Hadoop 1.x Architecture and its Daemons:

Hadoop 1.x Architecture is a history now because most of the Hadoop applications are using **Hadoop 2.x Architecture**. But still understanding of Hadoop 1.x Architecture will provide us the insights of how hadoop has evolved over the time.

Hadoop 1.x Major Components: HDFS and MapReduce.

1) HDFS:

HDFS is a Hadoop Distributed FileSystem, where our BigData is stored using Commodity Hardware. It is designed to work with Large DataSets with default block size of 64MB.

a) Name Node:

- There is a single instance of this process which runs on a cluster and that is on a master node.
- It used to store Metadata about DataNode like "How many blocks are stored in Data Nodes, Which DataNodes have data, DataNode Details, locations, timestamps etc."
- This process reads all the metadata from a file named fsimage and keeps it in memory. After this process is started, it updates metadata for newly added or removed files in RAM.
- It periodically writes the changes in one file called edits as edit logs. This process is heart of HDFS, if it is down HDFS is not accessible any more.

b) Secondary Name Node:

- For this also 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's NOT.
- It manages the metadata for the NameNode. 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 NameNode daemon is restarted it can always find updated information in fsimage file.

c) DataNode:

- There are many instances of this process running on various slave nodes.
- It is responsible for storing 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 data loss.
- This process periodically sends heart beats to NameNode to make NameNode aware that slave process is up and running.

2) MapReduce:

MapReduce is a Distributed and Batch Processing Programming Model. MapReduce also uses Commodity Hardware to process "High Volume of Variety of Data at High Velocity" in a reliable and fault-tolerant manner.

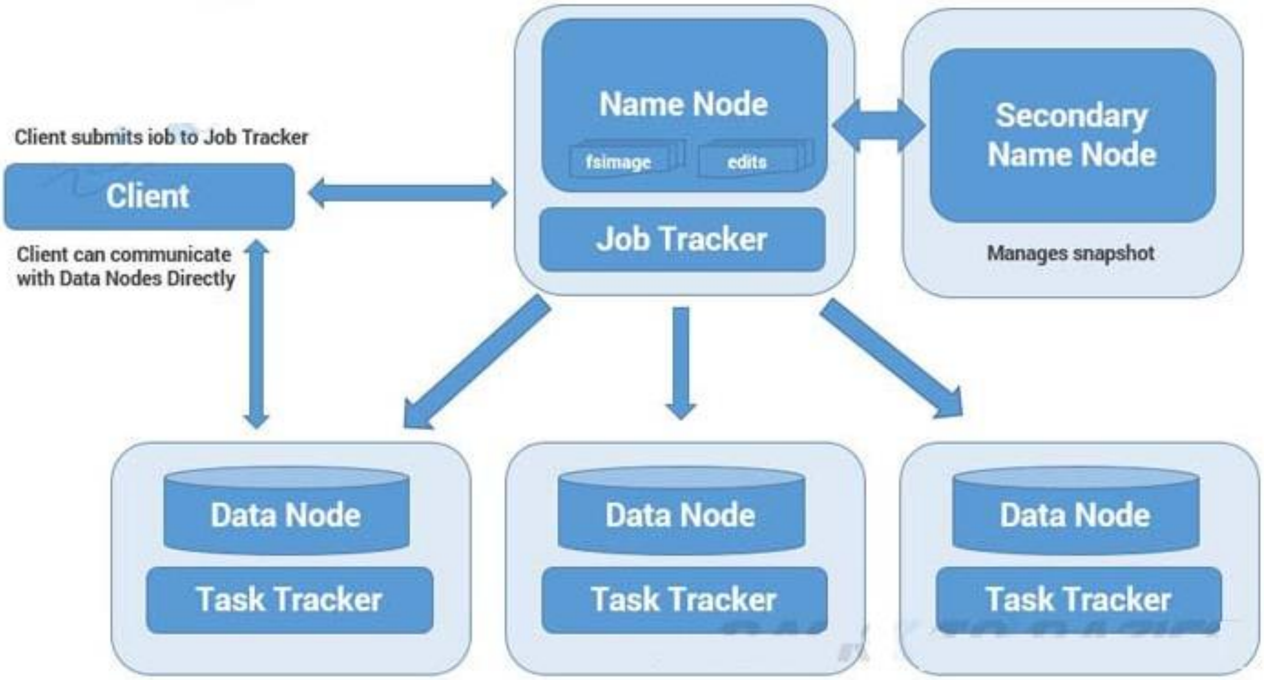
MapReduce component is again divided into two sub-components:

a) Job Tracker:

