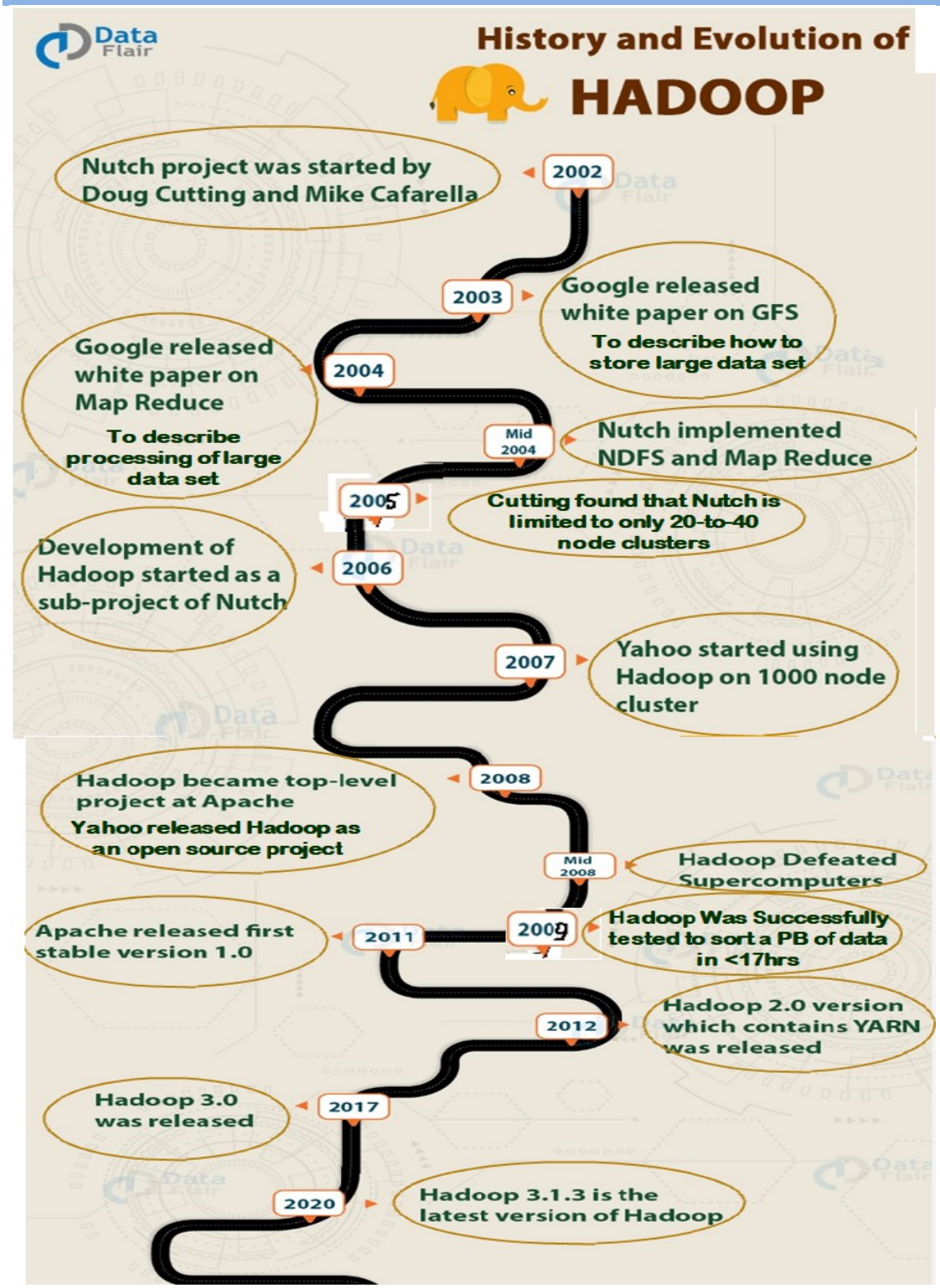
HADOOP

Syllabus:

Hadoop: *Features, Advantages of Hadoop, Versions of Hadoop, Hadoop ecosystem, Hadoop distributions, Hadoop Vs SQL. Introduction to Hadoop: Why Hadoop? RDBMS Vs Hadoop, Distributed computing challenges, History of Hadoop, Hadoop overview, use case of Hadoop, HDFS, Processing data with Hadoop, Managing resources and applications with Hadoop YARN.*

History of Hadoop



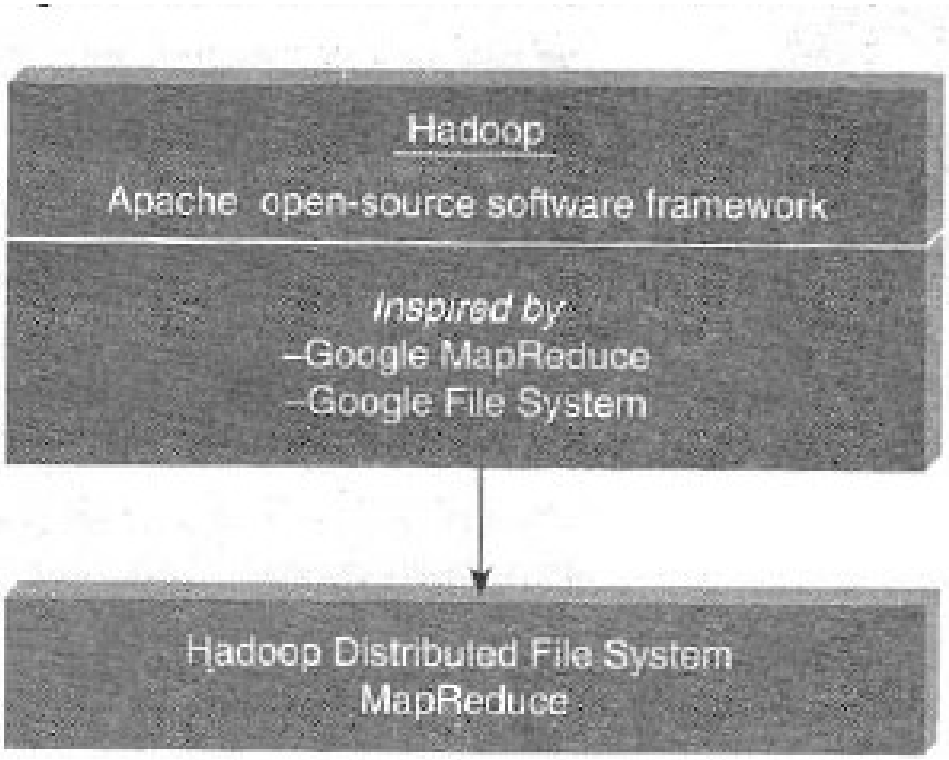


Figure 4.8 Hadoop.

- Hadoop is an **open-source software framework** for storing and processing **large datasets** ranging in size from **gigabytes to petabytes and more**.
- Hadoop was developed at the **Apache Software Foundation**.
- Hadoop is a framework **written in Java**, originally developed **by Doug Cutting in 2006** who named it after his son's toy elephant. He was working with Yahoo then.
- It was created to support distribution for **"Nutch",** the text search engine.
- Hadoop **uses Google's MapReduce and Google File System** technologies as its foundation.
- In 2008, Hadoop defeated the supercomputers and became the fastest system on the planet for sorting terabytes of data.
- Hadoop is now a **core part** of the computing infrastructure for companies such as **Yahoo, Facebook, LinkedIn, Twitter,** etc.

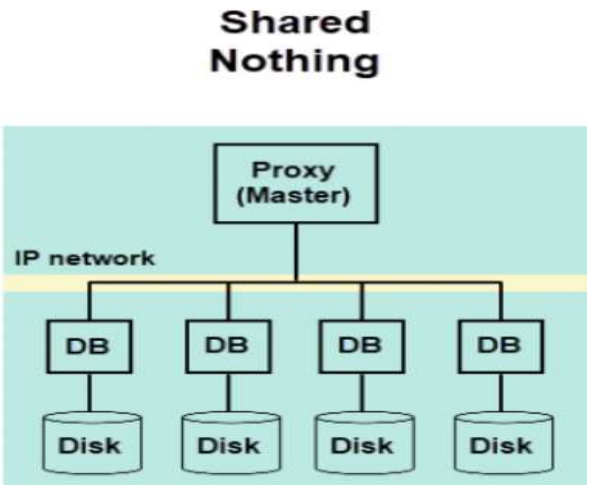
Hadoop Overview

- Hadoop Ecosystem is **neither a programming language nor a service**, it is a platform or framework which solves big data problems. You can consider **it as a suite** which encompasses a number of services (**ingesting, storing, analyzing and maintaining**) inside it.
- Apache Hadoop is an open source, Java-based software platform that **manages data processing and storage for big data applications.**
- Hadoop works by distributing large data sets and analytics jobs across nodes in a computing cluster, breaking them down into smaller workloads that can be run in parallel.
- Hadoop can process structured and unstructured data and scale up reliably from a single server to thousands of machines.
- It's distributed file system has the provision of rapid data transfer rates among nodes. It also allows the system to continue operating in case of node failure.

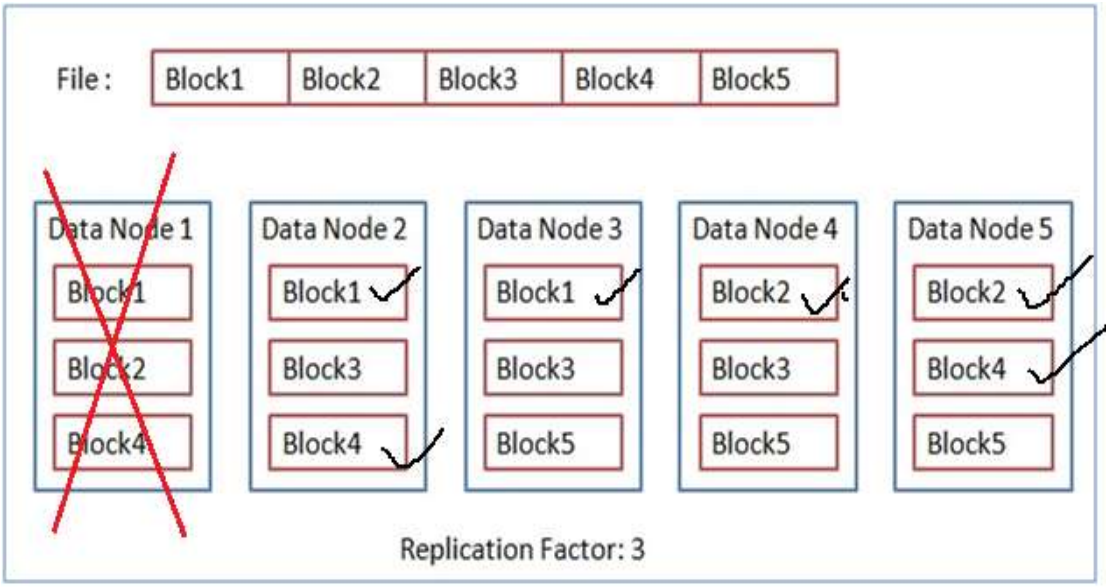
- Hadoop provides-
  - The Storage layer – HDFS
  - Batch processing engine – MapReduce
  - Resource Management Layer – YARN

Features of Hadoop

1. **Optimized for Massive Data**: Hadoop is designed to handle large amounts of data, including structured, semi-structured, and unstructured data. It uses commodity hardware, meaning inexpensive, off-the-shelf servers instead of specialized, high-end systems.
2. **Shared Nothing Architecture**: Hadoop uses a distributed architecture where each node operates independently without relying on shared resources, preventing bottlenecks and improving scalability.



3. Data Replication: Hadoop ensures fault tolerance by replicating data across multiple machines. If one machine fails, the data can still be accessed from other machines storing its replica.



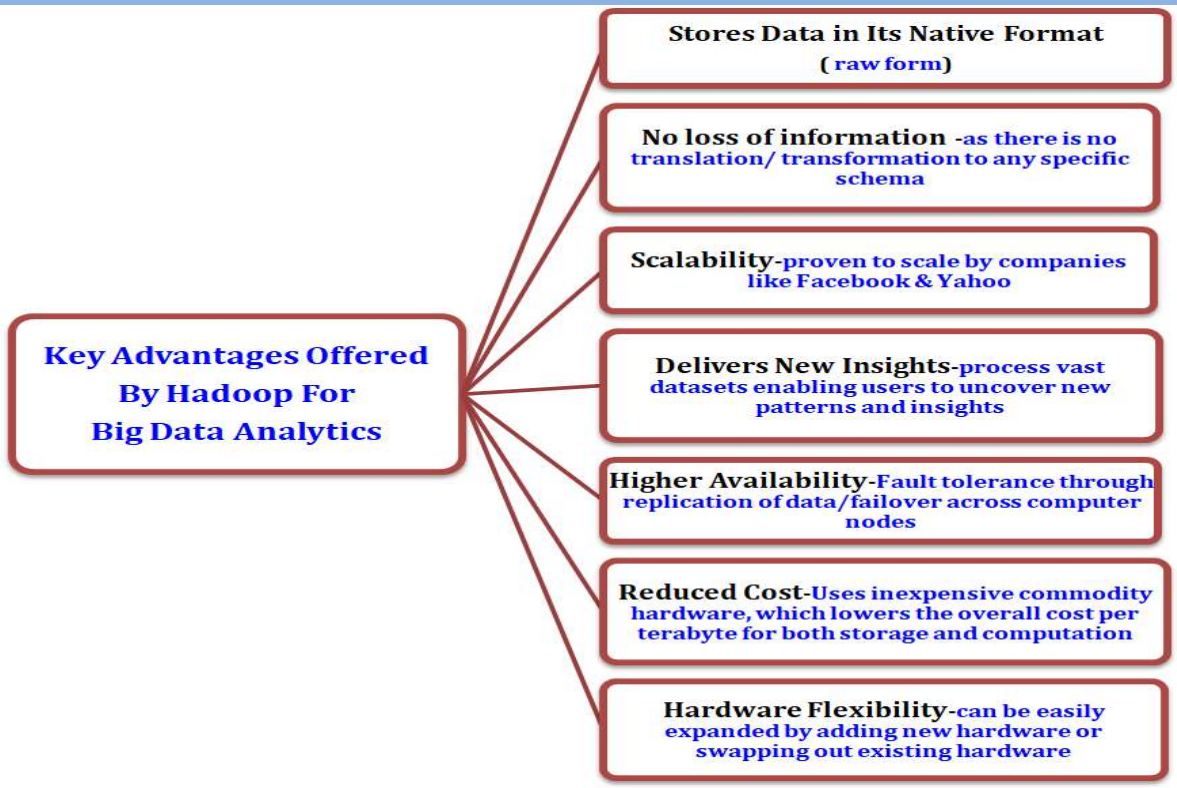
4. **High Throughput over Low Latency**: Hadoop is built for processing large amounts of data in bulk (batch processing). While it can handle high volumes, it is not optimized for low-latency tasks requiring immediate response.
5. **OLTP and OLAP Complement**: Hadoop complements both **On-Line Transaction Processing (OLTP)** and **On-Line Analytical Processing (OLAP)** systems. However, it is not a replacement for traditional relational database management systems (RDBMS) that are

typically used for OLTP tasks.

6. **Limitations with Non-Parallelizable Work**: Hadoop struggles with tasks that cannot be easily broken down into smaller, independent units (i.e., when the tasks have dependencies). Parallelization is key to Hadoop's efficiency, and when that's not possible, its performance suffers.
7. **Not Suitable for Small Files**: Hadoop is not optimized for processing small files. It works best with large datasets and files due to its overhead in managing data distribution and replication across its clusters.

These points describe how Hadoop is ideal for big data analytics but has limitations in areas like real-time data processing or when dealing with small datasets.

Advantages of Hadoop



The Figure outlines the key advantages offered by Hadoop for big data analytics.

1. **Stores Data in Its Native Format**: Hadoop allows you to store data in its raw form, without needing to preprocess or transform it into a specific format. This flexibility makes it ideal for handling a wide variety of data types (structured, semi-structured, and unstructured).

2. **No Loss of Information**: since data is not transformed or forced into a specific schema, there is any risk of losing information. This ensures that all details, regardless of format, are retained for analysis.

3. **Scalability**: Hadoop is proven to scale well, even when handling massive amounts of data. Large companies like Facebook and Yahoo have successfully demonstrated how Hadoop can scale horizontally by adding more commodity servers to handle increasing data loads without compromising performance.

4. **Delivers New Insights**: Hadoop's ability to process vast datasets enables users to uncover new patterns and insights that may not be possible with smaller datasets or traditional systems.

5. **Higher Availability**: Hadoop ensures high availability through data replication. If one node in the cluster fails, another node with the data replica can continue processing, offering fault tolerance and minimizing downtime.

6. **Reduced Cost**: Hadoop offers a cost-effective solution for storing and processing data. It uses inexpensive commodity hardware, which lowers the overall cost per terabyte for both storage and computation.

