MODULE II HADOOP ARCHITECTURE AND ECOSYSTEM

Hadoop Eco system –– Hadoop architecture – Hadoop cluster – HDFS – Working with distributed file System – MapReduce –- Spark – RDD – Storing and Querying Data.

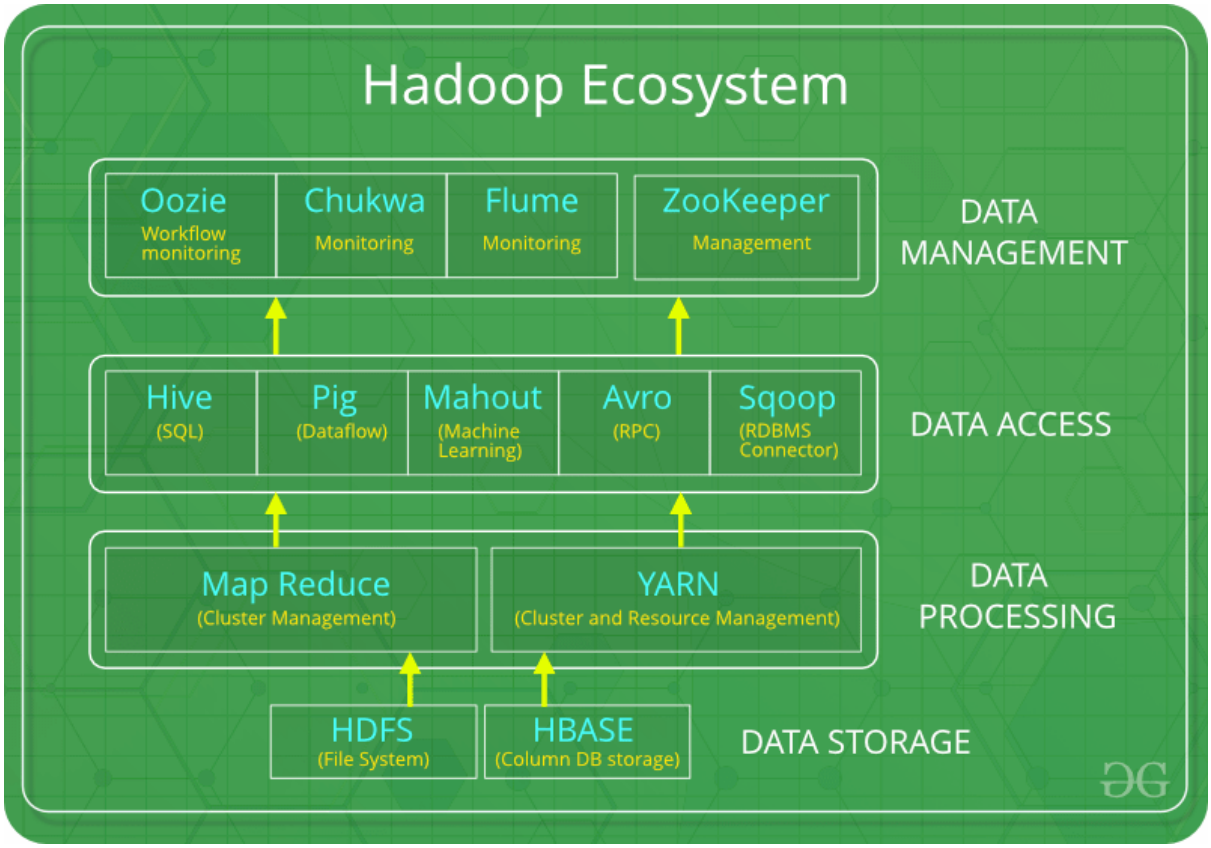
Hadoop Eco system:

Overview: Apache Hadoop is an open source framework intended to make interaction with big data easier, However, for those who are not acquainted with this technology, one question arises that what is big data ? Big data is a term given to the data sets which can't be processed in an efficient manner with the help of traditional methodology such as RDBMS. Hadoop has made its place in the industries and companies that need to work on large data sets which are sensitive and needs efficient handling. Hadoop is a framework that enables processing of large data sets which reside in the form of clusters. Being a framework, Hadoop is made up of several modules that are supported by a large ecosystem of technologies.

Introduction: *Hadoop Ecosystem* is a platform or a suite which provides various services to solve the big data problems. It includes Apache projects and various commercial tools and solutions. There are *four major elements of Hadoop* i.e. HDFS, MapReduce, YARN, and Hadoop Common Utilities. Most of the tools or solutions are used to supplement or support these major elements. All these tools work collectively to provide services such as absorption, analysis, storage and maintenance of data etc.

Following are the components that collectively form a Hadoop ecosystem:

- HDFS: Hadoop Distributed File System
- YARN: Yet Another Resource Negotiator
- MapReduce: Programming based Data Processing
- Spark: In-Memory data processing
- PIG, HIVE: Query based processing of data services
- HBase: NoSQL Database
- Mahout, Spark MLLib: Machine Learning algorithm libraries
- Solar, Lucene: Searching and Indexing
- Zookeeper: Managing cluster
- Oozie: Job Scheduling



Note: Apart from the above-mentioned components, there are many other components too that are part of the Hadoop ecosystem.
All these toolkits or components revolve around one term i.e. *Data*. That's the beauty of Hadoop that it revolves around data and hence making its synthesis easier.
HDFS:

- HDFS is the primary or major component of Hadoop ecosystem and is responsible for storing large data sets of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files.
- HDFS consists of two core components i.e.
    1. Name node
    1. Data Node
- Name Node is the prime node which contains metadata (data about data) requiring comparatively fewer resources than the data nodes that stores the actual data. These data nodes are commodity hardware in the distributed environment. Undoubtedly, making Hadoop cost effective.
- HDFS maintains all the coordination between the clusters and hardware, thus working at the heart of the system.

YARN:

- Yet Another Resource Negotiator, as the name implies, YARN is the one who helps to manage the resources across the clusters. In short, it performs scheduling and resource allocation for the Hadoop System.
- Consists of three major components i.e.
    1. Resource Manager
    1. Nodes Manager
    1. Application Manager
- Resource manager has the privilege of allocating resources for the applications in a system whereas Node managers work on the allocation of resources such as CPU, memory, bandwidth per machine and later on acknowledges the resource manager. Application manager works as an interface between the resource manager and node manager and performs negotiations as per the requirement of the two.

MapReduce:

- By making the use of distributed and parallel algorithms, MapReduce makes it possible to carry over the processing's logic and helps to write applications which transform big data sets into a manageable one.
- MapReduce makes the use of two functions i.e. Map() and Reduce() whose task is:
    1. *Map()* performs sorting and filtering of data and thereby organizing them in the form of group. Map generates a key-value pair based result which is later on processed by the Reduce() method.
    1. *Reduce()*, as the name suggests does the summarization by aggregating the mapped data. In simple, Reduce() takes the output generated by Map() as input and combines those tuples into smaller set of tuples.

PIG:
Pig was basically developed by Yahoo which works on a pig Latin language, which is Query based language similar to SQL.

- It is a platform for structuring the data flow, processing and analyzing huge data sets.
- Pig does the work of executing commands and in the background, all the activities of MapReduce are taken care of. After the processing, pig stores the result in HDFS.
- Pig Latin language is specially designed for this framework which runs on Pig Runtime. Just the way Java runs on the JVM.
- Pig helps to achieve ease of programming and optimization and hence is a major segment of the Hadoop Ecosystem.

HIVE:

- With the help of SQL methodology and interface, HIVE performs reading and writing of large data sets. However, its query language is called as HQL (Hive Query Language).
- It is highly scalable as it allows real-time processing and batch processing both. Also, all the SQL datatypes are supported by Hive thus, making the query processing easier.
- Similar to the Query Processing frameworks, HIVE too comes with two components: *JDBC Drivers* and *HIVE Command Line*.
- JDBC, along with ODBC drivers work on establishing the data storage permissions and connection whereas HIVE Command line helps in the processing of queries.

Mahout:

- Mahout, allows Machine Learnability to a system or application. Machine Learning, as the name suggests helps the system to develop itself based on some patterns, user/environmental interaction or on the basis of algorithms.
- It provides various libraries or functionalities such as collaborative filtering, clustering, and classification which are nothing but concepts of Machine learning. It allows invoking algorithms as per our need with the help of its own libraries.

Apache Spark:

- It's a platform that handles all the process consumptive tasks like batch processing, interactive or iterative real-time processing, graph conversions, and visualization, etc.
- It consumes in memory resources hence, thus being faster than the prior in terms of optimization.
- Spark is best suited for real-time data whereas Hadoop is best suited for structured data or batch processing, hence both are used in most of the companies interchangeably.

Apache HBase:

- It's a NoSQL database which supports all kinds of data and thus capable of handling anything of Hadoop Database. It provides capabilities of Google's BigTable, thus able to work on Big Data sets effectively.
- At times where we need to search or retrieve the occurrences of something small in a huge database, the request must be processed within a short quick span of time. At such times, HBase comes handy as it gives us a tolerant way of storing limited data

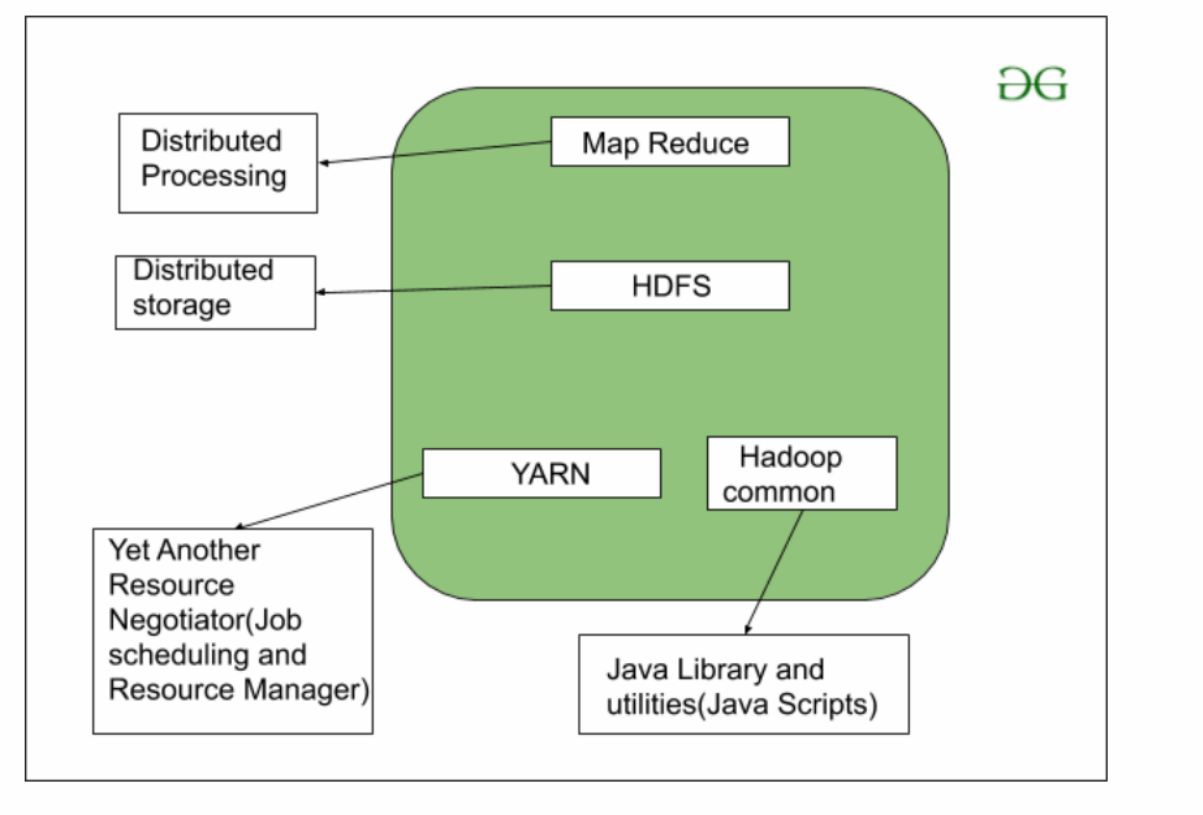
Other Components: Apart from all of these, there are some other components too that carry out a huge task in order to make Hadoop capable of processing large datasets. They are as follows:

- Solr, Lucene: These are the two services that perform the task of searching and indexing with the help of some java libraries, especially Lucene is based on Java which allows spell check mechanism, as well. However, Lucene is driven by Solr.
- Zookeeper: There was a huge issue of management of coordination and synchronization among the resources or the components of Hadoop which resulted in inconsistency, often. Zookeeper overcame all the problems by performing synchronization, inter-component based communication, grouping, and maintenance.
- Oozie: Oozie simply performs the task of a scheduler, thus scheduling jobs and binding them together as a single unit. There is two kinds of jobs .i.e Oozie workflow and Oozie coordinator jobs. Oozie workflow is the jobs that need to be executed in a sequentially ordered manner whereas Oozie Coordinator jobs are those that are triggered when some data or external stimulus is given to it.

