19AD501 – BIG DATA ANALYTICS

UNIT III

Introducing Hadoop –Hadoop Overview – RDBMS versus Hadoop – HDFS (Hadoop Distributed File System): Components and Block Replication – Processing Data with Hadoop – Introduction to MapReduce – Features of MapReduce – YARN, HBASE

Introducing Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers.

Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

In short Hadoop is an open source software framework for sorting and processing big data in distributed way on large clusters of commodity hardware.

Why Hadoop?
Its capability to handle massive amounts of data, different categories of data fairly quickly.

1. **Low cost:** It is an open source framework and uses commodity hardware to store enormous quantities of data.
2. **Computing Power:** Hadoop is based on distributed computing model, therefore more number of computing nodes, the more processing power at hand.
3. **Scalability:** When adding more nodes as the system grows and requires less administration.
4. **Storage Flexibility:** Hadoop provides convenience of storing as much as data as one needs and also added flexibility of deciding later as to how to use the stored data.
5. **Inherent Data Protection:** Hadoop protects the data and executing applications against hardware failure. If a node fails it automatically redirects the jobs that had been assigned to this node to the other functional and available nodes.

HADOOP OVERVIEW

Hadoop is open source software framework to store and process massive amounts of data in a distributed fashion on large clusters of commodity hardware. Basically, Hadoop accomplishes two tasks

1. Massive data storage
2. Faster data processing

Key Aspects of Hadoop

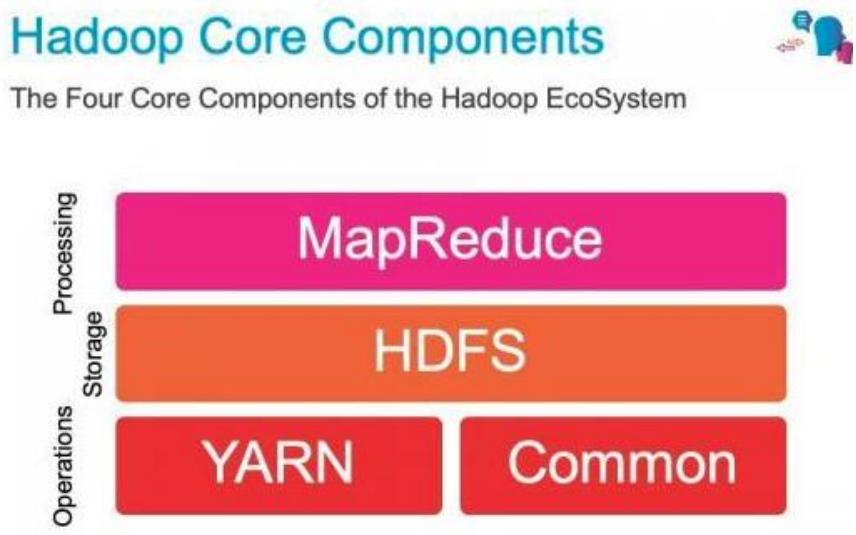
- **Open source**: It is free to download, use and contribute to
- **Frameworks**: Means everything that you will need to develop and execute and application is provided programs, tool etc.
- **Distributed**: Divides and stores data across multiple computers.

Computation/processing is done in parallel across multiple connected nodes.

- **Massive Storage**: Stores colossal, amount of data across nodes of low cost commodity hardware.
- **Faster Processing**: Large amounts of data is processed in parallel, yielding quick response.

Hadoop Core Components

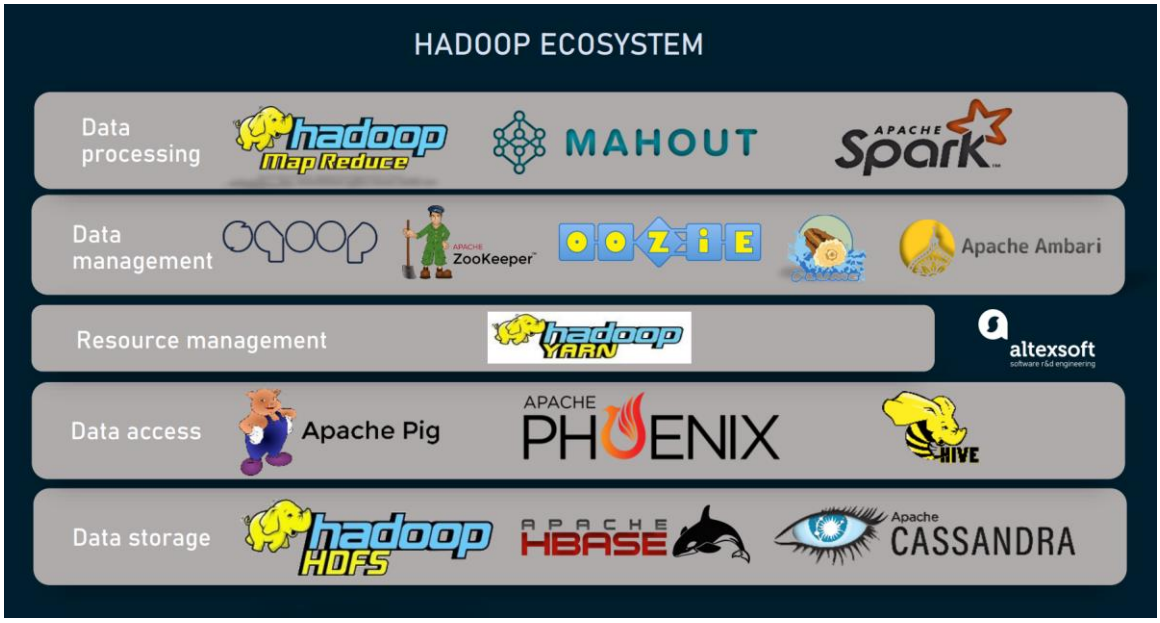
1. **Hadoop Common**: These are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
2. **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.
3. **Hadoop MapReduce**: This is YARN-based system for parallel processing of large data sets.
4. **Hadoop Yet Another Resource Negotiator (YARN):** This is a framework for job scheduling and cluster resource management.



