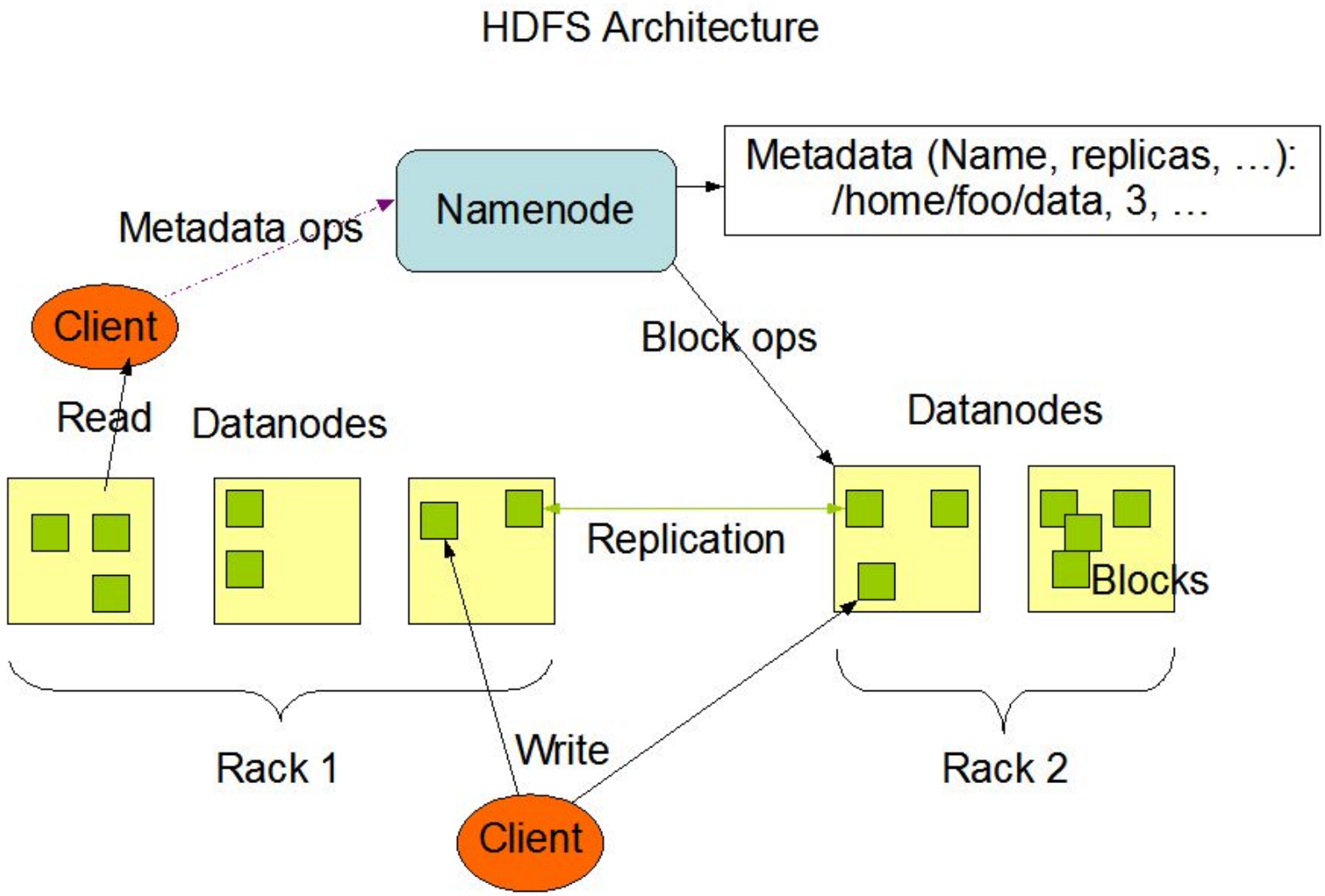
- **Управляющий узел, узел имен или сервер имен (NameNode)**

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.

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.

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.

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