What is big data?

Big data is nothing but large amount of data which is of the order of petabytes. Generated by

1. Banks- to keep an account of customer transactions
2. E-commerce companies save data of their customers and their transactions to draw insights, trends and is also used while giving offers
3. Telecom companies like Vodafone and Airtel utilize big data to suggest packs for customers analyzing their usage.
4. Social networking sites- As they have billions of users worldwide.

3 Vs of big data

1. **Velocity -** Speed at which data is increasing is very fast.
2. **Volume -** Size of the data is very large of the order of **petabytes.**
3. **Variety -** Not only structured but unstructured data like blobs, log files comprise of big data

Case: A company want to roll out offers to its top 10 customers. The offers need to be on products frequently used by those customers.

Issue/ Problem: Too large data - structured as well as unstructured to be stored, processed and analyzed

Solution: To store - Hadoop distributed file system
To process - Map-Reduce programs/ YARN
To analyze - Pig, Hive

Hadoop is open source and thus does not require any capital/investment.

Hadoop is an open-source framework written in java used to store and analyze big data. Hadoop is not an OLAP. It processes data by networking many physical systems which enables us to do parallel processing on big data. The increase in volume of the data is dealt by adding nodes to the clusters which can be easily done.

HDFS - Distributed file system in Hadoop

Hadoop architecture

Master (name node) - slave (data nodes) architecture

The architecture comprises of the name node and data node. **Name node** stores the meta data, information about location of files. As it is a single node it may become the single point of failure. And there are backup name nodes provided. Backup name node is activated once the actual name node stops sending heartbeat(signal) to the zookeeper node. Backup nodes are actually created by journal node. There are two files associated with the metadata:

- **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
- **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.

Namenode also takes care of the replicas and assigns nodes to these replicas once datanode fails.

Data node stores the actual files by dividing them in blocks. Upon instruction from name node data node reads and writes requests from the clients and also responsible for creating, deleting and replicating blocks.
Block: Default block size is 128 MB

Job Tracker

- The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using Name Node. In response, Name Node provides metadata to Job Tracker.

Task Tracker

- It works as a slave node for Job Tracker.
- It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper.

ADVANTAGES-
fast (Data is stored as cluster and thus retrieval is faster. Even processing tools are situated on same servers, thus reducing processing time.**) Scalable (** By adding nodes**), cost effective (**open source**), Failure resistant (**By replication of blocks.**)**
Fault tolerant (if any machine fails, the other machine containing the copy of that data automatically become active.**). Can be deployed on a low-cost hardware.**

YARN - YET ANOTHER RESOURCE MANAGER

WHY YARN?

Yarn allows Hbase , spark and other applications along with MapReduce to run simultaneously on the same cluster and thus utilizes the cluster better. Yarn was introduced with Hadoop 2.0 .

Yarn has 2 components

1. **Resource manager:** As the name suggests, manages the resources for the application running in the Hadoop cluster after accepting requests from the clients
   It has 2 components
   a) **Application manager:** It accepts the requests from the client. The Application manager is responsible to accept or reject the application when it is submitted to the Resource manager by the client.
   b) **Scheduler:** Schedules the jobs according to the resources
2. **Node manager:** Manages the memory resources inside the data/slave nodes. Assigned for each data node. Also sends monitoring information to resource manager.
3. **Application Master:** An application is a single job submitted to a framework. The application master is responsible for negotiating resources with the resource manager, tracking the status and monitoring progress of a single application. The application master requests the container from the node manager by sending a Container Launch Context(CLC) which includes everything an application needs to run. Once the application is started, it sends the health report to the resource manager from time-to-time.
4. **Container:** It is a collection of physical resources such as RAM, CPU cores and disk on a single node. The containers are invoked by Container Launch Context(CLC) which is a record that contains information such as environment variables, security tokens, dependencies etc.