Hadoop Ecosystem

Hadoop ecosystem support projects to enhance the functionality of hadoop core components. The Eco Projects are as follows

1. HIVE
2. PIG
3. SQOOP
4. HBASE
5. FLUME
6. OOZIE
7. AMBARI
8. MAHOUT
9. SPARK
10. ZOOKEEPER



Hadoop Conceptual Layer
It is conceptually divided into **Data Storage Layer** which stores huge volumes of data **Data Processing Layer** which processes data in parallel to extract richer and meaningful insights from data.

High-Level Architecture of Hadoop
Hadoop is distributed Master-Slave Architecture. Master Node is known as Name Node and slave nodes are known as DataNodes.

Master HDFS: Its main responsibility is partitioning the data storage across the slave nodes. It also keeps track of locations of data on DataNodes**.**

Master MapReduce: It decides and schedules computation task on slave nodes.

Why not RDBMS?
RDBMS is not suitable for storing and processing large files, images and videos. RDBMS is not a good choice when it comes to advanced analytics involving machine learning.

RDBMS Versus HADOOP

PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node based flat structure
Data	Suitable for structures data	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Performance	Data processing in GB's	Data Processing in PB's
Choice	When data needs consistent relationship	Big data processing, which does not require any consistent relationship between data

Software license	Proprietary	Open source
Project Environment	One project with multiple components	Eco System suite of java based projects
Architecture	Designed for client server architecture	Designed for distributed architecture
Hardware	High usage require high end server	Designed to run on commodity hardware.
File System	Relies on OS file system	Based on distributed file system
Updates	Stable product	Still evolving
Transactions	Support ACID transactions	Support BASE
Schema	Schema required on write	Schema required on read
Processor	Needs expensive hardware or high-end processor to store huge volumes of data.	In a hadoop cluster, a node requires only a processor, a network card, and few hard drives.
Cost	Cost around $10,000 to $14,000 per terabytes of storage	Cost around $4,000 per terabytes of storage.

HDFS (Hadoop Distributed File System)

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

- Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.
- HDFS holds very large amount of data and provides easier access. To store such huge data, the files split into blocks are stored across multiple machines.
- These files are stored in redundant fashion to rescue the system from possible data losses in case of failure.

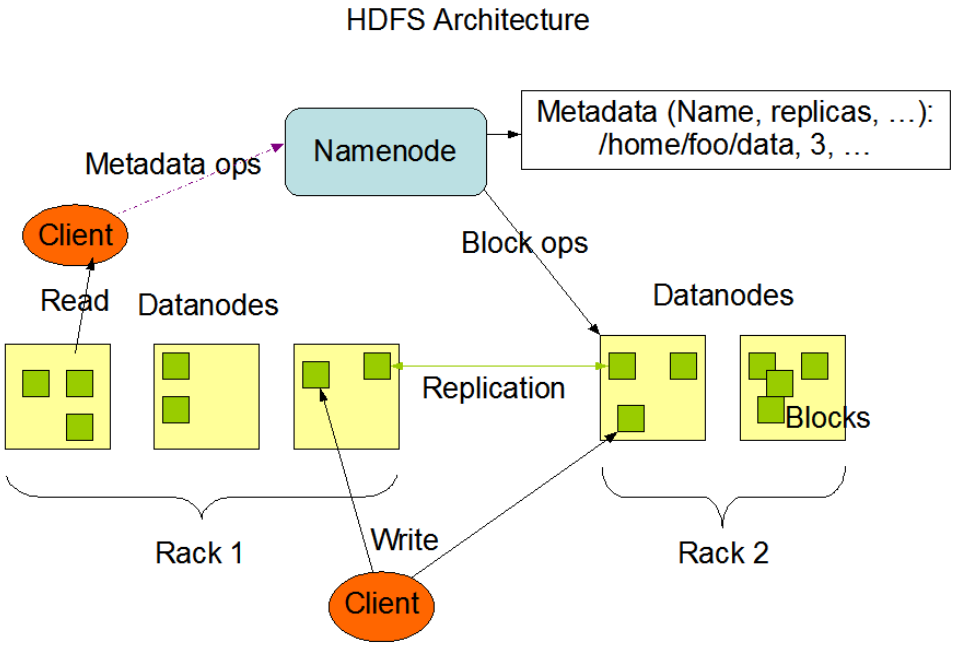
HDFS Architecture
HDFS uses a master/slave architecture where master consists of a single Name Node that manages the file system metadata and one or more slave. Data Nodes that store the actual data.

Name Node
The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes). NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients. Stores metadata for the files, like the directory structure of a typical File System.
Functions of Name Node

- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories. It also determines the mapping of blocks to DataNodes.
- It is the master daemon that maintains and manages the DataNodes (slave nodes)

- It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. 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
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog.



Data Node:

- The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. These nodes manage the data storage of their system.
- A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of Data Nodes.
- Data nodes store and retrieve blocks when they are requested by client or name node.
- They report back to name node periodically, with list of blocks that they are storing.
- The data node also perform operations such as block creation, deletion and replication as stated by the name node.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

Secondary NameNode:

Apart from these two daemons, there is a third daemon or a process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. And don't be confused about the Secondary NameNode being a backup NameNode because it is not.

Functions of Secondary NameNode:

- The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system.
- It is responsible for combining the EditLogs with FsImage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.

Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode

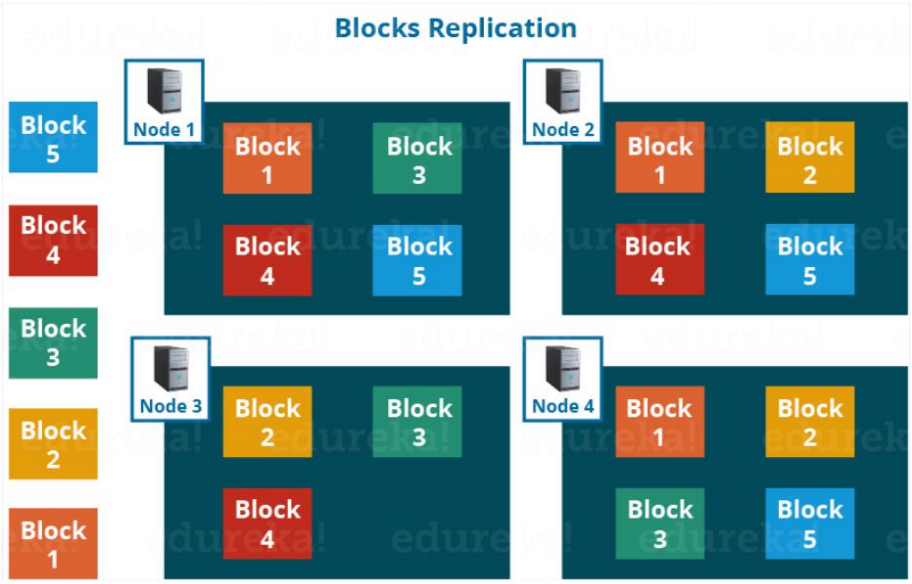
Block and Replication

Blocks are the nothing but the smallest continuous location on your hard drive where data is stored. In general, in any of the File System, you store the data as a collection of blocks.

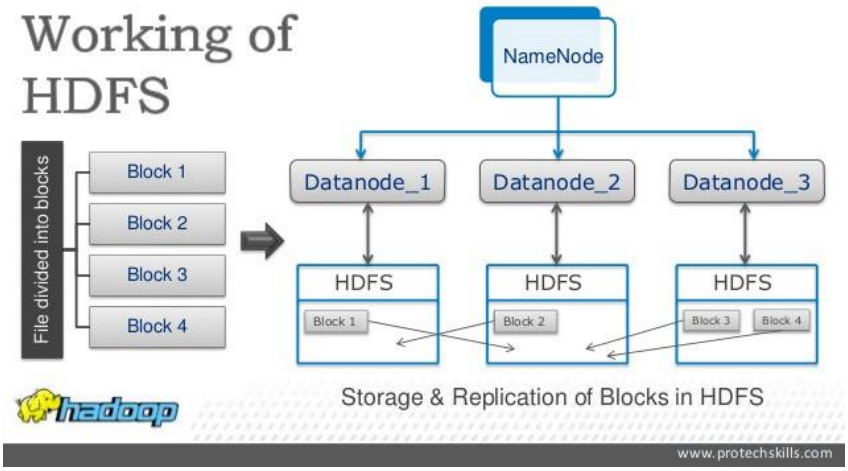
- Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes.
- These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block.
- The default block size in Hadoop version 1 is 64MB, but it can be increased as per the need to change in HDFS configuration. Default block size in Hadoop version 2 is 128 MB.

Replication Management:

HDFS provides a reliable way to store huge data in a distributed environment as data blocks. The blocks are also replicated to provide fault tolerance. The default replication factor is 3 which is again configurable.



- **Monitoring:** There is a continuous "**heartbeat**" communication between the data nodes to the name node.
- If a data node's heartbeat is not heard by the name node, the data node is considered to have failed and is no longer available.
- In this case, a replica is employed to replace the failed node, and a change is made to the replication scheme.



HDFS Read/ Write Architecture:
HDFS follows Write Once – Read Many Philosophy. So, you can't edit files already stored in HDFS. But, you can append new data by re-opening the file.

HDFS Write Architecture: Assume that the system block size is configured for 128 MB (default). So, the client will be dividing the file "example.txt" into 2 blocks – one of 128 MB (Block A) and the other of 120 MB (block B).

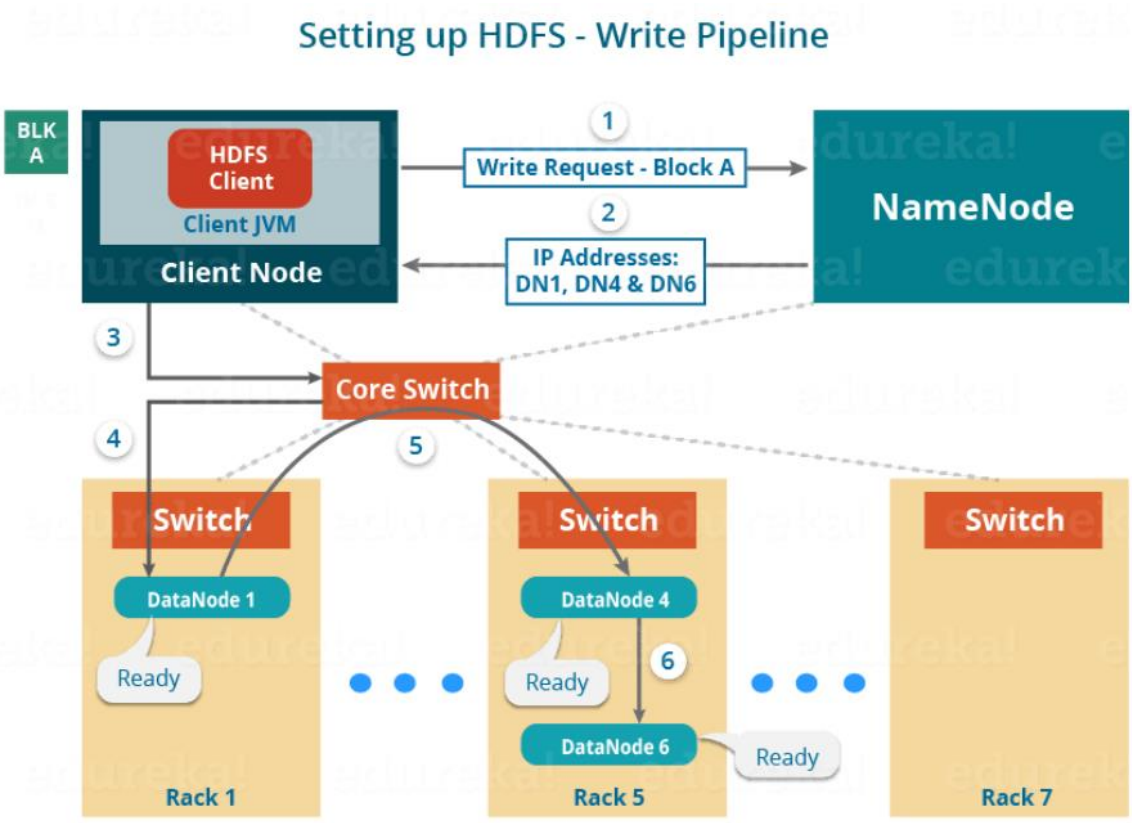
Now, the following protocol will be followed whenever the data is written into HDFS:

