UNIT-III INTRODUCTION TO BIG DATA AND HADOOP

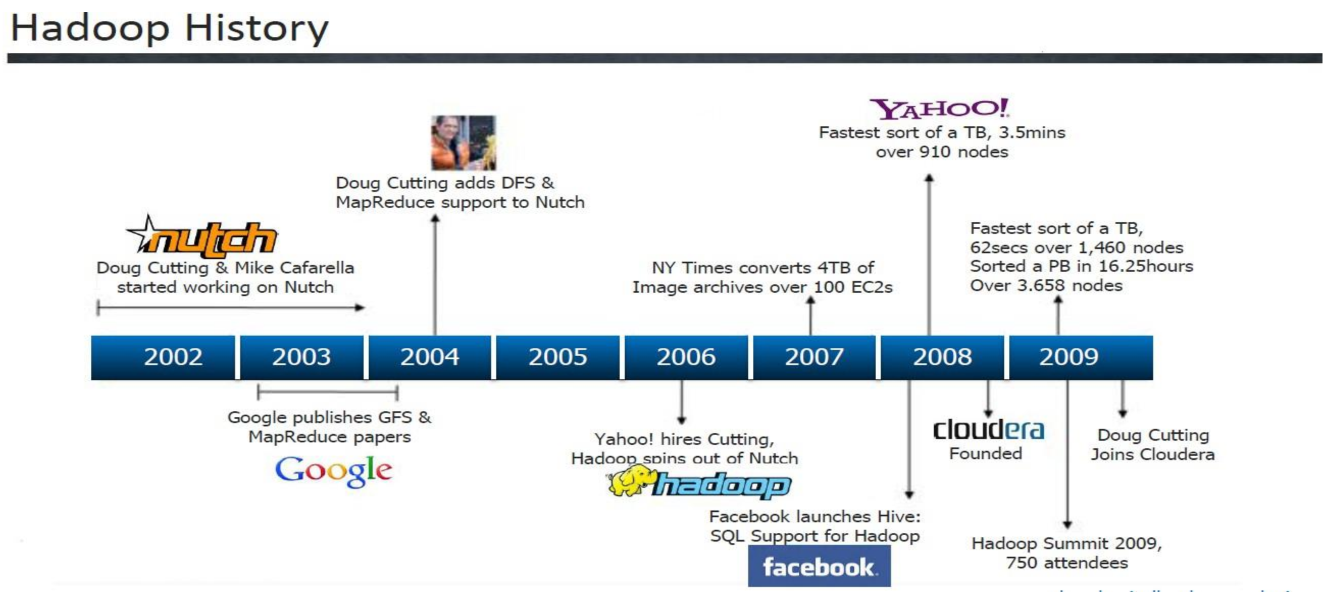
Introduction to Big Data, Types of Digital Data, Challenges of conventional systems - Web data, Evolution of analytic processes and tools, Analysis Vs reporting - Big Data Analytics, Introduction to Hadoop - Distributed Computing Challenges - History of Hadoop, Hadoop Eco System - Use case of Hadoop – Hadoop Distributors – HDFS – Processing Data with Hadoop – Map Reduce.

Introduction:

- Hadoop is an open-source software framework used for storing and processing Big Data in a distributed manner on large clusters of commodity hardware. Hadoop is licensed under Apache Software Foundation (ASF).
- Hadoop is written in the Java programming language and ranks among the highest-level Apache projects.
- **Doug Cutting** and **Mike J. Cafarella** developed Hadoop.
- By getting inspiration from **Google**, Hadoop is using technologies like **Map-Reduce** programming model as well as Google file system (**GFS**).
- It is optimized to handle massive quantities of data that could be structured, unstructured or semi-structured, using commodity hardware, that is, relatively inexpensive computers.
- It is intended to work upon from a single server to thousands of machines each offering local computation and storage. It supports the large collection of data set in a distributed computing environment.

History:

Hadoop came into existence and why it is so popular in the industry nowadays. So, it all started with two people, Mike Cafarella and Doug Cutting, who were in the process of building a search engine system that can index 1 billion pages. After their research, they estimated that such a system will cost around half a million dollars in hardware, with a monthly running cost of $30,000, which is quite expensive. However, they soon realized that their architecture would not be capable enough to work around with billions of pages on the web.

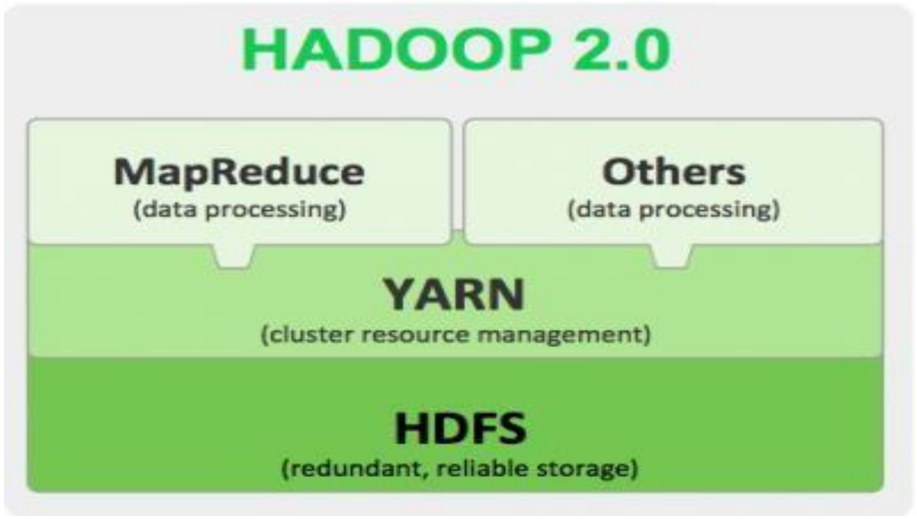

Hadoop History
Doug Cutting & Mike Cafarella started working on Nutch
Doug Cutting adds DFS & MapReduce support to Nutch
Google publishes GFS & MapReduce papers
Yahoo! hires Cutting, Hadoop spins out of Nutch
NY Times converts 4TB of Image archives over 100 EC2s
Facebook launches Hive: SQL Support for Hadoop
Fastest sort of a TB, 3.5mins over 910 nodes
Fastest sort of a TB, 62secs over 1,460 nodes Sorted a PB in 16.25hours Over 3.658 nodes
cloudera Founded
Doug Cutting Joins Cloudera
Hadoop Summit 2009, 750 attendees
2002
2003
2004
2005
2006
2007
2008
2009

Doug and Mike came across a paper, published in 2003, that described the architecture of Google's distributed file system, called GFS, which was being used in production at Google. Now, this paper on GFS proved to be something that they were looking for, and soon, they realized that it would solve all their problems of storing very large files that are generated as a part of the web crawl and indexing process. Later in 2004, Google published one more paper that introduced MapReduce to the world. Finally, these two papers led to the foundation of the framework called "***Hadoop***". **Doug Cutting, Mike Cafarella** and team took the solution provided by Google and started an Open Source Project called HADOOP in 2005 and Doug named it after his son's toy elephant. Now Apache Hadoop is a registered trademark of the Apache Software Foundation.

Hadoop Architecture:

- Apache Hadoop offers a scalable, flexible and reliable distributed computing big data framework for a cluster of systems with storage capacity and local computing power by leveraging commodity hardware.
- Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. In short, Hadoop framework is capable enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for huge amounts of data.
- Hadoop follows a Master Slave architecture for the transformation and analysis of large datasets using Hadoop MapReduce paradigm.
- The 3 **important Hadoop** ***core components*** that play a vital role in the Hadoop architecture are -