Hadoop – Architecture

As we all know Hadoop is a framework written in Java that utilizes a large cluster of commodity hardware to maintain and store big size data. Hadoop works on MapReduce Programming Algorithm that was introduced by Google. Today lots of Big Brand Companies are using Hadoop in their Organization to deal with big data, eg. Facebook, Yahoo, Netflix, eBay, etc. The Hadoop Architecture Mainly consists of 4 components.

- MapReduce
- HDFS(Hadoop Distributed File System)
- YARN(Yet Another Resource Negotiator)
- Common Utilities or Hadoop Common

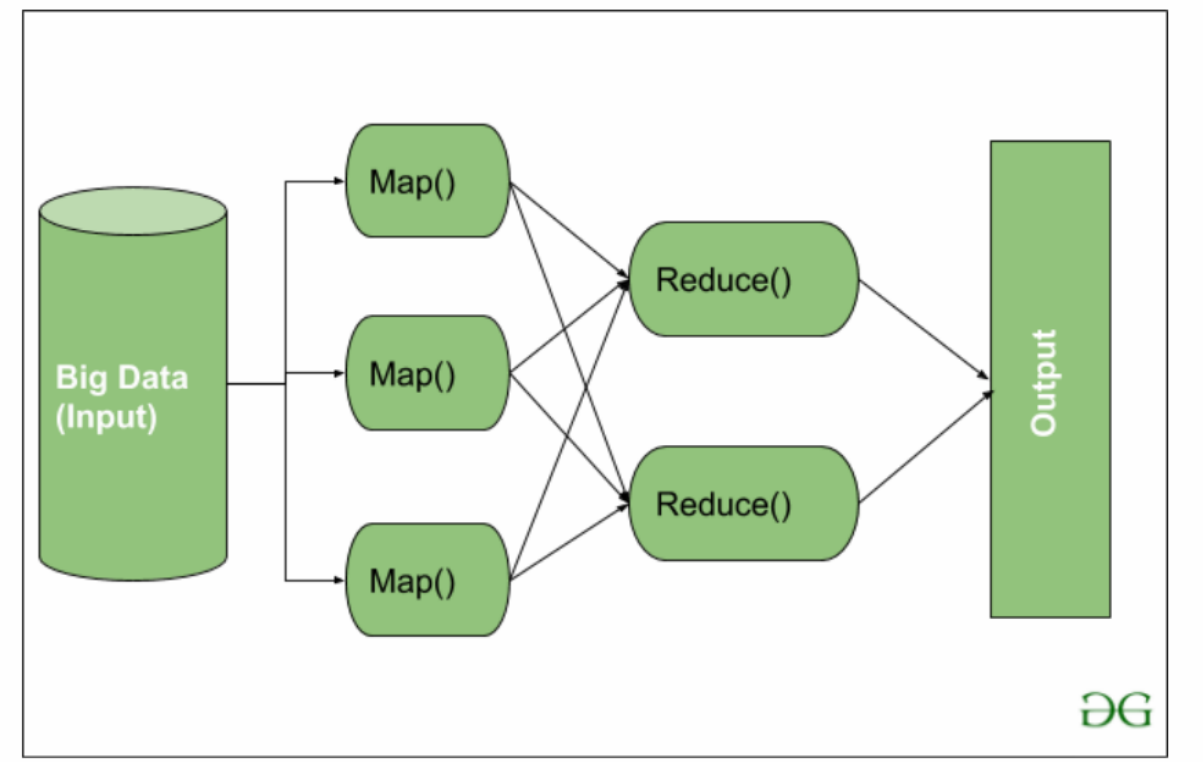


Let’s understand the role of each one of this component in detail.

1. MapReduce

MapReduce nothing but just like an Algorithm or a data structure that is based on the YARN framework. The major feature of MapReduce is to perform the distributed processing in parallel in a Hadoop cluster which Makes Hadoop working so fast. When you are dealing with Big Data, serial processing is no more of any use. MapReduce has mainly 2 tasks which are divided phase-wise:

In first phase, Map is utilized and in next phase Reduce is utilized.



Here, we can see that the *Input* is provided to the Map() function then it's *output* is used as an input to the Reduce function and after that, we receive our final output. Let's understand What this Map() and Reduce() does.

As we can see that an Input is provided to the Map(), now as we are using Big Data. The Input is a set of Data. The Map() function here breaks this DataBlocks into Tuples that are nothing but a key-value pair. These key-value pairs are now sent as input to the Reduce(). The Reduce() function then combines this broken Tuples or key-value pair based on its Key value and form set of Tuples, and perform some operation like sorting, summation type job, etc. which is then sent to the final Output Node. Finally, the Output is Obtained.

The data processing is always done in Reducer depending upon the business requirement of that industry. This is How First Map() and then Reduce is utilized one by one.

Let's understand the *Map Task* and *Reduce Task* in detail.

Map Task:

- RecordReader The purpose of *recordreader* is to break the records. It is responsible for providing key-value pairs in a Map() function. The key is actually is its locational information and value is the data associated with it.
- Map: A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.

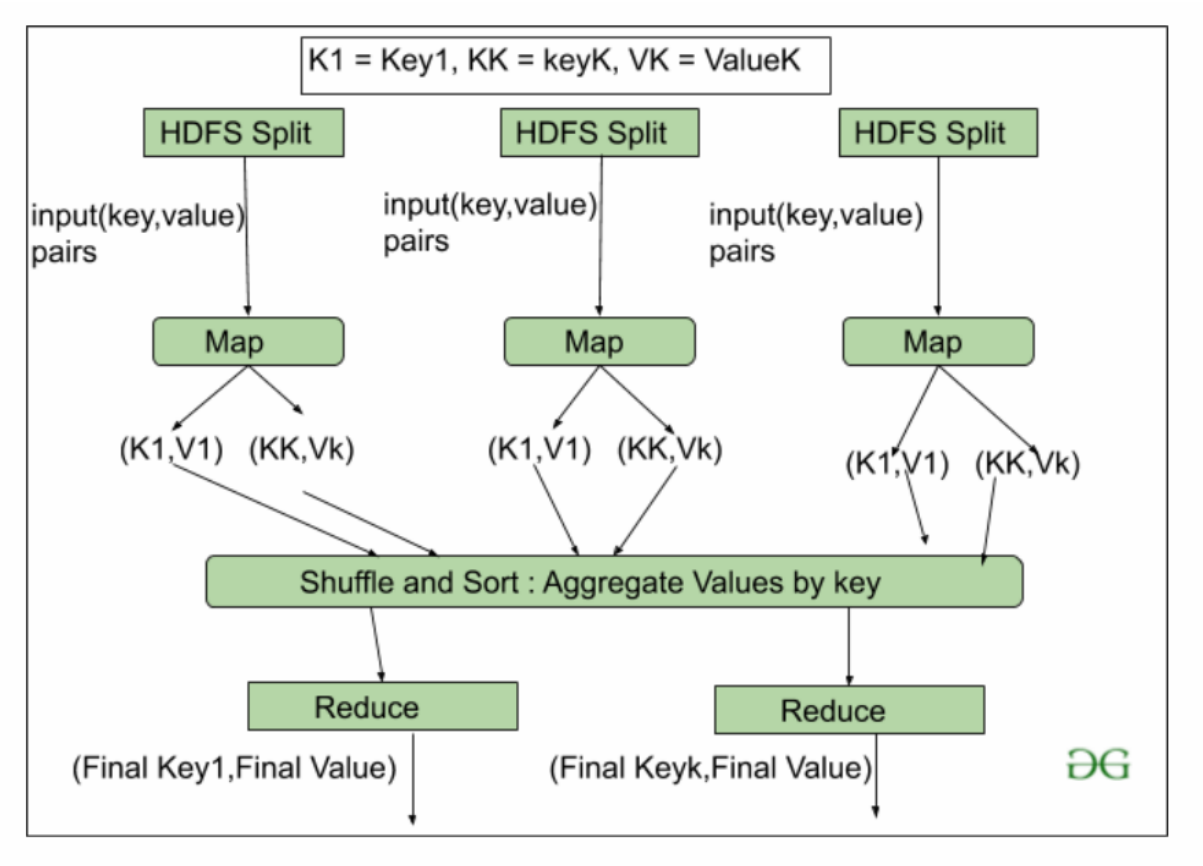
- Combiner: Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The intermediate key-value that are generated in the Map is combined with the help of this combiner. Using a combiner is not necessary as it is optional.
- Partitionar: Partitional is responsible for fetching key-value pairs generated in the Mapper Phases. The partitioner generates the shards corresponding to each reducer. Hashcode of each key is also fetched by this partition. Then partitioner performs it's(Hashcode) modulus with the number of reducers(*key.hashcode()%(number of reducers)).*

Reduce Task

- Shuffle and Sort: The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as *Shuffling*. Using the Shuffling process the system can sort the data using its key value.

Once some of the Mapping tasks are done Shuffling begins that is why it is a faster process and does not wait for the completion of the task performed by Mapper.

- Reduce: The main function or task of the Reduce is to gather the Tuple generated from Map and then perform some sorting and aggregation sort of process on those key-value depending on its key element.
- OutputFormat: Once all the operations are performed, the key-value pairs are written into the file with the help of record writer, each record in a new line, and the key and value in a space-separated manner.



2. HDFS

HDFS(Hadoop Distributed File System) is utilized for storage permission. It is mainly designed for working on commodity Hardware devices(inexpensive devices), working on a distributed file system design. HDFS is designed in such a way that it believes more in storing the data in a large chunk of blocks rather than storing small data blocks.

HDFS in Hadoop provides Fault-tolerance and High availability to the storage layer and the other devices present in that Hadoop cluster. Data storage Nodes in HDFS.

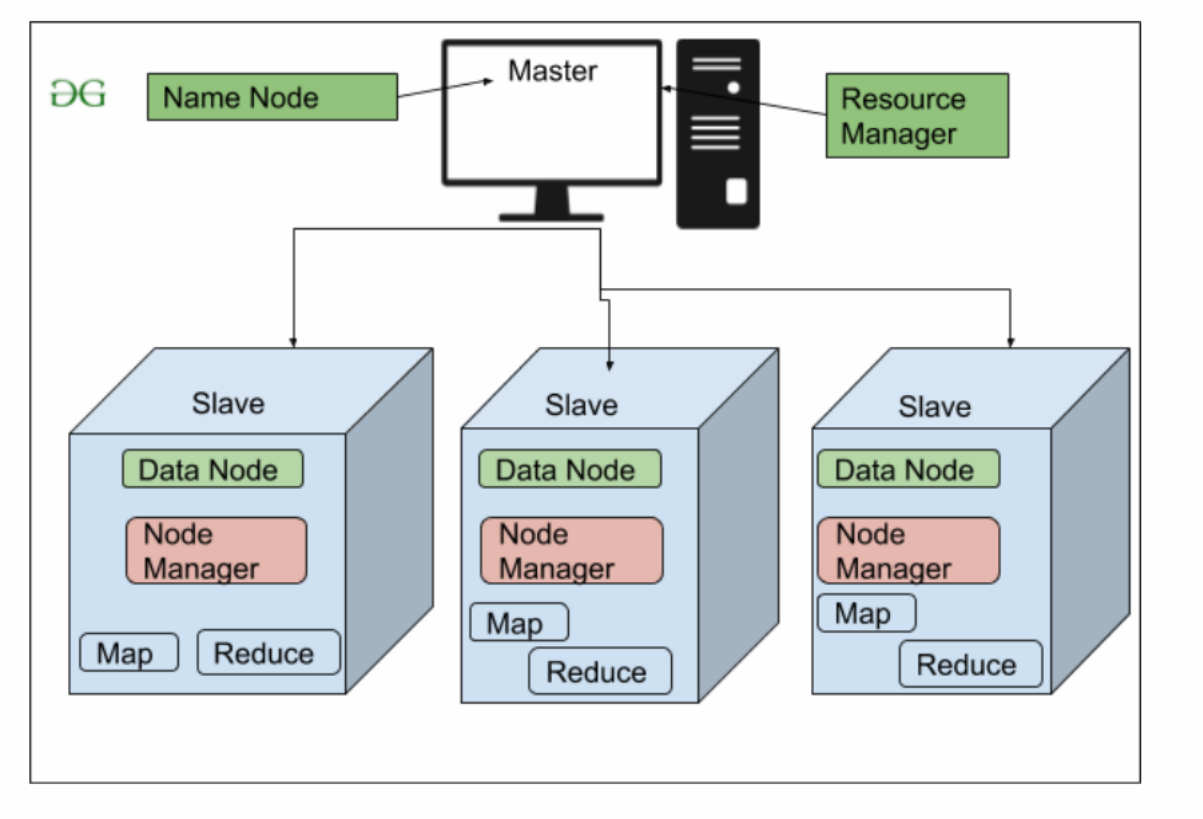
- NameNode(Master)
- DataNode(Slave)

NameNode:NameNode works as a Master in a Hadoop cluster that guides the Datanode(Slaves). Namenode is mainly used for storing the Metadata i.e. the data about the data. Meta Data can be the transaction logs that keep track of the user's activity in a Hadoop cluster.