KEY CONCEPTS:

A. Yarn is comprised of Resource Manager and Node Manager
B. There is only one Resource Manager which runs on Master Node
C. There will be multiple Node Managers running on each Data Node
D. Resource Manager deals with resource management to execute any Job/Application
E. Node Manager takes care of individual tasks/processes submitted to them
F. Please note that YARN is a generic Framework, its not only meant to execute Map Reduce Jobs. It can be used to execute any application, say main() of a Java Application.

Now, lets discuss about Job/Application Flow via YARN

(i) Client submits a Job to YARN.
(ii) The submitted Job can be a Map Reduce Job or any other application/process

(iii) This Job/application is picked by Resource Manager

(iv) Since there can be multiple Jobs/applications submitted to Resource Manager, hence Resource Manager will check the scheduling algorithm, available capacity to see if submitted Job/Application can be launched

(v) When Resource Manager finds that it can launch newly submitted Job/Application, it allocates a Container. Container is a set of resources (CPU,memory etc) required to launch the Job/Application

(vi) It checks which Node can take up this request, once it finds a Node then it contacts the appropriate Node Manager for the same

(vii) Node Manager will then actually allocate the resources required to execute the Job/application and will then launch Application Master Process within Container

(viii) Application Master Process is the main process for Job/Application execution. Please note that Application Master is Framework specific implementation. Map Reduce Framework has its own implementation of Application Master.

(ix) Application Master will check if additional resources or containers are required to execute the Job/Application. This is the case when we submit a Map Reduce Job where Multiple Mappers and Reducers will be required to accomplish the Job.

(x) If additional resources are required then Application Master will negotiate with Resource Manager to allocate resources/containers. It will be responsibility of Application Master to execute and monitor the individual tasks for an application/job.

(xi) The request made by Application Master to Resource Manager is known as Resource Request. The request contains the resources required to execute the individual Task and a location constraint. Location constraint is required as Task needs to be run in as proximity to data as possible to conserver network bandwidth. 12 As a response to Resource Request, Resource Manager will spawn a Node Manager on the selected Node. Node Manager will then allocate resources for the container. Within that container, task will run. This task is known as App Process.

(xii) If there are multiple Mappers then there will be multiple App Process running (in a container) on multiple Nodes. Each of them will send their heat beat to their Application Master Process. This is how Application Master will monitor individual Tasks it launches.

(xiii) Application Master will also send its Heart Beat Signal to Resource Manager to indicate status of Job/Application execution.

(xiv) Once any Application execution is completed then Application Master for that application will be de-registered.

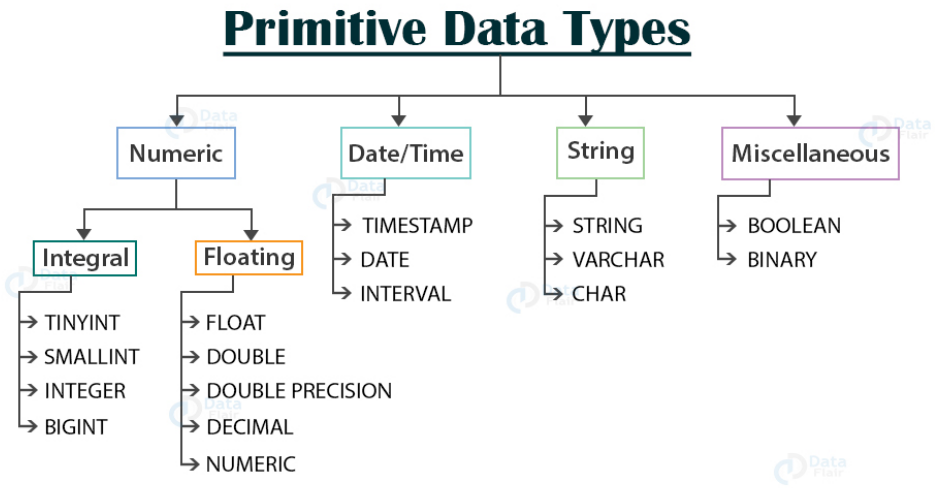
HIVE

- Data warehouse in Hdfs which allows processing in Sql language through Hiveql which is internally converted into MapReduce.
- Supports ddl, dml and user defined functions.
- Fast and scalable
- Storage types- text file, Hbase, RCFile
- Uses indexing to accelerate queries.