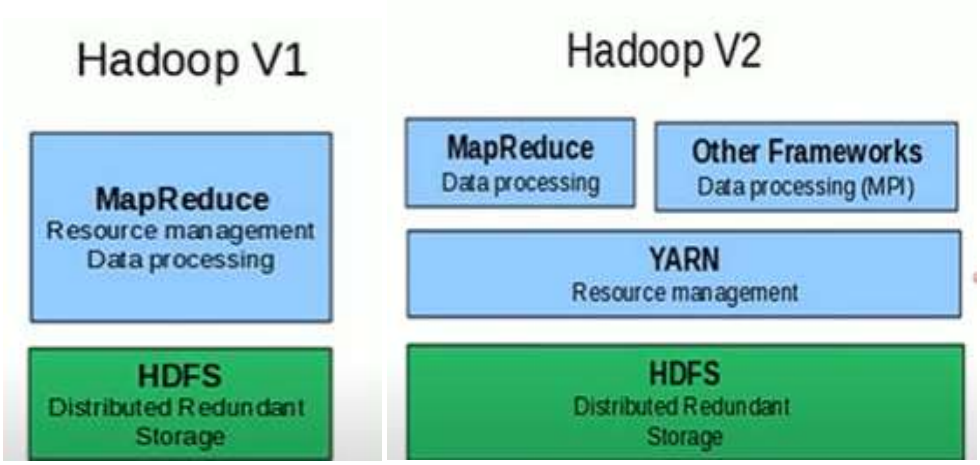
7. **Hardware Flexibility**: Hadoop clusters can be easily expanded by adding new hardware or swapping out existing hardware, making it highly flexible and adaptable as needs grow or hardware becomes obsolete.

These advantages make Hadoop a powerful tool for organizations dealing with large-scale data, enabling cost-efficient, scalable, and flexible big data analytics.

Versions of Hadoop

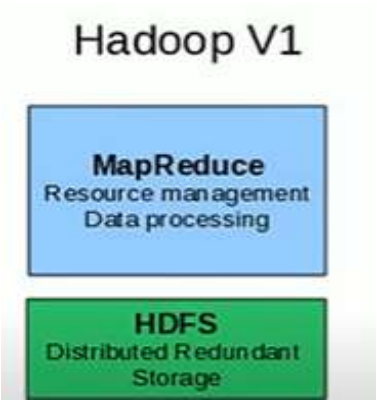
Two versions of Hadoop:

1. Hadoop 1.0
2. Hadoop 2.0



Hadoop 1.0

This version of Hadoop consists of 2 main parts:



1. **Data storage framework**
   - It is a general-purpose file system called Hadoop Distributed File System(**HDFS**).
   - It is schema-less
   - It simply stores data files(in any format)
2. **Data processing framework**
   - It is a simple functional programming model initially popularized by Google as **MapReduce.**
   - It is a programming model and processing engine that

divides a task into small sub-tasks (Map phase) and then combines the results (Reduce phase).

- It processes data in parallel on nodes that store the data, minimizing data movement.
- It uses two functions: the **MAP** and the **REDUCE** functions to process data.
  - **MAP**- Breaks down and maps data by splitting it into smaller chunks and processing them in parallel. The Map function takes input as key-value pairs, processes them, and produces another set of key-value pairs as output.
  - **REDUCE**- Combines and reduces the data by aggregating the data from the Map set. The Reduce function also takes input as key-value pairs, and produces key-value pairs as output.

The two functions seemingly work in isolation from one another, thus enabling the processing to be highly distributed in a highly-parallel, fault-tolerant and scalable way.

Hadoop 1.0- Limitations

1. MapReduce Programming Expertise:
Hadoop 1.x required significant proficiency in MapReduce programming, along with Java expertise, which made it challenging for those without such skills.

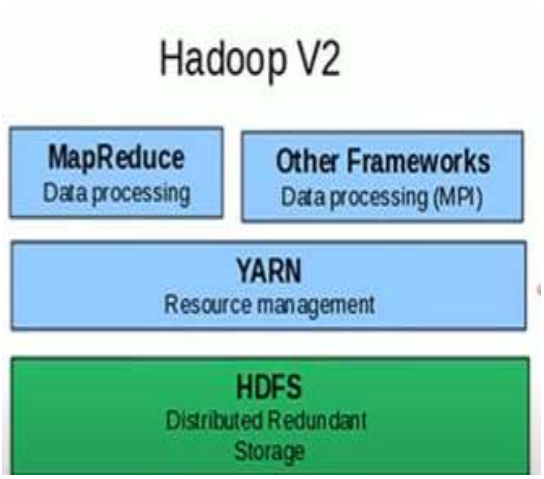
2. Batch Processing Only:
Hadoop 1.x was limited to batch processing, which was suitable for specific tasks like log analysis and large-scale data mining, but inadequate for other real-time or interactive processing needs.

3. Scalability Issues:
Due to reliance on a single JobTracker, Hadoop 1.x can have performance bottlenecks as the number of tasks and nodes increases.

4. Single Point of Failure:
The single Namenode and JobTracker present potential points of failure, meaning that if these components crash, the system might become unusable.

5. Fixed Slot Allocation:
The MapReduce tasks have fixed map and reduce slots per node, which could lead to inefficient resource utilization since unused slots cannot be dynamically allocated between map and reduce tasks.

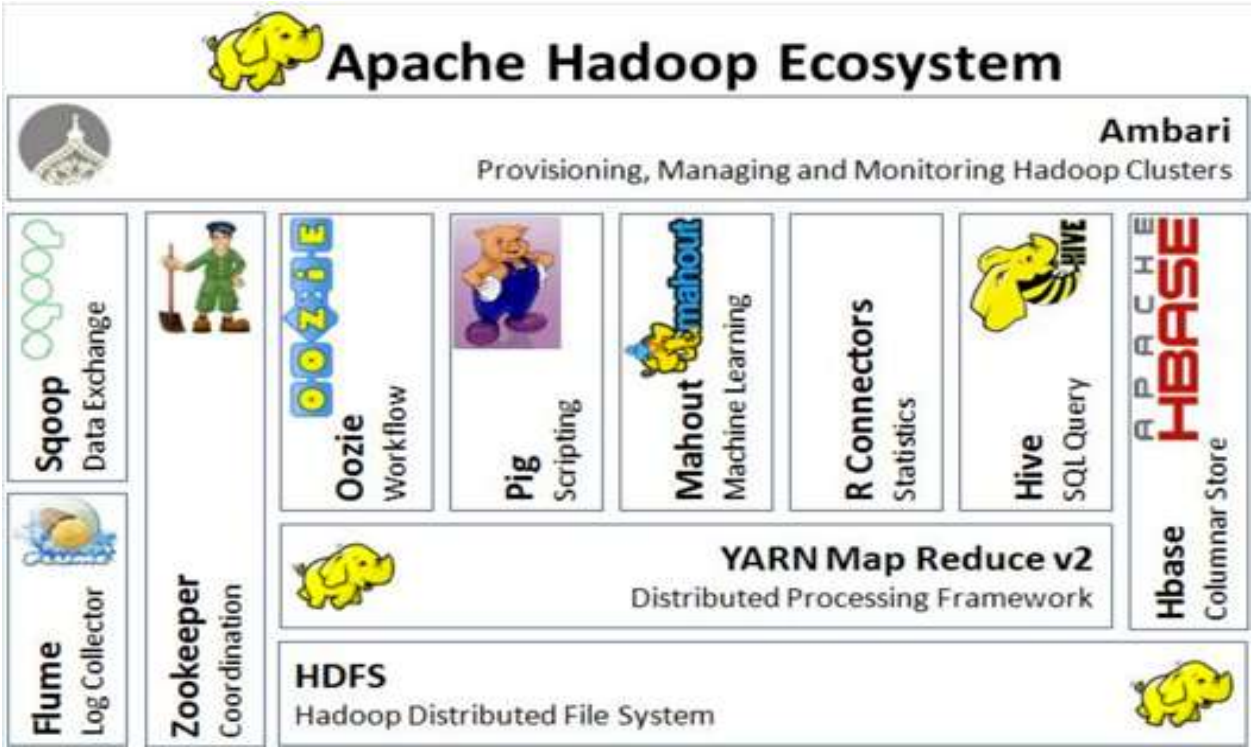
Hadoop 2.0



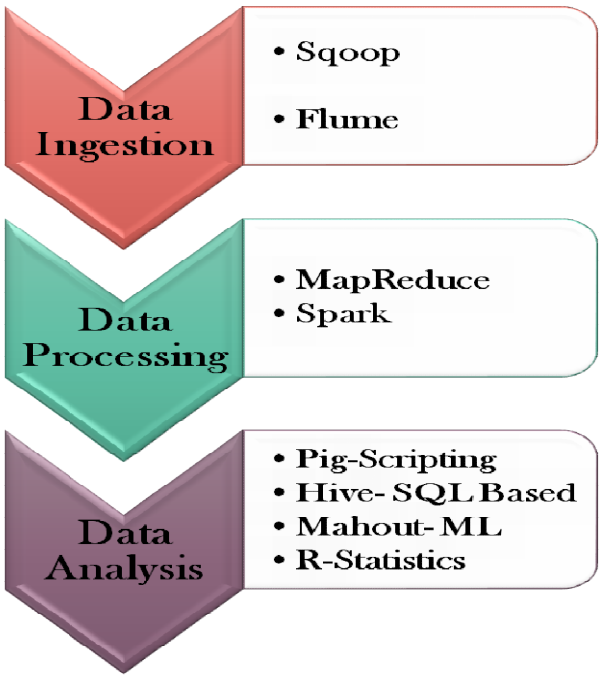
- Hadoop 2.0 is a major revision to the Hadoop framework, introducing significant improvements over Hadoop 1.0.
- HDFS continues to be the data storage framework.
- A new and separate resource management framework called YARN (Yet Another Resource Negotiator) has been added.
- YARN/MapReduce2 has been introduced in Hadoop 2.0.
- It is a layer that separates the resource management layer and the processing components layer.

- MapReduce2 moves Resource management (like infrastructure to monitor nodes, allocate resources and schedule jobs)into YARN
- Any Application capable of dividing itself into parallel tasks is supported by YARN.
- YARN coordinates the allocation of sub-taks of the submitted application, thereby further enhancing the flexibility, scalability and efficiency of the applications.
- It works by having an **ApplicationMaster** in place of the **JobTracker**, running applications on resources governed by a new **NodeManager**.
- **ApplicationMaster** is able to run any application and not just MapReduce. This, means, that the MapReduce Programming expertise is no longer required.
- It not only supports batch processing, but also real-time processing.
- MapReduce is no longer the only data processing option, other alternative data processing functions such as data standardization, master data management can now be performed natively in HDFS.

Overview of Hadoop ecosystem



- ***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.
- There are components available in Hadoop ecosystem for **Data Ingestion, Data Processing, Data Analysis**.

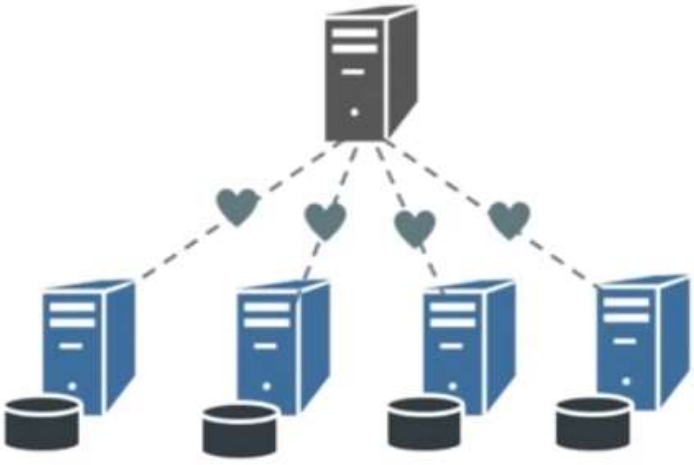


Storage Components:

1. HDFS (Hadoop Distributed File System)

2. HBase

HDFS (Hadoop Distributed File System):

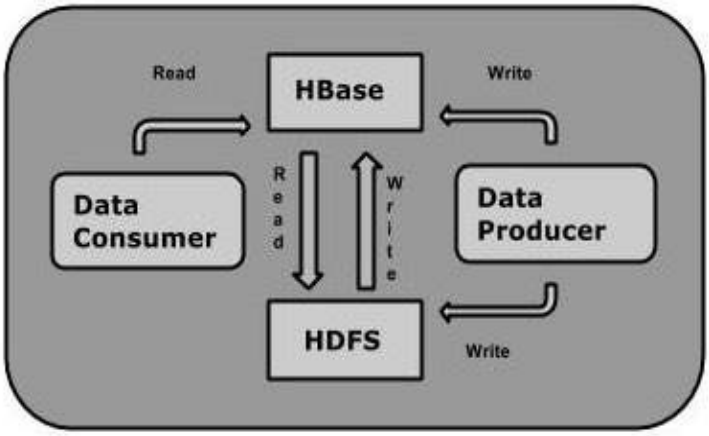


- It is the distributed storage unit of Hadoop.
- It provides streaming access to file system data as well as file permissions and authentication.
- It is based on GFS (Google File System).
- It is used to scale a single cluster node to hundreds and thousands of nodes.
- It handles large datasets running on commodity hardware.
- HDFS is highly fault-tolerant. It stores files across multiple machines. These files are stored in redundant fashion to allow for data recovery in case of failure.

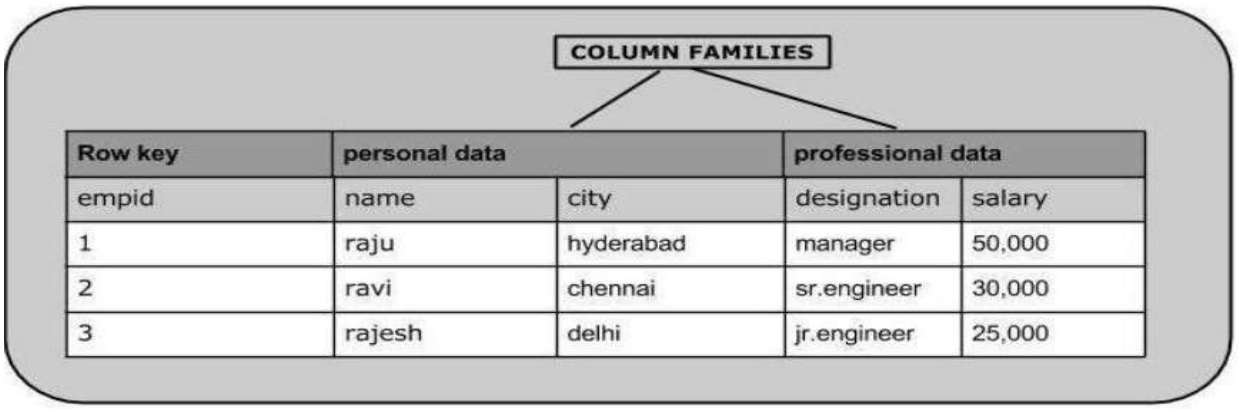
HBase:

- HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.
- It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.
- One can store the data in HDFS either directly or through HBase. Data consumer

reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



- It is based on Google BigTable.
- This is widely used by Facebook, Twitter, Yahoo, etc.
- It can be accessed by Apache Hive, Apache Pig, MapReduce, and store information in HDFS.

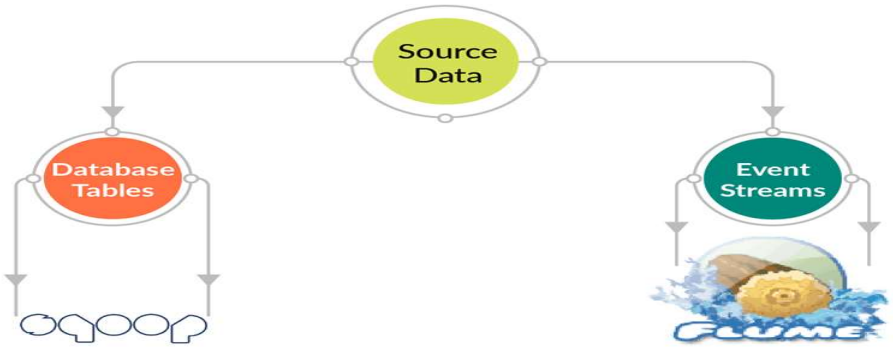


Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

- **HBase** is a **column-oriented database** and the tables in it are sorted by row.
- The table schema defines only column families, which are the **key value pairs**.
- A table have multiple column families and each column family can have any number of columns.

Data Ingestion

- **Data ingestion** in Hadoop is the process of **moving data from various sources into Hadoop**, a distributed file system that's optimized for handling large files.
- **Apache Sqoop** and **Apache Flume** are both open source tools that help with data ingestion in Hadoop. Both tools can be used to load data into HDFS, but **they differ in the types of data they ingest** and how they do it.



1. Sqoop:

- Sqoop stands for SQL to Hadoop.
- Its main functions are:
  a) Importing data from RDBMS such as MySQL, Oracle, DB2, etc. to Hadoop file system (HDFS, HBase, Hive).
  b) Exporting data from Hadoop File system (HDFS, HBase, Hive) to RDBMS (MySQL, Oracle, DB2).
- **Uses of Sqoop:**
  a) It has a connector-based architecture to allow plug-ins to connect to external systems such as MySQL, Oracle, DB2, etc.
  b) It can provision the data from external system on to HDFS and populate tables in Hive and HBase.
  c) It integrates with Oozie allowing you to schedule and automate import and export tasks.