- Only one instance of this process runs on a master node.
- Any MapReduce job is submitted to Job Tracker first.
- Job Tracker checks for the location of various file blocks used in MapReduce processing.
- Then it initiates the separate tasks on various DataNodes where blocks are present by communicating with Task Tracker Daemons.

b) Task Tracker:

- This process has multiple instances running on the slave nodes.
- It receives the job information from Job Tracker and initiates a task on that slave node.
- In most of the cases, Task Tracker initiates the task on the same node where the physical data block is present.
- Same as DataNode daemon, this process also periodically sends heart beats to Job Tracker to make Job Tracker aware that slave process is running.



Hadoop 1.x Architecture in Detail

Limitations of Hadoop 1.x Architecture:

- Single Point of Failure as there is no backup NameNode.
- Job scheduling, Resource Management and Job Monitoring are being done by Job Tracker which is tightly coupled with Hadoop. So Job Tracker is not able to manage resources outside Hadoop.
- Job Tracker has to coordinate with all task tracker, so in a very big cluster it will be difficult to manage huge number of task trackers altogether.
- Due to single NameNode, there is no concept of namespaces in Hadoop 1.x. So everything is managed under single namespace.
- Using Hadoop 1.x architecture, Hadoop Cluster can be scaled upto ~4000 nodes only. Scalability beyond that may cause performance degradation and increasing task failure ratio.

Understanding Hadoop 2.x Architecture and its Daemons:

- Hadoop 2.x has some common and new APIs which can be easily integrated with any third party applications to work with Hadoop.
- It has new Java APIs and features in HDFS and MapReduce which are known as HDFS2 and MR2 respectively.
- New architecture has added the architectural features like HDFS High Availability and HDFS Federation.
- Hadoop 2.x is not using Job Tracker and Task Tracker daemons for resource management now on-wards but it is using YARN (Yet Another Resource Negotiator) for Resource Management.

