HADOOP 2.X COMPONENTS

➢ Hadoop 2.x Core Components

• Entire infrastructure is divided into 2 parts - storage and processing
• Hadoop Distributed File System (HDFS) provides the storage mechanism.
Yet Another Resource Negotiator (YARN) provides the processing part.
• The HDFS provides storage mechanism using certain software daemons which work in a master and slave fashion
• There are two philosophies on which hadoop works - all of its basic software's are software daemons, and each of these daemons work in a master and slaves fashion.
• In total 5 Hadoop Daemons - 3 for HDFS and 2 for YARN. They work in a master and slave mode. Name Node, Secondary Name Node (Masters) and Data Nodes (Slaves) for HDFS and Resource Manager (Master) and Node Managers (Slaves) for YARN. Blocks
• Its the physical division of data file done by HDFS while storing the data.
• 128 MB of blocks size by default for Hadoop 2.x Computer Racks
• A computer rack is a physical chassis that can house multiple computers or servers simultaneously. It is a mounting rack that has the ability to install more than one computer and connects those computers using a network switch.

1) HDFS - Hadoop Distributed File System

HDFS is a distributed file system which stores the data in distributed manner. Rather than storing a complete file it divides a file into small blocks of 128 MB size by default. And distributes them across the cluster. Each blocks is replicated 3 times as per default configuration multiple times and is stored on different nodes to ensure data availability.
HDFS in Hadoop 2.x mainly has 3 daemons which are Name Node, Secondary Name Node and Data Node.

a) Name Node

Name node metadata

• **FSImage**

- **The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in FSImage at the name node. This is saved on the hard disk of the machine on admin machine on which Name Node daemon is running. (FS v1.0, FS v2.0 (FS v1.0+editLogs).....)**

• **EditLogs**

- **The change to the above FS images gets saved inside the transaction log called edit logs.**

• **The above two logs are together called Name node Meta data.**

• **The Namenode Daemon saves this above Meta data in the RAM of admin machine while the cluster is up and running.**

• **As soon as the cluster comes down, the edit logs get merged with FS Image.**

• **On a fresh restart, the Meta data gets loaded from the FS Image and the changes again start getting updated in edit logs.**

Name node Meta Data Operations / Working

• **Caters to the clients requests.(like Read / Write)**

• **Name node holds the entire Meta Data in its main memory (i.e. RAM) in the running cluster**

• **Manages the above Meta Data**

- **Creation, Deletion etc. of files and directories (also known as meta data operations)**

• **This is called as Namespace Management**

• **Manages Block Storage Management for slaves machines**

- **This includes heart beat management between Data Node and Name node**

- **Providing support for block replication, Block reporting etc.**

• **This is called as Block Management.**

b) Secondary Name Node

- For this also, only single instance of this process runs on a cluster
- This process can run on a master node (for smaller clusters) or can run on a separate node (in larger clusters) depends on the size of the cluster
- One misinterpretation from name is “This is a backup Name Node” but IT IS NOT!!!!!
- It manages the metadata for the Name Node. In the sense, it reads the information written in edit logs (by Name Node) and creates an updated file of current cluster metadata
- Then it transfers that file back to Name Node so that FSImage file can be updated
- So, whenever Name Node daemon is restarted it can always find updated information in FSImage file

c) Data Node

- There are many instances of this process running on various slave nodes(referred as Data nodes)
- It is responsible for storing the individual file blocks on the slave nodes in Hadoop cluster
- Based on the replication factor, a single block is replicated in multiple slave nodes(only if replication factor is > 1) to prevent the data loss
- Whenever required, this process handles the access to a data block by communicating with Name Node
- This process periodically sends heart bits to Name Node to make Name Node aware that slave process is running

2) YARN

- **MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. It Uses 3 YARN components: Resource Manager and Node Manager. These two components are responsible for executing distributed data computation jobs in Hadoop 2**
  • YARN, to address the overloading of Job Trackers in Hadoop 1.x. also divided the job tracker responsibilities of Hadoop 1.x into the following separate components :-

Resource Manager

- This is the processing master daemon with which clients interact by way of submitting the jobs.
- It is a daemon which gets initialized by the Hadoop Admins.
- Manages the entire processing part of a submitted Hadoop 2.x job.
- Listens to the heart beats from Node Manager Slave Daemons.
- Spawns 2 objects - Scheduler and Application Manager

Scheduler

- A scheduler is responsible for allocating resources to various running applications.
- Its a daemon under the umbrella of Resource Manager running on admin machines, therefore a single instance of scheduler will be there.
- It does so by allocating Containers on slave machines, which are nothing but temporary JVM instances and other resources.
- Application Masters negotiate resources / containers from them.

Container

- An abstract notion for resources on the YARN platform given by schedulers.
- It includes resources like CPU cores, disk space and RAM which are using by JVM instances running the analytics.
- Temporary existence hence not a daemon.
- They are supposed to run a single unit of the job (Mapper or Reducer) that is submitted by the client. A job will have multiple units ((Mapper or Reducer))
- Resource of the containers are released as soon as the tasks and job finishes.
- Everything which is not a daemon in YARN needs a container to execute itself.
- Uses java run time to instantiate and execute itself with the assigned resources (when told by Node Managers.)

Application Master / one per job

- Manages each instance of the application / job submitted. Representation of an instance of the job.
- Negotiates resources (i.e. containers) from the Schedulers to execute the job
- Temporary existence. Needs a container to run itself. Exits as soon as the job gets over.
- Execute tasks given to them and sends the latest updates on the task progress the application manager.
- Application Master will be initialized at one of the slaves.
- Application Master also will be initialized by the Node Manager.

Node Manager

- It a slave daemon running on slave machines.
- Sends heart beats to Resource Manager telling them that their overall status and well being.
- Provides per node services on the cluster
- It allocates / instantiates containers once it hears from the Application Managers (or Application Masters), oversees them to monitor the resources assigned while the job executes.
- Does not need a container to run itself at its a daemon and started at the time of configuring the cluster.

YARN Job Flow

- User Submit the job
- The job comes to Resource Manager
- Resource Manager has two components running
- Scheduler (Allocation Resources)
- Application Manager (Instantiates Application Masters by getting in touch with the Node Managers of the slave reference it has received from the scheduler)
- Application Manager will tell the scheduler that it needs a container / resources for instantiating an Application Master
- Scheduler will give reference of the slave (running a node manager) on which a container is available which can run the Application Master for the job.
- Application Manager will get in touch with the Node Manager of the slave reference received from Scheduler and tell it to instantiate the Container to run the Application Master.
- The Node Manager of the slave will instantiate the Container for Application Master.
- The Node Manager will instantiate the Application Master (representation of the job instance) inside that container.
- Because it has to run the job now, the Application Master will now negotiate resources (i.e. Containers) from Scheduler to run its tasks (because the job submitted will get splitter into tasks)
- The scheduler will allocate the Containers to the Application Master for the job.
- Application Master will get in touch with the Node Managers of all the containers assigned by the Scheduler to the Application Master for the job.
- All the respective Node Managers will instantiate the Containers and start executing the tasks. (Many such Node Managers).
- The containers will start giving the status updates to their respective Application Masters.
- The Application Master will give the job progress updates to the Application Manager.
- Application Manager will give status to the client.
- Application Manager will keep managing other jobs in the similar manner.