2. Flume:

- **Flume** efficiently ingests log data from many online sources and web servers into a centralized storage system (HDFS, HBase).
- Flume is also used to ingest massive amounts of event data produced by social networking sites like Facebook and Twitter and e-commerce sites like Amazon and Flipkart and log files into Hadoop.
- Flume has been **developed by Cloudera**.
- The default destination in Flume (called as sink in flume parlance) is HDFS.

Data Processing

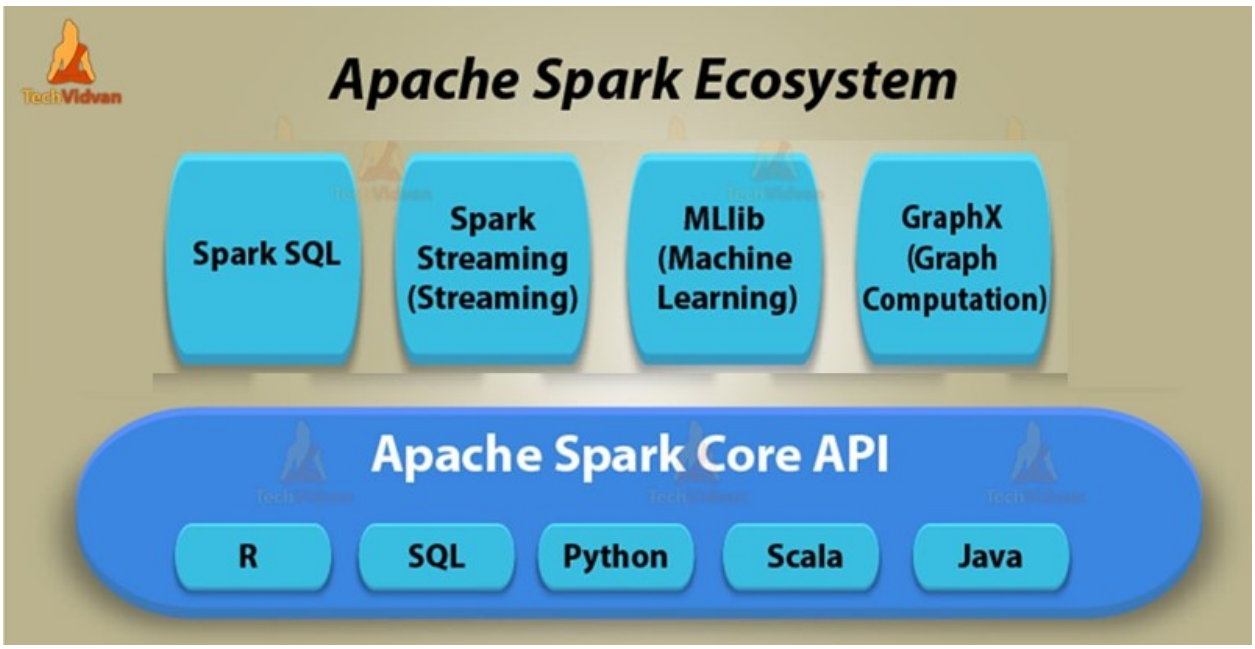
1. MapReduce:

- It is a **programming paradigm** that allows distributed and parallel processing of huge datasets.
- It is based on Google MapReduce.
- Google released a paper on MapReduce programming paradigm in 2004 and that became the genesis of Hadoop processing model.
- The MapReduce framework gets the input data from HDFS.
- There are two main phases: **Map phase** and **the Reduce phase**.

- The **map phase converts** the input data into another set of data (key-value pairs).
- This **new intermediate** dataset then serves as the input to the reduce phase.
- The **reduce phase** acts on the datasets **to combine** (aggregate and consolidate) and reduce them to a smaller set of tuples.
- The **result is then stored back in HDFS.**

2. Spark:

- It is both a programming model as well as a computing model. It is an open-source big data processing framework. It was originally developed in 2009 at UC Berkeley's AmpLab and became an open-source project in 2010. It is written in Scala. It provides in-memory computing for Hadoop.
- In Spark, workloads execute in memory rather than on disk owing to which it is much faster (10 to 100 times) than when the workload is executed on disk. However, if the datasets are too large to fit into the available system memory, it can perform conventional disk-based processing.
- It serves as a potentially faster and more flexible alternative to MapReduce. It accesses data from HDFS (Spark does not have its own distributed file system) but bypasses the MapReduce processing.
- Spark can be used with Hadoop coexisting smoothly with MapReduce (sitting on top of Hadoop YARN) or used independently of Hadoop (standalone).
- As a programming model, it works well with Scala, Python (it has API connectors for using it with Java or Python) or R programming language.
- The following are the Spark libraries:



a) Spark SQL: Spark also has support for SQL. Spark SQL uses SQL to help query data stored in disparate applications.

b) Spark streaming: It helps to analyze and present data in real time.

c) MLib: It supports machine learning such as applying advanced statistical operations on data in Spark Cluster.

d) GraphX: It helps in graph parallel computation.

Data analysis

1. Pig:

- It is a high-level scripting language used with Hadoop.
- It serves as an alternative to MapReduce.
- It has two parts:

(a) Pig Latin

- It is SQL-like scripting language. Pig Latin scripts are translated into MapReduce jobs which can then run on YARN and process data in the HDFS cluster.
- It was initially developed by Yahoo. It is immensely popular with developers who are not comfortable with MapReduce. However, SQL developers may have a preference for Hive.
- There is a "Load" command available to load the data from "HDFS" into Pig. Then one can perform functions such as grouping, filtering, sorting, joining etc.
- The processed or computed data can then be either displayed on screen or placed back into HDFS. It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analyzing huge data sets.

(b) Pig runtime- It is the runtime environment.

2. Hive:

- Hive is a data warehouse software project built on top of Hadoop.
- Three main tasks performed by Hive are summarization, querying and analysis.
- It supports queries written in a language called HQL or HiveQL which is a declarative SQL-like language.
- It converts the SQL-style queries into MapReduce jobs which are then executed on the Hadoop platform.

Hadoop ecosystem other components

Impala:

- It is a high performance SQL engine that runs on Hadoop cluster.
- It is ideal for interactive analysis.
- It has very low latency measured in milliseconds.
- It supports a dialect of SQL called Impala SQL.

ZooKeeper-is a coordination service for distributed applications.

Oozie-It is a workflow scheduler system to manage Apache Hadoop jobs.

Mahout-It is a scalable machine learning and data mining library.

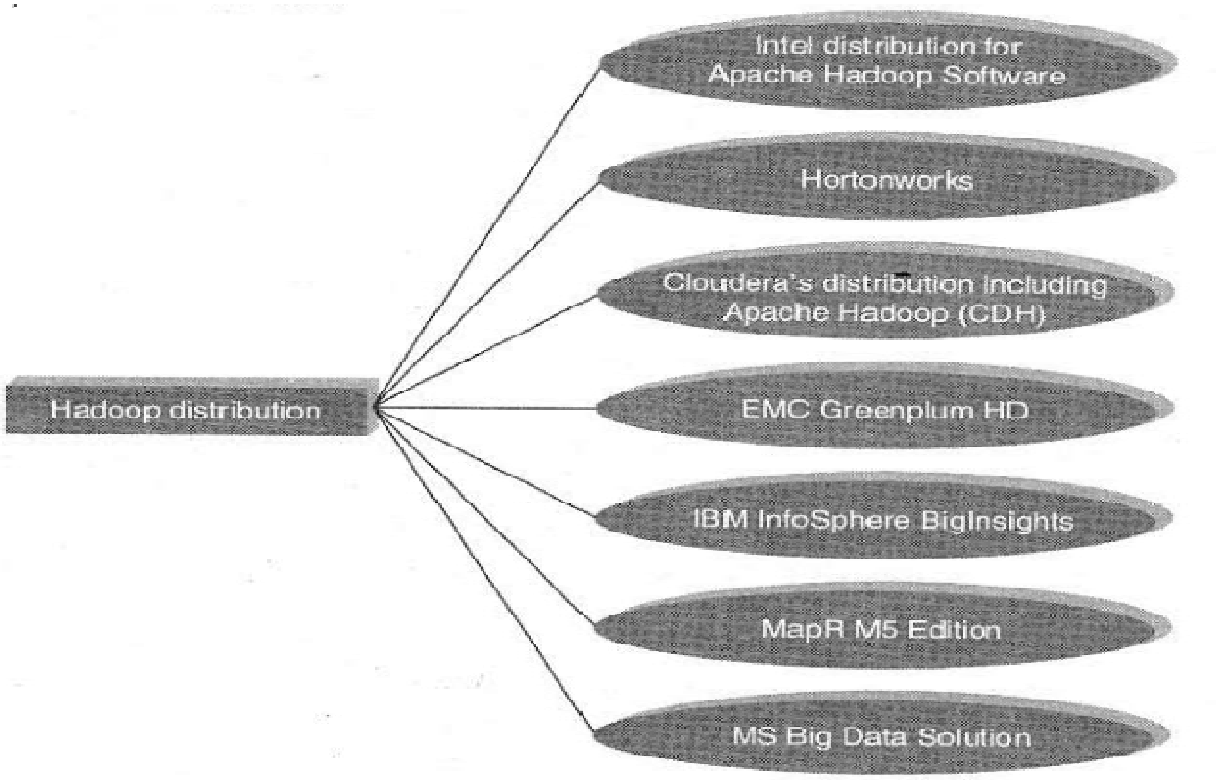
Chukwa-is a data collection system for managing large distributed systems.

Ambari-is a web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters.

Hadoop distributions

Hadoop is an open-source Apache project. Anyone can freely download the core aspects of Hadoop. The core aspects of Hadoop include the following:

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



Hortonworks: A well-known Hadoop distribution, which was later merged into Cloudera. It focused on open-source solutions.

Cloudera's distribution including Apache Hadoop (CDH): Cloudera's distribution that includes Hadoop and other tools for big data processing and analytics. It is one of the most popular distributions.

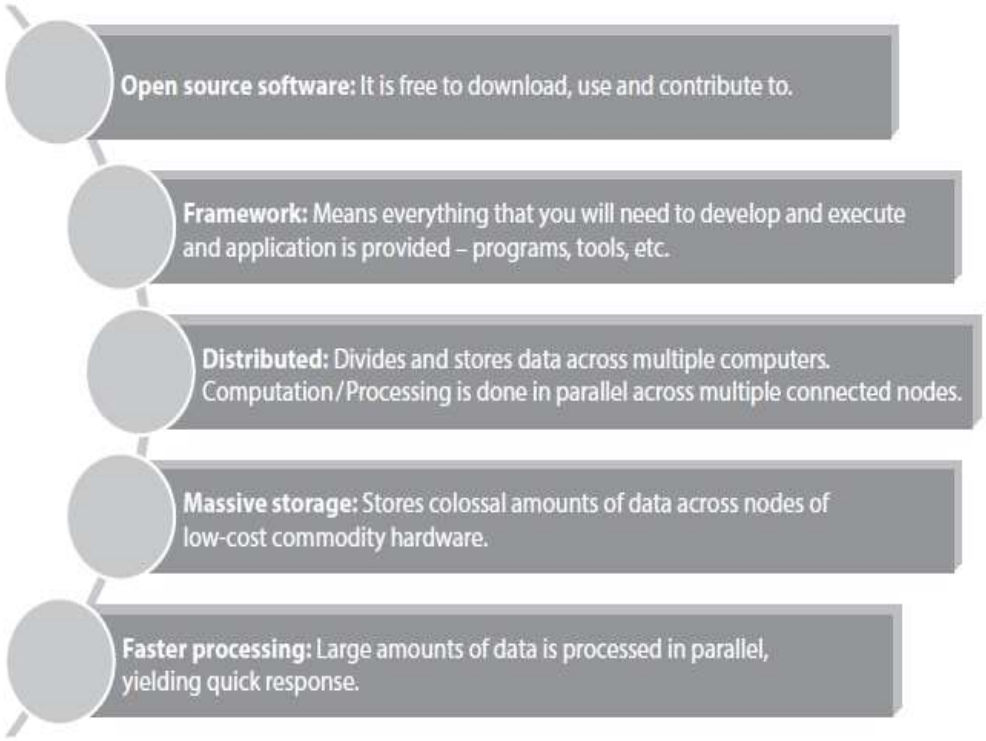
EMC Greenplum HD: A Hadoop distribution by EMC (now part of Dell), integrated with their Greenplum database solution.

IBM InfoSphere BigInsights: IBM's Hadoop distribution that integrates with their InfoSphere suite, aimed at enterprise-level big data processing.

MapR M5 Edition: A Hadoop distribution provided by MapR (now part of HPE), known for its high performance and reliability.

MS Big Data Solution: Microsoft's Hadoop distribution, integrated with its Azure cloud platform for scalable big data processing.

Key Aspects of Hadoop



Key Considerations of Hadoop

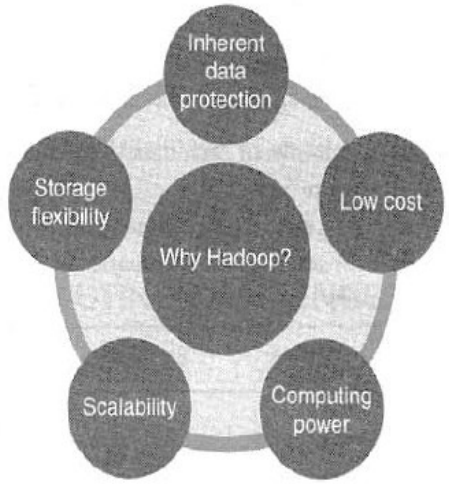


Figure 5.2 Key considerations of Hadoop.

The key consideration is its capability to **handle massive amounts of data, different categories of data - fairly quickly.**

The other considerations are (Figure 5.2):

1. Low cost: Hadoop is an open-source framework and uses commodity hardware (commodity hardware is relatively inexpensive and easy to obtain hardware) to store enormous quantities of data.

2. Computing power: Hadoop is based on distributed computing model which processes very large volumes of data fairly quickly. The more the number of computing nodes, the more the processing power at hand.

3. Scalability: This boils down to simply adding nodes as the system grows and requires much less administration.

4. Storage flexibility: Unlike the traditional relational databases, in Hadoop data need not be pre-processed before storing it. Hadoop provides the convenience of storing as much data as one need and also the added flexibility of deciding later as to how to use the stored data. In Hadoop, one can store unstructured data like images, videos, and free-form text.

5. Inherent data protection: Hadoop protects data and executing applications against hardware failure. If a node fails, it automatically redirects the jobs that had been assigned to this node to the other functional and available nodes and ensures that distributed computing does not fail. It goes a step further to store multiple copies (replicas) of the data on various nodes across the cluster.

Hadoop Conceptual Layer

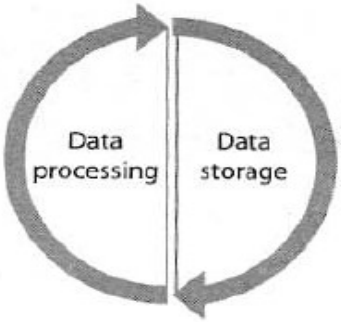
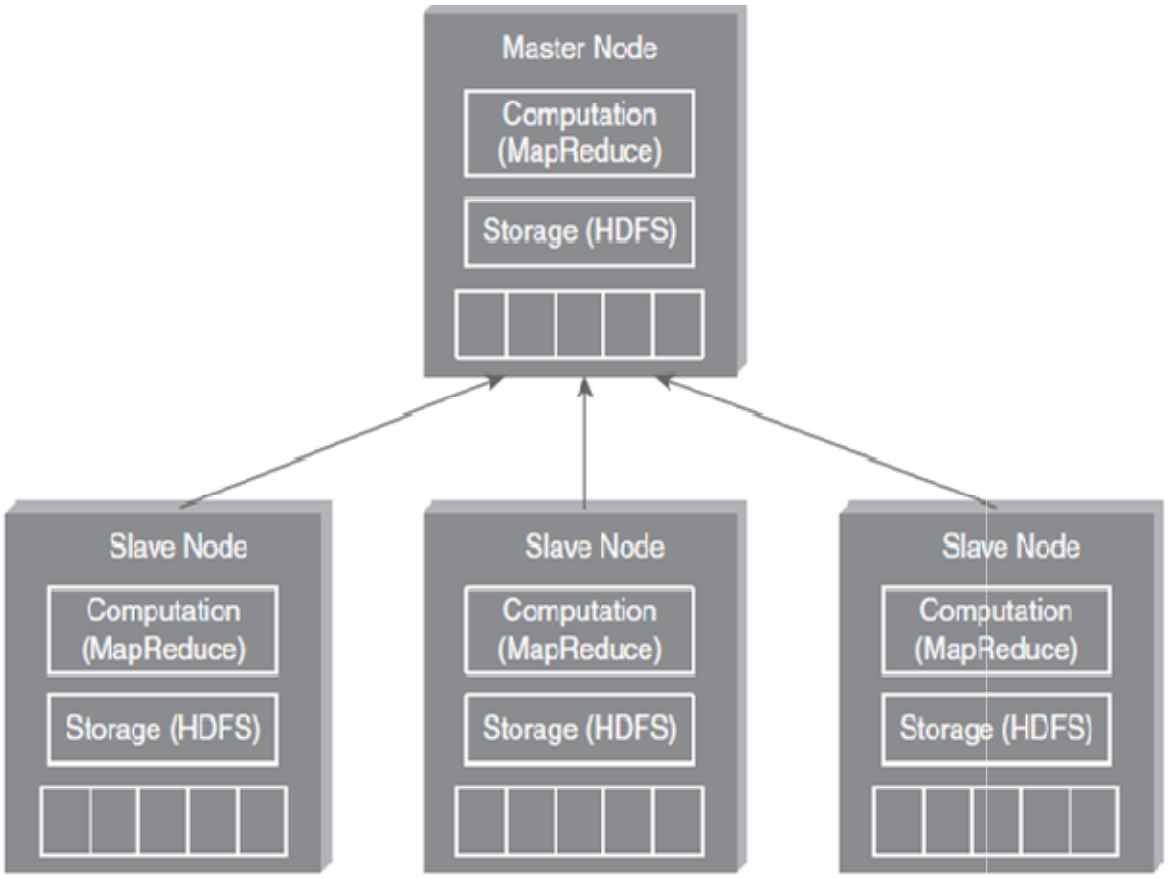


Figure 5.9 Hadoop conceptual layer.