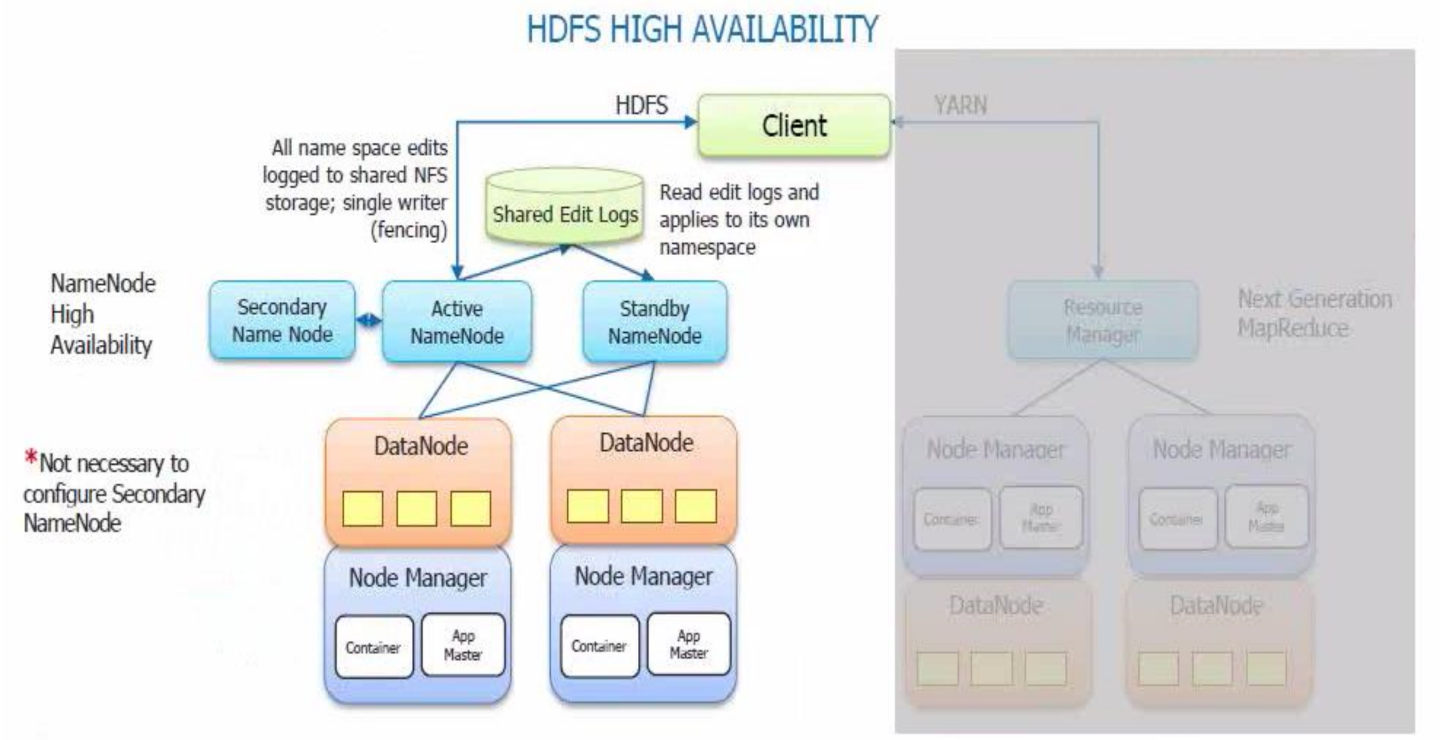
1) HDFS High Availability (HA):

Problem: As you know in Hadoop 1.x Architecture NameNode was a single point of failure, which means if your NameNode is down, you don't have access to your Hadoop Cluster. How to deal with this problem?

Solution: Hadoop 2.x is featured with NameNode HA which is referred as HDFS High Availability (HA).

- Hadoop 2.x support two NameNodes at a time one node is Active and another is Standby node.
- Active NameNode handles the client operation in the cluster.
- StandBy NameNode manages metadata at the same time via Shared Edit Logs and keeps in-sync with Active NameNode.
- When Active NameNode is down, Standby NameNode takes over and handles the client operation then after.
- HDFS HA can be configured by two ways:

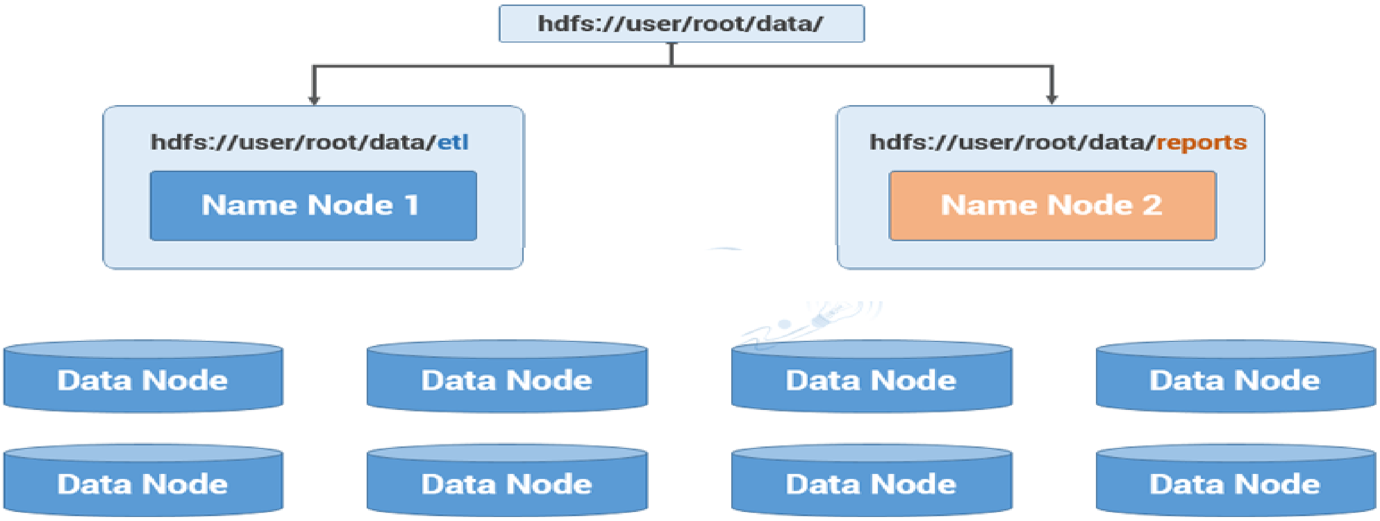
a) Using Shared NFS Directory
b) Using Quorum Journal Manager