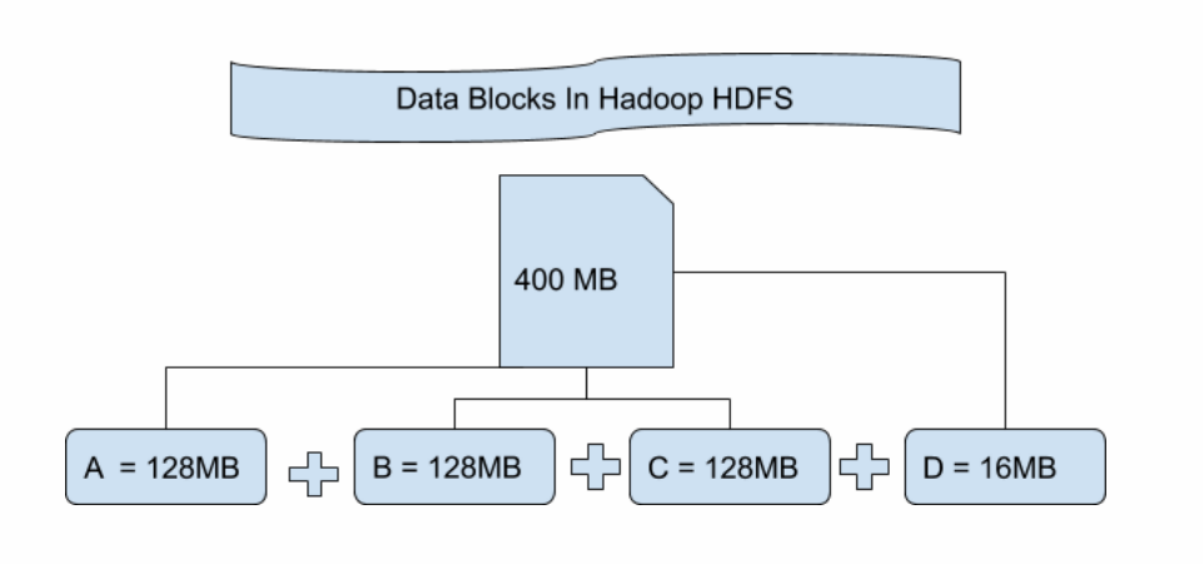
Meta Data can also be the name of the file, size, and the information about the location(Block number, Block ids) of Datanode that Namenode stores to find the closest DataNode for Faster Communication. Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.

DataNode: DataNodes works as a Slave DataNodes are mainly utilized for storing the data in a Hadoop cluster, the number of DataNodes can be from 1 to 500 or even more than that. The more number of DataNode, the Hadoop cluster will be able to store more data. So it is advised that the DataNode should have High storing capacity to store a large number of file blocks.

High Level Architecture Of Hadoop



File Block In HDFS: Data in HDFS is always stored in terms of blocks. So the single block of data is divided into multiple blocks of size 128MB which is default and you can also change it manually.



Let's understand this concept of breaking down of file in blocks with an example. Suppose you have uploaded a file of 400MB to your HDFS then what happens is this file got divided into blocks of 128MB+128MB+128MB+16MB = 400MB size. Means 4 blocks are created each of 128MB except the last one. Hadoop doesn't know or it doesn't care about what data is stored in these blocks so it considers the final file blocks as a partial record as it does not have any idea regarding it. In the Linux file system, the size of a file block is about 4KB which is very much less than the default size of file blocks in the Hadoop file system. As we all know Hadoop is mainly configured for storing the large size data which is in petabyte, this is what makes Hadoop file system different from other file systems as it can be scaled, nowadays file blocks of 128MB to 256MB are considered in Hadoop.

Replication In HDFS Replication ensures the availability of the data. Replication is making a copy of something and the number of times you make a copy of that particular thing can be expressed as it's Replication Factor. As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks.

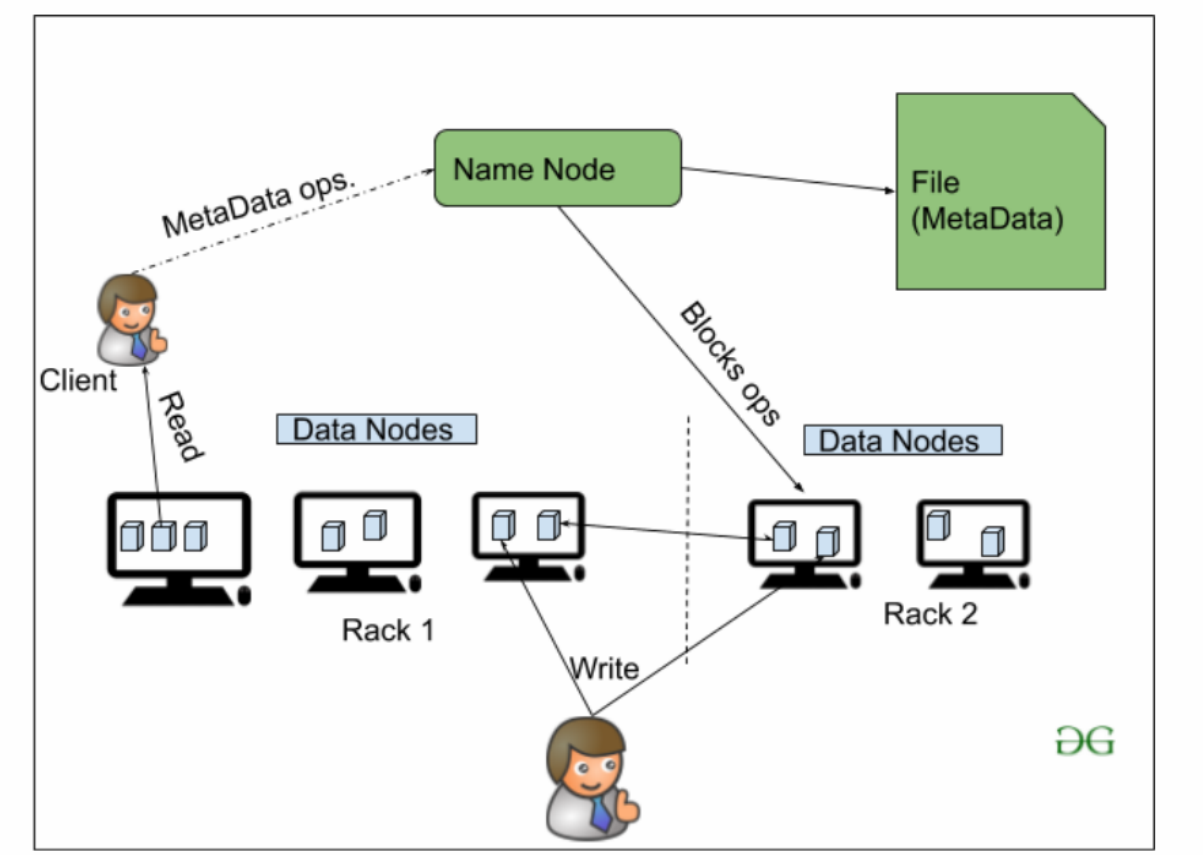
By default, the Replication Factor for Hadoop is set to 3 which can be configured means you can change it manually as per your requirement like in above example we have made 4 file blocks which means that 3 Replica or copy of each file block is made means total of 4×3 = 12 blocks are made for the backup purpose.

This is because for running Hadoop we are using commodity hardware (inexpensive system hardware) which can be crashed at any time. We are not using the supercomputer for our Hadoop setup. That is why we need such a feature in HDFS which can make copies of that file blocks for backup purposes, this is known as fault tolerance.

Now one thing we also need to notice that after making so many replica's of our file blocks we are wasting so much of our storage but for the big brand organization the data is very much important than the storage so nobody cares for this extra storage. You can configure the Replication factor in your *hdfs-site.xml* file.

Rack Awareness The rack is nothing but just the physical collection of nodes in our Hadoop cluster (maybe 30 to 40). A large Hadoop cluster is consists of so many Racks . with the help of this Racks information Namenode chooses the closest Datanode to achieve the maximum performance while performing the read/write information which reduces the Network Traffic.

HDFS Architecture



3. YARN(Yet Another Resource Negotiator)

YARN is a Framework on which MapReduce works. YARN performs 2 operations that are Job scheduling and Resource Management. The Purpose of Job schedular is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized. Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc. And the use of Resource Manager is to manage all the resources that are made available for running a Hadoop cluster.

Features of YARN

- Multi-Tenancy
- Scalability
- Cluster-Utilization
- Compatibility

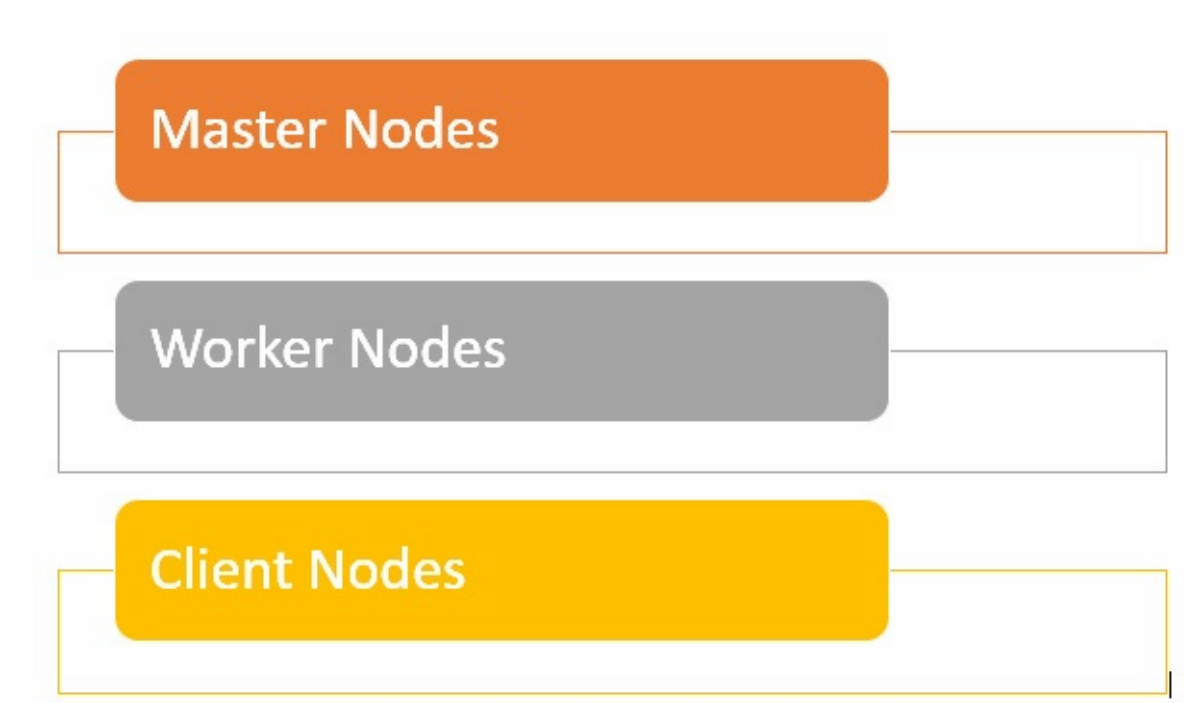
4. Hadoop common or Common Utilities

Hadoop common or Common utilities are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster. these utilities are used by HDFS, YARN, and MapReduce for running the cluster. Hadoop Common verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

HADOOP CLUSTER

Hadoop Cluster Architecture

Hadoop clusters are composed of a network of master and worker nodes that orchestrate and execute the various jobs across the Hadoop distributed file system. The master nodes typically utilize higher quality hardware and include a NameNode, Secondary NameNode, and JobTracker, with each running on a separate machine. The workers consist of virtual machines, running both DataNode and TaskTracker services on commodity hardware, and do the actual work of storing and processing the jobs as directed by the master nodes. The final part of the system are the Client Nodes, which are responsible for loading the data and fetching the results.