Hadoop is conceptually divided into **Data Storage Layer** which stores huge volumes of data and **Data Processing Layer** which processes data in parallel to extract richer and meaningful insights from data (Figure 5.9).

Hadoop High Level Architecture



Hadoop is a distributed Master-Slave Architecture. Master node is known as NameNode and slave no are known as DataNodes. Figure above depicts the Master-Slave Architecture of Hadoop Framework.

Master Node:

- **Computation (MapReduce)**: The master node manages and coordinates the processing tasks using the **MapReduce** framework. It assigns tasks to the slave nodes and monitors their execution.
- **Storage (HDFS)**: The master node also handles the file system namespace and controls access to files stored in **HDFS** (Hadoop Distributed File System). It tracks the metadata of the files (such as where file blocks are stored).

Slave Nodes: Each slave node performs two key roles:

- **Computation (MapReduce)**: Slave nodes execute the computation tasks (map and reduce operations) assigned by the master node.
- **Storage (HDFS)**: Slave nodes store actual data blocks within the HDFS. They are responsible for reading and writing to their local disks as directed by the master node.

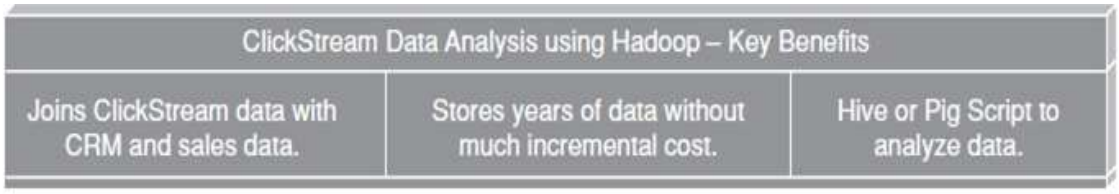
Master-Slave Relationship:

- The master node controls all operations, while the slave nodes carry out the actual data processing and storage tasks.
- The master node keeps track of which slave nodes have which pieces of data and distributes computational tasks based on the data locality to minimize network traffic.

Use case for Hadoop

ClickStream Data Analysis:

- ClickStream data (mouse clicks) helps you to understand the purchasing behavior of customers.
- ClickStream analysis helps online marketers to optimize their product web pages, promotional content, etc. to improve their business.



The ClickStream analysis (Figure 5.11) using Hadoop provides three key benefits:

1. Hadoop helps to join ClickStream data with other data sources such as Customer Relationship Management Data (Customer Demographics Data, Sales Data, and Information on Advertising Campaigns). This additional data often provides the much needed information to understand customer behavior.
2. Hadoop's scalability property helps you to store years of data without ample incremental cost. This helps you to perform temporal or year over year analysis on ClickStream data which your competitors may miss.
3. Business analysts can use Apache Pig or Apache Hive for website analysis. With these tools, you can organize ClickStream data by user session, refine it, and feed it to visualization or analytics tools.

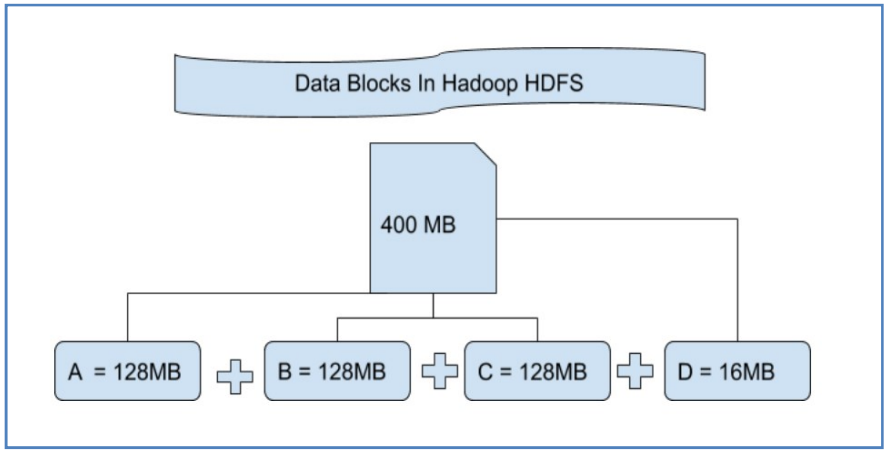
HDFS (HADOOP DISTRIBUTED FILE SYSTEM)

Some key points of HDFS are as follows:

1. It is the primary Storage component of Hadoop.
2. It was developed using Distributed File System design.
3. It was Modeled after Google File System.
4. It is Optimized for high throughput (HDFS leverages large block size and moves computation where data is stored).
5. You can replicate a file for a configured number of times, which is tolerant in terms of both software and hardware.
6. Re-replicates data blocks automatically on nodes that have failed.
7. You can realize the power of HDFS when you perform read or write on large files (gigabytes and larger).
8. It Sits on top of native file system

File Block in HDFS

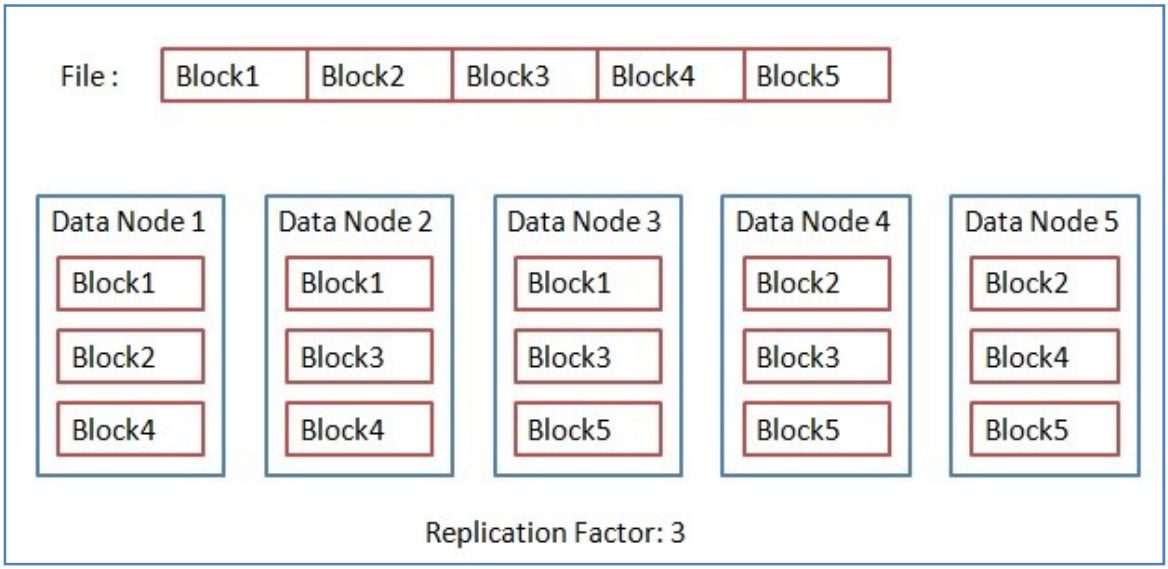
Data in HDFS is always stored in terms of blocks. So the single block of data is divided into multiple blocks of size 128MB which is default and you can also change it manually.



- 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. 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.

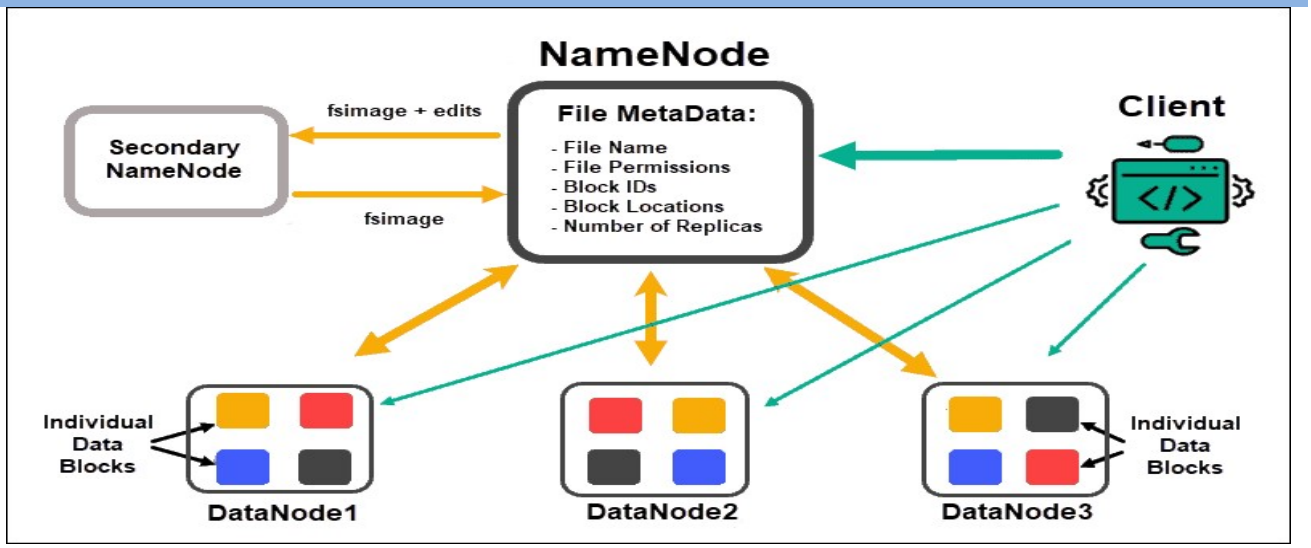
Replication in HDFS

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 5 file blocks which means that 3 Replica or copy of each file block is made means total of 5×3 = 15 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.**

HDFS Architecture and Daemons



HDFS follows the master-slave architecture. HDFS architecture consists of three Daemons which are:-

1. Namenode

- Single NameNode per cluster.
- Keeps the metadata details

2. Datanode

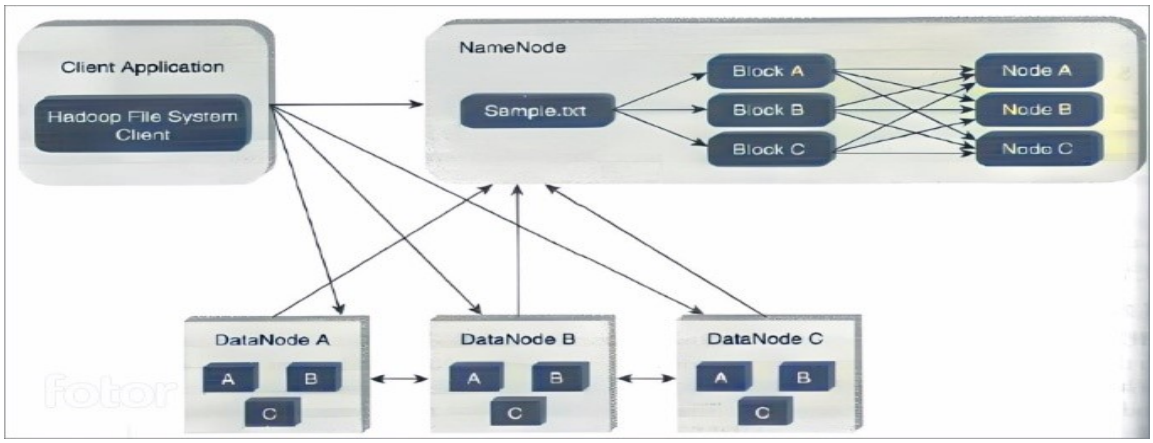
- Multiple DataNode per cluster
- Read/Write operations

3. Secondary Namenode.

- Housekeeping Daemon

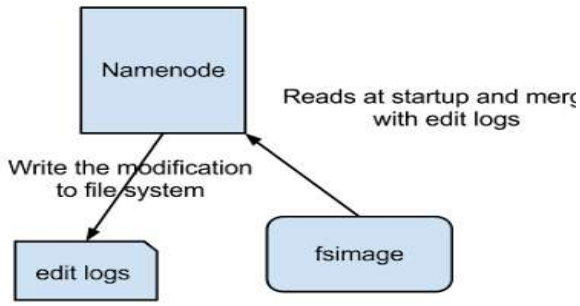
NameNode

- HDFS breaks a large file into smaller pieces called blocks.
- **NameNode** uses a rack ID to identify DataNodes in the rack. A rack is a collection of DataNodes within the cluster.
- NameNode keeps track of blocks of files placed on various DataNodes.
- NameNode manages file-related operations such as **read, write, create, and delete**.
- Its main job is managing the File System Namespace. It is as shown in Figure below:



- Namenode holds the metadata for the HDFS like Namespace information, block information etc. When in use, all this information is stored in main memory. But these information also

stored in disk for persistence storage.



- The image shows how Name Node stores information in disk.
- The two different files are:

1. fsimage - Its the snapshot of the filesystem when namenode started.

2. Edit logs - Its the sequence of changes made to the filesystem after namenode started.

- Only in the restart of namenode , edit logs are applied to fsimage to get the latest snapshot of the file system. Then it flushes out new version of **FsImage** on disk and **truncates the old EditLog** because the changes are updated in the **FsImage**.
- But namenode restart are rare in production clusters which means edit logs can grow very large for the clusters where namenode runs for a long period of time.

DataNode

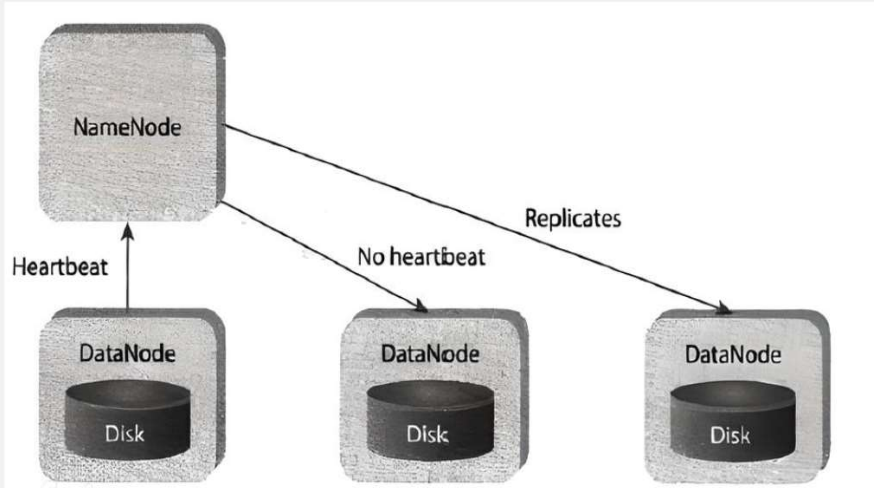


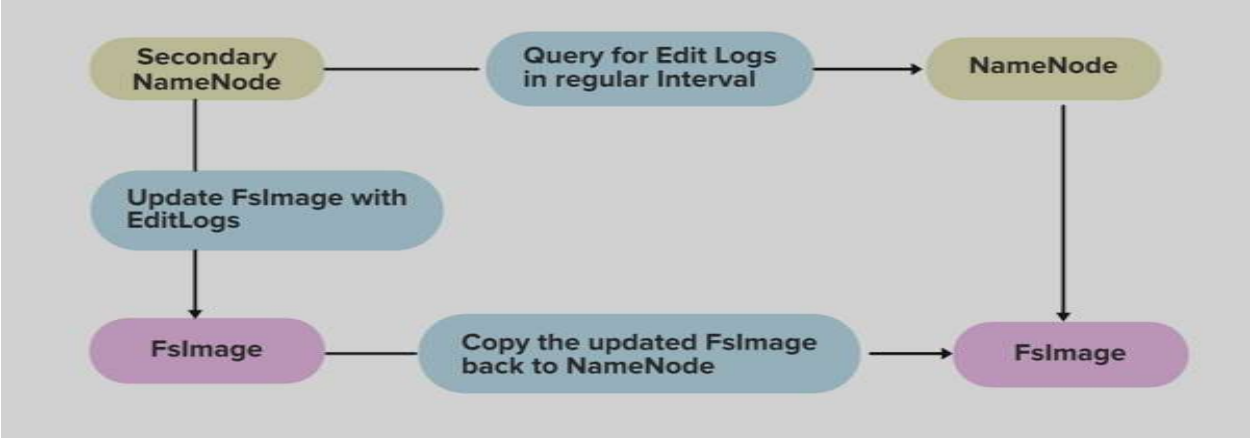
Figure 5.17 NameNode and DataNode Communication.