HDFS HIGH AVAILABILITY
HDFS
Client
YARN
All name space edits logged to shared NFS storage; single writer (fencing)
Shared Edit Logs
Read edit logs and applies to its own namespace
NameNode High Availability
Secondary Name Node
Active NameNode
Standby NameNode
Resource Manager
Next Generation MapReduce
DataNode
DataNode
Node Manager
Node Manager
Container
App Master
*Not necessary to configure Secondary NameNode
DataNode
DataNode

2) HDFS Federation:

Problem: HDFS uses namespace for managing directories, file and block level information in cluster. Hadoop 1.x architecture was able to manage only single namespace in a whole cluster with the help of the NameNode (which is a single point of failure in Hadoop 1.x). Once that Name Node is down you loose access of full cluster data. It was not possible for partial data availability based on name space.

Solution: Above problem is solved by HDFS Federation. Hadoop 2.x Architecture allows to manage multiple namespaces by enabling multiple NameNodes.



HDFS Federation

Hadoop 2.x Architecture in Detail:

Hadoop2 Architecture has mainly 2 set of daemons:

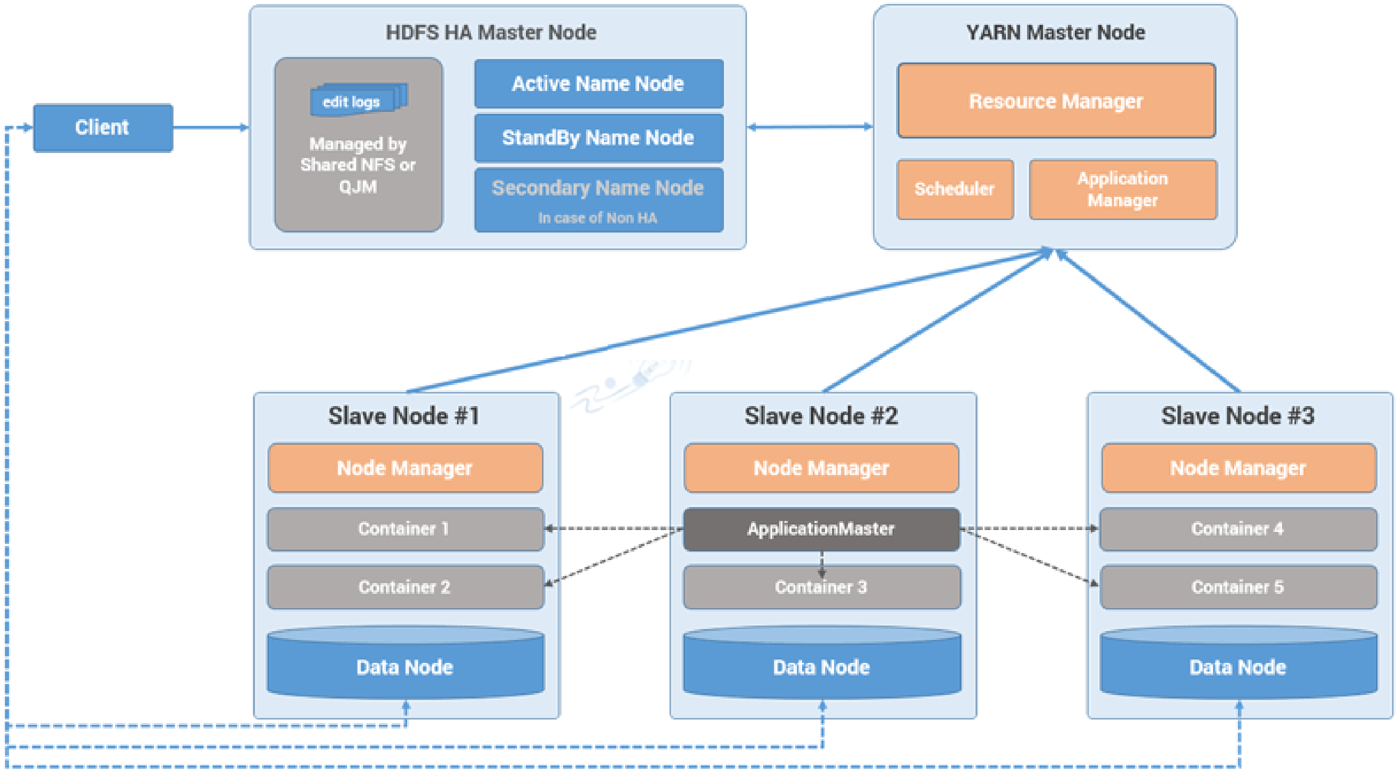
1) HDFS 2.x Daemons: NameNode, Secondary NameNode (not required in HA) and Data Nodes.

2) MapReduce 2.x Daemons (YARN): Resource Manager and Node Manager.

HDFS 2.x Daemons:

The working methodology of HDFS 2.x daemons is same as it was in Hadoop 1.x but with following differences.

- Hadoop 2.x allows Multiple Name Nodes for HDFS Federation.
- Hadoop 2.x allows HDFS High Availability in which it can have Active and StandBy Name Nodes (No Need of Secondary Name Node in HA).



Hadoop 2.x Architecture

MapReduce 2.x Daemons (YARN):
MapReduce2 has replaced Job Tracker and Task Tracker with YARN components Resource Manager and Node Manager respectively.

a) Resource Manager:

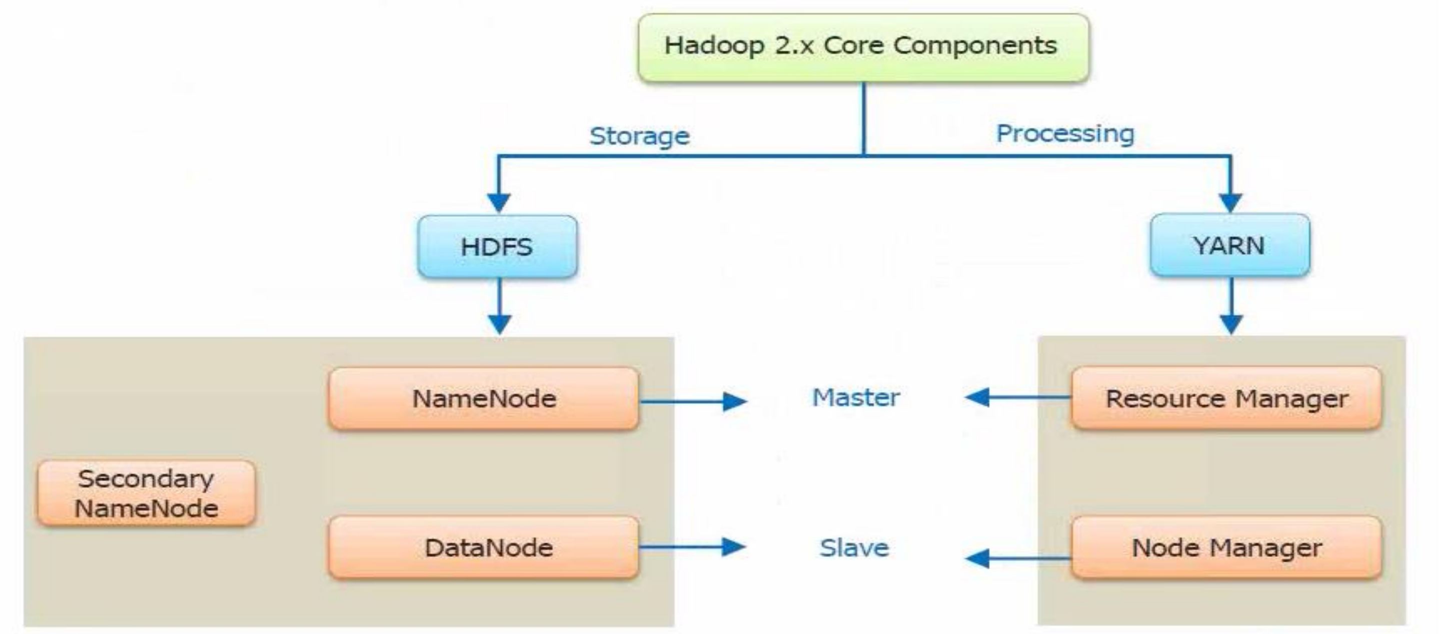
- This has a single instance and the process runs on master node.
- It is responsible for getting job submitted from client and schedule it on cluster, monitor running jobs on cluster and allocating proper resources on the slave node.
- It communicates with Node Manager on the slave node to track the resource utilization.
- It uses two other processes named *Application Manager* and *Scheduler* for MapReduce Task and Resource Management.

