Hadoop Fundamentals

Hadoop is an Apache-based open-source framework that is written in java and enables big datasets to be processed across clusters of computers using simple programming techniques. Hadoop is a distributed computing system that can expand from a single computer to thousands of clustered computers, with each unit providing local processing and storage. Hadoop can store and handle massive datasets ranging from gigabytes to petabytes.

Hadoop in Layman's Term

Hadoop is a Big Data problem-solving tool. When we have Petabytes of data to deal with, we need a lot of processing power and storage. Hadoop offers us a framework for doing this operation quickly.
Hadoop is built on a distributed design, which means it distributes and processes data over several clusters, nodes, and servers. It makes use of parallel computing in this manner to perform different data operations.

History of Hadoop

2002

The History of Hadoop started in the year 2002. In 2002, Doug Cutting and Mike Cafarella started the Apache Nutch Project to create an online search engine that could crawl and index web pages.
But after doing a lot of research they came to know that this would cost around $500,000, which was too expensive and impossible for Indexing billions of websites.

2003

So in 2003, Google released a search paper on GFS i.e, Google File System which detailed the GFS architecture and offered a solution for storing huge datasets in a distributed fashion. This study dealt with the

issue of keeping large files created during the web crawling and indexing process. However, this is just half part of the answer to their difficulty.

2004

In 2004, Google produced another article on MapReduce technology, which was the answer for processing very enormous datasets. For Doug Cutting and Mike Cafarella's Nutch project, this article represented the remaining half solution. Both algorithms (GFS and MapReduce) were there in Google's white paper at the time. But, these two strategies were not used by Google because it was not an open-source. So, Doug cutting along with Mike Cafarella started implementing Google's techniques (GFS & MapReduce) as open-source in the Apache Nutch project

2005

In 2005, Cutting found that the Nutch project is only limited to clusters of 20-40 nodes. For dealing with this limitation more manpower was needed as this could not be possible with just two people i.e, Cutting and Caferella. That's where he got a company called Yahoo which was interested in investing in their efforts. Yahoo had a large team of engineers that was eager to work on their project.

2006

So, in 2006 Cutting joined Yahoo. Through his Nutch project, he wanted to provide the world an open-source, reliable, scalable computing framework. So, he separated the distributed computing parts from nutch and formed a new project called Hadoop. He chosen this name called Hadoop as it was easy to pronounce and was unique, also his son was having a yellow toy elephant and its name was Hadoop.

2007

In 2007, Yahoo finally tested Hadoop on 1000 node clusters and started using it.

2008

In the year 2008, Yahoo presented Hadoop as an open-source project to the Apache Software Foundation(ASF). Also, the Apache Software Foundation successfully tested Hadoop on 4000 node clusters.

2009

After the successful testing of Hadoop, in 2009 it was again tested to sort a PB (PetaByte) of data in less than 17 hours for handling billions of searches and indexing millions of web pages. After this Cutting left Yahoo and joined Cloudera.

Why is Hadoop in demand?

Here, there are some features that will describe why Hadoop is in demand:-

1. Managing Big Data

   In today's world, there is an explosion of data. Especially the volume of unstructured data is increasing exponentially. Therefore in order to manage this huge amount of data we required a technology called Hadoop.

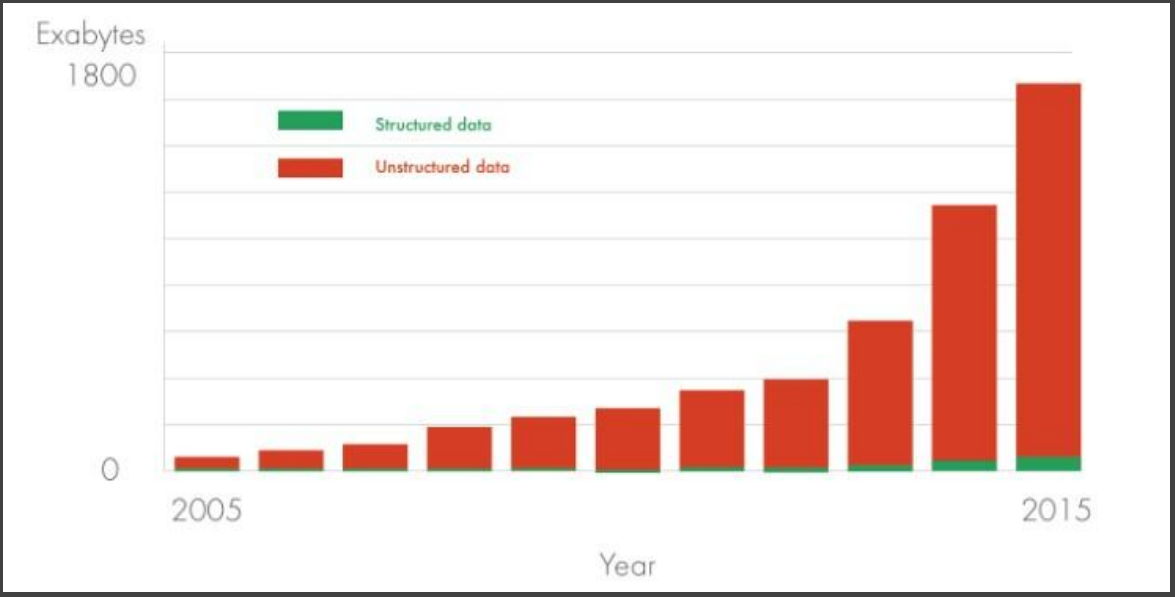

Fig:- Rise of Unstructured Data

According to Google, humans created 5 exabytes of data from the birth of civilization till 2003. And now after every two days, we are creating 5 exabytes of data. There is a growing need for a system that can manage this volume of data. So, Hadoop came to the rescue in this situation.

2. Exponential Growth of Big Data Market

   Companies are gradually recognising the value of big data in their operations. In India, the big data analytics industry will expand eightfold. According to NASSCOM, it will grow from USD 2 billion to USD 16 billion by 2025. Smart gadgets are becoming more prevalent in India's cities and villages as the country develops. The big data market will grow as a result of this.

3. Lack of Hadoop Professionals

   As we've seen, the Hadoop industry is constantly expanding, resulting in additional employment openings every day. Because the essential skills are in short supply, the majority of these Hadoop job openings stay unfilled. So now is the moment to put your big data skills to the test by understanding the technology

before it's too late. Make a career boost by becoming a Hadoop specialist.

4. Ease of Use
   At the back-end, the Hadoop framework is in charge of all the parallel data processing. While coding, we don't have to be concerned with the difficulties of distributed processing. All we have to do now is write the driver, mapper, and reducer functions. The Hadoop framework is in charge of storing and processing data in a distributed way. Ecosystem coding has been much simpler after the advent of Spark in Hadoop. Thousands of lines of code are required in MapReduce. In Spark, however, achieving the same functionality requires just a few lines of code.