1. Hadoop Distributed File System (HDFS)
2. Hadoop MapReduce
3. Yet Another Resource Negotiator (YARN)



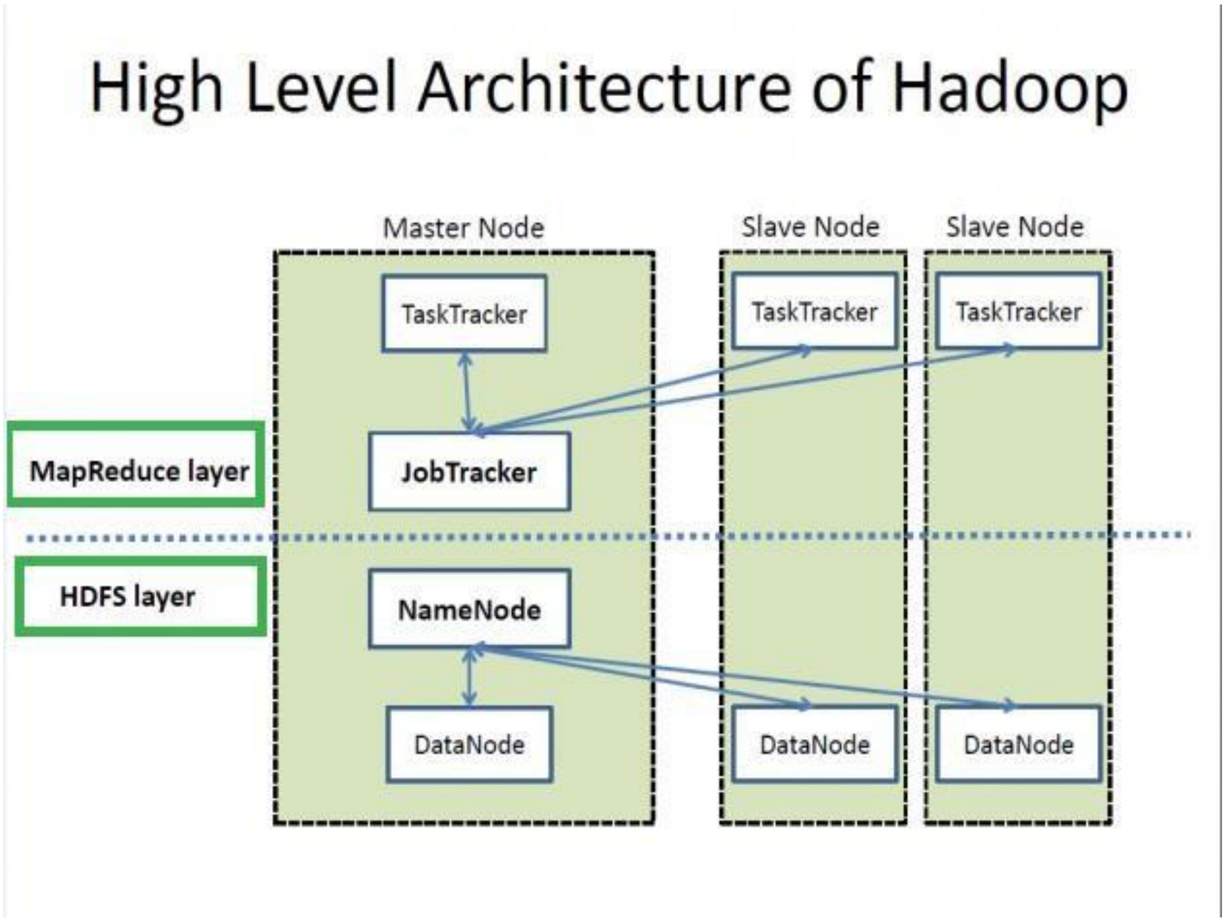
- **Hadoop Distributed File System (HDFS):**
  - Hadoop Distributed File System runs on top of the existing file systems on each node in a Hadoop cluster.
  - Hadoop Distributed File System is a block-structured file system where each file is divided into blocks of a pre-determined size.
  - Data in a Hadoop cluster is broken down into smaller units (called blocks) and distributed throughout the cluster. Each block is duplicated twice (for a total of three copies), with the two replicas stored on two nodes in a rack somewhere else in the cluster.
  - Since the data has a default replication factor of three, it is highly available and fault-tolerant.
  - If a copy is lost (because of machine failure, for example), HDFS will automatically re-replicate it elsewhere in the cluster, ensuring that the threefold replication factor is maintained.

- **Hadoop MapReduce:** This is for parallel processing of large data sets.
  - The MapReduce framework consists of a single **master node** (**JobTracker)** and n numbers of **slave nodes** (**Task Tracker**) where n can be 1000s. Master manages, maintains and monitors the slaves while slaves are the actual worker nodes.
  - Client submit a job to Hadoop. The job can be a mapper, a reducer or a list of input. The job is sent to job tracker process on master node. Each slave node runs a process through task tracker.

- The master 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.
- The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically.
- Master stores the metadata (data about data) while slaves are the nodes which store the data. The client connects with master node to perform any task.

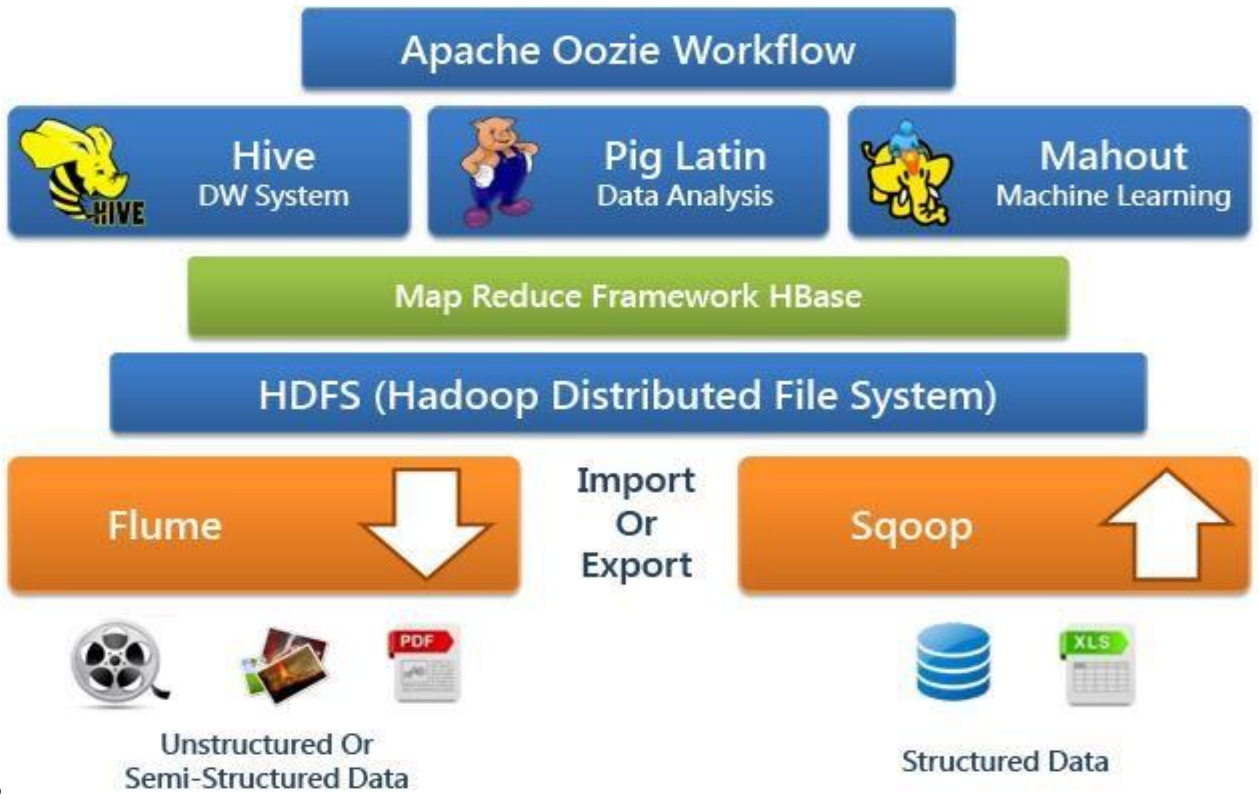
➢ **Hadoop YARN:** YARN (Yet Another Resource Negotiator) is the framework responsible for assigning computational resources for application execution and cluster management. YARN consists of three core components:

- ResourceManager (one per cluster)
- ApplicationMaster (one per application)
- NodeManagers (one per node)



Hadoop ecosystem:

- Hadoop Ecosystem is neither a programming language nor a service; it is a platform or framework which solves big data problems. You can consider it as a suite which encompasses a number of services (ingesting, storing, analyzing and maintaining) inside it. Let us discuss and get a brief idea about how the services work individually and in collaboration.
- The Hadoop ecosystem provides the furnishings that turn the framework into a comfortable home for big data activity that reflects your specific needs and tastes.
- The Hadoop ecosystem includes both official Apache open source projects and a wide range of commercial tools and solutions.
- Below are the Hadoop components, that together form a Hadoop ecosystem,
  - ✓ **HDFS** -> *Hadoop Distributed File System*
  - ✓ **YARN** -> *Yet Another Resource Negotiator*
  - ✓ **MapReduce** -> *Data processing using programming*
  - ✓ **Spark** -> In-memory Data Processing
  - ✓ **PIG, HIVE**-> *Data Processing Services using Query (SQL-like)*
  - ✓ **HBase** -> *NoSQL Database*
  - ✓ **Mahout, Spark MLlib** -> *Machine Learning*
  - ✓ **Apache Drill** -> *SQL on Hadoop*
  - ✓ **Zookeeper** -> *Managing Cluster*
  - ✓ **Oozie** -> *Job Scheduling*
  - ✓ **Flume, Sqoop** -> *Data Ingesting Services*
  - ✓ **Solr&Lucene** -> *Searching & Indexing*



s

Apache open source Hadoop ecosystem elements:

Spark, Pig, and Hive are three of the best-known Apache Hadoop projects. Each is used to create applications to process Hadoop data.