b) Node Manager:

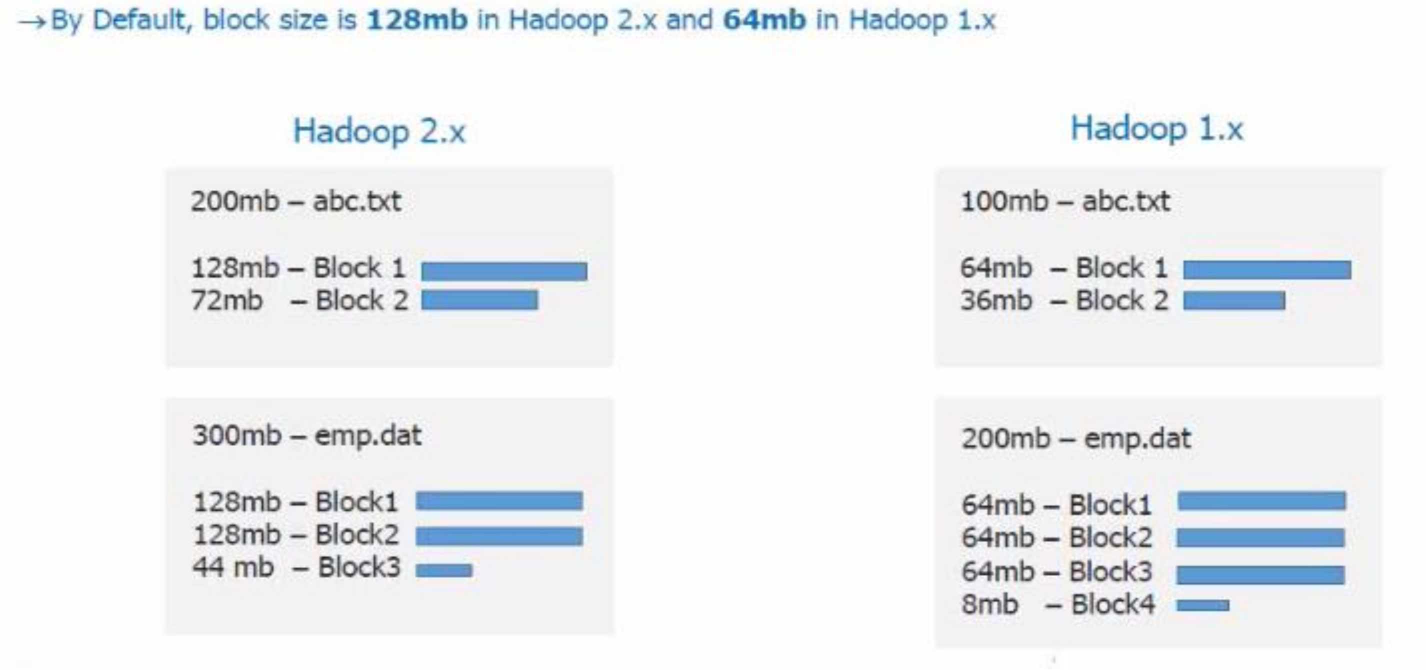
- This has multiple instances and the process runs on the slave nodes.
- It is responsible for coordinating with Resource Manager for task scheduling and tracking the resource utilization on the slave nodes.
- It also reports the resource utilization back to the Resource Manager.
- It uses two other processes named *Application Master* and *Container* for MapReduce Task Scheduling and Execution on the slave nodes.