- At first, the HDFS client will reach out to the NameNode for a Write Request against the two blocks, say, Block A & Block B.
- The NameNode will then grant the client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually.
- The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness that we have discussed earlier.
- Let's say the replication factor is set to default i.e. 3. Therefore, for each block the NameNode will be providing the client a list of (3) IP addresses of DataNodes. The list will be unique for each block.
- Suppose, the NameNode provided following lists of IP addresses to the client:
    - For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}
    - For Block B, set B = {IP of DataNode 3, IP of DataNode 7, IP of DataNode 9}
- Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
    - Now the whole data copy process will happen in three stages: Set up of Pipeline
    - Data streaming and replication
    - Shutdown of Pipeline (Acknowledgement stage)

1. Set up of Pipeline:

Before writing the blocks, the client confirms whether the DataNodes, present in each of the list of IPs, are ready to receive the data or not. In doing so, the client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that

block. Let us consider Block A. The list of DataNodes provided by the NameNode is: For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}.

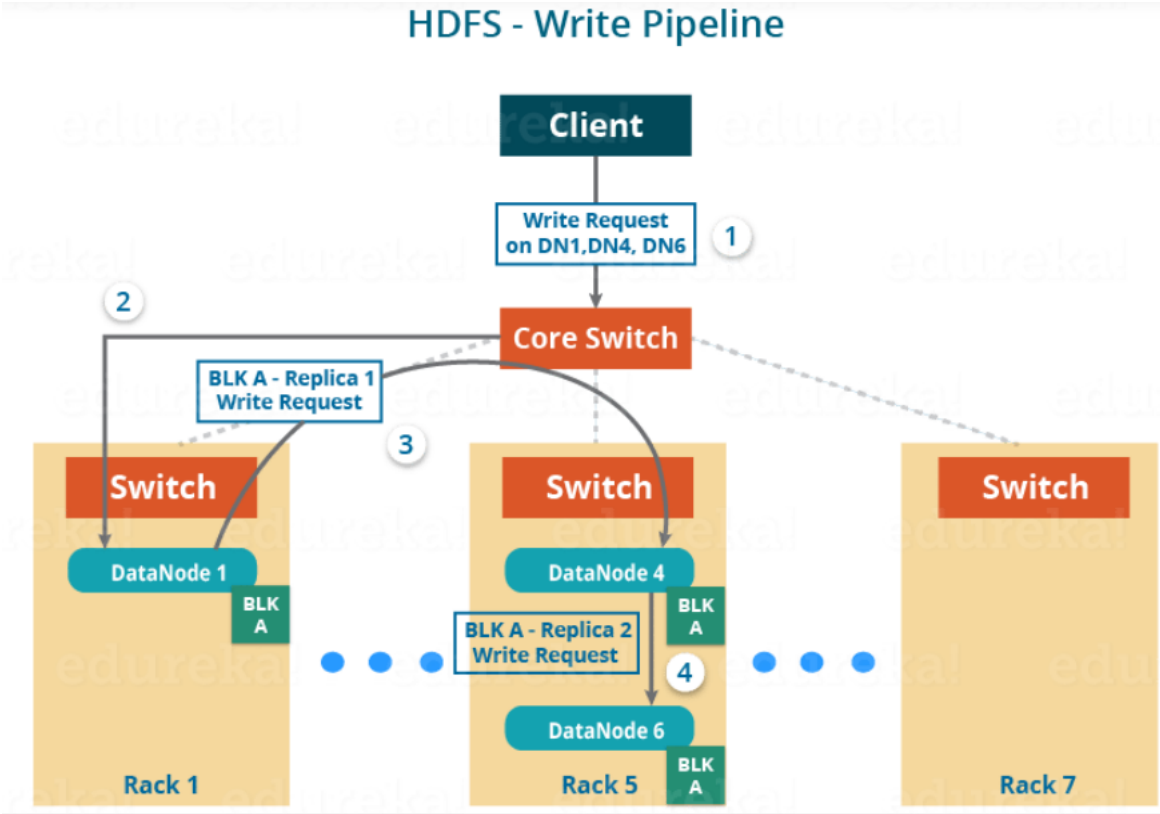


So, for block A, the client will be performing the following steps to create a pipeline:

- The client will choose the first DataNode in the list (DataNode IPs for Block A) which is DataNode 1 and will establish a TCP/IP connection.
- The client will inform DataNode 1 to be ready to receive the block. It will also provide the IPs of next two DataNodes (4 and 6) to the DataNode 1 where the block is supposed to be replicated.
- The DataNode 1 will connect to DataNode 4. The DataNode 1 will inform DataNode 4 to be ready to receive the block and will give it the IP of DataNode 6. Then, DataNode 4 will tell DataNode 6 to be ready for receiving the data.
- Next, the acknowledgement of readiness will follow the reverse sequence, i.e. From the DataNode 6 to 4 and then to 1.
- At last DataNode 1 will inform the client that all the DataNodes are ready and a pipeline will be formed between the client, DataNode 1, 4 and 6.
- Now pipeline set up is complete and the client will finally begin the data copy or streaming process.

2. Data Streaming:

As the pipeline has been created, the client will push the data into the pipeline. Now, don't forget that in HDFS, data is replicated based on replication factor. So, here Block A will be stored to three DataNodes as the assumed replication factor is 3. Moving ahead, the client will copy the block (A) to DataNode 1 only. The replication is always done by DataNodes sequentially.



So, the following steps will take place during replication:

- Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
- Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
- Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.