- **Spark:** Apache Spark is a framework for real time data analytics in a distributed computing environment. It executes in-memory computations to increase speed of data processing over Map-Reduce.
- **Hive:** Facebook created HIVE for people who are fluent with SQL. Basically, HIVE is a data warehousing component which performs reading, writing and managing large data sets in a distributed environment using SQL-like interface. The query language of Hive is called Hive Query Language (HQL), which is very similar like SQL. ***HIVE + SQL = HQL.*** It provides tools for ETL operations and brings some SQL-like capabilities to the environment.
- **Pig:** Pig is a procedural language for developing parallel processing applications for large data sets in the Hadoop environment. Pig is an alternative to Java programming for MapReduce, and automatically

generates MapReduce functions. Pig includes Pig Latin, which is a scripting language. Pig translates Pig Latin scripts into MapReduce, which can then run on YARN and process data in the HDFS cluster.

- **HBase:** HBase is a scalable, distributed, NoSQL database that sits atop the HFDS. It was designed to store structured data in tables that could have billions of rows and millions of columns. It has been deployed to power historical searches through large data sets, especially when the desired data is contained within a large amount of unimportant or irrelevant data (also known as sparse data sets).
- **Oozie:** Oozie is the workflow scheduler that was developed as part of the Apache Hadoop project. It manages how workflows start and execute, and also controls the execution path. Oozie is a server-based Java web application that uses workflow definitions written in hPDL, which is an XML Process Definition Language similar to JBOSS JBPM jPDL.
- **Sqoop:** Sqoop is bi-directional data injection tool. Think of Sqoop as a front-end loader for big data. Sqoop is a command-line interface that facilitates moving bulk data from Hadoop into relational databases and other structured data stores. Using Sqoop replaces the need to develop scripts to export and import data. One common use case is to move data from an enterprise data warehouse to a Hadoop cluster for ETL processing. Performing ETL on the commodity Hadoop cluster is resource efficient, while Sqoop provides a practical transfer method.

- **Ambari** – A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig, and Sqoop.
- **Flume** – A distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming event data.
- **Mahout** – A scalable machine learning and data mining library.
- **Zookeeper** – A high-performance coordination service for distributed applications.

The ecosystem elements described above are all open source Apache Hadoop projects. There are numerous commercial solutions that use or support the open source Hadoop projects.

Hadoop Distributions:

- ➢ Hadoop is an open-source, catch-all technology solution with incredible scalability, low cost storage systems and fast paced big data analytics with economical server costs.
- ➢ Hadoop Vendor distributions overcome the drawbacks and issues with the open source edition of Hadoop. These distributions have added functionalities that focus on:

- **Support:**
  Most of the Hadoop vendors provide technical guidance and assistance that makes it easy for customers to adopt Hadoop for enterprise level tasks and mission critical applications.
- **Reliability:**
  Hadoop vendors promptly act in response whenever a bug is detected. With the intent to make commercial solutions more stable, patches and fixes are deployed immediately.
- **Completeness:**
  Hadoop vendors couple their distributions with various other add-on tools which help customers customize the Hadoop application to address their specific tasks.
  - **Fault Tolerant:**
    Since the data has a default replication factor of three, it is highly available and fault-tolerant.

Here is a list of top Hadoop Vendors who play a key role in big data market growth

- ➢ Amazon Elastic MapReduce
- ➢ Cloudera CDH Hadoop Distribution
- ➢ Hortonworks Data Platform (HDP)
- ➢ MapR Hadoop Distribution
- ➢ IBM Open Platform (IBM Infosphere Big insights)
- ➢ Microsoft Azure's HDInsight -Cloud based Hadoop Distribution

Advantages of Hadoop:

The increase in the requirement of computing resources has made Hadoop a viable and extensively used programming framework. Modern day organizations can learn Hadoop and leverage their knowhow of managing processing power of their businesses.

1. Scalable: Hadoop is a highly scalable storage platform, because it can stores and distribute very large data sets across hundreds of inexpensive servers that operate in parallel. Unlike traditional relational database systems (RDBMS) that can’t scale to process large amounts of data, Hadoop enables businesses to run applications on thousands of nodes involving many thousands of terabytes of data.

2. Cost effective: Hadoop also offers a cost effective storage solution for businesses' exploding data sets. The problem with traditional relational database management systems is that it is extremely cost prohibitive to scale to such a degree in order to process such massive volumes of data. In an effort to reduce costs, many companies in the past would have had to down-sample data and classify it based on certain assumptions as to which data was the most valuable. The raw data would be deleted, as it would be too cost-prohibitive to keep. While this approach may have worked in the short term, this meant that when business priorities changed, the complete raw data set was not available, as it was too expensive to store.

3. Flexible: Hadoop enables businesses to easily access new data sources and tap into different types of data (both structured and unstructured) to generate value from that data. This means businesses can use Hadoop to derive valuable business insights from data sources such as social media, email conversations. Hadoop can be used for a wide variety of purposes, such as log processing, recommendation systems, data warehousing, market campaign analysis and fraud detection.

4. Speed of Processing: Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If you're dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes, and petabytes in hours.

5. Resilient to failure: A key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, that data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use.

Summary of UNIT – I:

- **Part – I - BigData**
    1. Data, Information, Knowledge application on data
    2. What is Bigdata, examples of Bigdata (TBs &PBs of data)
    3. Sources of Bigdata, Facts about Bigdata (ex: Aircrafts, social Media, tweets, e-commerce)
    4. Types of Bigdata (structured, unstructured, semi-structured)
    5. 3 Vs of Bigdata (properties of bigdata – volume, velocity, variety)
    6. Data Analytics, Types of Data Analytics
    7. Need of Data Analytics

- **Part – II - Hadoop**
    1. What is Hadoop
    2. History of Hadoop (Doug Cutting -2005)
    3. Evolution of Hadoop
    4. Basic Architecture of Hadoop (High level architecture also)
    5. Hadoop eco-system (various tools- HDFS, Mapreduce, spark, Hbase, pig, Hive, Zoo Keeper, Sqoop, Flume, Oozie etc).
    6. Hadoop distributions
    7. Benefits of Hadoop

Activities:

8. Sketch Noting
9. One minute Chat
10. Quescussion
11. Presentations
12. Question/Answers

Unit - II

Hadoop Distributed File System (HDFS): Introduction, Design Principles, Components of HDFS – Name Node, Secondary Name Node, Data Nodes, HDFS File Blocks, Storing File Blocks into HDFS from Client Machine, Rack Awareness, HDFS File Commands.

Map Reduce: Introduction, Architectural Components of Map Reduce, Functional Components of Map Reduce, Heartbeat Signal, Speculative Execution

HDFS:

- **Hadoop Distributed file system** is a Java based distributed file system that allows you to store large data across multiple nodes in a Hadoop cluster. So, if you install Hadoop, you get HDFS as an underlying storage system for storing the data in the distributed environment.
- Hadoop File System was developed using distributed file system design. It runs on commodity hardware. 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 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 also makes applications available to parallel processing.
- HDFS runs on top of the existing file systems on each node in a Hadoop cluster. It is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines.