Hadoop 2.x Core Components:

Storage: Identify the machine where data needs to be stored & Store the data.
Processing: Identify the machine where processing needs to be done & actual execution.



Blocks Size in Hadoop 1.x and 2.x:



→By Default, block size is **128mb** in Hadoop 2.x and **64mb** in Hadoop 1.x

Hadoop 2.x

200mb – abc.txt

128mb – Block 1
72mb – Block 2

300mb – emp.dat

128mb – Block1
128mb – Block2
44 mb – Block3

Hadoop 1.x

100mb – abc.txt

64mb – Block 1
36mb – Block 2

200mb – emp.dat

64mb – Block1
64mb – Block2
64mb – Block3
8mb – Block4

Why is HDFS Block size 128 MB in Hadoop 2.x?

- HDFS have huge data, i.e. Terabytes and Petabytes of data.
- So like Linux file system which have 4 KB block size and if we have block size 4KB for HDFS, then we would be having too many data blocks in HDFS and therefore too much of metadata.
- So managing this huge number of blocks and metadata will create huge overhead and traffic which we don't want.
- On the other hand, data block size can't be so large that the system is waiting a very long time for one last unit of data processing to finish its work.

Replication Management:

- Block replication provides fault tolerance. If one copy is not accessible and corrupted, we can read data from other copy.
- The default replication factor is 3 which is configurable. So, each block replicates three times and stored on different DataNodes.
- If we are storing a file of 128 MB in HDFS using the default configuration, we will end up occupying a space of 384 MB i.e. (3*128 MB).
- NameNode receives block report from DataNode periodically to maintain the replication factor.
- When a block is over-replicated/under-replicated the NameNode add or delete replicas as needed.