5. Hadoop is everywhere
   There isn't a single sector where Big Data hasn't had an impact. Healthcare, retail, government, finance, media, transportation, natural resources, and other industries have all been impacted by Big Data. People are becoming more data-savvy. This indicates that they are recognising the value of data. Hadoop is a platform for harnessing the power of data to benefit company operations.

Components of Hadoop Ecosystem

The major Components of Hadoop are:-

1. HDFS (Hadoop Distributed File System)
   HDFS is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes. HDFS is one of the major components of Apache Hadoop. Also HDFS follows master-slave architecture and it has the following elements:-
   a) NameNode

NameNode is also known as Master node. NameNode does not store any data but it consists of files and directories. NameNode holds Metadata, such as the number of blocks, their location, which Rack the data is saved on, which Datanode the data is stored on, and other information.

b) DataNode

DataNode is also known as Slave node and is responsible for storing actual data in HDFS. Data node responds to client requests for read and write operations. Datanode's replica block is made up of two files on the file system. The first file contains data, while the second contains metadata for the block. Data checksums are included in HDFS Metadata. Each Datanode establishes a connection with its matching Namenode and performs handshaking during startup. Handshaking is used to verify the namespace ID and DataNode software version. When a mismatch is discovered, DataNode shuts down automatically.

2. Map Reduce

Hadoop MapReduce is a key component of the Hadoop ecosystem that handles data processing. MapReduce is a programming framework that makes it simple to create programmes that process the massive amounts of organised and unstructured data contained in the Hadoop Distributed File System. Because MapReduce applications are parallel in nature, they are ideal for executing large-scale data analysis on a cluster of machines. As a result, the speed and reliability of clustered parallel processing is improved.

The MapReduce works in total two phases:-

a) Map

The map function turns a set of data into another set of data, with individual pieces split down into tuples (key/value pairs).

b) Reduce

The Reduce function takes the Map's output as an input and combines the data tuples based on the key, changing the value of the key accordingly.

3. YARN

YARN stands for Yet Another Resource Negotiator. It is a very efficient technology to manage the Hadoop cluster and a completely new way of processing data rightly at the centre of the Hadoop architecture.

How Hadoop solves Data Explosion problem?

Data explosion problems are everywhere in this digital world. Every companies whether it be big or small are now investing to solve problems in Hadoop. Here are some of the example of how Hadoop solves data explosion problem:-

1. Pattern Recognition

   Pattern recognition is a common big data challenge that Hadoop can successfully tackle since it can manage structured, semi-structured, and unstructured data. Real-time analytics is a typical use for pattern recognition. Hadoop can collect, store, and interpret data from sensors and linked devices in real time. Hadoop can proactively assist in churning possible safety hazards and assisting in damage control with this real-time data.

2. Index Building

Another big data difficulty is indexing vast amounts of data. Data generation has grown so common that limiting it is practically impossible. Hadoop can be used to create indexes for those millions of records. It effectively simplifies high-volume data search queries. An app that needs to announce correct scores for the top players on their list is a good example of this.

3. Recommendation Engines
   Almost every online platform now has a recommendation engine. It doesn't matter if it's an e-commerce site or a content-based site like YouTube. It is possible to create meaningful recommendations and change recommendations in a fraction of a second using Hadoop. Hadoop and predictive analysis essentially helps in the filtering of large amounts of data online in order to reduce down digestible options that enhance user engagement and conversions.

Hadoop 1.x

- Hadoop 1.x has only two major components:
  1) MapReduce
  2) HDFS
- In Hadoop 1.x, MapReduce does both batch processing and Cluster management.
- In Hadoop 1 there is only one NameNode to manage the entire namespace.
- The whole stack is affected when a Namenode fails in Hadoop 1.
- Hadoop 1.x is limited to 4000 nodes per cluster and it does not support horizontal scalability.

Hadoop 2.x

- Hadoop 2.x has three major components:
  1) MapReduce
  2) HDFS
  3) YARN
- In Hadoop 2.x, YARN does cluster management
- In Hadoop 2.x, there is multi NameNode
- In Hadoop 2.x, Hive, Pig and HBase are all equipped to handle NameNode failure
- Hadoop 2.x supports 10000 nodes per cluster and it also supports horizontal scalability.

Hadoop 2.x and Hadoop 3.x

Here are the difference between Hadoop 2.x and Hadoop 3.x on basis of certain features:-

Features	Hadoop 2.x	Hadoop 3.x
License	In Hadoop 2.x, Apache 2.0 is used for license	In Hadoop 3.x also, Apache 2.0 is used for license
Minimum supported version of Java	Minimum supported version of java is java 7	Minimum supported version of java is java 7
Fault - Tolerance	Fault tolerance is handled by replication. HDFS by default replicates each block three times for a number of	Fault tolerance is handled by Erasure coding. Erasure Coding is to use in the place of Replication, which provides the

	purposes	same level of fault tolerance
Data Balancing	For data balancing, it uses HDFS balancer	For data, balancing, it uses Intra-data node balancer, which is invoked via the HDFS disk balancer CLI
Storage Overhead example	If there is 6 block so there will be 18 blocks occupied the space because of the replication scheme	If there is 6 block so there will be 9 blocks occupied the space 6 block and 3 for parity

Hadoop 1.x Components

The major components of Hadoop 1.x are:-

a) HDFS
b) MapReduce

Also known as 'Two Pillars' of Hadoop 1.x.

Let's discuss these two pillars of Hadoop in detail.

HDFS

HDFS stands for Hadoop Distributed File System. Here, Commodity Hardware is used to store the BigData. It's made to work with large data sets, having a 64MB block size as the default (We can change it as per our Project requirements).

Now again this HDFS component is divided into sub-components:

a) Name Node
b) Data Node

Name Node

The Name Node is placed into the Master Node. It is mainly used to hold Meta Data about Data Nodes such as "How many blocks are saved in Data Nodes, What is the file size, Which Data Nodes have data, Slave Node Details, Data Node Locations, Timestamps, and so on."

Data Node

The Data Node is placed into Slave Nodes.It is mainly used to store actual data i.e, real data. It generally stores data in data slots of size 64MB.

MapReduce

MapReduce is a programming model for distributed data processing or batch processing. Like HDFS, MapReduce relies on commodity hardware to process "High volumes of variety of Data at High Velocity rate" in a reliable and fault-tolerant way.

Now again this HDFS component is divided into sub-components:

c) Job Tracker
d) Task Tracker

Job Tracker

MapReduce Tasks are assigned to Task Trackers in the Cluster of Nodes using Job Tracker. When prior Task Trackers fail or shutdown circumstances occur, it reassigns the identical duties to other Task Trackers.

Job Tracker keeps track of all Task Tracker's statuses, such as up/running, Failed, and Recovered.