- There are multiple DataNodes per cluster.
- A DataNode also continuously sends "heartbeat" message to NameNode to ensure the connectivity between the NameNode and DataNode.
- In case there is no heartbeat from a DataNode, the NameNode replicates that DataNode within the cluster and keeps on running as if nothing had happened.

Secondary NameNode

- Secondary Namenode helps to overcome the of large editlogs issues by taking over responsibility of merging editlogs with fsimage from the namenode.
- The Secondary NameNode takes a snapshot of HDFS metadata at intervals specified in the Hadoop configuration.

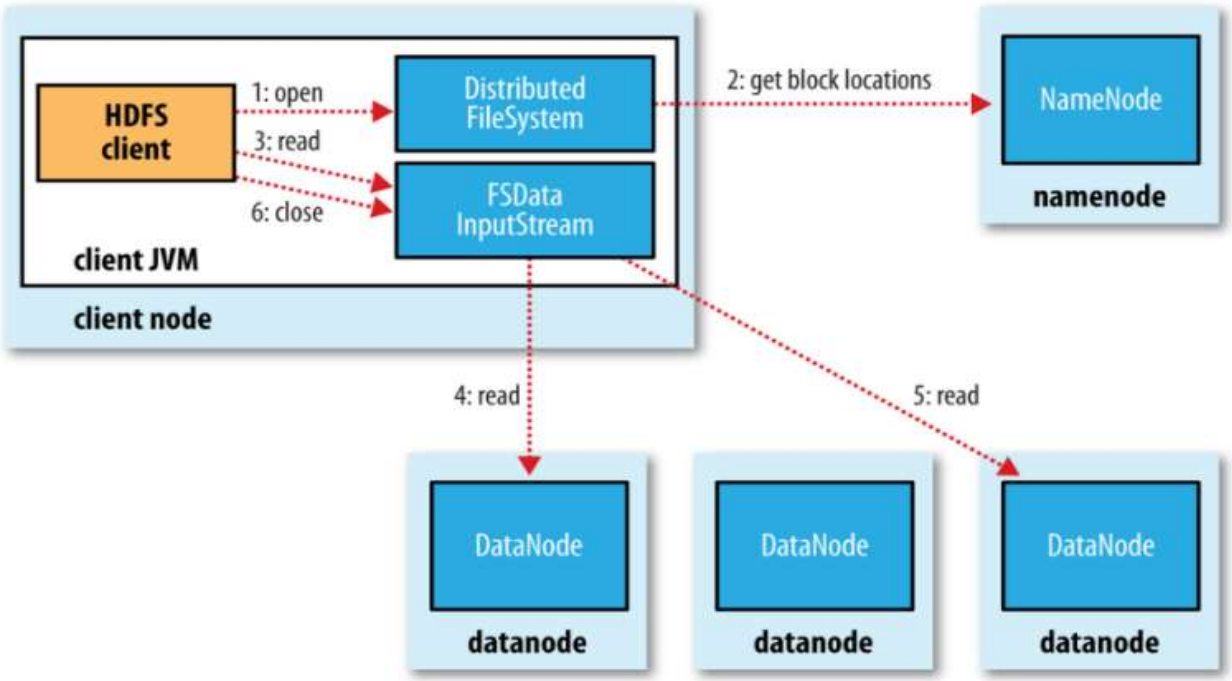
- Since the memory requirements of Secondary NameNode are the same as NameNode, it is better run NameNode and Secondary NameNode on different machines.
- In case of failure of the NameNode, Secondary NameNode can be configured manually to bring up the cluster.
- However, the Secondary NameNode does not record any real-time changes that happen to the HDFS metadata.



- The figure shows the working of Secondary Namenode:
  1. It gets the **edit logs** from the namenode in regular intervals and applies to **fsimage**
  2. Once it has **new fsimage**, it **copies back to namenode.**
  3. Namenode will **use this fsimage** for the next restart, which will reduce the startup time.

HDFS -Anatomy of File Read

Figure below depicts the anatomy of File Read.



The Steps involved in File Read are:

Step 1: The client opens the file it wishes to read by calling open() on the File System Object(which for HDFS is an instance of Distributed File System).

Step 2: Distributed File System(DFS) calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file. For each block, the name node

returns the addresses of the data nodes that have a copy of that block. The DFS returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the data node and name node I/O.

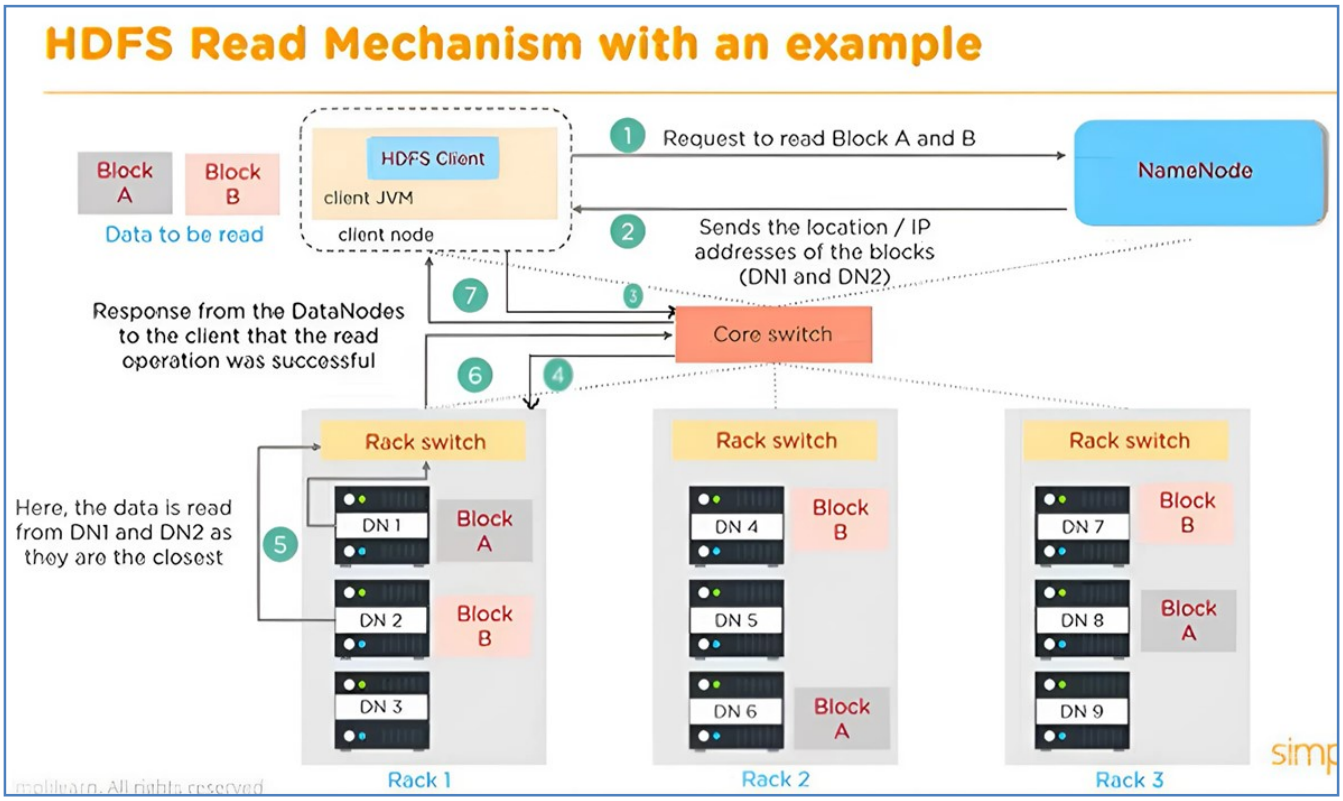
Step 3: The client then calls read() on the stream. DFSInputStream, which has stored the info node addresses for the primary few blocks within the file, then connects to the primary (closest) data node for the primary block in the file.

Step 4: Data is streamed from the data node back to the client, which calls read() repeatedly on the stream.

Step 5: When the end of the block is reached, DFSInputStream will close the connection to the data node, then finds the best data node for the next block. This happens transparently to the client, which from its point of view is simply reading an endless stream. Blocks are read as, with the DFSInputStream opening new connections to data nodes because the client reads through the stream. It will also call the name node to retrieve the data node locations for the next batch of blocks as needed.

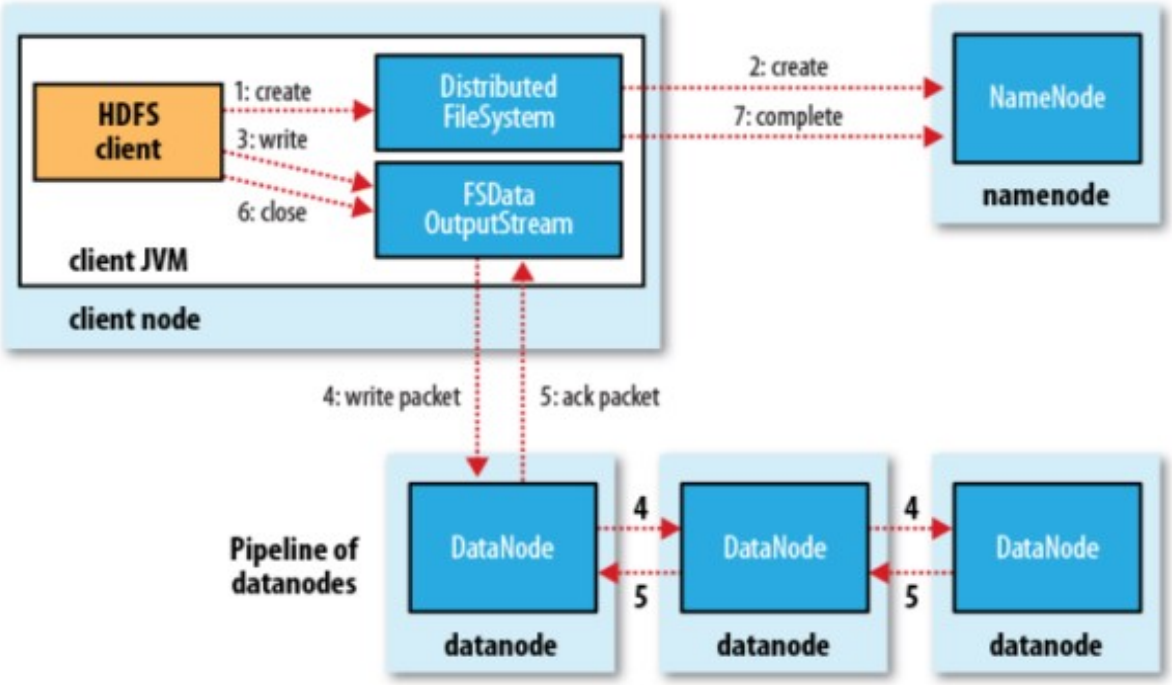
Step 6: When the client has finished reading the file, a function is called, close() on the FSDataInputStream.

Example: (Explain below figure in your own words)



HDFS -Anatomy of File Write

Figure below depicts the anatomy of File write.



The Steps involved in File Write are:

Step 1. The client calls create() on DistributedFileSystem to create a file.

Step 2. An RPC call to the NameNode happens through the DistributedFileSystem to create a new file. The NameNode performs various checks to create a new file (checks whether such a file exists or not). Initially, the NameNode creates a file without associating any data blocks to the file. The DistributedFileSystem returns an FSDataOutputStream to the client to perform write.

Step 3. As the client writes data, data is split into packets by DFSOutputStream, which is then written to an internal queue, called data queue. DataStreamer consumes the data queue. The DataStreamer requests

the NameNode to allocate new blocks by selecting a list of suitable DataNodes to store replicas. This list of DataNodes makes a pipeline. Here, we will go with the default replication factor of three, so there will be three nodes in the pipeline for the first block.

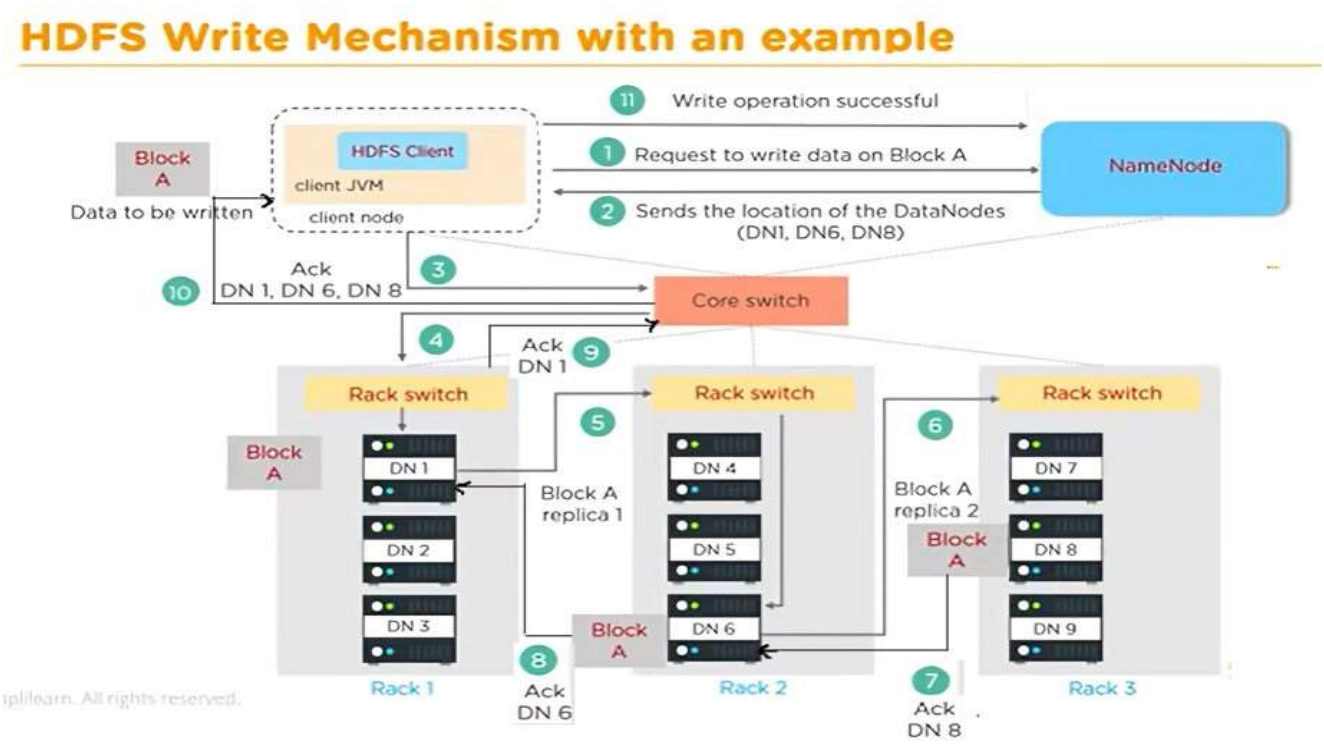
Step 4. DataStreamer streams the packets to the first DataNode in the pipeline. It stores packet and forwards it to the second DataNode in the pipeline. In the same way, the second DataNode stores the packet and forwards it to the third DataNode in the pipeline.

Step 5. In addition to the internal queue, DFSOutputStream also manages an "Ack queue" of packets that are waiting for the acknowledgement by DataNodes. A packet is removed from the "Ack queue" only if it is acknowledged by all the DataNodes in the pipeline.

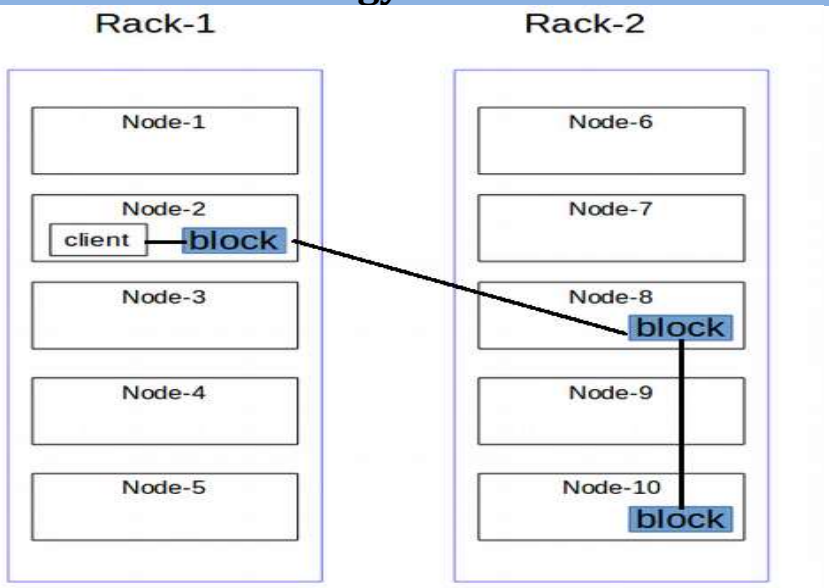
Step 6. When the client finishes writing the file, it calls close() on the stream.