Features of HDFS:

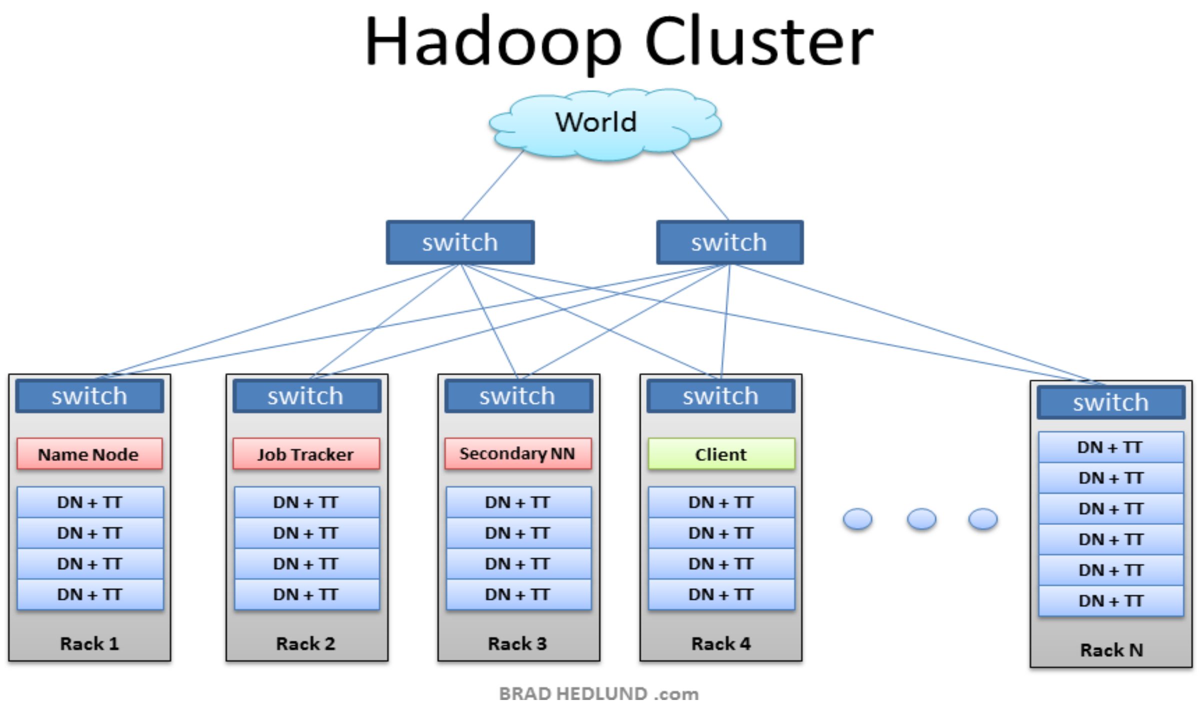
- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

Basic terminology:

Node: A node is simply a computer. This is typically non-enterprise, commodity hardware for nodes that contain data.

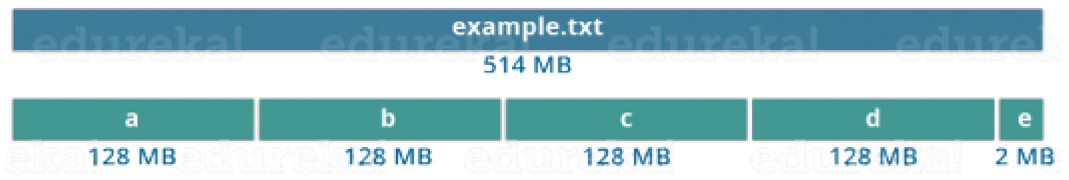
Rack: Collection of nodes is called as a rack. A rack is a collection of 30 or 40 nodes that are physically stored close together and are all connected to the same network switch.
Network bandwidth between any two nodes in the same rack is greater than bandwidth between two nodes on different racks.

Cluster: A Hadoop Cluster (or just cluster from now on) is a collection of racks.



File Blocks:

Blocks are 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. Similarly, HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster. The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement. All blocks of the file are the same size except the last block, which can be either the same size or smaller. The files are split into 128 MB blocks and then stored into the Hadoop file system. The Hadoop application is responsible for distributing the data block across multiple nodes.



Let's take an example where we have a file "example.txt" of size 514 MB as shown in above figure. Suppose that we are using the default configuration of block size, which is 128 MB. Then, 5 blocks will be created. The first four blocks will be of 128 MB. But, the last block will be of 2 MB size only.

Components of HDFS:

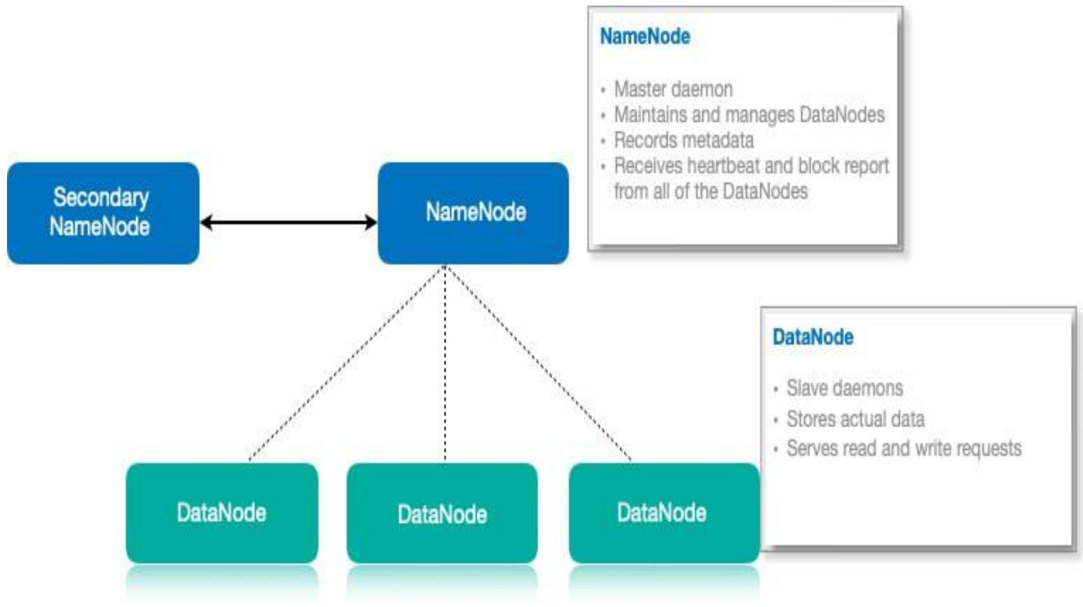
HDFS is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single **NameNode (Master node)** and all the other nodes are **DataNodes (Slave nodes).** HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

NameNode:

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. The HDFS architecture is built in such a way that the user data never resides on the NameNode. Name node contains metadata and the data resides on DataNodes only.

Functions of NameNode:

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- Manages the file system namespace.
- 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 records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- 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.
- In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

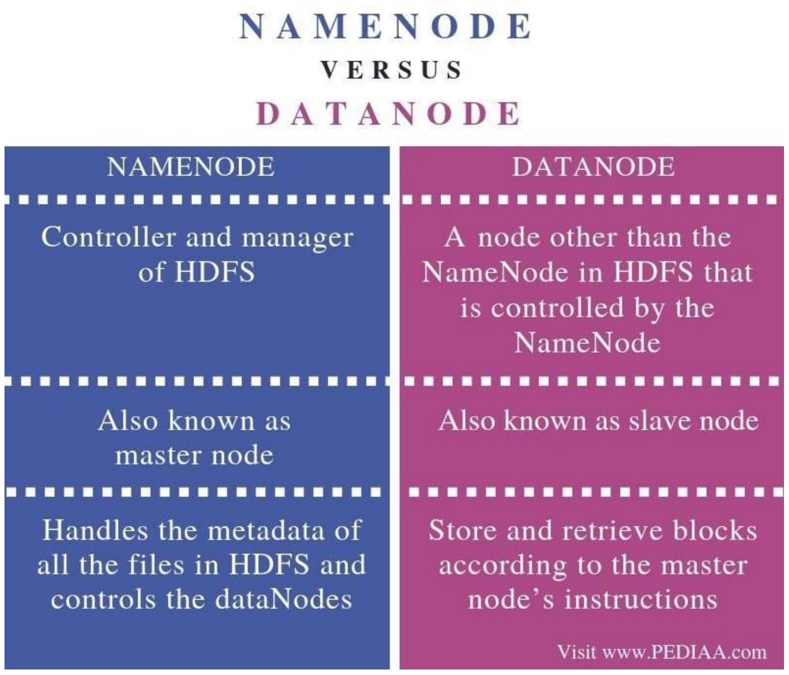


DataNode:

Data Nodes are the slave nodes in HDFS. Unlike NameNode, DataNode is commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. The Data Node is a block server that stores the data in the local file ext3 or ext4.