Task Tracker

Task Tracker completes the tasks that Job Tracker has assigned and reports back to Job Tracker on their progress.

How Hadoop 1.x Components Work

This architecture is used by Hadoop 1.x components to interface with one another and to work in parallel, in a reliable and fault-tolerant manner.

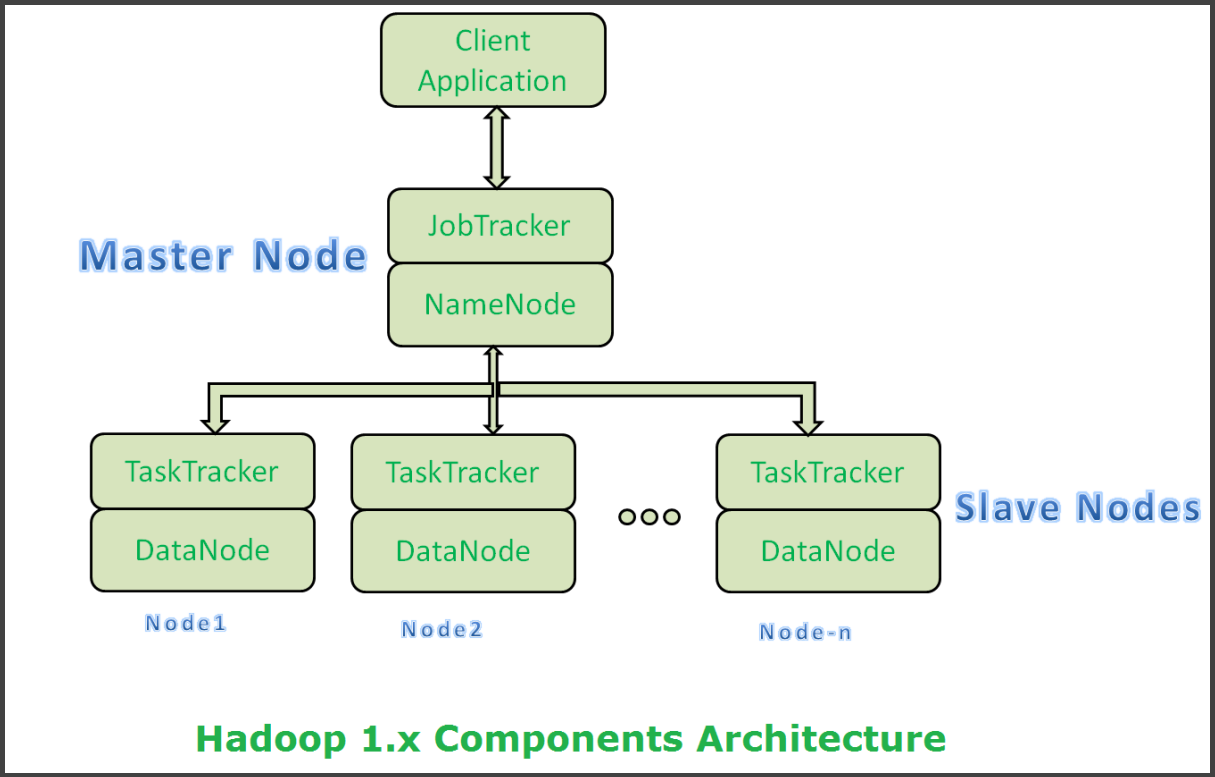

Fig:- Hadoop 1.x components Architecture

Here in the figure, as you can see there is one master node and one slave node. These both contain two Hadoop components:

- HDFS Component
- MapReduce Component

Now let's first talk about the HDFS Component:

Master Node's HDFS Component is also known as Name Node. This Name Node component is used to store MetaData.

Slave Node's HDFS Component is also known as Data Node. This

Data Node component is used to store the actual information.

Now about MapReduce Component:

Master Node's MapReduce component is also known as Job Tracker. This Job Tracker takes care of assigning tasks to task tracker and receiving results from them.

Slave Node's MapReduce coponent is also known as Task Tracker. This Task Tracker actually performs the task given by Job Tracker by using MapReduce Batch Processing tool.

Hadoop 2.x

The major components of Hadoop 2.x are:-

a) HDFS
b) YARN
c) MapReduce

Also known as the 'Three Pillars' of Hadoop 2.x. Here major key component change is YARN.

Let's discuss these component in detail:

HDFS

HDFS stands for Hadoop Distributed File System. As you may recall, in Hadoop 1.x architecture, the Name Node was a single point of failure, which meant that if your Name Node daemon went down for any reason, you couldn't access your Hadoop Cluster. However, in Hadoop 2.x architecture, the Name Node is no longer a single point of failure.

To deal with this problem, Hadoop 2.x is featured with Name Node HA which is referred to as HDFS High Availability (HA).

- Hadoop 2.x supports two Name Nodes at a time one node is active and another is standby node
- Active Name Node handles the client operations in the cluster

- At any given time when Active Name Node is down, Standby Name Node takes over and handles the client operation thereafter.

YARN

YARN stands for Yet Another Resource Negotiator. It is a resource management layer in Hadoop and also allows various data processing engines such as interactive processing, graph processing, batch processing, and stream processing to run and process data stored in HDFS (Hadoop Distributed File System).

MapReduce

MapReduce is a data processing module for batch processing or distributed data processing. It's also known as "MR V1" because it's a Hadoop 1.x component with some new features.