- Master nodes are responsible for storing data in HDFS and overseeing key operations, such as running parallel computations on the data using MapReduce.
- The worker nodes comprise most of the virtual machines in a Hadoop cluster, and perform the job of storing the data and running computations. Each worker node runs the DataNode and TaskTracker services, which are used to receive the instructions from the master nodes.
- Client nodes are in charge of loading the data into the cluster. Client nodes first submit MapReduce jobs describing how data needs to be processed and then fetch the results once the processing is finished.

What is cluster size in Hadoop?

A Hadoop cluster size is a set of metrics that defines storage and compute capabilities to run Hadoop workloads, namely :

- Number of nodes : number of Master nodes, number of Edge Nodes, number of Worker Nodes.
- Configuration of each type node: number of cores per node, RAM and Disk Volume.

What are the advantages of a Hadoop Cluster?

- Hadoop clusters can boost the processing speed of many big data analytics jobs, given their ability to break down large computational tasks into smaller tasks that can be run in a parallel, distributed fashion.
- Hadoop clusters are easily scalable and can quickly add nodes to increase throughput, and maintain processing speed, when faced with increasing data blocks.

- The use of low cost, high availability commodity hardware makes Hadoop clusters relatively easy and inexpensive to set up and maintain.
- Hadoop clusters replicate a data set across the distributed file system, making them resilient to data loss and cluster failure.
- Hadoop clusters make it possible to integrate and leverage data from multiple different source systems and data formats.
- It is possible to deploy Hadoop using a single-node installation, for evaluation purposes.

What are the challenges of a Hadoop Cluster?

- Issue with small files - Hadoop struggles with large volumes of small files - smaller than the Hadoop block size of 128MB or 256MB by default. It wasn't designed to support big data in a scalable way. Instead, Hadoop works well when there are a small number of large files. Ultimately when you increase the volume of small files, it overloads the Namenode as it stores namespace for the system.
- High processing overhead - reading and writing operations in Hadoop can get very expensive quickly especially when processing large amounts of data. This all comes down to Hadoop's inability to do in-memory processing and instead data is read and written from and to the disk.
- Only batch processing is supported - Hadoop is built for small volumes of large files in batches. This goes back to the way data is collected and stored which all has to be done before processing starts. What this ultimately means is that streaming data is not supported and it cannot do real-time processing with low latency.
- Iterative Processing - Hadoop has a data flow structure is set-up in sequential stages which makes it impossible to do iterative processing or use for ML.

Hadoop – Cluster, Properties and its Types

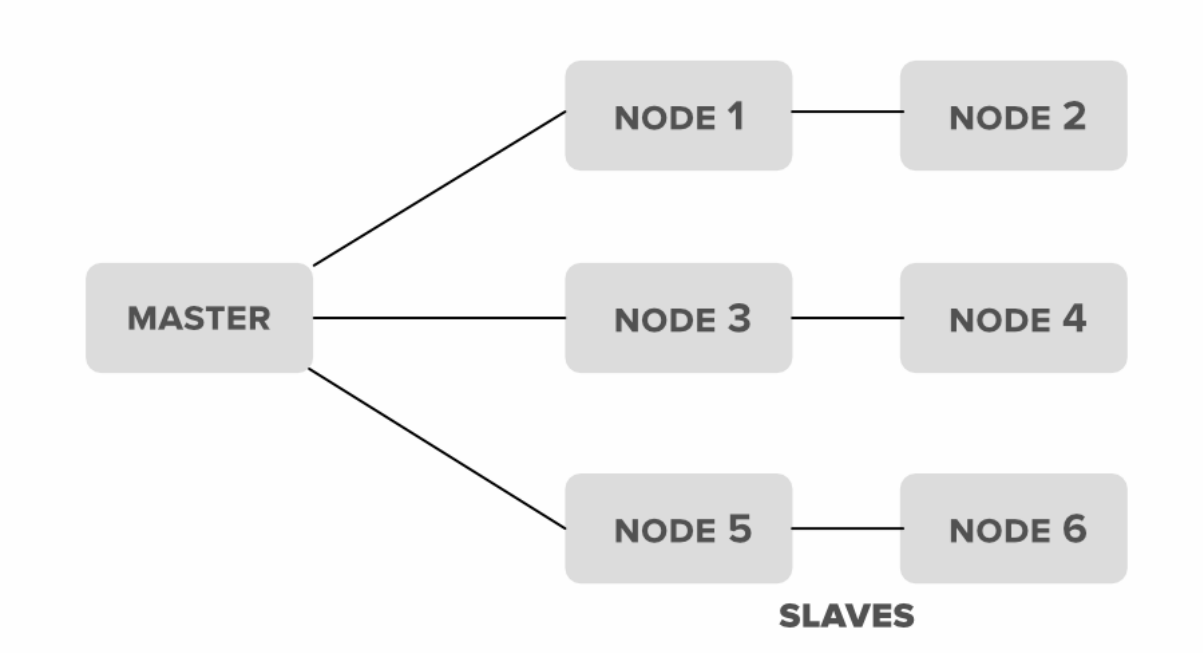
Before we start learning about the Hadoop cluster first thing we need to know is what actually *cluster* means. Cluster is a collection of something, a simple computer cluster is a group of various computers that are connected with each other through LAN(Local Area Network), the nodes in a cluster share the data, work on the same task and this nodes are good enough to work as a single unit means all of them to work together.

Similarly, a *Hadoop cluster* is also a collection of various commodity hardware(devices that are inexpensive and amply available). This Hardware components work together as a single unit. In the Hadoop cluster, there are lots of nodes (can be computer and servers) contains Master and Slaves, the Name node and Resource Manager works as Master and data node, and Node Manager works as a Slave. The purpose of Master nodes is to guide the slave nodes in a single Hadoop cluster. We design Hadoop clusters for storing, analyzing,

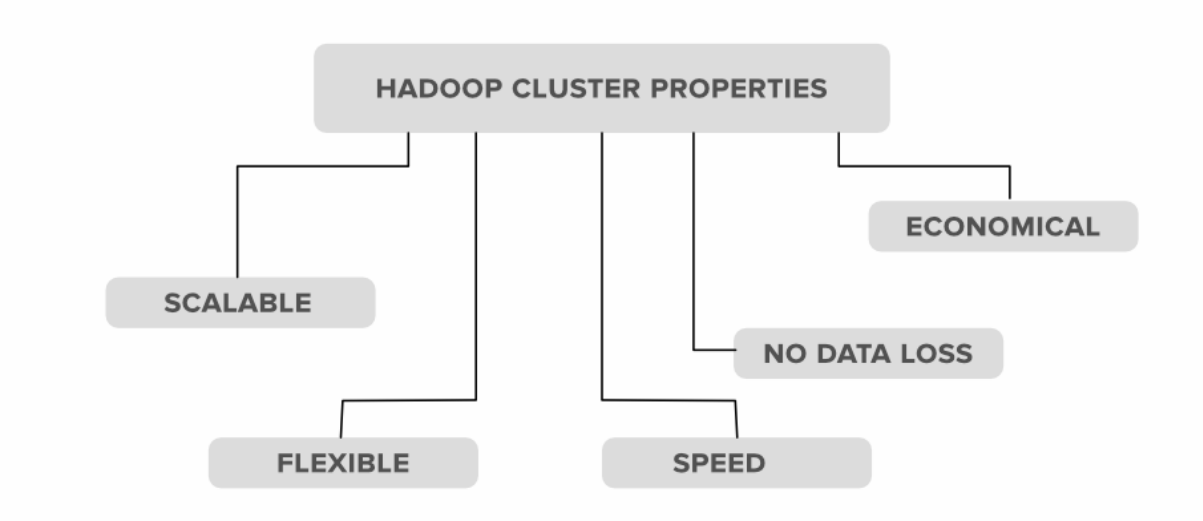
understanding, and for finding the facts that are hidden behind the data or datasets which contain some crucial information. The Hadoop cluster stores different types of data and processes them.

- Structured-Data: The data which is well structured like Mysql.
- Semi-Structured Data: The data which has the structure but not the data type like XML, Json (Javascript object notation).
- Unstructured Data: The data that doesn't have any structure like audio, video.

Hadoop Cluster Schema:



Hadoop Clusters Properties



1. Scalability: Hadoop clusters are very much capable of scaling-up and scaling-down the number of nodes i.e. servers or commodity hardware. Let's see with an example of what

actually this scalable property means. Suppose an organization wants to analyze or maintain around 5PB of data for the upcoming 2 months so he used 10 nodes(servers) in his Hadoop cluster to maintain all of this data. But now what happens is, in between this month the organization has received extra data of 2PB, in that case, the organization has to set up or upgrade the number of servers in his Hadoop cluster system from 10 to 12(let's consider) in order to maintain it. The process of scaling up or scaling down the number of servers in the Hadoop cluster is called scalability.

2. Flexibility: This is one of the important properties that a Hadoop cluster possesses. According to this property, the Hadoop cluster is very much Flexible means they can handle any type of data irrespective of its type and structure. With the help of this property, Hadoop can process any type of data from online web platforms.

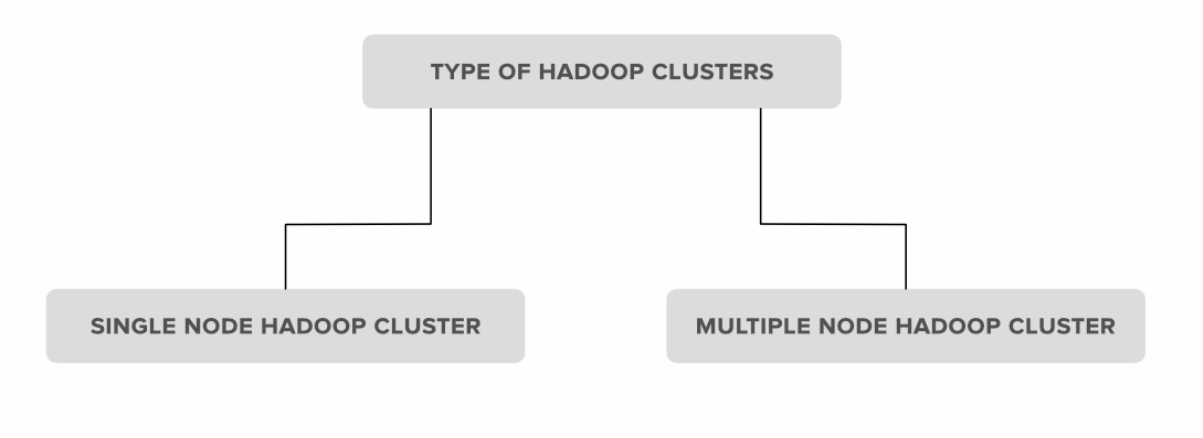
3. Speed: Hadoop clusters are very much efficient to work with a very fast speed because the data is distributed among the cluster and also because of its data mapping capability's i.e. the MapReduce architecture which works on the Master-Slave phenomena.

4. No Data-loss: There is no chance of loss of data from any node in a Hadoop cluster because Hadoop clusters have the ability to replicate the data in some other node. So in case of failure of any node no data is lost as it keeps track of backup for that data.

5. Economical: The Hadoop clusters are very much cost-efficient as they possess the distributed storage technique in their clusters i.e. the data is distributed in a cluster among all the nodes. So in the case to increase the storage we only need to add one more another hardware storage which is not that much costliest.