Step 7. This flushes all the remaining packets to the DataNode pipeline and waits for relevant acknowledgments before communicating with the NameNode to inform the client that the creation of the file is complete.

Example: (Explain below figure in your own words)



HDFS-Replica Placement Strategy



The replica placement policy followed by Hadoop framework is as follows-

- **For the default case, when the replication factor is 3**

a) Put one replica on the same machine where the client application (application which is using the file) is, if the client is on a DataNode. Otherwise choose a random DataNode for storing the replica.
b) Store another replica on a node in a different (remote) rack.
c) The last replica is also stored on the same remote rack but the node where it is stored is different.

- **In case replication factor is greater than 3**

a) For the first 3 replicas policy as described above is followed.
b) From replica number 4 onward node location is determined randomly while keeping the number of replicas per rack below the upper limit (which is basically (replicas - 1) / racks + 2).

Working with HDFS Commands

1. Objective: To create a directory (say, sample) in HDFS.

hadoop fs -mkdir /sample

2. Objective: To copy a file from local file system to HDFS.

hadoop fs –put /root/sample/test.txt /sample/test.txt

3. Objective: To copy a file from HDFS to local file system.

hadoop fs –get /sample/test.txt /root/sample/testsample.txt

4. Objective: To get the list of directories and files at the root of HDFS

hadoop fs –ls /

5. Objective: To get the list of complete directories and files of HDFS.

hadoop fs –ls –R /

6. Objective: To copy a file from local file system to HDFS via copyFromLocal command

hadoop fs **–copyFromLocal /root** ***/sample/test.txt /sample/testsample.txt***

7. Objective: To copy a file from Hadoop file system to local file system via copyToLocal command

hadoop fs **–copyToLocal** ***/sample/test.txt /root/sample/testsample1.txt***

8. Objective: To display contents of an HDFS file on console

hadoop fs **–cat /sample/test.txt**

9. Objective: To copy a file from one directory to another on HDFS

hadoop fs **–cp /sample/test.txt /sample1**

10. Objective: To remove a directory from HDFS

hadoop fs -rm-r /sample1

Processing Data with Hadoop

- **MapReduce Programming** is a software framework. MapReduce Programming helps you to process massive amounts of data in parallel.
- In MapReduce Programming,
  - The input dataset is split into independent chunks.
  - **Map** tasks process these independent chunks completely in a parallel manner. The output produced by the map tasks serves as intermediate data and is stored on the local disk of that server. The output of the mappers are automatically shuffled and sorted by the framework. MapReduce Framework sorts the output based on keys. This sorted output becomes the input to the reduce tasks.
  - **Reduce** task provides reduced output by combining the out- put of the various mappers. Job inputs and outputs are stored in a file system. MapReduce framework also takes care of the other tasks such as scheduling, monitoring, re-executing failed tasks, etc.
- Hadoop Distributed File System and MapReduce Framework run on the same set of nodes. This configuration allows effective scheduling of tasks on the nodes where data is present (Data Locality). This in turn results in very high throughput.
- There are two daemons associated with MapReduce Programming.
  - A single master **Job Tracker** per cluster and one slave **TaskTracker** per cluster-node.

- The **Job Tracker** is responsible for scheduling tasks to the ***TaskTrackers, monitoring the task, and re-executing the task just in case the TaskTracker fails.***
- The **TaskTracker** executes the task. Refer Figure 5.21.

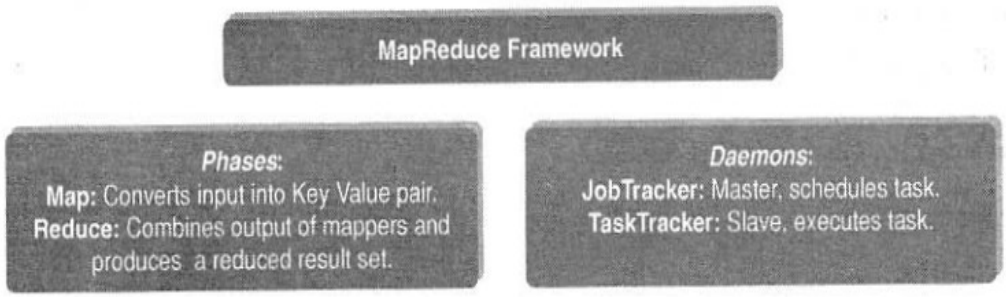


Figure 5.21 MapReduce Programming phases and daemons.

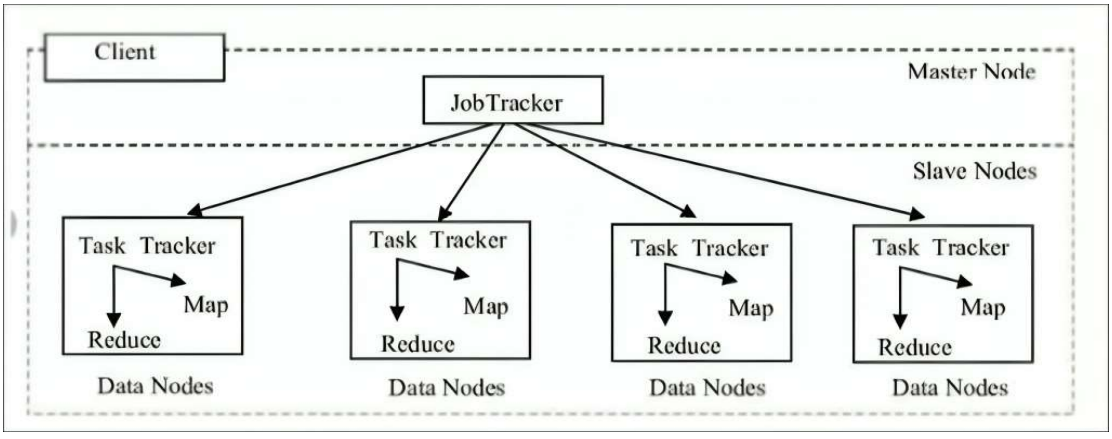
MapReduce deamons

1. JobTracker:

- It provides connectivity between Hadoop and your application.
- When you submit code to cluster, Job Tracker creates the execution plan by deciding which task to assign to which node.
- It also monitors all the running tasks.
- When a task fails, it automatically re-schedules the task to a different node after a predefined number of retries.
- Job Tracker is a master daemon responsible for executing overall MapReduce job.
- There is a single Job Tracker per Hadoop cluster.

2. TaskTracker:

- This daemon is responsible for executing individual tasks that is assigned by the JobTracker.
- There is a single TaskTracker per slave.
- Task Tracker continuously sends heartbeat message to Job Tracker. When the Job Tracker fails to receive a heartbeat from a TaskTracker, the Job Tracker assumes that the TaskTracker has failed and resubmits the task to another available node in the cluster.
- Once the client submits a job to the JobTracker, it partitions and assigns diverse MapReduce tasks for each TaskTracker in the cluster.
- Figure 5.22 depicts Job Tracker and TaskTracker interaction.



MapReduce programming Architecture

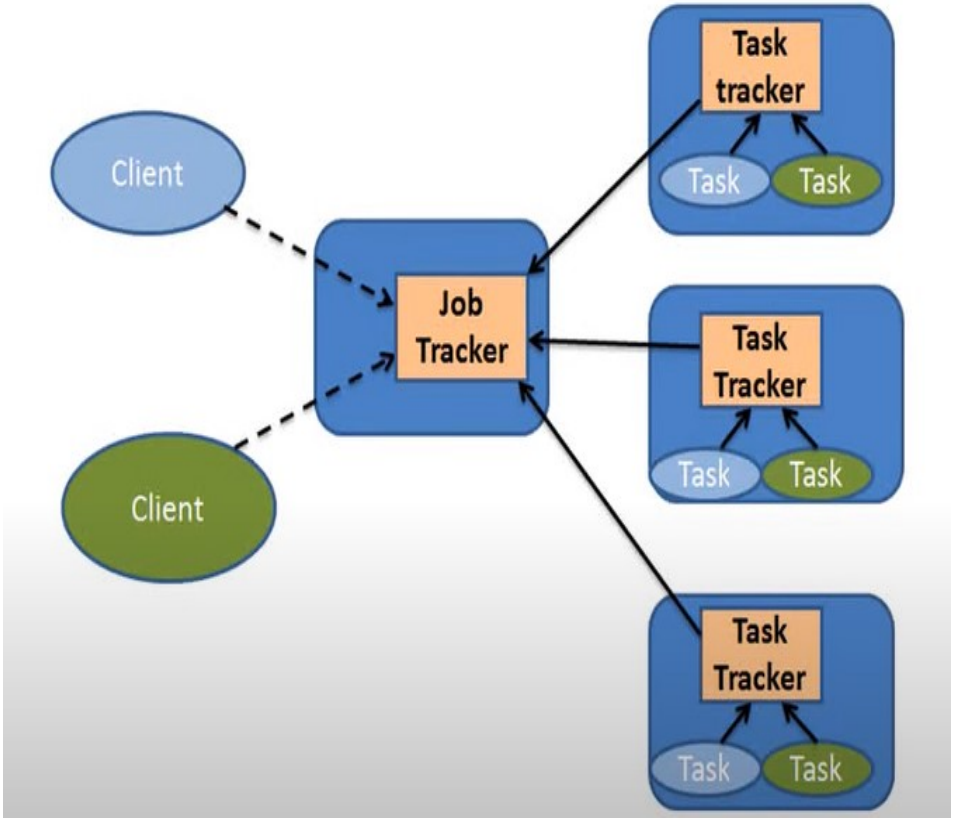


Fig: Map Reduce Programming Architecture

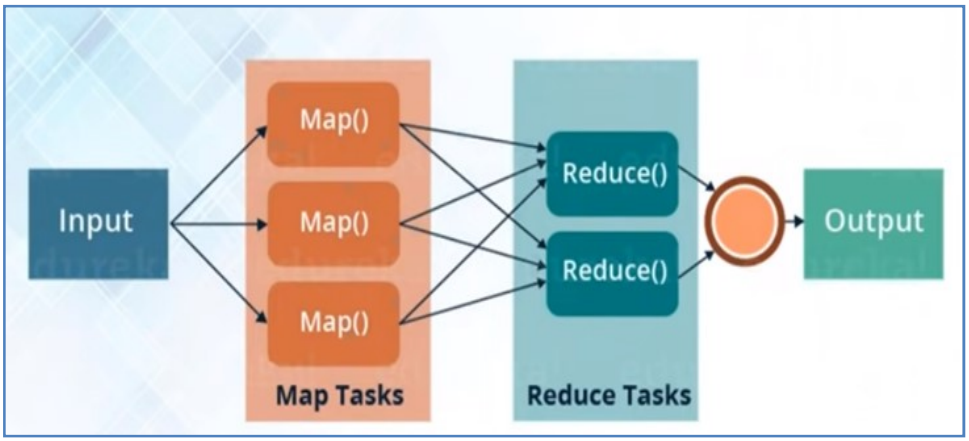


Fig: Map Reduce Programming Architecture

Figures above describe the working model of MapReduce Programming. The following steps describe how MapReduce performs its task.

1. First, the input dataset is split into multiple pieces of data (several small subsets).

2. Next, the framework creates a master and several workers processes and executes the worker processes remotely.

3. Several map tasks work simultaneously and read pieces of data that were assigned to each map. The map worker uses the map function to extract only those data that are present on their server and generates key/value pair for the extracted data.

4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of the specified mapper.

5. When the map workers complete their work, the master instructs the reduce workers to begin their work. The reduce workers in turn contact the map workers to get the key/value data for their partition. The data thus received is shuffled and sorted as per keys.
6. Then it calls reduce function for every unique key. This function writes the output to the file.
7. When all the reduce workers complete their work, the master transfers the control to the user program.

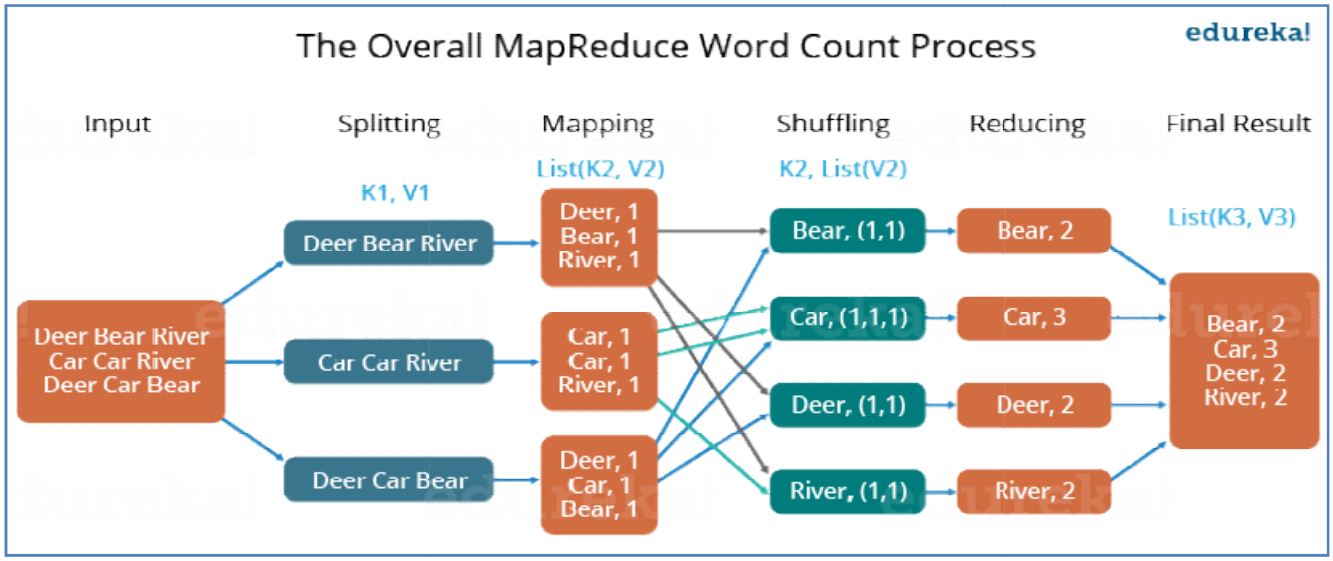
MapReduce programming example

A Word Count Example of MapReduce

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

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

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



- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a value (1) to each of the tokens or words. The rationale behind giving a value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – **Dear, 1; Bear, 1; River, 1.** The mapping process remains the same on all the nodes.
- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. **For example, Bear, [1,1]; Car, [1,1,1].., etc.**
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as **– Bear, 2.**
- Finally, all the output key/value pairs are then collected and written in the output file.

Word Count MapReduce Programming using Java

The MapReduce Programming requires three things.

1. Driver Class: This class specifies Job Configuration details.

2. Mapper Class: This class overrides the Map Function based on the problem statement.

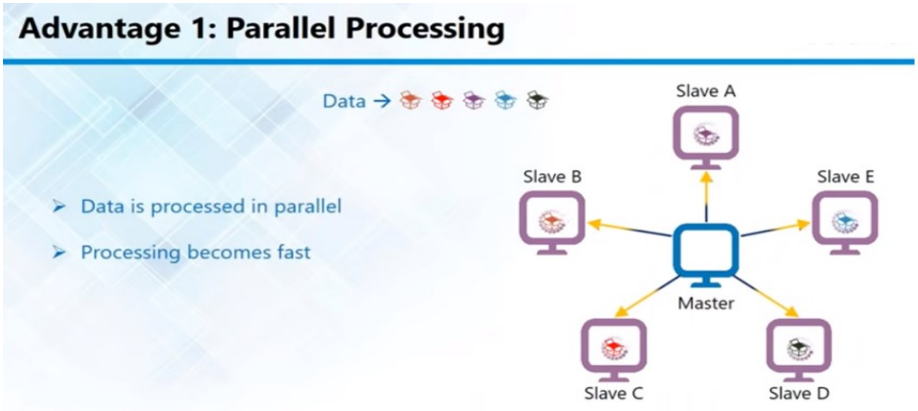
3. Reducer Class: This class overrides the Reduce Function based on the problem statement.