Limitations

- Cannot process unstructured data
- Not used for transactional processing, and thus cannot be used to process real time, streaming data.
- High latency (time b/w writing a query and receiving o/p)

DATATYPES

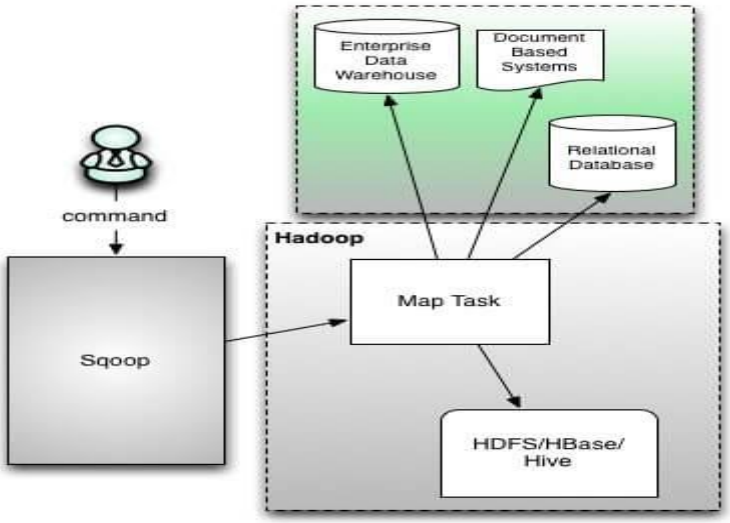


SQOOP - Sqoop is a command line interface application which is used to transfer big data from various relational database such as MySQL, oracle, sap Hana to Hadoop file system (HDFS, HIVE, HBASE etc.) and vice versa efficiently.

IMPORTANT FEATURES OF SQOOP

- Full Load
- Incremental Load
- Parallel import/export
- Import results of SQL query
- Compression
- Connectors for all major RDBMS Databases
- Kerberos Security Integration
- Load data directly into Hive/Hbase
- Support for Accumulo

HOW SQOOP WORKS



Here only Map phase will run and reduce is not required because the complete import and export process doesn't require any aggregation and so there is no need for reducers in Sqoop.

1. Sqoop asks for metadata information from Relation DB.
2. Relational DB returns the required request.
3. Based on metadata information Sqoop generates java classes.
4. Based on primary id partitioning happens in table as multiple mappers will importing data as the same time.

PIG - High level Apache project and is internally written in java.
Allows users to explicitly specify the sequence of programs
Easy to code for new developers
Eliminates complexity of java and thus opens up MapReduce to more users.
Follows lazy evaluation

Runs in 2 modes

1. Local mode - On Client side **(Start: pig -x local) (grunt interface)**
Executes in a single JVM
Files are installed and run using the local host
Thus input and output files are stored in local file system

2. Map Reduce mode **(Start: pig)**
Default mode also known as the Hadoop mode
Pig Latin is internally converted as a MapReduce job in Hadoop cluster
Input and output are present on the hdfs