3. Shutdown of Pipeline or Acknowledgement stage:

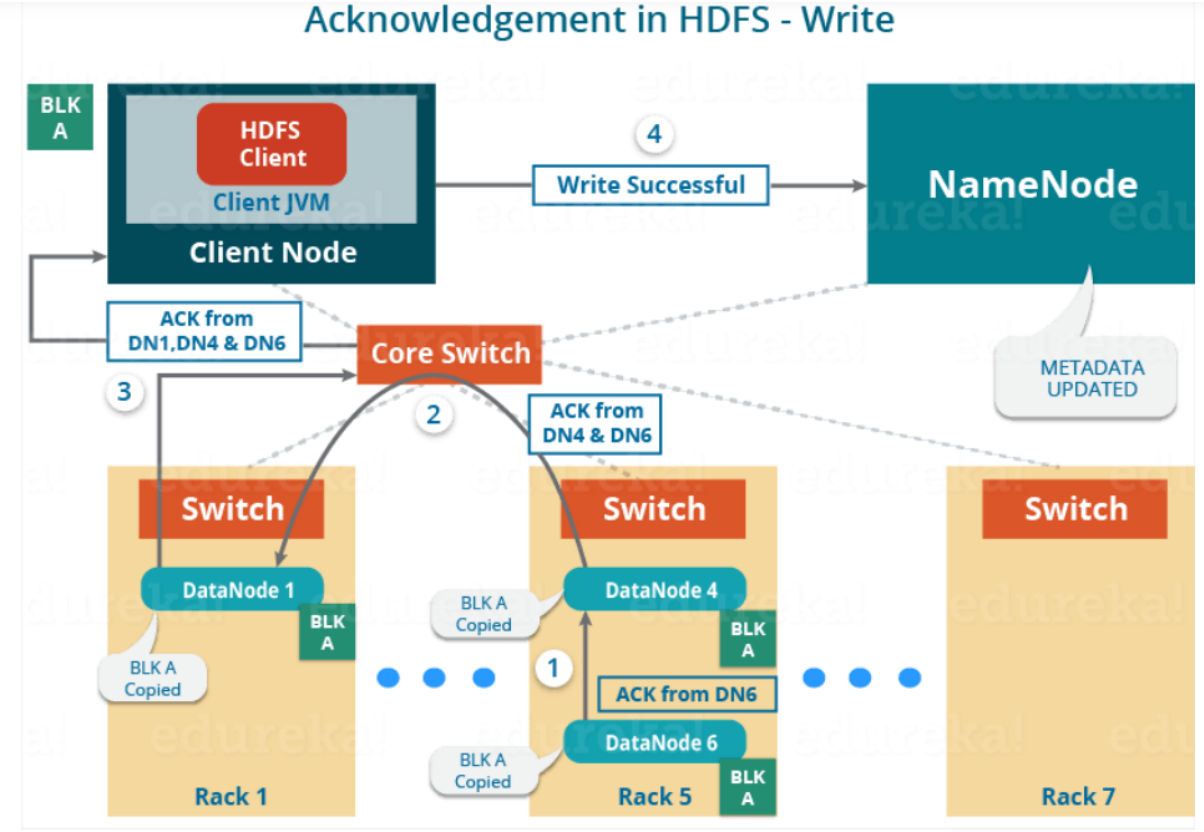
Once the block has been copied into all the three DataNodes, a series of acknowledgements will take place to ensure the client and NameNode that the data has been written successfully. Then, the client will finally close the pipeline to end the TCP session.

As shown in the figure below, the acknowledgement happens in the reverse sequence i.e. from DataNode 6 to 4 and then to 1. Finally, the DataNode 1 will push three acknowledgements (including its own) into the pipeline and send it to the client. The client will inform NameNode that data has been written successfully. The NameNode will update its metadata and the client will shut down the pipeline.

Similarly, Block B will also be copied into the DataNodes in parallel with Block A. So, the following things are to be noticed here:

- The client will copy Block A and Block B to the first DataNode **simultaneously**.

- Therefore, in our case, two pipelines will be formed for each of the block and all the process discussed above will happen in parallel in these two pipelines.
- The client writes the block into the first DataNode and then the DataNodes will be replicating the block sequentially.