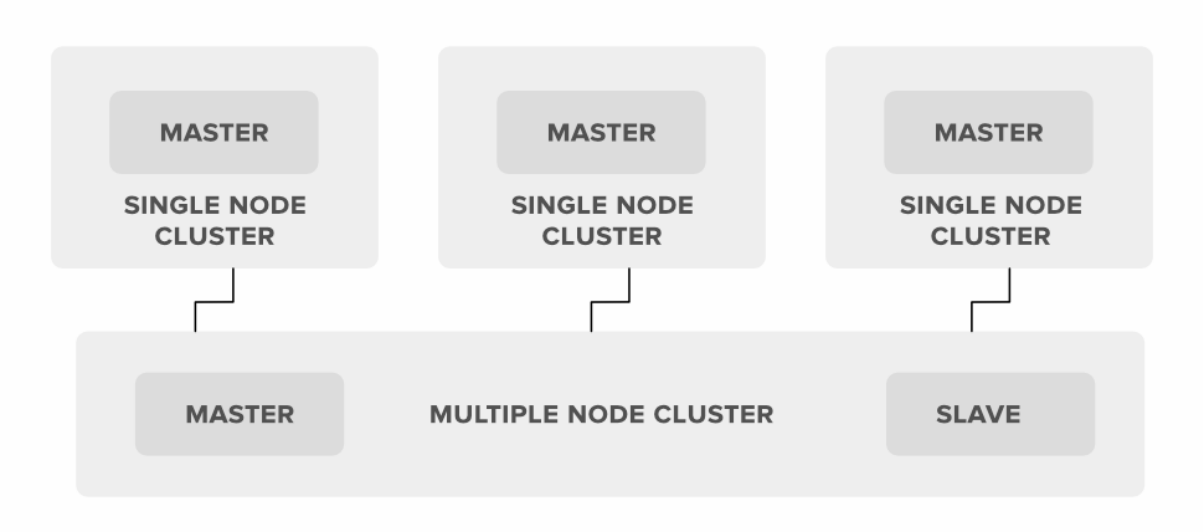
Types of Hadoop clusters

1. Single Node Hadoop Cluster
2. Multiple Node Hadoop Cluster



1. Single Node Hadoop Cluster: In Single Node Hadoop Cluster as the name suggests the cluster is of an only single node which means all our Hadoop Daemons i.e. Name Node, Data Node, Secondary Name Node, Resource Manager, Node Manager will run on the same system or on the same machine. It also means that all of our processes will be handled by only single JVM(Java Virtual Machine) Process Instance.

2. Multiple Node Hadoop Cluster: In multiple node Hadoop clusters as the name suggests it contains multiple nodes. In this kind of cluster set up all of our Hadoop Daemons, will store in different-different nodes in the same cluster setup. In general, in multiple node Hadoop cluster setup we try to utilize our higher processing nodes for Master i.e. Name node and Resource Manager and we utilize the cheaper system for the slave Daemon's i.e.Node Manager and Data Node.



Hadoop Distributed File System(HDFS)

What is HDFS

Hadoop comes with a distributed file system called HDFS. In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application.

It is cost effective as it uses commodity hardware. It involves the concept of blocks, data nodes and node name.

Where to use HDFS

- Very Large Files: Files should be of hundreds of megabytes, gigabytes or more.
- Streaming Data Access: The time to read whole data set is more important than latency in reading the first. HDFS is built on write-once and read-many-times pattern.
- Commodity Hardware:It works on low cost hardware.

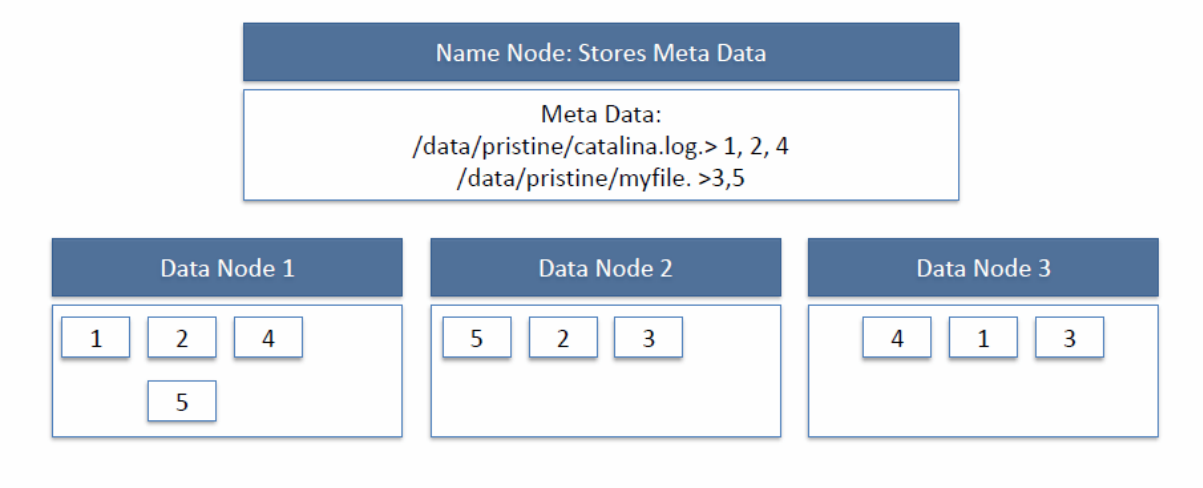
Where not to use HDFS

- Low Latency data access: Applications that require very less time to access the first data should not use HDFS as it is giving importance to whole data rather than time to fetch the first record.
- Lots Of Small Files:The name node contains the metadata of files in memory and if the files are small in size it takes a lot of memory for name node's memory which is not feasible.
- Multiple Writes:It should not be used when we have to write multiple times.

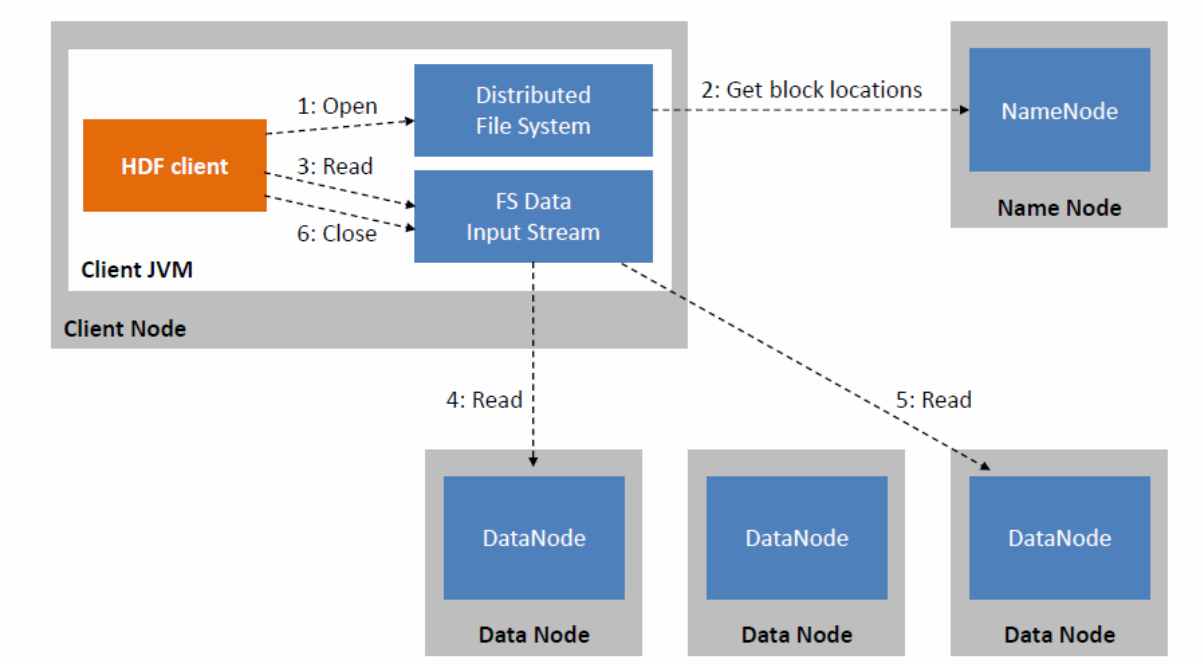
HDFS Concepts

1. Blocks: A Block is the minimum amount of data that it can read or write.HDFS blocks are 128 MB by default and this is configurable.Files n HDFS are broken into block-sized chunks,which are stored as independent units.Unlike a file system, if the file is in HDFS is smaller than block size, then it does not occupy full block?s size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only.The HDFS block size is large just to minimize the cost of seek.
2. Name Node: HDFS works in master-worker pattern where the name node acts as master.Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names and location of each block.The metadata are small, so it is stored in the memory of name node,allowing faster access to data. Moreover the HDFS cluster is accessed by multiple clients concurrently,so all this information is handled bya single machine. The file system operations like opening, closing, renaming etc. are executed by it.
3. Data Node: They store and retrieve blocks when they are told to; by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.

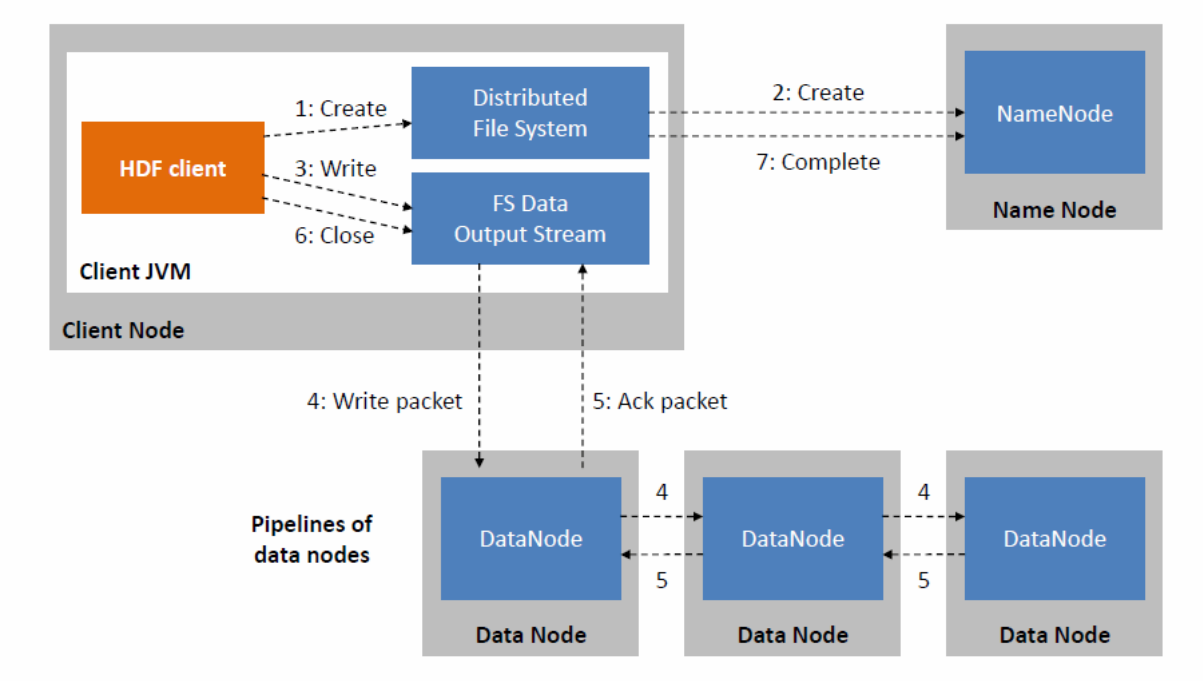
HDFS DataNode and NameNode Image:



HDFS Read Image:



HDFS Write Image:



Since all the metadata is stored in name node, it is very important. If it fails the file system can not be used as there would be no way of knowing how to reconstruct the files from blocks present in data node. To overcome this, the concept of secondary name node arises.

Secondary Name Node: 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.

Starting HDFS

The HDFS should be formatted initially and then started in the distributed mode. Commands are given below.

To Format $ hadoop namenode -format

To Start $ start-dfs.sh

HDFS Basic File Operations

1. Putting data to HDFS from local file system
    - First create a folder in HDFS where data can be put form local file system.

$ hadoop fs -mkdir /user/test

- Copy the file "data.txt" from a file kept in local folder /usr/home/Desktop to HDFS folder /user/ test

$ hadoop fs -copyFromLocal /usr/home/Desktop/data.txt /user/test

- Display the content of HDFS folder

$ Hadoop fs -ls /user/test

2. Copying data from HDFS to local file system
    - $ hadoop fs -copyToLocal /user/test/data.txt /usr/bin/data_copy.txt
3. Compare the files and see that both are same
    - $ md5 /usr/bin/data_copy.txt /usr/home/Desktop/data.txt

Recursive deleting

- hadoop fs -rmr <arg>

Example:

- hadoop fs -rmr /user/sonoo/

HDFS Other commands

The below is used in the commands

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system

- put <localSrc><dest>

Copies the file or directory from the local file system identified by localSrc to dest within the DFS.

- copyFromLocal <localSrc><dest>

Identical to -put

- copyFromLocal <localSrc><dest>

Identical to -put

- moveFromLocal <localSrc><dest>

Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.

- get [-crc] <src><localDest>

Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.

- cat <filen-ame>

Displays the contents of filename on stdout.

- moveToLocal <src><localDest>

Works like -get, but deletes the HDFS copy on success.