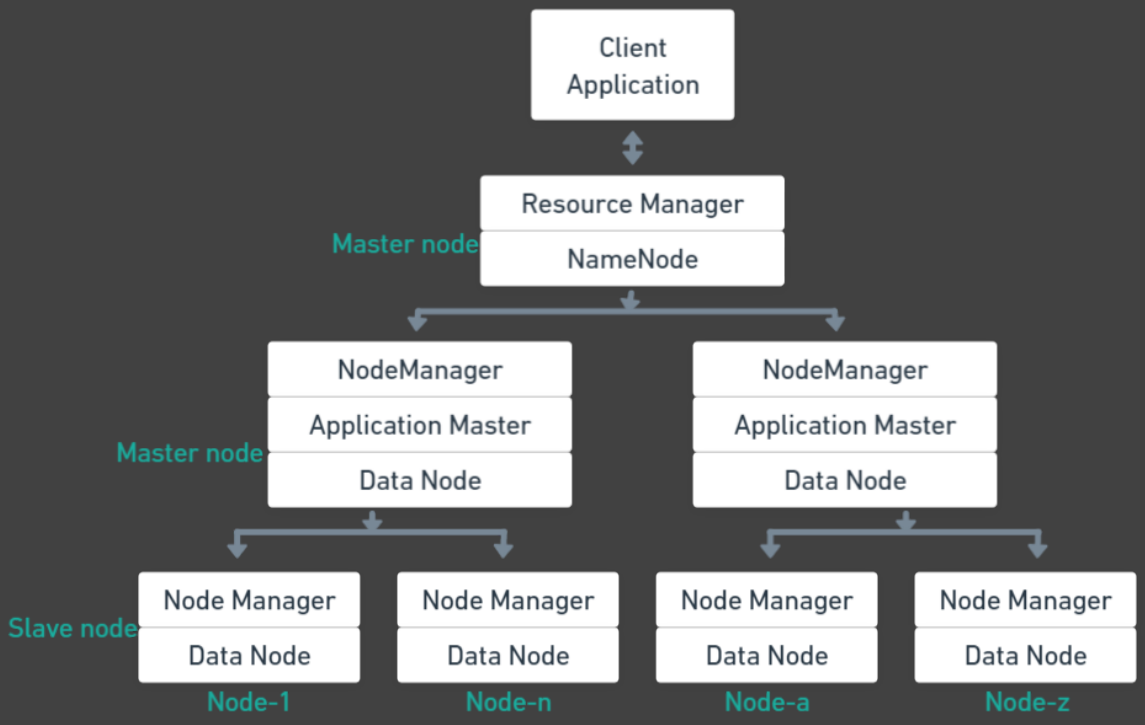

Fig:- Hadoop 2.x components Architecture

In Hadoop 2.x, there are some additional components as compared to Hadoop 1.x. Let's discuss all those component in detail:

Resource Manager

Resource Manager is a Per-Cluster Level Component. Resource Manager is again divided into two components:

- Scheduler
- Application Manager

Schedulers' task is to distribute resources to the running applications. It only deals with the scheduling of tasks and hence it performs no tracking and no monitoring of applications.

Whereas Application Manager manages applications running in the cluster. Tasks, such as the starting of Application Master or monitoring, are done by the Application Manager.

Application Master

Every job that is submitted to the framework is an application, and each application has its own Application Master. The following are the responsibilities of the Application Master:

- It manages errors and coordinates the execution of the application in the cluster.
- It negotiates with the Resource Manager for resources.
- It collaborates with the Node Manager to execute and monitor the tasks of other components.
- Heartbeats are delivered to the Resource Manager at regular intervals to check on its health and to update data based on its resource demands.

Node Manager

Node Manager is a component that is installed on a per-node basis. It is responsible for:

- Managing the life-cycle of the Container.

- Monitoring the usage of each Container's resources.

Container

- Containers are found on each Master Node and Slave Node.
- In HDFS, a container is a chunk of memory (Either Name Node or Data Node).
- Containers in Hadoop 2.x are comparable to Data Slots in Hadoop 1.x.

HDFS

HDFS is a distributed file system of Hadoop or you can understand it as the primary distributed storage used by Hadoop applications, that runs on commodity hardware and can handle massive data collections. It is used to scale an Apache Hadoop cluster from a few nodes to hundreds (or even thousands) of nodes. HDFS is one of Apache Hadoop's primary components, along with MapReduce and YARN. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data.

Some of the key features of Hadoop are:-

- HDFS is well suited for distributed storage and distributed processing using commodity hardware.
- It is fault tolerant, scalable, and extremely simple to expand.
- HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.
- Hadoop is written in Java and runs on all major operating systems.
- To communicate directly with HDFS, Hadoop includes shell-like commands.

HDFS Architecture

The Architecture of HDFS includes name node, secondary name node, data node, checkpoint node, backup node, and blocks. HDFS is fault-tolerant, and it is managed through the replication process. The Name node and data Node coordinates store huge files in a distributed structure across the cluster systems.

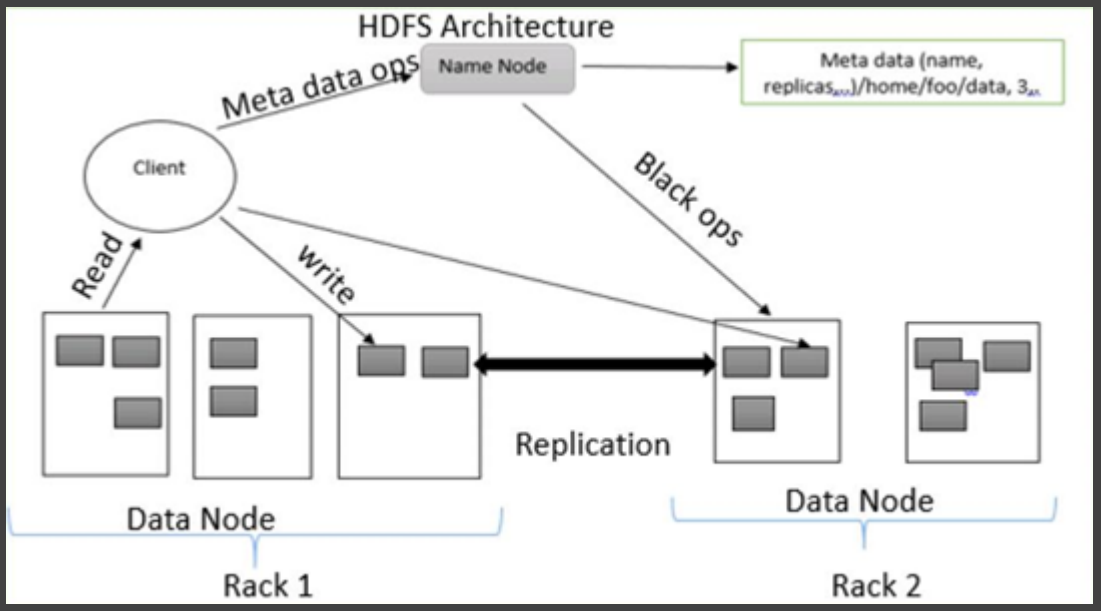

Fig:- Hadoop 2.x components Architecture

In general, HDFS follows master-slave architecture. These are the components present in HDFS architecture:-

1. **Name Node**

The Name Node is placed into the Master Node. It handles all the blocks which are present on DataNodes. Name Node performs the following tasks:-

- It manages all the blocks of Data.
- Name Node also provides file access to users.
- It also keeps all the records of blocks present on Data Node.
- Name Node keeps track of all file information such as if a file is renamed or its content is modified or destroyed. It also immediately logs the changes in EditLogs.

2. **Secondary Name Node**

Secondary NameNode is also called a checkpoint node because it performs regular checkpoints. It acts as a helper for primary NameNode. Secondary Name Node performs the following tasks:-

- Secondary Name Node combines FsImage and EditLogs from the NameNode.
- It reads all file system data/information from NameNode's storage memory and writes it to the filesystem's hard drive.
- It downloads the FsImage and EditLogs from Name Node at regular intervals, reads the modification information done by Edit Logs files, and notes the modification to the FsImage. This process creates a new FsImage, which is then sent back to the NameNode. Whenever the NameNode starts, it will use these FsImage.