Advantages of MapReduce

The two biggest advantages of MapReduce are:

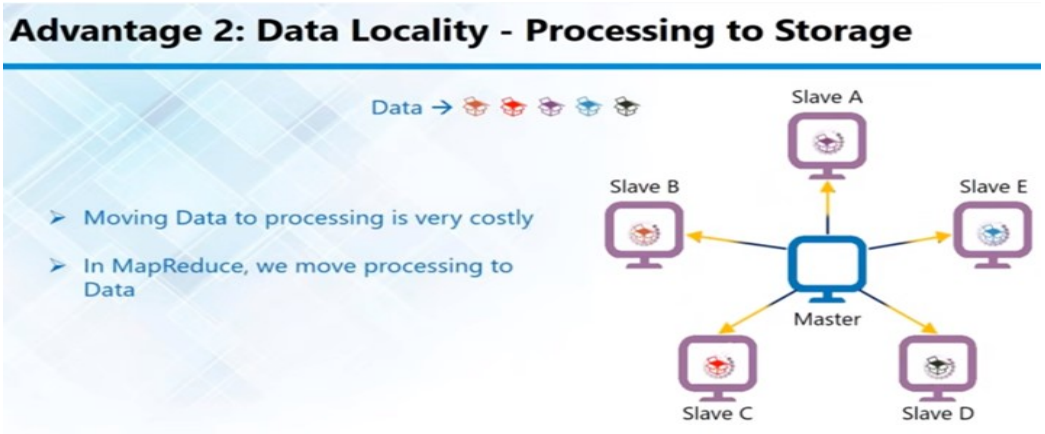
1. Parallel Processing:

- In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

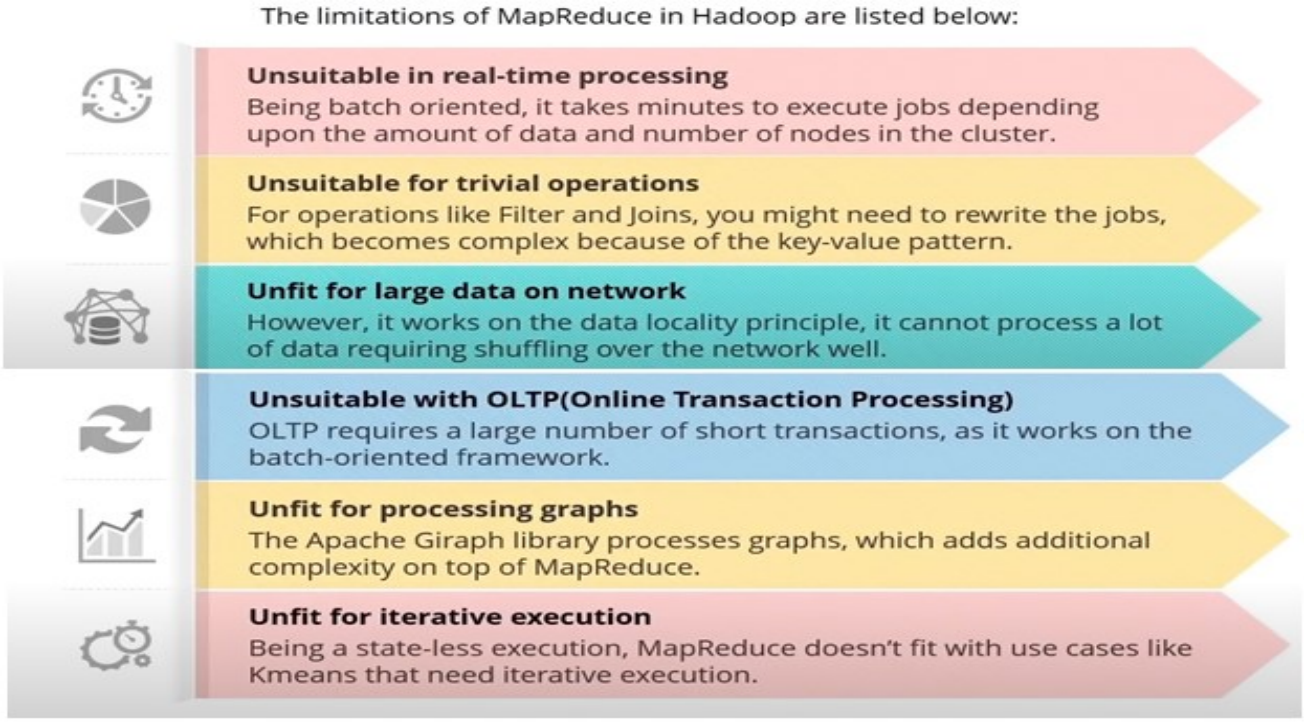


2. Data Locality:

- Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework.
- This allows us to have the following advantages:
  - It is very cost-effective to move processing unit to the data.
  - The processing time is reduced as all the nodes are working with their part of the data in parallel.
  - Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.



Disadvantages of MapReduce



MapReduce Programming Workflow, Architecture

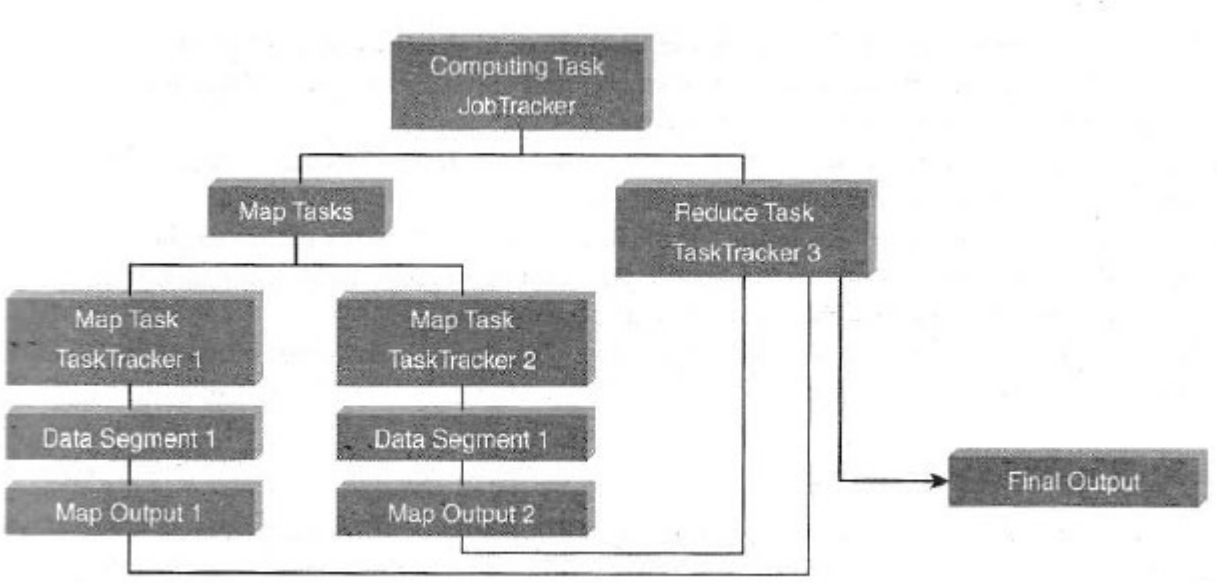


Figure 5.23 MapReduce programming workflow.

The figure 5.23 gives the MapReduce programming workflow:

- **Computing Task (JobTracker):**
  The JobTracker is responsible for distributing tasks to different nodes within the Hadoop cluster. It manages the entire MapReduce job lifecycle.
- **Map Tasks:**
  Data is split into segments, and Map Tasks are distributed to different nodes (here referred to as TaskTracker 1 and TaskTracker 2). Each segment is processed in parallel.
- **TaskTracker 1 and TaskTracker 2:**
  These are individual nodes that handle the actual processing of map tasks. TaskTrackers

report back to the JobTracker.

- **Data Segments:**
  The input data is divided into manageable chunks, each handled by a map task.
- **Map Output:**
  After the map tasks are complete, the intermediate output is stored (shown as Map Output 1 and Map Output 2).
- **Reduce Tasks:**
  Once the map tasks are done, the outputs are shuffled and sorted, then sent to Reduce Tasks (e.g., TaskTracker 3), which combine the results to produce the final output.
- **Final Output:**
  The result of the reduce phase is the final output of the distributed computation.

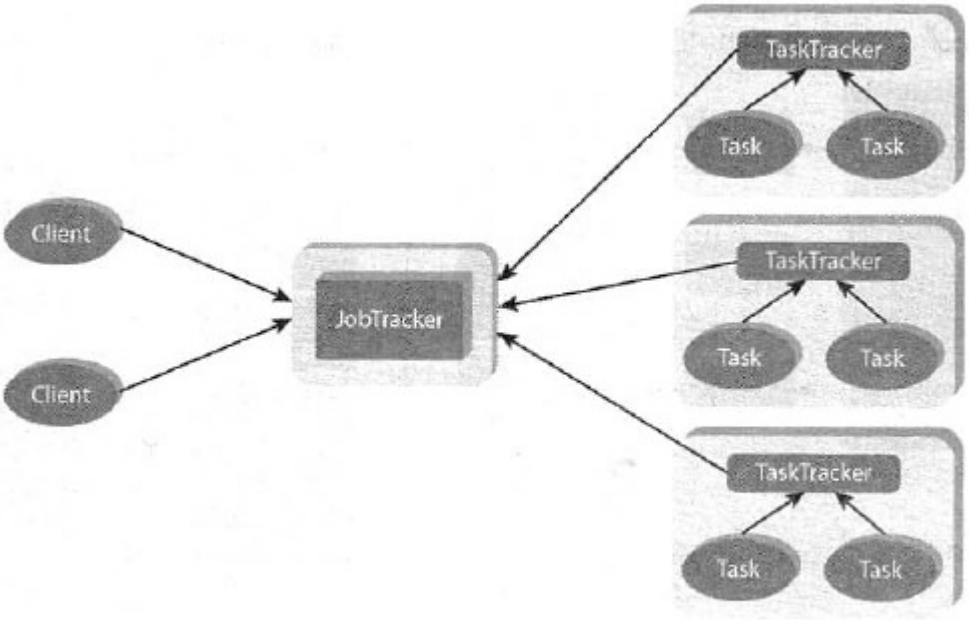


Figure 5.24 MapReduce programming architecture.

The figure 5.24 gives the MapReduce Architecture which explains the Client-JobTracker-TaskTracker Interaction.

1. First, the input dataset is split into multiple pieces of data (several small subsets).

2. Next, the framework creates a master and several workers processes and executes the worker processes remotely.

3. Several map tasks work simultaneously and read pieces of data that were assigned to each map The map worker uses the map function to extract only those data that are present on their server and generates key/value pair for the extracted data.

4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of the specified mapper.

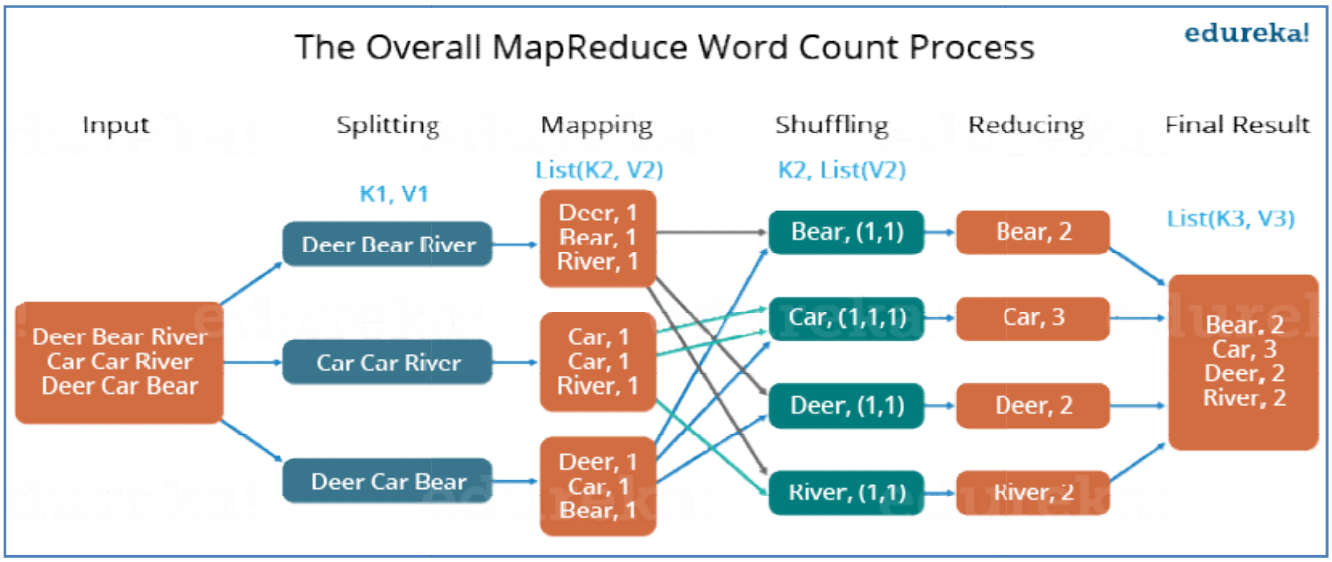
5. When the map workers complete their work, the master instructs the reduce workers to begin their work. The reduce workers in turn contact the map workers to get the key/value data for their partition. The data thus received is shuffled and sorted as per keys.

6. Then it calls reduce function for every unique key. This function writes the output to the file.

7. When all the reduce workers complete their work, the master transfers the control to the user program.

A Word Count Example of MapReduce

- Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:
  - **Dear, Bear, River, Car, Car, River, Deer, Car and Bear**
- Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a value (1) to each of the tokens or words. The rationale behind giving a value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Word Count MapReduce Programming using Java

The MapReduce Programming requires three things.

1. Driver Class: This class specifies Job Configuration details.

2. Mapper Class: This class overrides the Map Function based on the problem statement.

3. Reducer Class: This class overrides the Reduce Function based on the problem statement.

Wordcounter.java: Driver Program

```
package com.app;
import java.io.IOException;
import org.apache.hadoop.fs. Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCounter {
public static void main (String[ ] args) throws IOException, InterruptedException,
ClassNotFoundException {
Job job = new Job ();
job.setJobName ("wordcounter");
job.setJarByClass (WordCounter.class); job.setMapperClass (WordCounterMap.class);
job.setReducerClass (WordCounterRed.class); job.setOutputKeyClass (Text.class);
job.setOutputValueClass (IntWritable.class);
FileInputFormat.addInput Path (job, new Path ("/sample/word. txt"));
FileOutputFormat.setOutputPath (job, new Path ("/sample/wordcount"));
System.exit (job.waitForCompletion (true)? 0: 1);
}
```

Explanation:

This code defines a MapReduce job that reads a text file, processes it using a mapper to count words, and outputs the result. It uses the newer org.apache.hadoop.mapreduce.Job API and demonstrates the general steps to set up a MapReduce job: defining the mapper, reducer, input/output types, and input/output paths, and finally running the job.

WordCounterMap.java: Map Class

```
package com.app;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce. Mapper;

public class WordCounterMap extends Mapper<LongWritable, Text, Text, IntWritable> {
 @Override
protected void map (LongWritable key, Text value, Context context) throws IOException, InterruptedException
                {
                        String [ ] words=value.toString().split (",");
                        for (String word: words) {
                                context.write (new Text (word), new IntWritable (1));
                }
}
```

Explanation:

- **Input**: Each line of text is processed as a string (`Text`), and the `split ()` function breaks it into words based on commas.
- **Output**: For each word, the mapper outputs a key-value pair where the key is the word (as `Text`), and the value is `1` (as `IntWritable`).
- The map method thus transforms each line of the input into multiple key-value pairs that represent the words in the line and their count.

Example:

Suppose the input file contains the following line:

```
apple,banana,apple,orange
```

When this line is processed by the `map()` method:

1. The line is split into an array: `["apple", "banana", "apple", "orange"]`.
2. The mapper emits the following intermediate key-value pairs:

```
("apple", 1)
("banana", 1)
("apple", 1)
("orange", 1)
```

These intermediate results will be passed to the **Reducer** stage, where the counts for each word will be aggregated.

WordCounterReduce.java: Reduce Class

```
package com.infosys;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce. Reducer;

public class WordCounterRed extends Reducer<Text, IntWritable, Text, IntWritable>{
        @Override
        protected void reduce (Text word, Iterable<IntWritable> values, Context context)
                                        throws IOException, InterruptedException {
Integer count =0;
for (IntWritable val: values) {
count + val.get();
}
context.write (word, new IntWritable (count));
}
}
```

Explanation:

- **Input to Reducer**:
  - The `Reducer` receives key-value pairs where the **key** is a word (e.g., `"apple"`), and the **value** is an iterable list of counts (e.g., `[1, 1, 1]` for "apple").
- **Processing**:
  - It iterates over the list of values, sums them up, and produces the total count for each word.

- **Output from Reducer**:
  - For each word, the `Reducer` writes a key-value pair where the key is the word and the value is the total count (e.g., `("apple", 3)`).