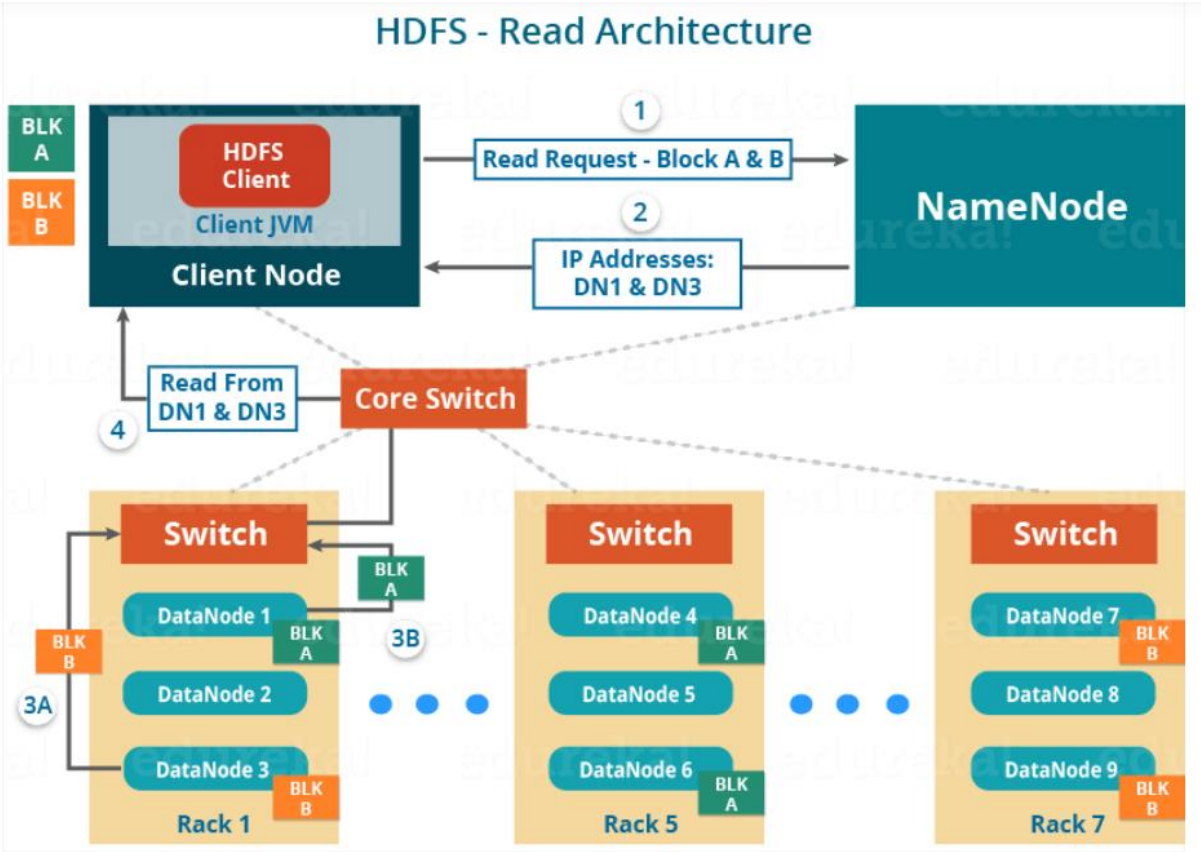


HDFS Read Architecture:

HDFS Read architecture is comparatively easy to understand. Let's take the above example again where the HDFS client wants to read the file "example.txt" now. Now, following steps will be taking place while reading the file:

- The client will reach out to NameNode asking for the block metadata for the file "example.txt".
- The NameNode will return the list of DataNodes where each block (Block A and B) are stored.
- After that client, will connect to the DataNodes where the blocks are stored.
- The client starts reading data parallel from the DataNodes (Block A from DataNode 1 and Block B from DataNode 3).
- Once the client gets all the required file blocks, it will combine these blocks to form a file.

While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.



MAPREDUCE

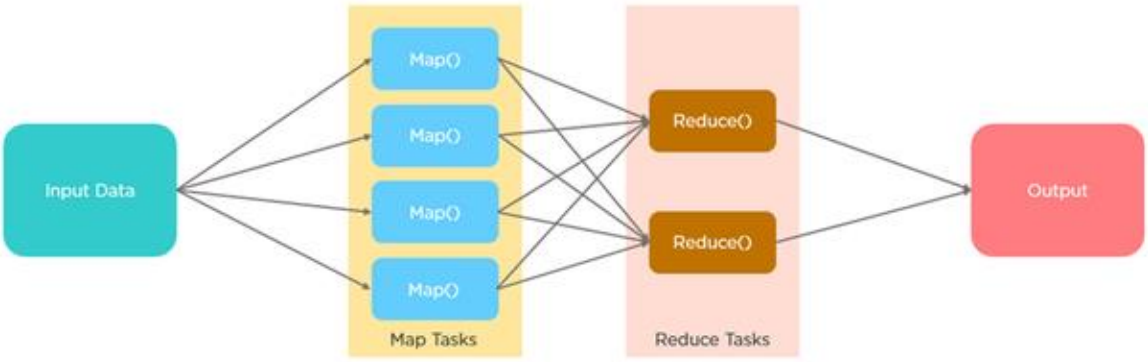
- MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analysing huge volumes of complex data.
- MapReduce is a processing technique and a program model for distributed computing based on java.
- The MapReduce algorithm contains two important tasks, namely **Map and Reduce**.
- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.
- The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers.

The Algorithm

MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.

1. **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
2. **Reduce stage**: This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After

processing, it produces a new set of output, which will be stored in the HDFS.



Phases

Input Phase − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

Map − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.

Intermediate Keys − They key-value pairs generated by the mapper are known as intermediate keys.

Combiner − It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper.

Shuffle and Sort − It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

Reducer − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

Output Phase − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Working

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.
- Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

The MapReduce Framework

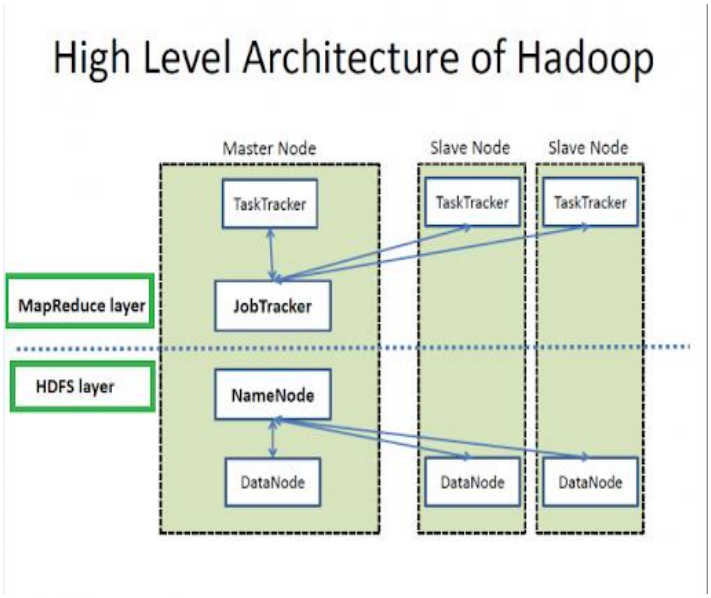
The MapReduce framework consists of a single master **JobTracker** and one slave **TaskTracker** per cluster-node.

Job Tracker

- The master Job Tracker is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks.

Task Tracker

- The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically.
- The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted.



YARN

- Hadoop Yet Another Resource Negotiator (YARN): This is a framework for job scheduling and cluster resource management. A resource management framework for scheduling and handling resource requests from distributed applications.

Why YARN?