3. Data Node

The Data Node is placed into Slave Nodes.It is mainly used to store actual data i.e, real data. Name Node performs the following tasks:-

- Every data is stored on Data Nodes.
- It generally stores data in data slots of size 64MB.
- It does all file actions as requested by the user, such as reading file content and writing new data to files.
- It also follows all directions given, such as building partnerships, deleting some blocks on DataNode, and so on.

4. Checkpoint Node

A checkpoint node is a node that periodically creates a checkpoint of files. Checkpoint the node in HDFS, then retrieve the FsImage and EditLogs from NameNode, combine them, and transmit the new image to NameNode. The most recent checkpoint is saved in the same directory as the name node's directory. As a result, the checkpointed image is always accessible if it is required.

5. **Backup Node**

   A backup node performs the same role as a checkpoint node in terms of checkpointing. The Backup node in Hadoop stores the most recent and updated copy of the file system namespace. Because the Backup node is synced with the current NameNode, there is no need to download FsImage and editsLogs files from the active NameNode to establish a checkpoint. Because it saves namespace into a local FsImage file and resets editions, the Backup node's function is more precise.

6. **Blocks**

   Users' data is stored in HDFS files, which are subsequently partitioned into smaller parts. The DataNodes keep track of these segments. A block refers to the set of features found on DataNodes. These blocks have a default block size of 128 MB. By setting HDFS, the size of the block can be modified to meet the needs of the user.

   The block length is equal to the data size if the data size is less than the block size. If the data is 135 MB, for example, two blocks will be created. One will have a default size of 128 MB, whereas the other will have a size of only 7 MB, not 128 MB. As a result, a significant amount of space and disc clock time is saved.

HDFS Features

Here are some of the important features of HDFS:-

- **Fault tolerant**

  In HDFS, fault tolerance refers to the system's resilience in the event of a breakdown. The HDFS is so fault-tolerant that if one machine fails, the copy of that data is automatically activated on the other machine.

- **Scalability**
  HDFS is highly scalable, with the ability to expand hundreds of nodes in a single cluster.

- **Replication**
  The node containing the data may be lost due to unfavourable conditions. To avoid such issues, HDFS always keeps a copy of the data on a different system.
- **High Availability**
  Hadoop's high availability functionality ensures that data is available even if a NameNode or Data Node fails. HDFS makes data block replicas, if one of the Data Node fails, the user can still access his data from the other Data Nodes that carry a copy of the same data block.
  And if the active Name Node fails, the passive node takes the active Name Node's responsibilities. As a result, data will be available to the user even if the machine crashes.

- **Distributed Data Storage**
  Distributed data storage is one of the most significant characteristics of HDFS, and it is one of the reasons why Hadoop is so strong. Data is separated into various chunks and stored in nodes in this system.

- **Portability**
  HDFS is constructed in such a way that it can be moved from one platform to another with ease.

Name Node and Data Node

HDFS cluster has two types of nodes operating in a master-slave pattern i.e, a Name node and a number of Data Nodes. Name Node stores the metadata and Data Node stores the actual/real data.

Let’s first discuss some key points of Name Node then we will move towards Data Node.

Name Node

- The Name Node manages the file system namespace that consists of files and directories
- These files and directories are represented on the Name Node by inodes which stores all information about files and directories such as permissions, modification and access times,namespace and disk space quotas.
- Name Node maintains file-system tree and its metadata for all files and directories in the tree.
- Inodes and memory blocks which store metadata of file system is called image(fsimage).
- Namenode stores these image in RAM and persistent record of images is stored in namenode's local file system called checkpoint.
- Along with the checkpoint, Name Node maintains a write ahead log file called journal, in its local file system.
- Any changes made in HDFS is tracked inside the journal, which grows in size unless the changes are persisted and merged with the checkpoint.

Data Node

- Each block in Data Node consists of two files. The first file contains the actual data and the second file contains metadata like checksum of data stored.
- When a Data Node initializes and joins with Name Node, a namespace ID is assigned to it and that namespace ID is persistently stored in all Data Nodes. Namespace ID is used to maintain file system integrity.

- When a system start, all Data Nodes of cluster sends handshake message to namenode to prove its identity that datanode belong to given namenode namespace.
- If name Node finds that namespace ID is not correct for any Data Node then that Data Node is not allowed to join the cluster and Data Node shut-down automatically. Thus, namespace ID restricts Data Node with different namespace IDs to join clusters. On successful handshake process, Data Node is registered with Name Node and becomes part of hadoop cluster.
- Also the storage ID is assigned to each Data Nodes when it joins a Name Node for the very first time and it never changes after that. That's why this storage ID is an internal identifier of the Data Node.
- By the time, when Data Nodes are registered with the Name Node, they send block report to Name Node. This Block report contains block ID, generation stamp and block length for each block replicas datanode possess.
- Along with block reports, Data Node also sends a lightweight message to Name Node called Heartbeat to mark their presence in the cluster. The default heartbeat interval is three seconds.
- Now, if in case Name Node does not receive a heartbeat from a Data Node in ten minutes then the Name Node marks the datanode to be dead.
- Name Node does not send message to data Node directly. It sends response of heartbeat and instructs Data Node to provide block reports, replicate blocks, shut down nodes, etc.

Secondary Name Node

In the Hadoop cluster, in order to deal with single point failure of Name Node, Hadoop maintains a standby node called as Secondary Name Node in different server other than where Name Node is existing. Suppose for an example the hadoop cluster has not been started for months and the log file is very huge. Then at the time when cluster

restarts, Name Node needs to restore the image(metadata of file system) from the checkpoint created earlier and merge it with the journal. The time taken to restart will depend on the size of journal. Here comes the secondary namenode for rescue. Here the main purpose of using secondary namenode is to periodically merge journal with checkpoint and create new journal and checkpoint. The steps which secondary namenode follows are :

- Secondary namenode downloads both checkpoint and journals from namenode.
- Merge both of them locally and create new checkpoint and empty journal.
- The updated checkpoint is returned to Name Node.
- So, the secondary namenode avoids overhead of merging logs at the moment of restarting the cluster and Name Node restores faster.

Job Tracker

- The job tracker's main functions are resource management (managing task trackers), resource availability tracking, and task life cycle management (tracking its progress, fault tolerance etc.)
- The JobTracker process is normally operated on a separate node rather than on a DataNode.
- The client sends requests to JobTracker for MapReduce execution.
- HDFS will continue to work if the JobTracker is offline, however MapReduce execution will be disabled and ongoing MapReduce jobs will be halted.
- JobTracker process is critical to the Hadoop cluster in terms of MapReduce execution.

Task Tracker

- The task tracker has the simple role of following the job tracker's instructions and updating the job tracker's progress status on a regular basis.
- Task Tracker runs on almost all the Data Nodes.
- In MRv2, task tracker was replaced by Node Manager.
- TaskTrackers run Mapper and Reducer tasks on DataNodes.
- JobTracker will allocate Mapper and Reducer jobs to TaskTrackers to complete.
- The TaskTracker will be in constant connection with the JobTracker, signalling the task's progress.