PIG DATATYPES
int long float double array bytearray(BLOB) chararray

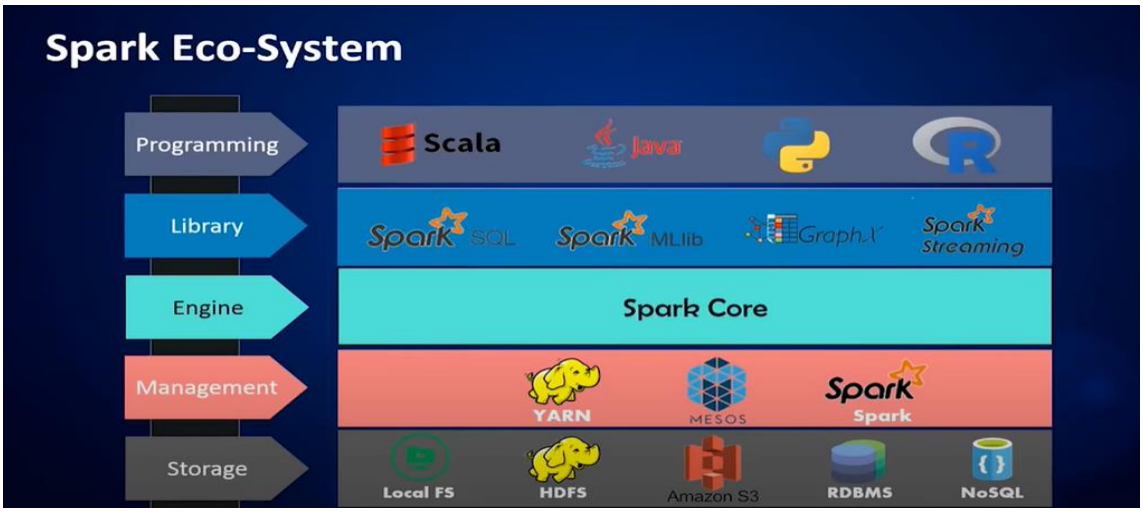
COMPLEX DATATYPES

tuple - ()
map - []
bag - collection of tuples - {(), (), ….}

SPARK - Spark is an open source, scalable, massively parallel, in memory execution environment for running analytics applications. Like MapReduce, spark also runs on cluster but uses random access memory to cache and process data which makes it 100 times faster than MapReduce for small workloads. It is called as **LIGHTNING-FAST ANALYTICS ENGINE FOR BIG DATA AND ML.**

FEATURES OF SPARK

1. **SPEED:** 100x faster on RAM and 10x faster on local disc than Hadoop map-reduce
2. **CACHING:** It has a simple programming layer that provides powerful caching and disk persistence capabilities
3. **REAL-TIME PROCESSING:** One of the main reasons Spark was introduced is Real time processing possible due to in-memory computation.
4. **POLYGLOT:** Provides several APIs in JAVA, SCALA, PYTHON, SQL
5. **DEPLOYMENT:** Can be deployed using Mesos, yarn or even its own cluster manager.



SPARK -CORE ENGINE

It is the most vital component of Spark ecosystem, which is responsible for basic I/O functions, scheduling, monitoring Etc. The entire Apache spark ecosystem is built on the top of this core execution engine

LIBRARIES

The **spark SQL** component is used to leverage the power of declarative queries and optimize storage by executing SQL queries on spark data, which is present in the rdds and other external sources

Spark streaming component allows developers to perform batch processing and streaming of data in the same application
Machine learning library. It eases the deployment and development of scalable machine learning pipelines, like summary statistics correlations feature extraction transformation functions optimization algorithms etc.
Graph x component lets the data scientist to work with graph and non-graph sources to achieve flexibility and resilience in graph construction and transformation
Spark R: R package that allows us to use Apache Spark to perform selection, aggregation, filtering operations. Also supports distributed machine learning using ml library