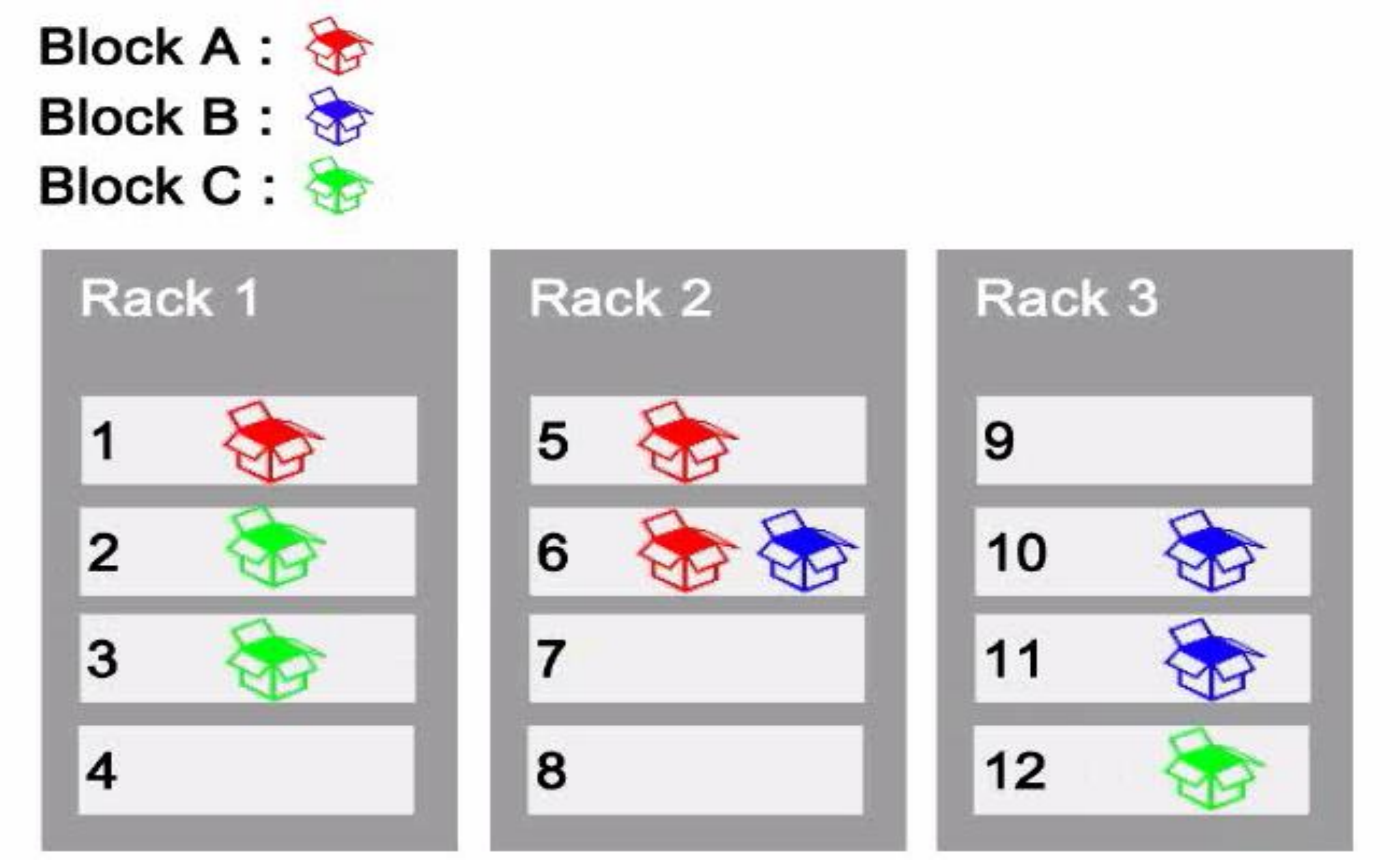
Rack Awareness:

- In a large cluster of Hadoop, in order to improve the network traffic while reading/writing HDFS file, NameNode chooses the DataNode which is closer to the same rack or nearby rack for read/write request.
- NameNode achieves rack information by maintaining the rack ids of each DataNode.
- Rack Awareness in Hadoop is the concept that chooses DataNode based on the rack information.
- NameNode makes sure that all the replicas are not stored on the same rack or single rack.
- It follows Rack Awareness Algorithm to reduce latency as well as fault tolerance.
- We know that default replication factor is 3 according to Rack Awareness.

Algorithm first replica of a block will store on a local rack. The next replica will store another block within the same rack. The third replica will store block on different rack.

Rack Awareness is important to improve:

- High Availability and Reliability.
- Performance of the cluster.
- Improve Network Bandwidth.


Block A :
Block B :
Block C :
Rack 1
1
2
3
4
Rack 2
5
6
7
8
Rack 3
9
10
11
12