Hadoop installation modes

Hadoop is available in three installation modes: stand-alone, pseudo-distributed, and fully-distributed. Let’s see these three mode:-

Standalone mode

By default, Hadoop is configured to run in a standalone or non-distributed mode. Standalone mode is primarily used for debugging when HDFS isn't required.

Pseudo-distributed mode

Pseudo-distributed mode is also referred to as a single-node cluster because both the NameNode and DataNode are located on the same system.

Fully-distributed mode

This fully distributed node is the production mode of Hadoop where multiple nodes will be running.

HDFS Read and Write operations

HDFS follows the concept of write once read many models so, we cannot edit files already stored in HDFS, but we can append the data by reopening the file. Here we will discuss about how client read and write the data from HDFS. Let’s first discuss about the read operation:-

Read Operations

- The client opens the file for reading by calling ‘open()’ method.
- This object uses RPC to connect to the namenode and retrieve metadata such as the file's block positions. Please keep in mind that these are the addresses of the file's first few blocks.
- Addresses of DataNodes that have a copy of that block are returned in response to this metadata request.
- An object of type FSDataInputStream is returned to the client once DataNode addresses have been received. DFSInputStream, which manages interactions with DataNode and NameNode, is included in FSDataInputStream. A client calls the'read()' method in step 4 of the preceding diagram, which causes DFSInputStream to connect to the first DataNode with the first block of a file.

- Data is read in streams, with the client repeatedly calling the read() method. This read() action will continue until the end of the block has been reached.
- DFSInputStream terminates the connection and continues on to find the next DataNode for the next block once the end of a block has been reached.
- When a client is finished reading, it uses the close() method to close the connection.

Write Operations

- The client creates the file by invoking DistributedFileSystem's create() method (DFS).
- DFS sends an RPC request to the name node to create a new file in the namespace of the file system with no blocks associated with it. The name node runs a series of checks to ensure that the file doesn't already exist and that the client has the necessary rights to create it. If these tests are successful, the name node creates a record for the new file; if the file cannot be created, the client is thrown an IOException. For the client to begin writing data to, the DFS returns an FSDataOutputStream.

- The DFSOutputStream separates the data into packets, which it writes to an indoor queue called the info queue, because the client writes data. The DataStreamer consumes the data queue and is responsible for requesting additional blocks from the name node by selecting from a pool of suitable data nodes to store the replicas. A pipeline is formed by a list of data nodes, and we'll suppose the replication level is three in this case, so there are three nodes in the pipeline. The DataStreamer sends the packets to the pipeline's primary data node, which stores and transmits each packet to the pipeline's second data node.
- Similarly, the second data node saves the packet and sends it to the pipeline's third (and final) data node.
- The DFSOutputStream maintains a "ack queue," which is an internal backlog of packets waiting to be acknowledged by data nodes.
- This action sends all remaining packets to the data node pipeline and waits for acknowledgements before connecting to the name node to determine whether the file is complete.

Local File System and HDFS

In terms of storage let’s understand the difference between Local file system and HDFS:-

- **Local File System**
  The Linux operating system's basic file system is known as the Local file system and stores any data file as it is in single copy. It uses the tree format to store data files. Any user can immediately access data files here. The data blocks are not replicated by LFS. It has always been used to store and process personal information (small data).

- **Distributed File System**
  When we need to store and process a large data file approx 1 TB in size then the Operating System's Local file system is insufficient

so, for that we g o for Distributed File System(DFS) in these situations. DFS divides any data file into multiple blocks before storing it. This file system works on the master-slave architecture with the Master being NameNode and the slaves being DataNodes.

What is a Rack?

In a Hadoop cluster, a rack is a group of 30-40 DataNodes or machines in a single data center or location. Traditional network design connects these DataNodes in a rack to the NameNode via a network switch. There will be numerous racks in a huge Hadoop cluster.

Rack Awareness

Rack awareness is the process of making Hadoop aware of which machines belong to which rack and how these racks are connected within the Hadoop cluster. In a Hadoop cluster, the Name Node stores all of the Data Node's rack ids. While storing the data blocks, Name Node selects the nearest DataNode based on the rack information. Rack awareness in Hadoop is defined as knowing how different data nodes are dispersed across racks or knowing the cluster architecture in the Hadoop cluster in simple terms. Rack awareness is crucial since it assures data reliability and aids in data recovery in the event of a rack failure.

Why Rack Awareness?

The following are the reasons for Hadoop's Rack Awareness:

- To increase the availability and dependability of data.
- To enhance the cluster's performance.
- To increase network capacity.
- To avoid losing data if the entire rack fails, despite the fact that the likelihood of a rack failure is far lower than that of a node failure.
- When at all feasible, keep bulk data on the rack.

- In-rack is assumed to provide faster bandwidth and lower latency.

Advantage of implementing Rack Awareness

- Hadoop's rack awareness aids replica placement, ensuring great reliability and fault tolerance.
- Rack awareness guarantees that Read/Write requests to replicas are sent to the nearest or same rack. This increases reading speed while lowering writing costs.
- With the rack awareness policy's we store the data in different Racks so no way to lose our data.
- Rack Awareness makes use of block transfers within the rack to increase network bandwidth. Data access requirements are met with the least amount of network trip possible in order to reduce network overheads.
- Rack Awareness assists the NameNode in assigning tasks to nodes in the network topology that are closer to data.
- Rack awareness can aid MapReduce jobs as well. Because it knows where the map's data is stored, it can run the map task on that machine, saving a significant amount of bandwidth and time.

HDFS Commands

We have divided the HDFS commands into three phase i.e, Basic, intermediate and Advanced one. Let's discuss each one of that one by one.

Basic HDFS Commands:-		
Sr. No.	HDFS Command	Description
1	$ Hadoop version	Prints Hadoop version

2	$ Hadoop fs -ls	List the contents of the root directory in HDFS
3	$ Hadoop fs -df hdfs:/	Report the amount of space used and available on a currently mounted filesystem
4	$ Hadoop balancer	The HDFS balancer re-balances data across the DataNodes, moving blocks from over-utilized to under-utilized nodes.
5	$ Hadoop fs -help	Help Command
Intermediate HDFS Commands:-		
6	$ Hadoop fs -mkdir /user/Cloudera/	creates a directory at the specified HDFS location
7	$ Hadoop fs -put data/sample.txt /user/training/Hadoop	Copies data from one location to another
8	$ Hadoop fs -du -s -h /user/Cloudera/	See the space occupied by a particular directory in HDFS
9	$ Hadoop fs -rm -r /user/cloudera/pigjobs/	Remove a directory in Hadoop
10	$ hadoop fs -rm -skipTrash hadoop/retail/*	Removes all the files in the given directory
11	$ hadoop fs -expunge	To empty the trash