- setrep [-R] [-w] rep <path>

Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)

- touchz <path>

Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.

- test -[ezd] <path>

Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.

- stat [format] <path>

Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y).

HDFS Features and Goals

The Hadoop Distributed File System (HDFS) is a distributed file system. It is a core part of Hadoop which is used for data storage. It is designed to run on commodity hardware.

Unlike other distributed file system, HDFS is highly fault-tolerant and can be deployed on low-cost hardware. It can easily handle the application that contains large data sets.

Let's see some of the important features and goals of HDFS.

Features of HDFS

- Highly Scalable - HDFS is highly scalable as it can scale hundreds of nodes in a single cluster.
- Replication - Due to some unfavorable conditions, the node containing the data may be loss. So, to overcome such problems, HDFS always maintains the copy of data on a different machine.
- Fault tolerance - In HDFS, the fault tolerance signifies the robustness of the system in the event of failure. The HDFS is highly fault-tolerant that if any machine fails, the other machine containing the copy of that data automatically become active.
- Distributed data storage - This is one of the most important features of HDFS that makes Hadoop very powerful. Here, data is divided into multiple blocks and stored into nodes.
- Portable - HDFS is designed in such a way that it can easily portable from platform to another.

Goals of HDFS

- Handling the hardware failure - The HDFS contains multiple server machines. Anyhow, if any machine fails, the HDFS goal is to recover it quickly.
- Streaming data access - The HDFS applications usually run on the general-purpose file system. This application requires streaming access to their data sets.
- Coherence Model - The application that runs on HDFS require to follow the write-once-ready-many approach. So, a file once created need not to be changed. However, it can be appended and truncate.

Distributed File System

What is Distributed File System?

A distributed file system (DFS) is a file system that is distributed on various file servers and locations. It permits programs to access and store isolated data in the same method as in the local files. It also permits the user to access files from any system. It allows network users to share information and files in a regulated and permitted manner. Although, the servers have complete control over the data and provide users access control.

DFS's primary goal is to enable users of physically distributed systems to share resources and information through the Common File System (CFS). It is a file system that runs as a part of the operating systems. Its configuration is a set of workstations and mainframes that a LAN connects. The process of creating a namespace in DFS is transparent to the clients.

DFS has two components in its services, and these are as follows:

1. Local Transparency
2. Redundancy

Local Transparency

It is achieved via the namespace component.

Redundancy

It is achieved via a file replication component.

In the case of failure or heavy load, these components work together to increase data availability by allowing data from multiple places to be logically combined under a single folder known as the "DFS root".

It is not required to use both DFS components simultaneously; the namespace component can be used without the file replication component, and the file replication component can be used between servers without the namespace component.

Features

There are various features of the DFS. Some of them are as follows:

Transparency

There are mainly four types of transparency. These are as follows:

1. Structure Transparency

The client does not need to be aware of the number or location of file servers and storage devices. In structure transparency, multiple file servers must be given to adaptability, dependability, and performance.

2. Naming Transparency

There should be no hint of the file's location in the file's name. When the file is transferred form one node to other, the file name should not be changed.

3. Access Transparency

Local and remote files must be accessible in the same method. The file system must automatically locate the accessed file and deliver it to the client.

4. Replication Transparency

When a file is copied across various nodes, the copies files and their locations must be hidden from one node to the next.

Scalability

The distributed system will inevitably increase over time when more machines are added to the network, or two networks are linked together. A good DFS must be designed to scale rapidly as the system's number of nodes and users increases.

Data Integrity

Many users usually share a file system. The file system needs to secure the integrity of data saved in a transferred file. A concurrency control method must correctly synchronize concurrent access requests from several users who are competing for access to the same file. A file system commonly provides users with atomic transactions that are high-level concurrency management systems for data integrity.

High Reliability

The risk of data loss must be limited as much as feasible in an effective DFS. Users must not feel compelled to make backups of their files due to the system's unreliability. Instead, a file system should back up key files so that they may be restored if the originals are lost. As a high-reliability strategy, many file systems use stable storage.

High Availability

A DFS should be able to function in the case of a partial failure, like a node failure, a storage device crash, and a link failure.

Ease of Use

he UI of a file system in multiprogramming must be simple, and the commands in the file must be minimal.

Performance

The average time it takes to persuade a client is used to assess performance. It must perform similarly to a centralized file system.

Distributed File System Replication

Initial versions of DFS used Microsoft's File Replication Service (FRS), enabling basic file replication among servers. FRS detects new or altered files and distributes the most recent versions of the full file to all servers.

Windows Server 2003 R2 developed the "DFS Replication" (DFSR). It helps to enhance FRS by only copying the parts of files that have changed and reducing network traffic with data compression. It also gives users the ability to control network traffic on a configurable schedule using flexible configuration options.

History of Distributed File System

The DFS's server component was firstly introduced as an additional feature. When it was incorporated into Windows NT 4.0 Server, it was called "DFS 4.1". Later, it was declared a standard component of all Windows 2000 Server editions. Windows NT 4.0 and later versions of Windows have client-side support.

Linux kernels 2.6.14 and later include a DFS-compatible SMB client VFS known as "cifs". DFS is available in versions Mac OS X 10.7 (Lion) and later.

Working of Distributed File System

There are two methods of DFS in which they might be implemented, and these are as follows:

1. Standalone DFS namespace
2. Domain-based DFS namespace

Standalone DFS namespace

It does not use Active Directory and only permits DFS roots that exist on the local system. A Standalone DFS may only be acquired on the systems that created it. It offers no-fault liberation and may not be linked to other DFS.

Domain-based DFS namespace

It stores the DFS configuration in Active Directory and creating namespace root at domainname>dfsroot> or FQDN>dfsroot>.

DFS namespace

SMB routes of the form are used in traditional file shares that are linked to a single server.

\\<SERVER>\<path>\<subpath>

Domain-based DFS file share paths are identified by utilizing the domain name for the server's name throughout the form.

\\<DOMAIN.NAME>\<dfsroot>\<path>

When users access such a share, either directly or through mapping a disk, their computer connects to one of the accessible servers connected with that share, based on rules defined by the network administrator. For example, the default behavior is for users to access the nearest server to them; however, this can be changed to prefer a certain server.

Applications of Distributed File System

There are several applications of the distributed file system. Some of them are as follows:

Hadoop

Hadoop is a collection of open-source software services. It is a software framework that uses the MapReduce programming style to allow distributed storage and management of large amounts of data. Hadoop is made up of a storage component known as Hadoop Distributed File System (HDFS). It is an operational component based on the MapReduce programming model.

NFS (Network File System)

A client-server architecture enables a computer user to store, update, and view files remotely. It is one of various DFS standards for Network-Attached Storage.

SMB (Server Message Block)

IBM developed an SMB protocol to file sharing. It was developed to permit systems to read and write files to a remote host across a LAN. The remote host's directories may be accessed through SMB and are known as "shares".

NetWare

It is an abandon computer network operating system that is developed by Novell, Inc. The IPX network protocol mainly used combined multitasking to execute many services on a computer system.

CIFS (Common Internet File System)

CIFS is an accent of SMB. The CIFS protocol is a Microsoft-designed implementation of the SIMB protocol.

Advantages and Disadvantages of Distributed File System

There are various advantages and disadvantages of the distributed file system. These are as follows:

Advantages

There are various advantages of the distributed file system. Some of the advantages are as follows:

1. It allows the users to access and store the data.
2. It helps to improve the access time, network efficiency, and availability of files.
3. It provides the transparency of data even if the server of disk files.
4. It permits the data to be shared remotely.
5. It helps to enhance the ability to change the amount of data and exchange data.

Disadvantages

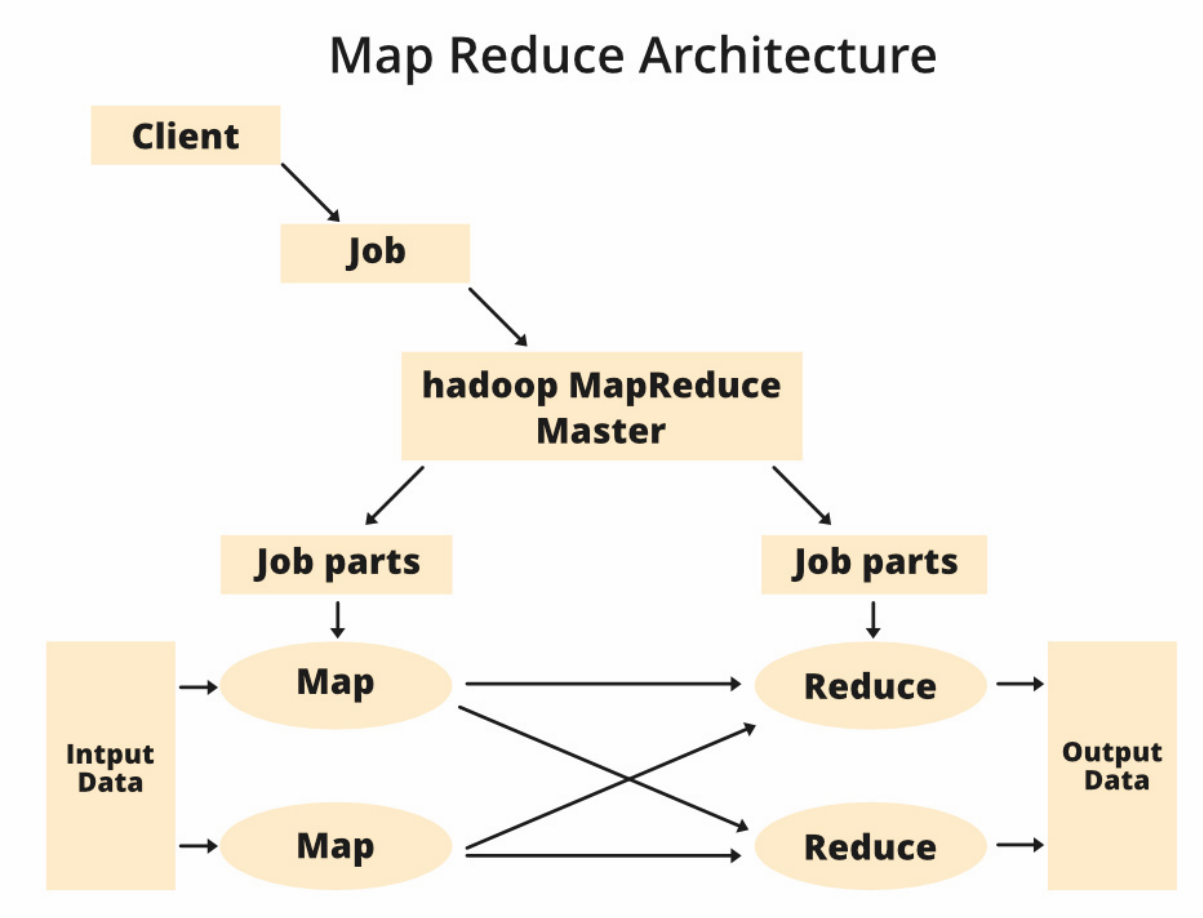
There are various disadvantages of the distributed file system. Some of the disadvantages are as follows:

1. In a DFS, the database connection is complicated.
2. In a DFS, database handling is also more complex than in a single-user system.
3. If all nodes try to transfer data simultaneously, there is a chance that overloading will happen.
4. There is a possibility that messages and data would be missed in the network while moving from one node to another.

MapReduce Architecture

MapReduce and HDFS are the two major components of Hadoop which makes it so powerful and efficient to use. MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages with various different-different optimizations. The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster network and to reduce the processing power. The MapReduce task is mainly divided into two phases Map Phase and Reduce Phase.

MapReduce Architecture:



Components of MapReduce Architecture:

1. **Client:** The MapReduce client is the one who brings the Job to the MapReduce for processing. There can be multiple clients available that continuously send jobs for processing to the Hadoop MapReduce Manager.
2. **Job:** The MapReduce Job is the actual work that the client wanted to do which is comprised of so many smaller tasks that the client wants to process or execute.
3. **Hadoop MapReduce Master:** It divides the particular job into subsequent job-parts.
4. **Job-Parts:** The task or sub-jobs that are obtained after dividing the main job. The result of all the job-parts combined to produce the final output.
5. **Input Data:** The data set that is fed to the MapReduce for processing.
6. **Output Data:** The final result is obtained after the processing.