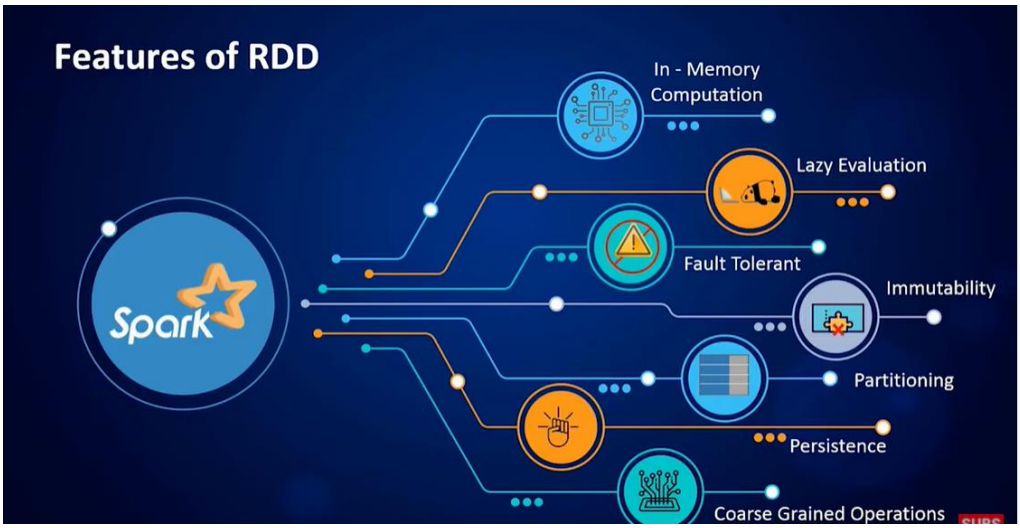
RDD- Resilient Distributed Datasets

Resilient: Data can be restored on failure
Distributed: Data is distributed on different nodes
Datasets: Group of data

WHY RDD?
In previous computing systems, the intermediate data/output needs to be stored on distributed file system, and these multiple I/O operations to distributed file system is time consuming. Moreover, the replications and serializations made the process even slower.
How does RDD overcome this?
RDD uses fault tolerant in-memory computing (ON RAM) and hence make the computing 10 to 100 times faster.

RDD is considered as the backbone of spark and is the fundamental datatype in spark that can handle both structured as well as unstructured data. RDD is an immutable (read only) collection of objects.
Anything you do on **spark** creates an RDD. if you are reading a data or transforming an existing RDD, a **new RDD** is created
Each data set present in an RDD is divided into logical partitions, which may be computed on different nodes of the cluster due to this you can perform Transformations or actions on the complete data parallelly w/o having to worry about the distribution because spark takes care of that

FEATURES OF RDD



1. **IN-MEMORY COMPUTATION**
2. **LAZY EVALUATION:** Result is not computed unless an action is specified
3. **FAULT TOLERANT:** in case of RDDs they track the data lineage information (DAG - Directed acyclic graph) to rebuild the last data automatically and this is how it provides fault tolerance to the system.
4. **IMMUTABILITY:** Data can be created and retrieved any time but once defined cannot be altered.
5. **PARTITIONING:** Fundamental of parallelism, data is logically divided and stored in partitions
6. **Persistence:** RDDs can be cached in memory at any instance that would be used many times in future
7. **Coarse Grained Operations:** Applies to all elements of the datasets.

3 ways to create RDDs

1. **By parallelizing collections (arrays, tuples)**
   rdd = sc.parallelize(collection_name, no_of_partition)
2. **Transformation on existing rdd**
   var a1 = Array(1,2,3,4,5,6,7,8,9,10)
   val r1 = sc.parallelize(a1)
   val new_rdd = r1.map (data => data*2)
3. **From external files**
   val rdd = sc.<filetype> (local or hdfs dir)