12	$ hadoop fs -copyFromLocal /home/cloudera/sample/ /user/cloudera/flume/
	
	$ hadoop fs -copyToLocal /user/cloudera/pigjobs/* /home/cloudera/oozie/
		copies data from and to local to HDFS

Advanced HDFS Commands:-		
14	$ sudo -u hdfs hadoop fs -chmod 777 /user/cloudera/flume/	change file permissions
15	$ hadoop fs -setrep -w 5 /user/cloudera/pigjobs/	set data replication factor for a file
16	$ Hadoop fs -count hdfs:/	Count the number of directories, files, and bytes under hdfs
17	$ sudo -u hdfs hdfs dfsadmin -safemode leave	make namenode exit safe mode
18	$hadoop namenode -format	Hadoop format a namenode

HDFS Web Interface

HDFS provides a web server that could be used to perform basic file browsing and status monitoring. The NameNode exposes this by default on port 50070. With a web browser, accessing http://namenode:50070/ will produce a page with summary information about the cluster's health, capacity, and usage (similar to the information returned by bin/hadoop dfsadmin -report).

Setting dfs.http.address in conf/hadoop-site.xml changes the IP and port where the web interface listens. The format must be address:port. Use 0.0.0.0 to accept requests from all addresses.

We can use this interface to browse HDFS using a rudimentary file-browser interface. On port 50075, each DataNode provides its file browser interface. Set the dfs.datanode.http.address configuration key to something other than 0.0.0.0:50075 to override this. This interface allows you to view log files generated by Hadoop daemons, which is useful for distributed debugging and troubleshooting.

Fault tolerance in HDFS

Fault tolerance refers to a system's ability to work or operate even

under adverse situations (like components failure).

- HDFS is fault-tolerant because it replicates data on different DataNodes.
- The data blocks are stored in different DataNodes. If one node crashes, the data can still be retrieved from other DataNodes.
- Prior to Hadoop 3, faults were handled through the replica construction process.
- It replicates users' data across many machines in the HDFS cluster. As a result, if one of the machines in the cluster fails, data can still be accessed from other machines that have the same copy of the data.
- HDFS also maintains the replication factor by creating a replica of data on other available machines in the cluster if suddenly one machine fails.
- Hadoop 3 introduced Erasure Coding to offer fault tolerance. Erasure Coding in HDFS improves storage efficiency while maintaining the same level of fault tolerance and data durability as standard HDFS deployments based on replication.

Name Node Failure Management

- In HDFS, each Datanode in the cluster sends a heartbeat to the Namenode at a predetermined period.
- If it receives any heartbeats, the Data Nodes are in good working order. In other words, we can determine whether a Datanode is active or not.
- If the Namenode does not receive the heartbeat signal, it assumes that the Datanode is either dead or inoperable.
- Once the datanodes have been declared dead. Based on the replication factor defined in the hdfs-site.xml file, data blocks on the failed Datanode are duplicated on other Datanodes.
- When the failed datanodes come back up, the Name node will take over management of the replication factor. This is how Namenode handles data node failure.

By this way Name Node manages the failure

Input Split and Data Blocks

Data Blocks

- Large files are divided into little parts called as Blocks in Hadoop HDFS.
- It is a continuous location on the hard drive where data is stored.
- The way, FileSystem stores data as a collection of blocks. In the same way, HDFS stores each file as blocks.
- The Hadoop application is responsible for distributing the data block across multiple nodes.
- The default size of the HDFS block is 128 MB which we can configure as per our requirement.
- All blocks of the file are of the same size except the last block, which can be of same size or smaller.
- The files are split into 128 MB blocks and then stored into Hadoop FileSystem.

Input Split

- InputSplit represents the data that will be processed by a single Mapper.
- The split is separated into records, and the map processes each record (which is a key-value pair).
- The number of InputSplits equals the number of map tasks. The data for a MapReduce task is initially saved in input files, which are commonly kept in HDFS.
- The InputFormat variable specifies how these input files will be split and read. InputFormat is in charge of generating InputSplit.
- Split size is set to be about equal to block size by default.
- InputSplit is user-defined, and the user can set the split size in the MapReduce program based on the size of the data.

Why is HDFS block size 128 MB?

The default size of a block in HDFS is 128 MB (Hadoop 2. x) and 64 MB (Hadoop 1. x) which is much larger as compared to the Linux system where the block size is 4KB. The reason of having this huge block size is to minimize the cost of seek and reduce the meta data information generated per block.

Hadoop RecordReader

The record reader breaks the data into key/value pairs for input to the Mapper. Hadoop RecordReader provides Key-value pairs for the mapper using the data within the bounds created by the InputSplit. The "start" byte position in the file where the RecordReader should begin creating key/value pairs and the "end" byte position in the file where it should cease reading records are the "start" and "end" respectively. The data is loaded from its source and then turned into key-value pairs appropriate for reading by the Mapper in Hadoop RecordReader. It keeps in touch with the InputSplit until the file reading is finished.

InputFormat in Hadoop

Hadoop InputFormat provides the input-specification for execution of the Map-Reduce job. InputFormat divides the input file into InputSplits, each of which is assigned to a different Mapper. InputFormat is the initial phase in the MapReduce job execution. It describes how to split up and read input files. It is also responsible for creating the input splits and dividing them into records. Here are some of the features of InputFormat:-

- The files or other objects for input are selected by InputFormat.
- It also specifies data splitting. It specifies the size of individual Map tasks as well as the server that may be used to execute them.
- Hadoop InputFormat defines the RecordReader. It is also responsible for reading actual records from the input file

TextInputFormat

TextInputFormat is the default input file format Hadoop, means if we do not specify any file formats, RecordReader will assume that the input file format is textinputformat. TextInputFormat considers each line of

each input file as another record and performs no parsing. This is mainly used for unformatted data or line-based records like log files.
Some of the features of TextInputFormat are:-

- It is used to read lines of text files.
- It basically helps in the generation of key-value pairs from text.
- The text files are then broken into lines with the help of line feed i.e, moving one line forward to check the end of the line. This is called as splits.
- After splits are created, key-value pairs are generated with the help of TextInputFormat.
- Data elements in MapReduce are always structured as a Key-Value pair. As a result, TextInputFormat aids in the generation of key and value pairs.

Types of Hadoop InputFormat

In Hadoop, there are various types of MapReduce InputFormat that are used for various purposes. Let's discuss each one of that one by one:-

1. FileInputFormat

   It is the basic class for all InputFormats that work with files. Hadoop FileInputFormat determines the location of data files in the input directory. FileInputFormat is given a path containing files to read when we start a Hadoop job. All files are read by FileInputFormat, which separates them into one or more InputSplits.