In **MapReduce**, we have a client. The client will submit the job of a particular size to the Hadoop MapReduce Master. Now, the MapReduce master will divide this job into further equivalent job-parts. These job-parts are then made available for the Map and Reduce Task. This Map and Reduce task will contain the program as per the requirement of the use-case that the particular company is solving. The developer writes their logic to fulfill the requirement that the industry requires. The input data which we are using is then fed to the Map Task and the Map will generate intermediate key-value pair as its output. The output of Map i.e. these key-value pairs are then fed to the Reducer and the final output is stored on the HDFS. There can be n number of Map and Reduce tasks made available for processing the data as per the requirement. The algorithm for Map and Reduce is made with a very optimized way such that the time complexity or space complexity is minimum.

Let's discuss the MapReduce phases to get a better understanding of its architecture:

The MapReduce task is mainly divided into **2 phases** i.e. Map phase and Reduce phase.

1. **Map:** As the name suggests its main use is to map the input data in key-value pairs. The input to the map may be a key-value pair where the key can be the id of some kind of address and value is the actual value that it keeps. The *Map()* function will be executed in its memory repository on each of these input key-value pairs and generates the intermediate key-value pair which works as input for the Reducer or *Reduce()* function.

2. **Reduce:** The intermediate key-value pairs that work as input for Reducer are shuffled and sort and send to the *Reduce()* function. Reducer aggregate or group the data based on its key-value pair as per the reducer algorithm written by the developer.

How Job tracker and the task tracker deal with MapReduce:

1. **Job Tracker:** The work of Job tracker is to manage all the resources and all the jobs across the cluster and also to schedule each map on the Task Tracker running on the same data node since there can be hundreds of data nodes available in the cluster.

2. **Task Tracker:** The Task Tracker can be considered as the actual slaves that are working on the instruction given by the Job Tracker. This Task Tracker is deployed on each of the nodes available in the cluster that executes the Map and Reduce task as instructed by Job Tracker.

There is also one important component of MapReduce Architecture known as **Job History Server**. The Job History Server is a daemon process that saves and stores historical information about the task or application, like the logs which are generated during or after the job execution are stored on Job History Server.

APACHE SPARK

What is Apache Spark?

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing. You'll find it used by organizations from any industry, including at FINRA, Yelp, Zillow, DataXu, Urban Institute, and CrowdStrike.

What is the history of Apache Spark?

Apache Spark started in 2009 as a research project at UC Berkley's AMPLab, a collaboration involving students, researchers, and faculty, focused on data-intensive application domains. The goal of Spark was to create a new framework, optimized for fast iterative processing like machine learning, and interactive data analysis, while retaining the scalability, and fault tolerance of Hadoop MapReduce. The first paper entitled, "Spark: Cluster Computing with Working Sets" was published in June 2010, and Spark was open sourced under a BSD license. In June, 2013, Spark entered incubation status at the Apache Software Foundation (ASF), and established as an Apache Top-Level Project in February, 2014. Spark can run standalone, on Apache Mesos, or most frequently on Apache Hadoop.

How does Apache Spark work?

Hadoop MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm. Developers can write massively parallelized operators, without having to worry about work distribution, and fault tolerance. However, a challenge to MapReduce is the sequential multi-step process it takes to run a job. With each step, MapReduce reads data from the cluster, performs operations, and writes the results back to HDFS. Because each step requires a disk read, and write, MapReduce jobs are slower due to the latency of disk I/O.

Spark was created to address the limitations to MapReduce, by doing processing in-memory, reducing the number of steps in a job, and by reusing data across multiple parallel operations. With Spark, only one-step is needed where data is read into memory, operations performed, and the results written back—resulting in a much faster execution. Spark also reuses data by using an in-memory cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset. Data re-use is accomplished through the creation of DataFrames, an abstraction over Resilient Distributed Dataset (RDD), which is a collection of objects that is cached in memory, and reused in multiple Spark operations. This dramatically lowers the latency making Spark multiple times faster than MapReduce, especially when doing machine learning, and interactive analytics.

Key differences: Apache Spark vs. Apache Hadoop

Outside of the differences in the design of Spark and Hadoop MapReduce, many organizations have found these big data frameworks to be complimentary, using them together to solve a broader business challenge.

Hadoop is an open source framework that has the Hadoop Distributed File System (HDFS) as storage, YARN as a way of managing computing resources used by different applications, and an implementation of the MapReduce programming model as an execution engine. In a typical Hadoop implementation, different execution engines are also deployed such as Spark, Tez, and Presto.

Spark is an open source framework focused on interactive query, machine learning, and real-time workloads. It does not have its own storage system, but runs analytics on other storage systems like HDFS, or other popular stores like Amazon Redshift, Amazon S3, Couchbase, Cassandra, and others. Spark on

Hadoop leverages YARN to share a common cluster and dataset as other Hadoop engines, ensuring consistent levels of service, and response.

What are the benefits of Apache Spark?

There are many benefits of Apache Spark to make it one of the most active projects in the Hadoop ecosystem. These include:

Fast

Through in-memory caching, and optimized query execution, Spark can run fast analytic queries against data of any size.

Developer friendly

Apache Spark natively supports Java, Scala, R, and Python, giving you a variety of languages for building your applications. These APIs make it easy for your developers, because they hide the complexity of distributed processing behind simple, high-level operators that dramatically lowers the amount of code required.

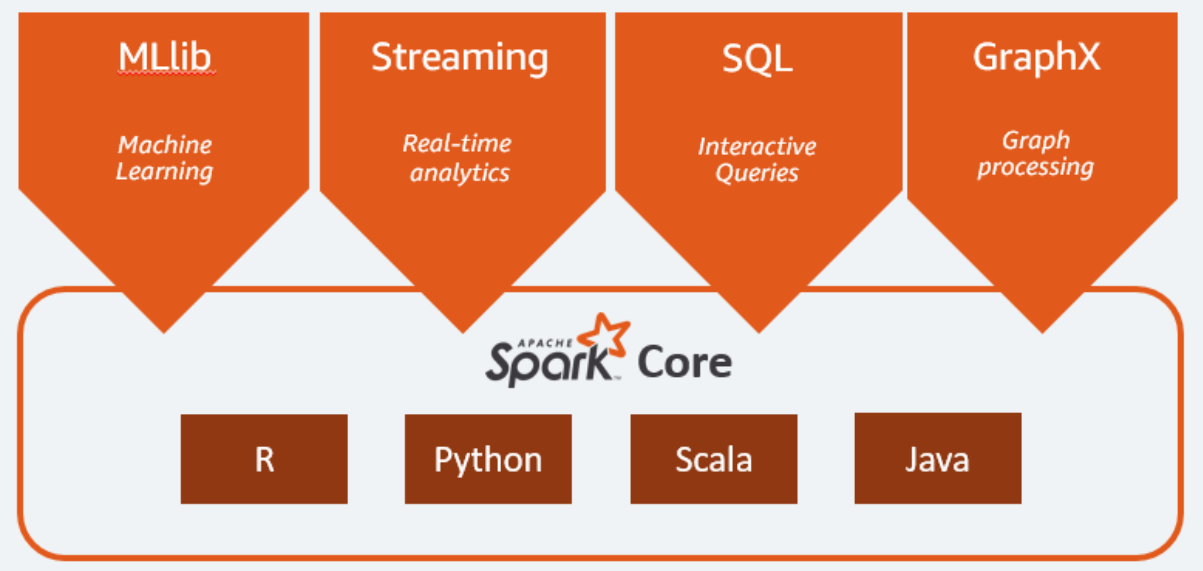
Multiple workloads

Apache Spark comes with the ability to run multiple workloads, including interactive queries, real-time analytics, machine learning, and graph processing. One application can combine multiple workloads seamlessly.

What are Apache Spark Workloads?

The Spark framework includes:

- Spark Core as the foundation for the platform
- Spark SQL for interactive queries
- Spark Streaming for real-time analytics
- Spark MLlib for machine learning
- Spark GraphX for graph processing



Spark Core

Spark Core is the foundation of the platform. It is responsible for memory management, fault recovery, scheduling, distributing & monitoring jobs, and interacting with storage systems. Spark Core is exposed through an application programming interface (APIs) built for Java, Scala, Python and R. These APIs hide the complexity of distributed processing behind simple, high-level operators.

MLlib

Machine Learning

Spark includes MLlib, a library of algorithms to do machine learning on data at scale. Machine Learning models can be trained by data scientists with R or Python on any Hadoop data source, saved using MLlib, and imported into a Java or Scala-based pipeline. Spark was designed for fast, interactive computation that runs in memory, enabling machine learning to run quickly. The algorithms include the ability to do classification, regression, clustering, collaborative filtering, and pattern mining.

Spark Streaming

Real-time

Spark Streaming is a real-time solution that leverages Spark Core's fast scheduling capability to do streaming analytics. It ingests data in mini-batches, and enables analytics on that data with the same application code written for batch analytics. This improves developer productivity, because they can use

the same code for batch processing, and for real-time streaming applications. Spark Streaming supports data from Twitter, Kafka, Flume, HDFS, and ZeroMQ, and many others found from the Spark Packages ecosystem.

Spark SQL

Interactive Queries

Spark SQL is a distributed query engine that provides low-latency, interactive queries up to 100x faster than MapReduce. It includes a cost-based optimizer, columnar storage, and code generation for fast queries, while scaling to thousands of nodes. Business analysts can use standard SQL or the Hive Query Language for querying data. Developers can use APIs, available in Scala, Java, Python, and R. It supports various data sources out-of-the-box including JDBC, ODBC, JSON, HDFS, Hive, ORC, and Parquet. Other popular stores—Amazon Redshift, Amazon S3, Couchbase, Cassandra, MongoDB, Salesforce.com, Elasticsearch, and many others can be found from the Spark Packages ecosystem.

GraphX

Graph Processing

Spark GraphX is a distributed graph processing framework built on top of Spark. GraphX provides ETL, exploratory analysis, and iterative graph computation to enable users to interactively build, and transform a graph data structure at scale. It comes with a highly flexible API, and a selection of distributed Graph algorithms.

What are the use cases of Apache Spark?

Spark is a general-purpose distributed processing system used for big data workloads. It has been deployed in every type of big data use case to detect patterns, and provide real-time insight. Example use cases include:

Financial Services

Spark is used in banking to predict customer churn, and recommend new financial products. In investment banking, Spark is used to analyze stock prices to predict future trends.

Healthcare

Spark is used to build comprehensive patient care, by making data available to front-line health workers for every patient interaction. Spark can also be used to predict/recommend patient treatment.

Manufacturing

Spark is used to eliminate downtime of internet-connected equipment, by recommending when to do preventive maintenance.

Retail

Spark is used to attract, and keep customers through personalized services and offers.

How deploying Apache Spark in the cloud works?

Spark is an ideal workload in the cloud, because the cloud provides performance, scalability, reliability, availability, and massive economies of scale. ESG research found 43% of respondents considering cloud as their primary deployment for Spark. The top reasons customers perceived the cloud as an advantage for Spark are faster time to deployment, better availability, more frequent feature/functionality updates, more elasticity, more geographic coverage, and costs linked to actual utilization.

What are the AWS offerings for Apache Spark?

Amazon EMR is the best place to deploy Apache Spark in the cloud, because it combines the integration and testing rigor of commercial Hadoop & Spark distributions with the scale, simplicity, and cost effectiveness of the cloud. It allows you to launch Spark clusters in minutes without needing to do node provisioning, cluster setup, Spark configuration, or cluster tuning. EMR enables you to provision one, hundreds, or thousands of compute instances in minutes. You can use Auto Scaling to have EMR automatically scale up your Spark clusters to process data of any size, and back down when your job is complete to avoid paying for unused capacity. You can lower your bill by committing to a set term, and saving up to 75% using Amazon EC2 Reserved Instances, or running your clusters on spare AWS compute capacity and saving up to 90% using EC2 Spot.