Functions of DataNode:

- The actual data is stored on DataNodes.
- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default; this frequency is set to 3 seconds.



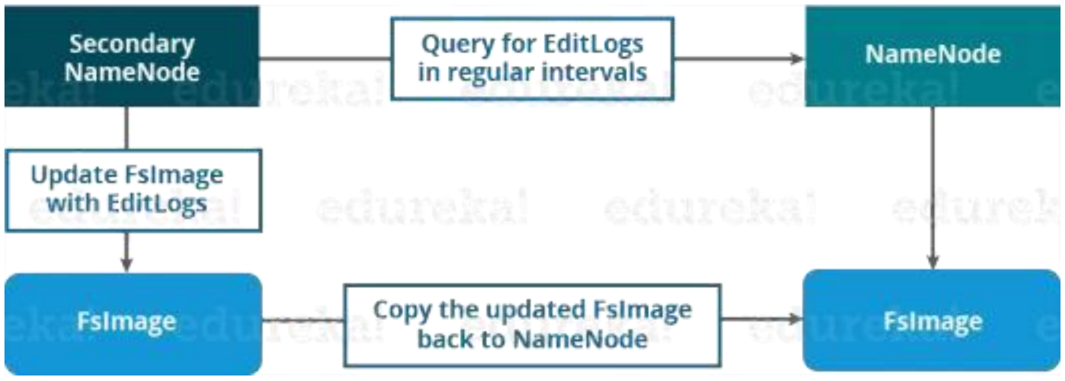
NAMENODE VERSUS DATANODE

NAMENODE	DATANODE
Controller and manager of HDFS	A node other than the NameNode in HDFS that is controlled by the NameNode
Also known as master node	Also known as slave node
Handles the metadata of all the files in HDFS and controls the dataNodes	Store and retrieve blocks according to the master node's instructions

Visit www.PEDIAA.com

Secondary NameNode:

It is a separate physical machine which acts as a helper of name node. It performs periodic check points. It communicates with the name node and take snapshot of meta data which helps minimize downtime and loss of data. The Secondary NameNode works concurrently with the primary NameNode as a **helper daemon.**

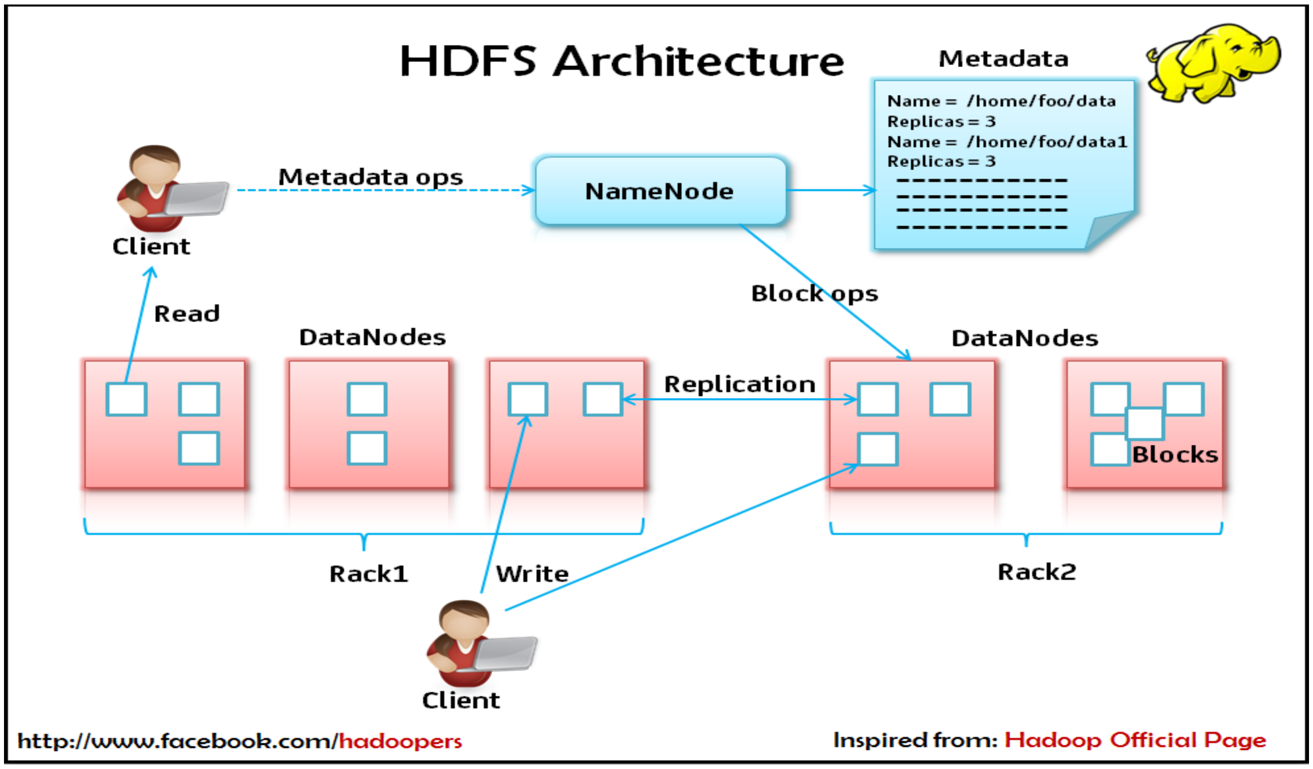

Secondary NameNode
Query for EditLogs in regular intervals
NameNode
Update FsImage with EditLogs
FsImage
Copy the updated FsImage back to NameNode
FsImage

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.

HDFS Architecture: Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.



Storing data into HDFS:

HDFS stores data in a reliable fashion using **replication and distribution**. Here is the series of steps that happen when a client writes a file in hdfs:

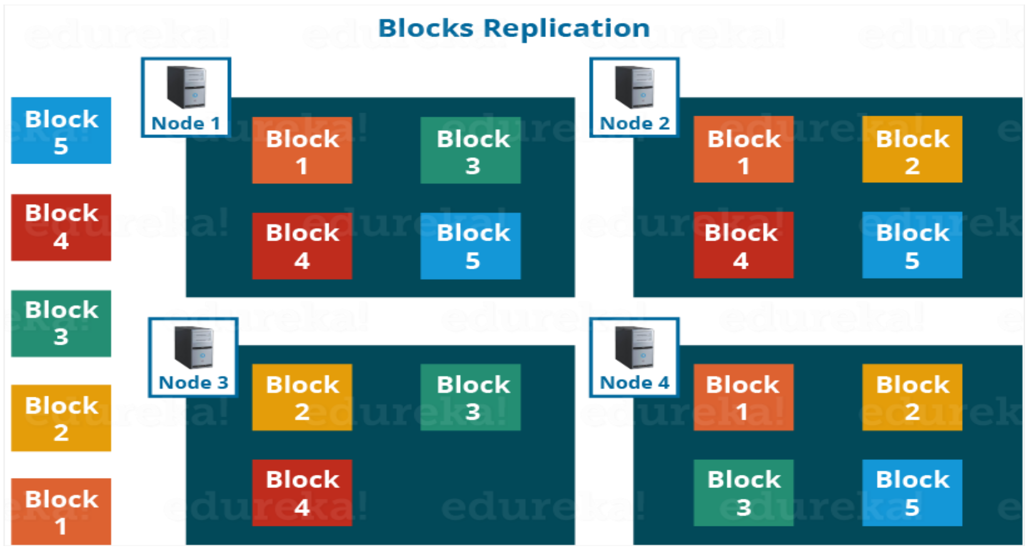
1. Client requests Namenode to create the file. It passes size of file as a parameter
2. Namenode responds with location of nodes where client can store data. By default there'll be 3 locations per block. If file size is 200mb, there'll be 2 blocks, first 128 mb, 2nd 72 mb. Similarly depending on the size, you'll have n number of blocks.
3. Client directly starts writing data to the first datanode out of three given by namenode. Please note that if there are 2 blocks to be written client can start writing them in parallel.
4. When the first datanode has stored the block, it replies to the client with success and now it passes on the same block to 2nd datanode. 2nd datanode will write this block and pass it on to 3rd datanode.
5. So basically writing of blocks from client to datanodes happens in parallel but replication happens in series.