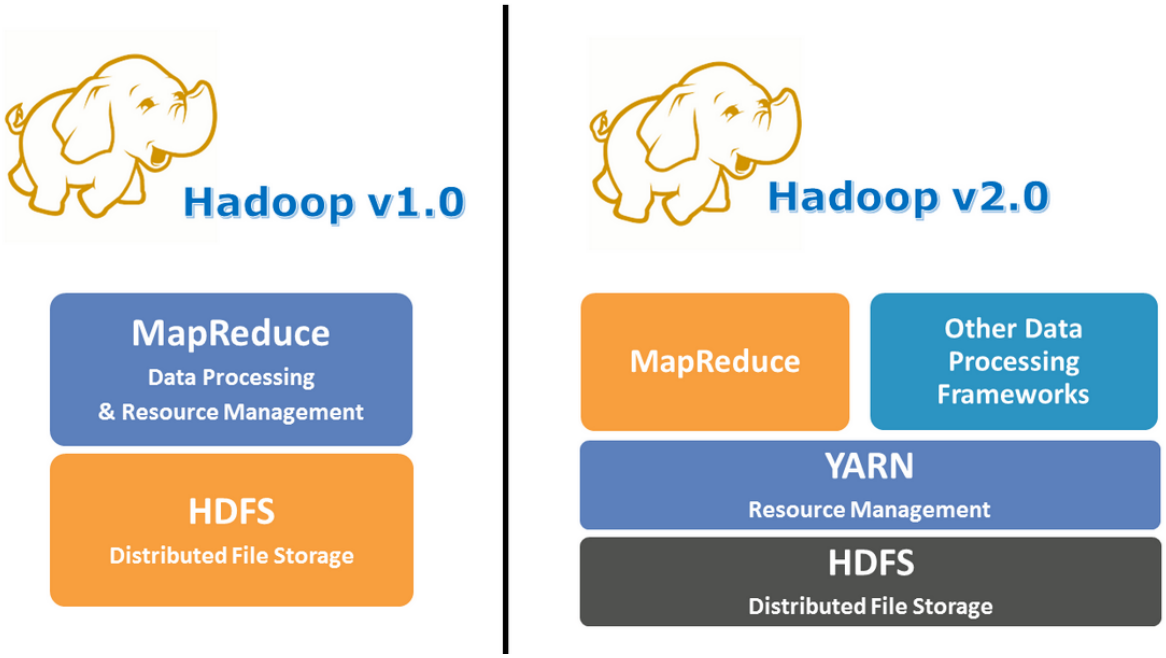
- In Hadoop version 1.0 which is also referred to as MRV1(MapReduce Version 1), MapReduce performed both processing and resource management functions. It consisted of a Job Tracker which was the single master. This design resulted in scalability bottleneck due to a single Job Tracker.

- The practical limits of such a design are reached with a cluster of 5000 nodes and 40,000 tasks running concurrently.

- Apart from this limitation, the utilization of computational resources is inefficient in MRV1. Also, the Hadoop framework became limited only to MapReduce processing paradigm.

To overcome all these issues, YARN was introduced in Hadoop version 2.0 in the year 2012

by Yahoo and Hortonworks. The basic idea behind YARN is to relieve MapReduce by taking over the responsibility of Resource Management and Job Scheduling. YARN started to give Hadoop the ability to run non-MapReduce jobs within the Hadoop framework.



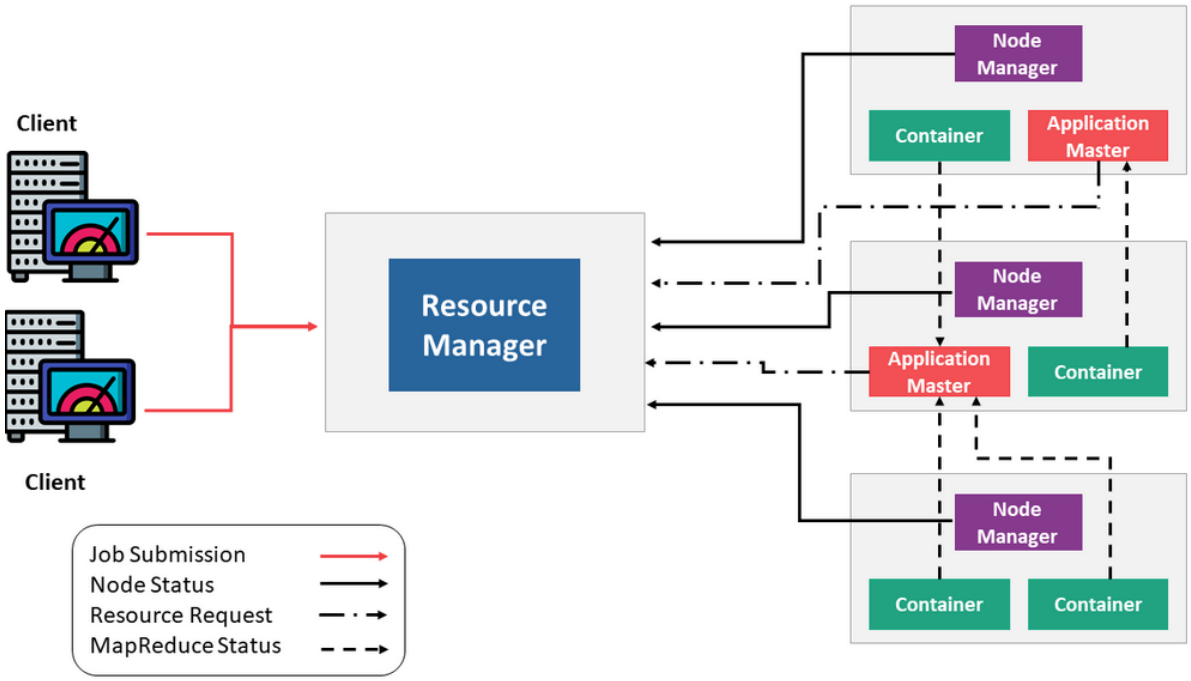
YARN allows different data processing methods like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS. Therefore YARN opens up Hadoop to other types of distributed applications beyond MapReduce.

YARN enabled the users to perform operations as per requirement by using a variety of tools like ***Spark*** for real-time processing, ***Hive*** for SQL, ***HBase*** for NoSQL and others.

YARN Architecture

Apart from Resource Management, YARN also performs Job Scheduling. YARN performs all your processing activities by allocating resources and scheduling tasks. Apache Hadoop YARN Architecture consists of the following main components :

1. **Resource Manager:** Runs on a master daemon and manages the resource allocation in the cluster.
2. **Node Manager:** They run on the slave daemons and are responsible for the execution of a task on every single Data Node.
3. **Application Master:** Manages the user job lifecycle and resource needs of individual applications. It works along with the Node Manager and monitors the execution of tasks.
4. **Container:** Package of resources including RAM, CPU, Network, HDD etc on a single node.