Apache Spark - RDD

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − **parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

Data Sharing is Slow in MapReduce

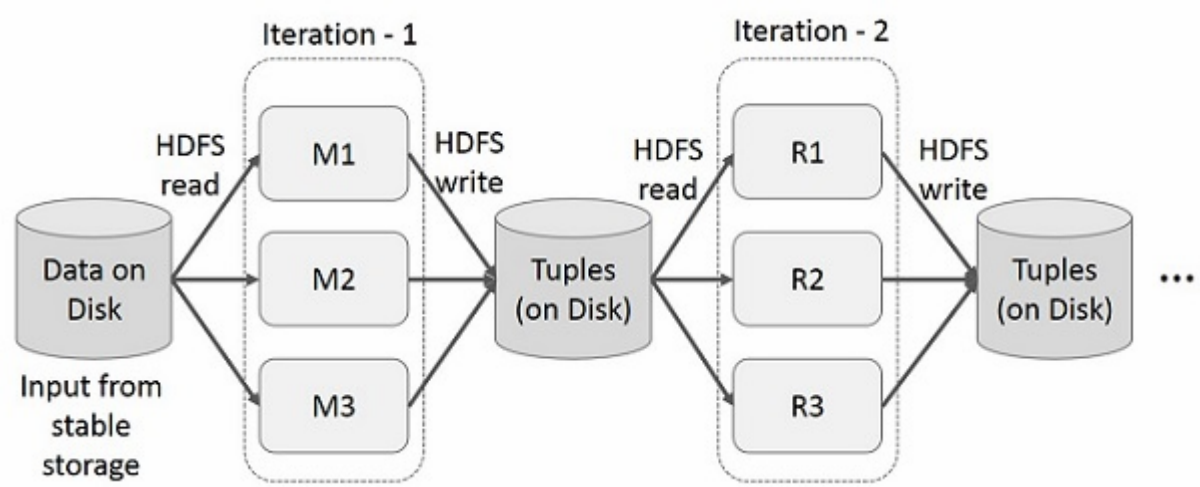
MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

Unfortunately, in most current frameworks, the only way to reuse data between computations (Ex − between two MapReduce jobs) is to write it to an external stable storage system (Ex − HDFS). Although this framework provides numerous abstractions for accessing a cluster's computational resources, users still want more.

Both **Iterative** and **Interactive** applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Regarding storage system, most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Iterative Operations on MapReduce

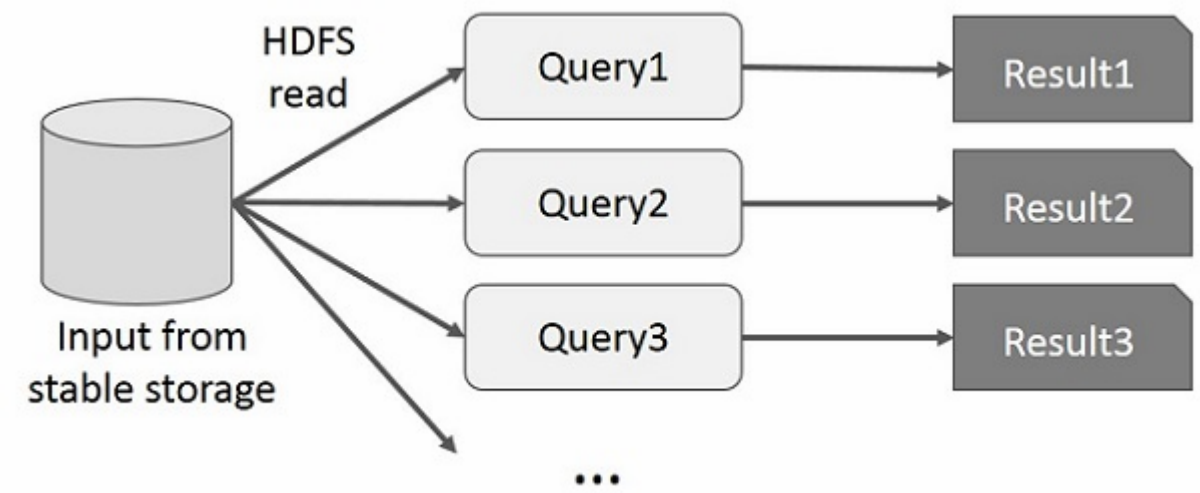
Reuse intermediate results across multiple computations in multi-stage applications. The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



Interactive Operations on MapReduce

User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominate application execution time.

The following illustration explains how the current framework works while doing the interactive queries on MapReduce.



Data Sharing using Spark RDD

Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

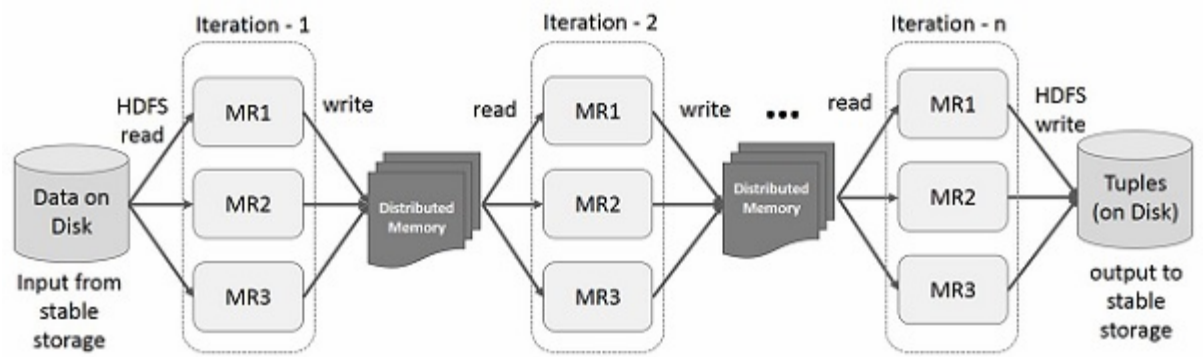
Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is **R**esilient **D**istributed **D**atasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

Let us now try to find out how iterative and interactive operations take place in Spark RDD.

Iterative Operations on Spark RDD

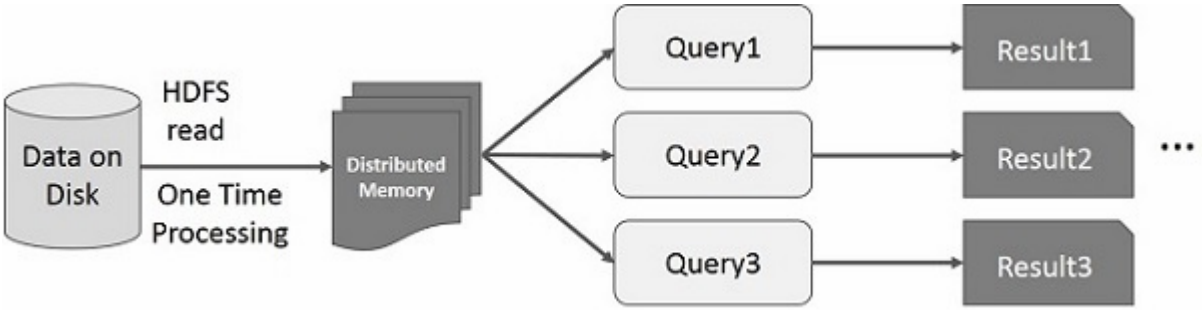
The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

Note − If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.



Interactive Operations on Spark RDD

This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also **persist** an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

Storing and Querying Data.

Storing and Querying Data in Hadoop and Spark

In the Hadoop ecosystem, storing and querying data efficiently is crucial for handling large-scale datasets. Below is an overview of how data storage and querying work in both Hadoop and Spark.

1. Storing Data

Data storage in Hadoop is primarily handled by **HDFS (Hadoop Distributed File System)**, while Spark relies on both HDFS and other distributed storage systems.

A. Hadoop Distributed File System (HDFS)

- HDFS is a distributed storage system that stores data across multiple machines in a Hadoop cluster.
- It follows a **Master-Slave Architecture**:
    - **NameNode** (Master) manages metadata (directory structure, file permissions, and block locations).
    - **DataNodes** (Slaves) store actual data blocks.
- Files are divided into **blocks (default 128MB or 256MB)** and replicated across multiple nodes for fault tolerance.

- HDFS supports **batch processing** and is optimized for large files (not small files).

B. Other Storage Options in Hadoop and Spark

- **Apache HBase**: A NoSQL database for real-time read/write access to large datasets.
- **Apache Hive**: A data warehouse that stores structured data and allows SQL-like querying.
- **Apache ORC and Parquet**: Columnar storage formats that optimize storage and query performance.
- **Amazon S3, Google Cloud Storage, and Azure Blob Storage**: Cloud-based storage options that integrate with Hadoop and Spark.

2. Querying Data

Querying data in the Hadoop ecosystem can be done using different tools depending on the use case:

A. Querying in Hadoop

1. **Apache Hive**
    - SQL-like querying language called **HiveQL**.
    - Converts queries into **MapReduce, Tez, or Spark jobs**.
    - Best suited for batch processing of structured data.
2. **Apache Pig**
    - A high-level scripting language (Pig Latin) for data transformation.
    - Converts Pig scripts into **MapReduce jobs**.
3. **Apache HBase**
    - NoSQL database with fast read/write operations.
    - Supports key-value lookups but does not support SQL natively.

B. Querying in Spark

1. **Spark SQL**
    - Allows querying structured data using **SQL and DataFrames**.
    - Supports **JDBC/ODBC** for database connectivity.
    - Can read from multiple sources: HDFS, Hive, Parquet, JSON, and relational databases.
2. **RDD (Resilient Distributed Dataset) Queries**
    - Provides transformations (map, filter, reduceByKey) for querying unstructured data.
    - Less optimized than DataFrames or Spark SQL but gives more control.
3. **DataFrames and Datasets**
    - More efficient than RDDs.
    - Uses **Catalyst Optimizer** for query optimization.
4. **Spark Streaming**
    - Allows querying real-time data streams from Kafka, Flume, and other sources.

Comparison: Hadoop vs. Spark for Storing & Querying

Feature	**Hadoop (HDFS, Hive)**	**Spark (RDD, DataFrames, SQL)**
Storage	HDFS (Batch-oriented)	HDFS, S3, HBase, Parquet
Query Language	HiveQL, Pig Latin	SQL, DataFrames, RDD transformations
Speed	Slower (MapReduce-based)	Faster (In-memory processing)
Use Case	Large-scale batch processing	Real-time & batch processing

Conclusion

- **HDFS** is the core storage system in Hadoop, but other storage solutions like HBase, ORC, and Parquet are also used.
- **Querying in Hadoop** is mainly done through Hive, Pig, and HBase.
- **Spark SQL and DataFrames** provide an efficient way to query data with better performance than traditional MapReduce.

Examples of Querying Data in Hive and Spark SQL

Below are practical examples of querying data in **Apache Hive** and **Apache Spark SQL**.

1. Querying Data in Apache Hive

Step 1: Create a Table in Hive

Hive stores data in **HDFS** and allows querying with **HiveQL** (SQL-like syntax).

sql

CopyEdit

```
CREATE TABLE employees (
    emp_id INT,
    name STRING,
    department STRING,
    salary FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Step 2: Load Data into Hive

Assume we have a CSV file (employees.csv) stored in HDFS:

CopyEdit

```
1, John, IT, 50000
```

2, Alice, HR, 60000

3, Bob, Finance, 70000

We load the data into the Hive table:

sql

CopyEdit

LOAD DATA INPATH '/user/hadoop/employees.csv' INTO TABLE employees;

Step 3: Run Queries

Query for employees with a salary greater than 55,000:

sql

CopyEdit

SELECT * FROM employees WHERE salary > 55000;

Aggregate query: Find the **average salary per department**:

sql

CopyEdit

SELECT department, AVG(salary) AS avg_salary

FROM employees

GROUP BY department;

2. Querying Data in Spark SQL

Apache Spark allows querying structured data using **DataFrames and SQL**.

Step 1: Load Data into Spark

Assume the same employees.csv file.

Using Spark DataFrame API (Python Example)

python

CopyEdit

from pyspark.sql import SparkSession

```
# Initialize Spark Session
spark = SparkSession.builder.appName("SparkSQLExample").getOrCreate()

# Load CSV data into DataFrame
df = spark.read.option("header",
"true").csv("hdfs:///user/hadoop/employees.csv", inferSchema=True)

# Show Data
df.show()
```

Step 2: Register as Temporary Table

python

CopyEdit

```
df.createOrReplaceTempView("employees")
```

Step 3: Run SQL Queries

Query for employees with a salary greater than 55,000:

python

CopyEdit

```
result = spark.sql("SELECT * FROM employees WHERE salary > 55000")
result.show()
```

Find **average salary per department**:

python

CopyEdit

```
result = spark.sql("SELECT department, AVG(salary) AS avg_salary FROM
employees GROUP BY department")
result.show()
```

Comparison of Hive and Spark SQL

Feature	Apache Hive	Apache Spark SQL
Processing Speed	Slower (MapReduce-based)	Faster (In-memory processing)
Query Language	HiveQL (SQL-like)	SQL/DataFrame API
Use Case	Batch processing	Batch & real-time processing
Storage	HDFS	HDFS, S3, HBase, etc.

Conclusion

- **Hive** is suitable for batch queries over large datasets stored in **HDFS**.
- **Spark SQL** is **faster** and allows querying using both SQL and **DataFrames**.
- **Spark** is preferred for real-time and interactive data analysis.