2. TextInputFormat

   It's the standard InputFormat. Each line of each input file is treated as a separate record by this InputFormat. It doesn't parse anything. TextInputFormat is useful for unformatted data or records that are based on lines, such as log files. Hence,

   - Key :- is the byte offset of the file's first line (not the entire file, only one split), making it unique when coupled with the file name.

- Value :- It's the line's content, without the line terminators.

3. KeyValueTextInputFormat

   It works in a similar way to TextInputFormat in that each line of input is treated as a single record. While TextInputFormat treats the entire line as the value, the KeyValueTextInputFormat uses a tab character ('/t') to divide the line into key and value. The value is the remainder of the line after the tab character, while the key is everything up to the tab character.

4. SequenceFileInputFormat

   SequenceFileInputFormat is a Hadoop input format for reading sequence files. Series files are binary files that store binary key-value pairs in a sequence. Sequence files compress data in blocks and allow for direct serialization and deserialization of a variety of data types (not just text). Both the Key and the Value are user-defined in this case.

5. SequenceFileAsInputFormat

   It's a SequenceFileInputFormat variation. The sequence file key values are converted to Text objects in this way. As a result, it converts the keys and values by running 'tostring()' on them. As a result, SequenceFileAsTextInputFormat allows sequence files to be used as streaming input.

6. SequenceFileAsBinaryInputFormat

   Hadoop SequenceFileAsBinaryInputFormat is a SequenceFileInputFormat that extracts the keys and values of a sequence file as an opaque binary object.

7. NLineInputFormat

   Hadoop NLineInputFormat is a variant of TextInputFormat in which the keys are the line's byte offset and the values are the

line's contents. With TextInputFormat and KeyValueTextInputFormat, each mapper receives a variable number of lines of input, which varies depending on the size of the split and the length of the lines. We use NLineInputFormat if we want our mapper to receive a fixed number of lines of input. In this:

- N is the number of lines of input received by each mapper. Each mapper receives exactly one line of input by default (N=1).
- If N=2, each split will have two lines. The first two Key-Value pairs will be given to one mapper, while the second two Key-Value pairs will be given to another mapper.

8. DBInputFormat

The Hadoop DBInputFormat is an InputFormat that uses JDBC to read data from a relational database. We must be careful not to overwhelm the database from which we are reading too many mappers because it lacks portioning capabilities. As a result, it's best for importing tiny datasets and maybe merging them with huge datasets from HDFS using MultipleInputs. LongWritables is the key, and DBWritables is the value.

OutputFormat in Hadoop

Hadoop's default output format is TextOuputFormat, which publishes records as lines of text. Text files are created as output files if the file output format is not specified explicitly.

- Hadoop RecordWriter takes Reducer's output data and writes it to output files.
- Output Format determines how these output key-value pairs are written in output files by RecordWriter.

- The functions Output Format and InputFormat are very similar. Hadoop's OutputFormat instances are used to write to files on HDFS or local drive.
- The output format for a Map-Reduce job is described by OutputFormat. based on the output specification.
- The RecordWriter implementation provided by OutputFormat is used to write the job's output files. A FileSystem is used to store the output files.

Types of Hadoop OutputFormat

1. TextOutputFormat

   TextOutputFormat is the default Hadoop reduction Output Format in MapReduce. It outputs (key, value) pairs on individual lines of text files, and its keys and values can be of any type because TextOutputFormat converts them to strings using toString(). The tab character that separates each key-value combination can be adjusted using the MapReduce.output.textoutputformat.separator parameter. KeyValue Because it separates lines into key-value pairs based on a configurable separator, TextOutputFormat is used to read these output text files.

2. SequenceFileOutputFormat

   It is an intermediate format used between MapReduce jobs, which quickly serialize arbitrary data types to the file, and the corresponding SequenceFileInputFormat, which deserializes the file into the same types and presents the data to the next mapper in the same manner as it was emitted by the previous reducer, because these are compact and easily compressible. The static methods on SequenceFileOutputFormat govern compression.

3. SequenceFileOutputFormat

It's a different type of SequenceFileInputFormat. It also saves keys and values in binary format to a sequence file.

4. MapFileOutputFormat

   It's a variant of FileOutputFormat. It also saves the results as map files. In order, the framework adds a key to a MapFile. As a result, we must guarantee that the reducer emits keys in the correct order.

5. MultipleOutputs

   It allows data to be written to files with names determined from the output keys and values, or from any string.

6. LazyOutputFormat

   Even if the output files are empty, FileOutputFormat will occasionally create them. LazyOutputFormat is an OutputFormat wrapper that ensures that the output file is produced only when the record for a certain partition is emitted.

7. DBOutputFormat

   In Hadoop, the DBOutputFormat Output Format is used to write to relational databases and HBase. It saves the output of reduction to a SQL table. It accepts key-value pairs with a type that extends DBwritable as the key. With a batch SQL query, the returned RecordWriter publishes only the key to the database.

What is Partitioner in Hadoop?

The Partitioner is responsible for partitioning the keys of the intermediate map output. The partition is determined using the hash function and key (or a subset of key). Each mapper's output is partitioned based on the key value, with records with the same key value going into the same partition (within each mapper), and each partition then being passed to a reducer. The partition class defines

where a given (key, value) pair will be placed. After the map phase, and before the reduce phase, the partition phase occurs.

Need of Partitioner in Hadoop?

The MapReduce job takes an input data set and outputs a list of key-value pairs, which is the outcome of the map phase, in which the input data is split into tasks, each task processes the split, and each map outputs a list of key-value pairs. The map output is then delivered to the reduce task, which applies the user-defined reduce function to the map outputs. However, before the reduce step, the map output is partitioned and sorted based on the key.

This partitioning indicates that all the values for each key be grouped together and that all the values of a single key go to the same reducer, allowing the map output to be distributed evenly across the reducer.

Map Only Job in Hadoop

In Hadoop, a Map-Only job is one in which the mapper does all tasks, the reducer performs none, and the mapper's output is the final output. It accomplishes this by breaking down the job submitted into a series of individual tasks i.e, sub-job.

So in map reduce what used to be done is, it first breaks each individual element into tuples i.e, key-value pairs in mapper phase and then in reducer phase it takes the output from the map and combines those tuples based on key. But in the case when we just need to perform the operation, we don’t need aggregation then at that time we prefer ‘Map-Only Job’.

Advantages of Map Only Job

There is a key, sort, and shuffle phase in between the map and reductions phases. Sort and shuffle are in charge of ascending the keys

and then grouping items based on the same keys. This step is quite expensive, and if we don't need it, we should skip it. By skipping it, we'll also skip the sort and shuffle phases. This also reduces network congestion since in shuffling, a mapper's output travels to the reducer, and enormous data must travel to the reducer when the data size is large.

In a map-only task, the output of the mapper is written to local disk before being sent to the reducer, whereas in a map-only job, the output is written directly to HDFS. This saves time and money in the long run. In addition, Hadoop Map Only does not require a partitioner or combiner, which speeds up the operation.