Components of YARN

You can consider YARN as the brain of your Hadoop Ecosystem.

Resource Manager

- It is the ultimate authority in resource allocation.
- On receiving the processing requests, it passes parts of requests to corresponding node managers accordingly, where the actual processing takes place.
- It is the arbitrator of the cluster resources and decides the allocation of the available resources for competing applications.
- Optimizes the cluster utilization like keeping all resources in use all the time against various constraints such as capacity guarantees, fairness, and SLAs.
- It has two major components: a) Scheduler b) Application Manager

a) Scheduler

- The scheduler is responsible for allocating resources to the various running applications subject to constraints of capacities, queues etc.
- It is called a pure scheduler in ResourceManager, which means that it does not perform any monitoring or tracking of status for the applications.
- If there is an application failure or hardware failure, the Scheduler does not guarantee to restart the failed tasks.
- Performs scheduling based on the resource requirements of the applications.
- It has a pluggable policy plug-in, which is responsible for partitioning the cluster resources among the various applications. There are two such plug-ins: **Capacity Scheduler** and **Fair Scheduler**, which are currently used as Schedulers in ResourceManager.

b) Application Manager

- It is responsible for accepting job submissions.
- Negotiates the first container from the Resource Manager for executing the application specific Application Master.
- Manages running the Application Masters in a cluster and provides service for restarting the Application Master container on failure.

Node Manager

- It takes care of individual nodes in a Hadoop cluster and manages user jobs and workflow on the given node.
- It registers with the Resource Manager and sends heartbeats with the health status of the node.
- Its primary goal is to manage application containers assigned to it by the resource manager.
- It keeps up-to-date with the Resource Manager.
- Application Master requests the assigned container from the Node Manager by sending it a Container Launch Context(CLC) which includes everything the application needs in order to run. The Node Manager creates the requested container process and starts it.
- Monitors resource usage (memory, CPU) of individual containers.
- Performs Log management.
- It also kills the container as directed by the Resource Manager.

Application Master

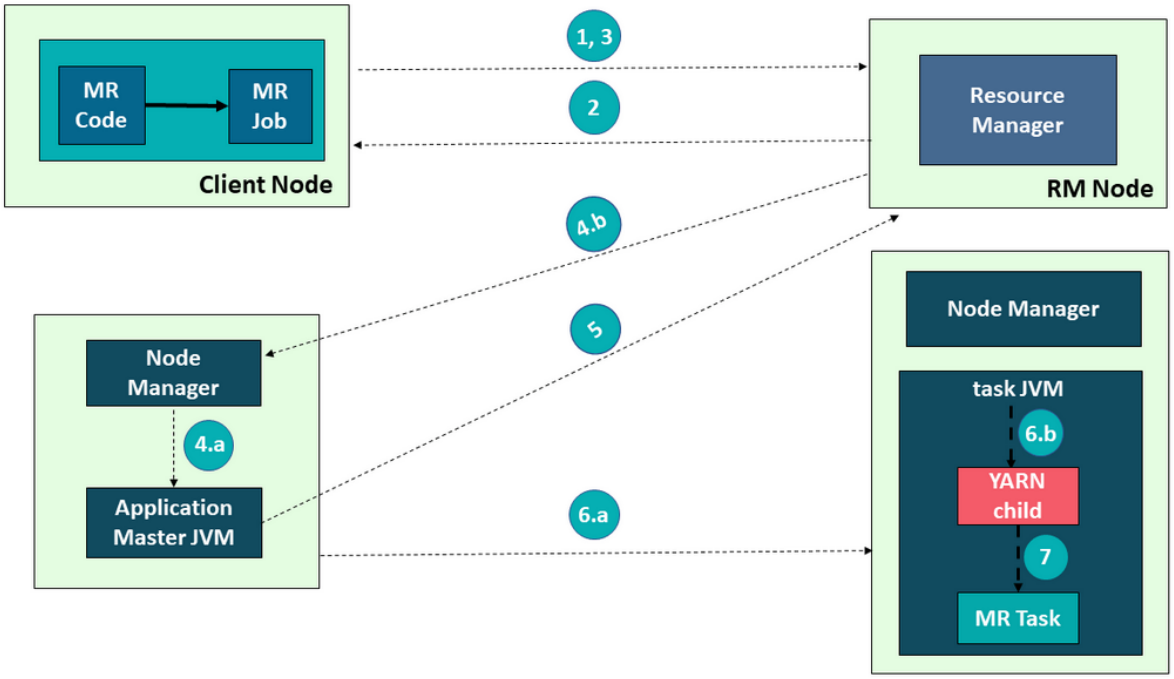
- An application is a single job submitted to the framework. Each such application has a unique Application Master associated with it which is a framework specific entity.
- It is the process that coordinates an application's execution in the cluster and also manages faults.
- Its task is to negotiate resources from the Resource Manager and work with the Node Manager to execute and monitor the component tasks.
- It is responsible for negotiating appropriate resource containers from the ResourceManager, tracking their status and monitoring progress.
- Once started, it periodically sends heartbeats to the Resource Manager to affirm its health and to update the record of its resource demands.

Container

- It is a collection of physical resources such as RAM, CPU cores, and disks on a single node.
- YARN containers are managed by a container launch context which is container life-cycle(CLC). This record contains a map of environment variables, dependencies stored in a remotely accessible storage, security tokens, payload for Node Manager services and the command necessary to create the process.
- It grants rights to an application to use a specific amount of resources (memory, CPU etc.) on a specific host.

Application Workflow in Hadoop YARN

Refer to the given image and see the following steps involved in Application workflow of Apache Hadoop YARN:



1. Client submits an application
2. Resource Manager allocates a container to start Application Manager
3. Application Manager registers with Resource Manager
4. Application Manager asks containers from Resource Manager
5. Application Manager notifies Node Manager to launch containers
6. Application code is executed in the container
7. Client contacts Resource Manager/Application Manager to monitor application's status
8. Application Manager unregisters with Resource Manager