Blocks of same file can go to different nodes, at least the replicated blocks will always be on different nodes. The first block is always on the datanode which is nearest to the client, 2nd and 3rd blocks are stored based on free capacity of the datanodes and/or rack awareness.

What is Replication Management?

- HDFS performs replication to provide Fault Tolerant and to improve data reliability.
- There could be situations where the data is lost in many ways- node is down, Node lost the network connectivity, a node is physically damaged, and a node is intentionally made unavailable for horizontal scaling.
- For any of the above-mentioned reasons, data will not be available if the replication is not made. HDFS usually maintains 3 copies of each Data Block in different nodes and different Racks. By doing this, data is made available even if one of the systems is down.
- Downtime will be reduced by making data replications. This improves the reliability and makes HDFs fault tolerant.
- Block replication provides fault tolerance. If one copy is not accessible and corrupted, we can read data from other copy.

- The number of copies or replicas of each block of a file in HDFS Architecture is replication factor. ***The default replication factor is 3*** which are again configurable. So, each block replicates three times and stored on different DataNodes.
- So, as you can see in the figure below where each block is replicated three times and stored on different DataNodes (considering the default replication factor): 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 (3*128 MB).



Rack Awareness in HDFS Architecture:

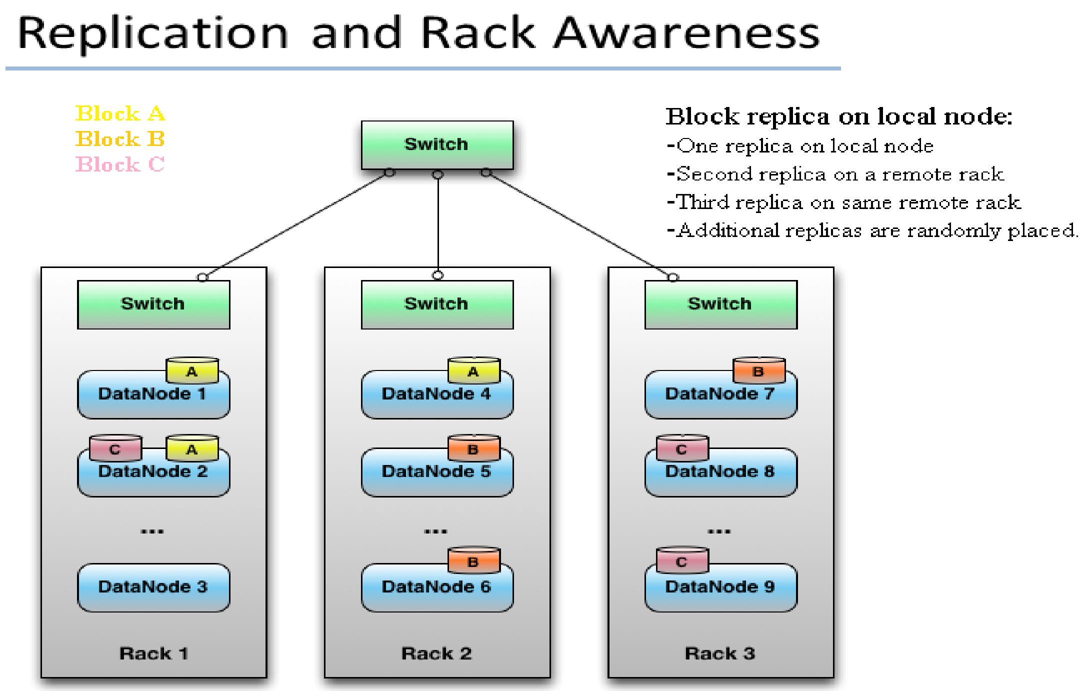
- **Rack**- It the collection of machines around 30-40. All these machines are connected using the same network switch and if that network goes down then all machines in that rack will be out of service. Thus we say rack is down.
- Rack Awareness was introduced by Apache Hadoop to overcome this issue. **Rack awareness** is the knowledge that how the data nodes are distributed across the rack of Hadoop cluster.
- In the 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 to Read /write request. NameNode achieves rack information by maintaining the rack ids of each DataNode. This concept that chooses Datanodes based on the rack information is called **Rack Awareness** in Hadoop.

- In HDFS, 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.
- As we know the default **Replication Factor** is **3** and Client want to place a file in HDFS, then Hadoop places the replicas as follows:

  1) The first replica is written to the data node creating the file, to improve the write performance because of the write affinity.

  2) The second replica is written to another data node within the same rack, to minimize the cross-rack network traffic.

  3) The third replica is written to a data node in a different rack, ensuring that even if a switch or rack fails, the data is not lost (Rack awareness).
- This configuration is maintained to make sure that the File is never lost in case of a Node Failure or even an entire Rack Failure.


Replication and Rack Awareness
Block A
Block B
Block C
Switch
Block replica on local node:
-One replica on local node
-Second replica on a remote rack
-Third replica on same remote rack
-Additional replicas are randomly placed.
Switch
Switch
Switch
A
DataNode 1
C
A
DataNode 2
...
DataNode 3
Rack 1
A
DataNode 4
B
DataNode 5
...
B
DataNode 6
Rack 2
B
DataNode 7
C
DataNode 8
...
C
DataNode 9
Rack 3

Advantages of Rack Awareness:

- Minimize the writing cost and Maximize read speed – Rack awareness places read/write requests to replicas on the same or nearby rack. Thus minimizing writing cost and maximizing reading speed.

- Provide maximize network bandwidth and low latency – Rack awareness maximizes network bandwidth by blocks transfer within a rack. This is particularly beneficial in cases where tasks cannot be assigned to nodes where their data is stored locally.
- Data protection against rack failure – By default, the namenode assigns 2nd & 3rd replicas of a block to nodes in a rack different from the first replica. This provides data protection even against rack failure

MapReduce:

MapReduce is a programming framework that allows us to perform parallel and distributed processing on huge data sets in distributed environment.

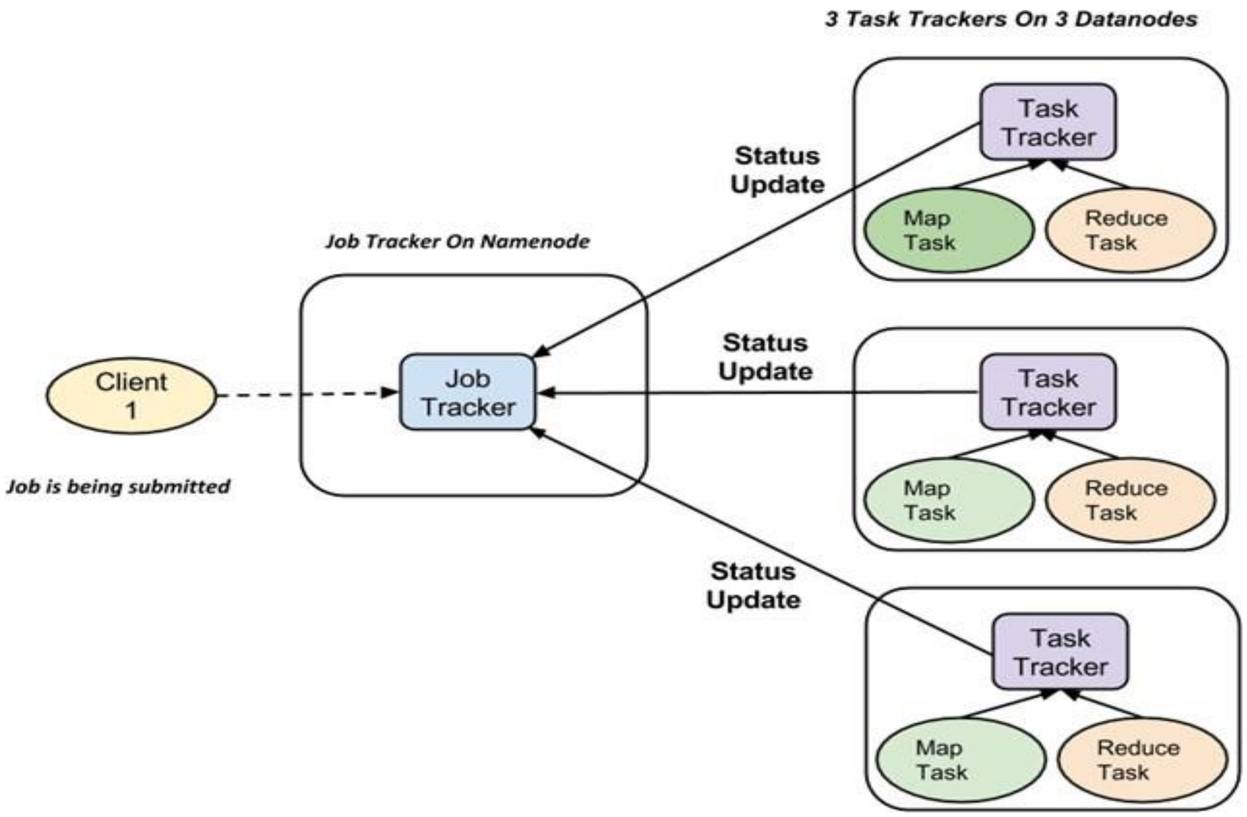
Map Reduce can be implemented by its components:

1. **Architectural components**
   a) Job Tracker
   b) Task Trackers
2. **Functional components**
   a) Mapper (map)
   b) Combiner (Shuffler)
   c) Reducer (Reduce)

Architectural Components:

- The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called:
  1. A Job tracker : Acts like a master (responsible for complete execution of submitted job)
  2. Multiple Task Trackers : Acts like slaves, each of them performing the job
- For every job submitted for execution in the system, there is one Job tracker that resides on Name node and there are multiple task trackers which reside on Data node.

- A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
- It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
- Execution of individual task is then look after by task tracker, which resides on every data node executing part of the job.
- Task tracker's responsibility is to send the progress report to the job tracker.
- In addition, task tracker periodically sends 'heartbeat' signal to the Job tracker so as to notify him of current state of the system.
- Thus job tracker keeps track of overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.



Functional Components:

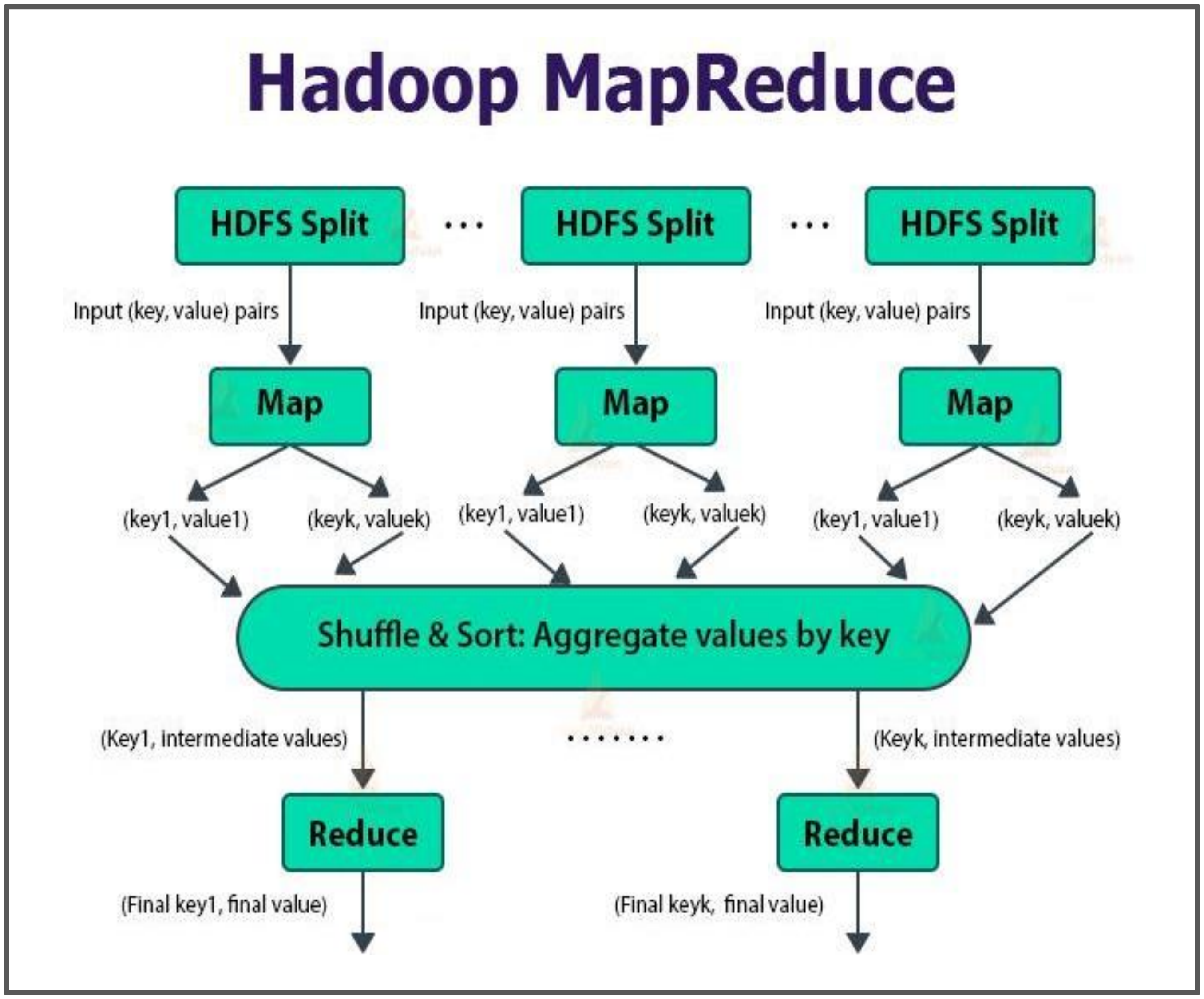
A job (complete Work) is submitted to the master, Hadoop divides the job into phases , map phase and reduce phase. In between Map and Reduce, there is small phase called shuffle & Sort in MapReduce.

1. Map tasks
2. Reduce tasks

Map Phase: This is very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. The map takes key/value pair as input. Key is a reference to the input. Value is the data set on which to operate. Map function applies the business logic to every value in input. Map produces an output is called intermediate output. An output of map is stored on the local disk from where it is shuffled to reduce nodes.

Reduce Phase: In MapReduce **Reduce** takes intermediate Key / Value pairs as input and process the output of the mapper. Key/value pairs provided to reduce are sorted by key. Usually, in the reducer, we do aggregation or summation sort of computation. A function defined by user supplies the values for a given key to the Reduce function. Reduce produces a final output as a list of key/value pairs. This final output is stored in HDFS and replication is done as usual.

Shuffling: This phase consumes output of mapping phase. Its task is to consolidate the relevant records from Mapping phase output.