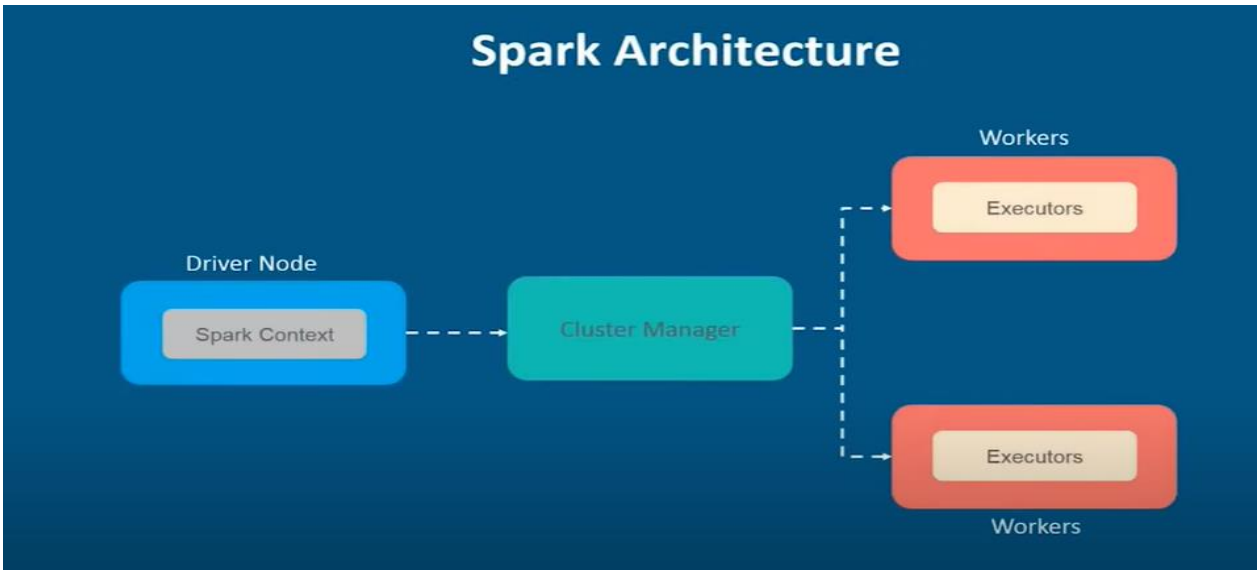
OPERATIONS OF RDD

1. **Transformation:** As earlier discussed, rdds are immutable, hence any transformation on existing rdd creates a new one.
   Narrow Transformation: Can be applied to single partition of parent rdd to create new rdd. No shuffling is required. ex-Map, flatmap
   Wide Transformations: Applied on multiple partitions of parent rdd as data is present on multiple partitions. Ex- reducebykey
2. **Action:** These commands produce results, and all operations are performed only when these commands are called as sparks follows lazy evaluation.

Workloads in spark

1. **Batch mode:** It is a batch job. We write a batch job and then schedule it, works through a queue
2. **Interactive mode:** It is an interactive shell where you go and execute the commands one by one. So you will execute one command check the result and then execute other command.
3. **Streaming mode:** Program runs continuously. As and when data is generated, transformation and actions are carried out on data to get results

SPARK ARCHITECTURE



DRIVER NODE- Master Node

In your master node, you have the driver program, which drives your application. The

code you are writing behaves as a driver program or if you are using the interactive shell, the shell acts as the driver program. First step is driver creates a spark context
Spark Context is a gateway to spark functionalities, any command created on spark has to go through spark context
Cluster manager: The role of the cluster manager is to allocate resources to applications on the worker node.
The purpose of Spark Context is to coordinate the spark applications, running as independent sets of processes on a cluster. To run on a cluster, the Spark Context connects to a different type of cluster managers and then perform the following tasks: -

1. It acquires executors on nodes in the cluster.
2. Then, it sends your application code to the executors. Here, the application code can be defined by JAR or Python files passed to the Spark Context.
3. At last, the Spark Context sends tasks to the executors to run.

Worker Node
The worker node is a slave node. Its role is to run the application code in the cluster.

Executor
An executor is a process launched for an application on a worker node.
It runs tasks and keeps data in memory or disk storage across them.
It read and write data to the external sources.
Every application contains its executor.

Task- A unit of work that will be sent to one executor.

Spark shell on LOCAL HOST 4040
Introduction to RDD Lineage
Basically, evaluation of RDD is lazy in nature. It means a series of transformations are performed on an RDD, which is not even evaluated immediately.
While we create a new RDD from an existing Spark RDD, that new RDD also carries a pointer to the parent RDD in Spark. That is the same as all the dependencies between the RDDs those are logged in a graph, rather than the actual data. It is what we call as lineage graph. **Lineage graph tracks the series of transformation for a particular rdd - It is the logical plan**.
DIRECT ACYCLIC GRAPH
Physical execution plan or execution is known as **DAG** of stages. As soon as one applies the action on the final rdd the lineage is sent to **catalyst optimizer** which creates the final plan which is sent to DAG. DAG creates the physical plan and divide the plan into stages depicting which stages can be run in parallel and which have dependencies.