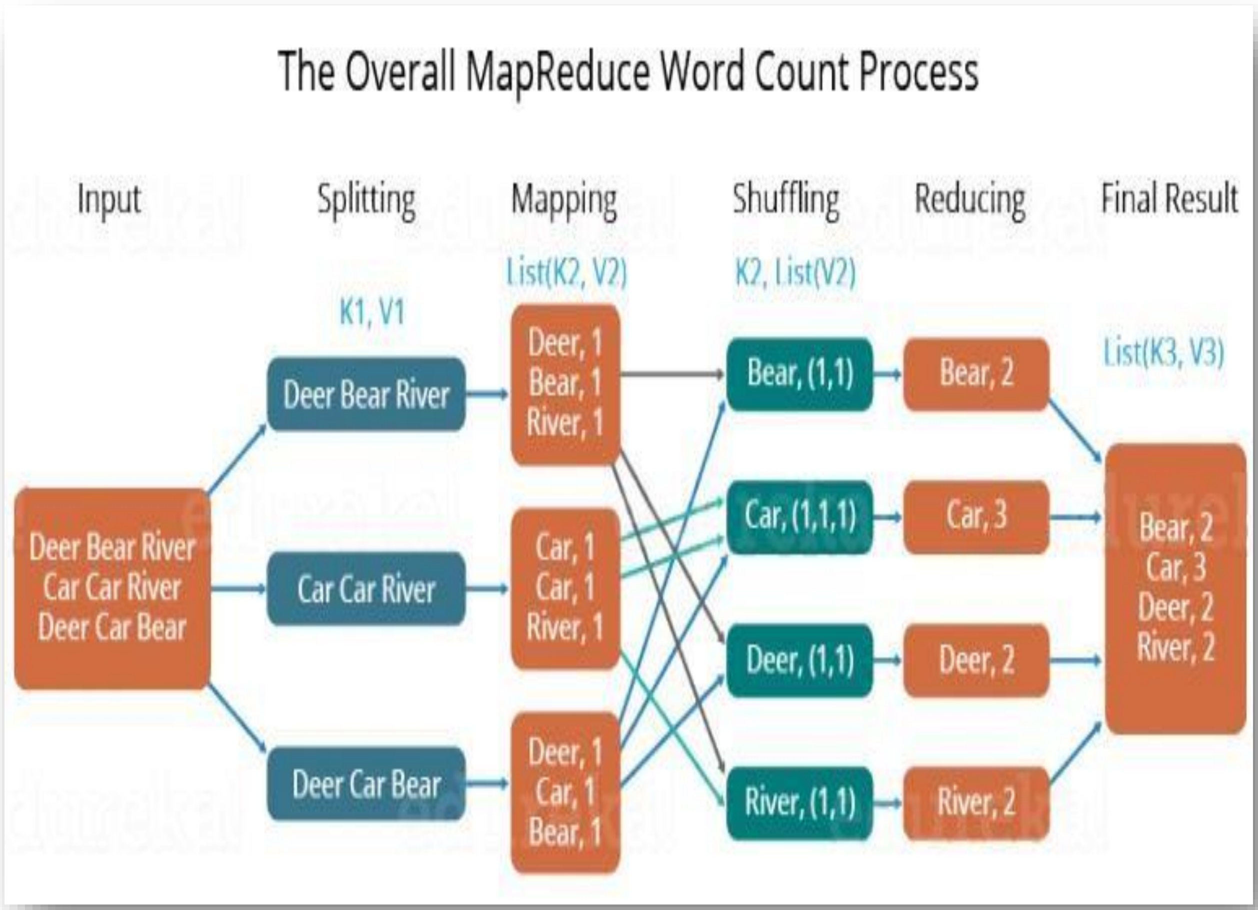


A Word Count Example of MapReduce:

Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we need to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input in three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mapper and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, will occur once.
- Now, a list of key-value pair will be created where the key is the individual word and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.

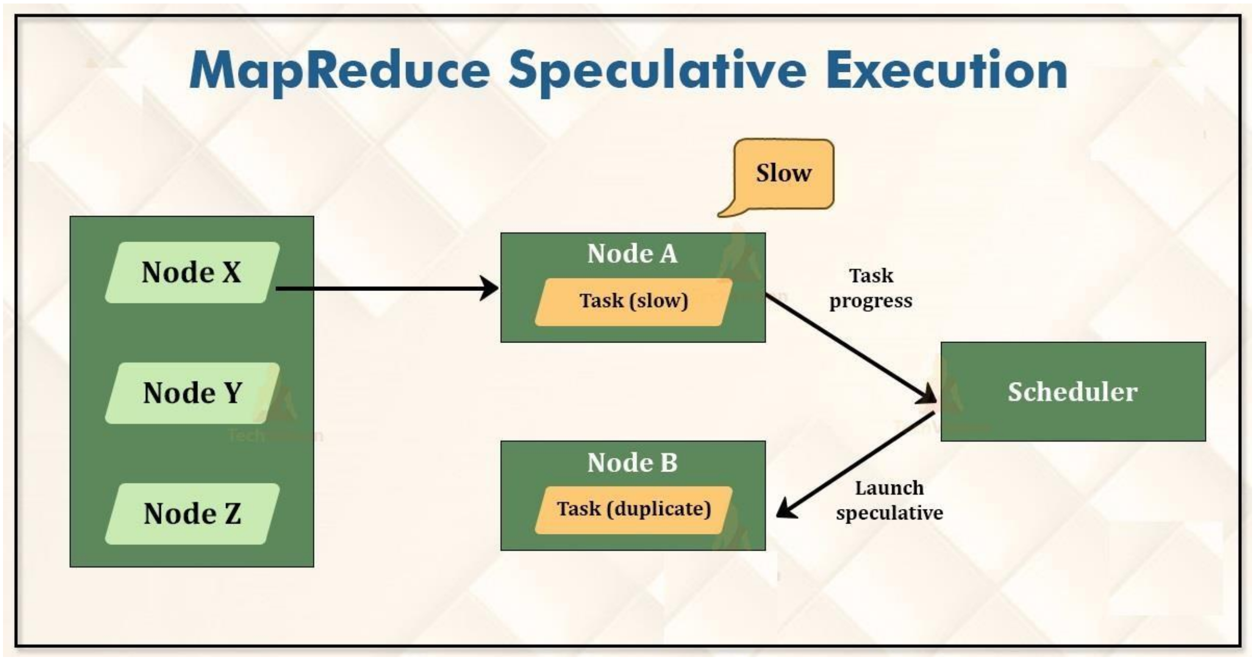
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Heartbeat Signal:

- ➢ HDFS follows a master slave architecture. Namenode (master) stores metadata about the data and Datanodes store/process the actual data (and its replications).
- ➢ Now the namenode should know if any datanode in a cluster is down (power failure/network failure) otherwise it will continue assigning tasks or sending data/replications to that dead datanode.
- ➢ **Heartbeat** is a mechanism for *detecting datanode failure* and ensuring that the link between datanodes and namenode is intact. In Hadoop , Name node and data node do communicate using **Heartbeat**. Therefore, Heartbeat is the signal that is sent by the datanode to the namenode after the regular interval of time to indicate its presence, i.e. to indicate that it is available.
- ➢ **The default heartbeat interval is 3 seconds**. If the DataNode in HDFS does not send heartbeat to NameNode in ten minutes, then NameNode considers the DataNode to be out of service and the Blocks replicas hosted by that DataNode to be unavailable.
- ➢ Hence once the heartbeat stops sending a signal to NameNode, then NameNode perform certain tasks such as replicating the blocks present in DataNode to other DataNodes to make the data is highly available and ensuring **data reliability**.
- ➢ NameNode that receives the Heartbeats from a DataNode also carries information like total storage capacity, the fraction of storage in use, and the number of data transfers currently in progress. For the NameNode's block allocation and load balancing decisions, we use these statistics.

Speculative Execution

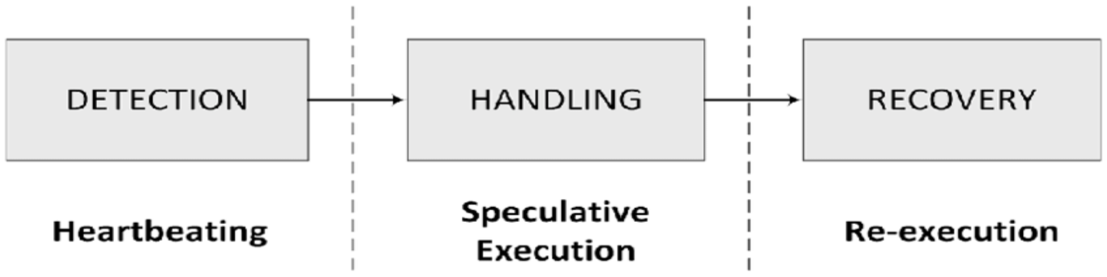
- In Hadoop, **MapReduce** breaks jobs into tasks and these tasks run parallel rather than sequential, thus reduces overall execution time. This model of execution is sensitive to slow tasks (even if they are few in numbers) as they slow down the overall execution of a job.
- There may be various reasons for the slowdown of tasks, including hardware degradation or software misconfiguration, but it may be difficult to detect causes since the tasks still complete successfully, although more time is taken than the expected time.
- The Hadoop framework does not try to diagnose or fix the slow-running tasks. The framework tries to detect the task which is running slower than the expected speed and launches another task, which is an equivalent task as a backup. The backup task is known as the speculative task, and this process is known as speculative execution in Hadoop.



- As the name suggests, Hadoop tries to speculate the slow running tasks, and runs the same tasks in the other nodes parallel. Whichever task is completed first, that output is considered for proceeding further, and the slow running tasks are killed.
- Firstly, all the tasks for the job are launched in Hadoop MapReduce. The speculative tasks are launched for those tasks that have been running for some time (at least one minute) and have

not made any much progress, on average, as compared with other tasks from the job. The speculative task is killed if the original task completes before the speculative task, on the other hand, the original task is killed if the speculative task finishes before it.

- In conclusion, we can say that Speculative execution is the key **feature of Hadoop** that improves job efficiency. Hence, it reduces the job execution time.


DETECTION
HANDLING
RECOVERY
Heartbeating
Speculative Execution
Re-execution