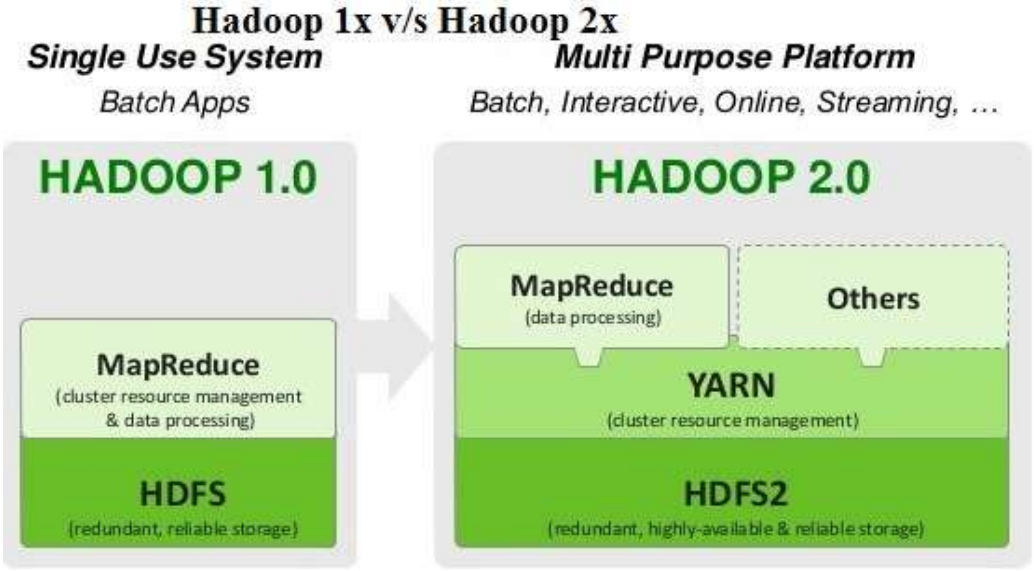
The output might look like this:

```
apple   3
banana  2
orange  1
```

This output indicates that "apple" appeared 3 times, "banana" appeared 2 times, and "orange" appeared 1 time in the input data.

Managing Resources and Applications With Hadoop YARN (Yet Another Resource Negotiator)

Apache Hadoop YARN is a sub-project of Hadoop 2.x. Hadoop 2.x is YARN-based architecture. It is a general processing platform. YARN is not constrained to MapReduce only. You can run multiple applications in Hadoop 2.x in which all applications share a common resource management. Now Hadoop can be used for various types of processing such as Batch, Interactive, Online, Streaming, Graph, and others. It is as shown in figure below:



Limitations of Hadoop 1.0 Architecture:

1. Single NameNode is responsible for managing entire namespace for Hadoop Cluster.
2. It has a restricted processing model which is suitable for batch-oriented MapReduce jobs.
3. Hadoop MapReduce is not suitable for interactive analysis.
4. Hadoop 1.0 is not suitable for machine learning algorithms, graphs, and other memory intensive algorithms.
5. **MapReduce** is responsible for **cluster resource management and data processing**.
6. **HDFS Limitation:** Name node saves all its file metadata in main memory. so it can quickly become overwhelmed with load on the system increasing.

Hadoop 2: HDFS

HDFS 2 consists of two major components: **(a) namespace, (b) blocks storage service.** Namespace service takes care of file-related operations, such as creating files, modifying files, and directories. The block storage service handles data node cluster management, replication.

HDFS 2 Features:

1. Horizontal scalability.

HDFS Federation uses multiple independent NameNodes for horizontal scalability. NameNodes are independent of each other. It means, NameNodes does not need any coordination with each other. The DataNodes are common storage for blocks and shared by all NameNodes. All DataNodes in the cluster registers with each NameNode in the cluster.

2. High availability.

High availability of NameNode is obtained with the help of Passive Standby NameNode. In Hadoop 2.x, Active-Passive NameNode handles failover automatically. All namespace edits are recorded to a shared NFS storage and there is a single writer at any point of time. Passive NameNode reads edits from shared storage and keeps updated metadata information. In case of Active NameNode failure, Passive NameNode becomes an Active NameNode automatically. Then it starts writing to the shared storage. Figure 5.26 describes the Active-Passive NameNode interaction.

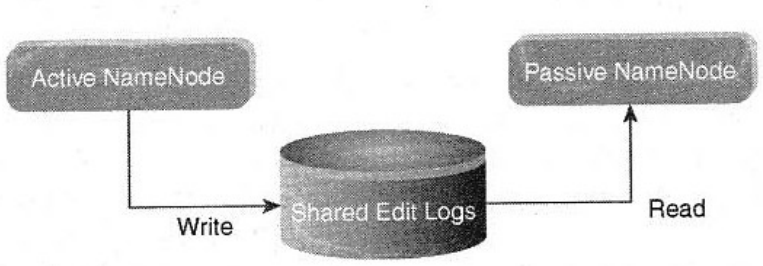


Figure 5.26 Active and Passive NameNode interaction.

Hadoop 2 YARN: Taking Hadoop beyond Batch

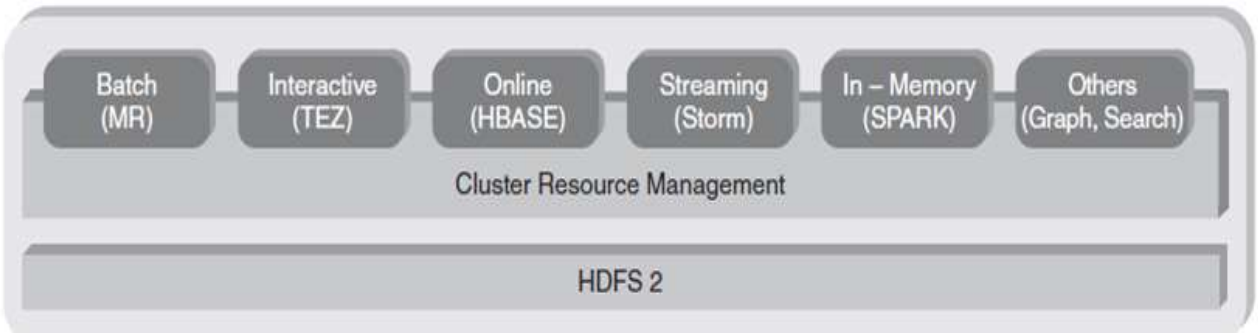


Figure above is “Hadoop YARN", and it presents an overview of Hadoop's YARN (Yet Another Resource Negotiator) architecture. YARN

YARN is the **resource management layer** of Hadoop. It manages resources across the cluster, ensuring that computing tasks (like batch jobs, streaming, etc.) get the resources they need (CPU, memory, etc.). YARN decouples resource management from the actual processing engine, making Hadoop more versatile for various types of workloads.

Above the resource management layer, different workloads are supported, as illustrated in the

boxes:

- **Batch (MR - MapReduce):** Traditional Hadoop **MapReduce** workloads that involve processing large amounts of data in parallel in a batch-oriented manner. MapReduce was the original processing model for Hadoop.
- **Interactive (TEZ): Tez** is an **interactive processing engine** that optimizes the execution of complex directed acyclic graphs (DAGs) of tasks. It provides faster query execution compared to MapReduce.
- **Online (HBase): HBase** is a **NoSQL database** that runs on top of HDFS and supports online, real-time access to large datasets. It's used for real-time read/write access to data.
- **Streaming (Storm): Apache Storm** is a **stream processing engine** that can process data streams in real-time. It supports low-latency, continuous processing for time-sensitive data.
- **In-Memory (Spark): Apache Spark** is a popular **in-memory data processing framework** that speeds up batch processing by keeping intermediate data in memory rather than writing it to disk like MapReduce.
- **Others (Graph, Search):** This includes other use cases, such as:
  - **Graph processing** (e.g., Apache Graph for graph-based computation)
  - **Search** engines that integrate with Hadoop for large-scale data indexing and querying.

Fundamental Idea

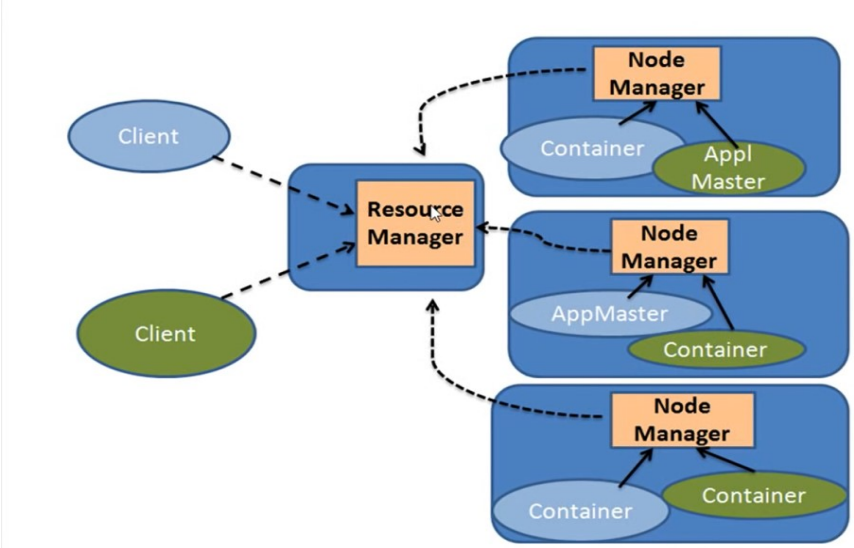
- The fundamental idea behind this architecture is splitting the JobTracker responsibility of resource management and Job Scheduling/Monitoring into separate daemons. Daemons that are part of YARN Architecture are described below.
- **A Global ResourceManager:** Its main responsibility is to distribute resources among various applications in the system. It has two main components:
  - **Scheduler:** Decides the allocation of resources to various running applications, it is a pure scheduler and it does not monitor or track the status of the application.
  - **Application Manager:** Accepts the job, negotiating resources for executing the application specific application master, Restarting the application master during its failure.
- **NodeManager:** This is a per-machine slave daemon. NodeManager responsibility is launching the application containers for application execution. NodeManager monitors the resource usage such as memory, CPU, disk, network, etc. It then reports the usage of resources to the global ResourceManager.
- **Per-application ApplicationMaster:** This is an application-specific entity. Its responsibility is to negotiate required resources for execution from the ResourceManager. It works along with the NodeManager for executing and monitoring component tasks.

Basic Components- Yarn Architecture

Apart from Resource Management, YARN also performs Job Scheduling. YARN performs all your processing activities by allocating resources and scheduling tasks. Apache Hadoop YARN Architecture consists of the following main components as shown in figure below.

Yarn components:

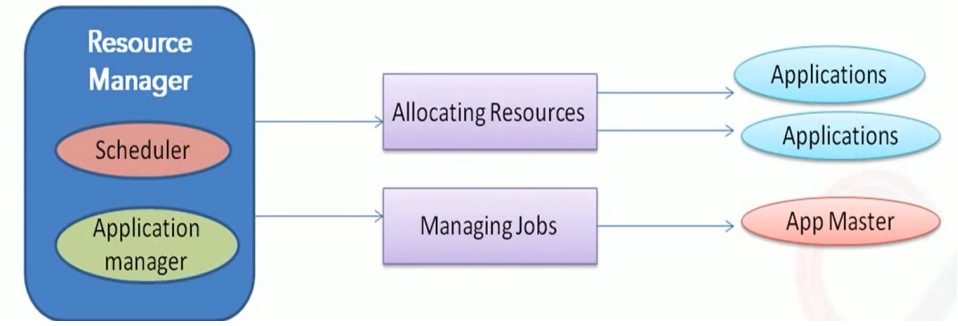
1. Client
2. Resource Manager
3. Node Manager
4. Application Master
5. Container



1. **Client:** It submits map-reduce jobs.

2. Resource Manager

- It is the master daemon of YARN and is responsible for resource assignment and management among all the applications.
- On receiving the processing requests, it passes parts of requests to corresponding node managers accordingly, where the actual processing takes place.
- It is the arbitrator of the cluster resources and decides the allocation of the available resources for competing applications.



- It has two major components: **a) Scheduler b) Application Manager**

a) Scheduler

- The scheduler is responsible for allocating resources to the various running applications subject to constraints of capacities, queues etc.
- Performs scheduling based on the resource requirements of the applications.

b) Application Manager

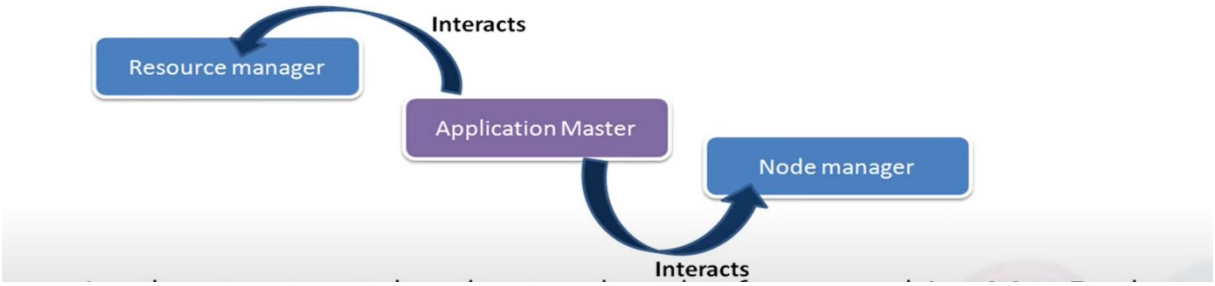
- It is responsible for accepting job submissions.

- Negotiates the first container from the Resource Manager for executing the application specific Application Master.
- Manages running the Application Masters in a cluster and provides service for restarting the Application Master container on failure.

3. **Node Manager**

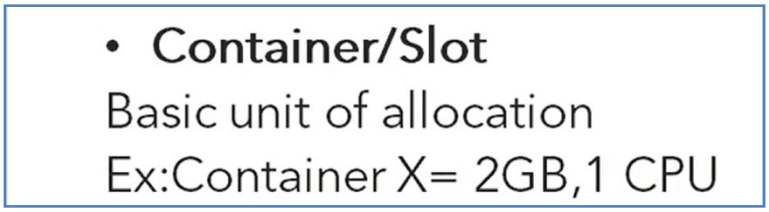
- It takes care of individual nodes in a Hadoop cluster and manages user jobs and workflow on the given node.
- It registers with the Resource Manager and sends heartbeats with the health status of the node.
- Its primary goal is to manage application containers assigned to it by the resource manager.
- It keeps up-to-date with the Resource Manager.

4. **Application Master**



- An application is a single job submitted to the framework. Each such application has a unique Application Master associated with it which is a framework specific entity.
- It is the process that coordinates an application's execution in the cluster and also manages faults.
- Its task is to negotiate resources from the Resource Manager and work with the Node Manager to execute and monitor the component tasks.
- It is responsible for negotiating appropriate resource containers from the Resource Manager, tracking their status and monitoring progress.
- Once started, it periodically sends heartbeats to the Resource Manager to affirm its health and to update the record of its resource demands.

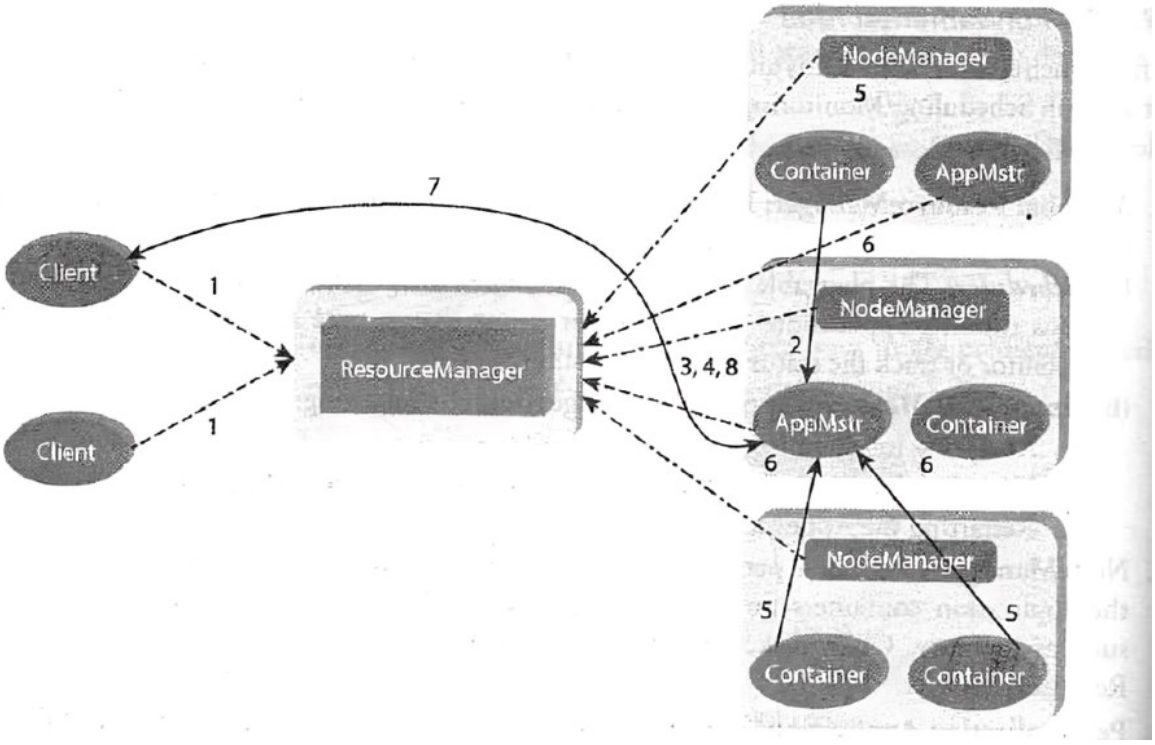
5. **Container**



- It is a collection of physical resources such as RAM, CPU cores, and disks on a single node.
- YARN containers are managed by a container launch context. This record contains a map of environment variables, dependencies stored in a remotely accessible storage, security tokens, payload for Node Manager services and the command necessary to create the process.
- It grants rights to an application to use a specific amount of resources (memory, CPU etc.) on a specific host.

Steps Involved in YARN Architecture

Figure below depicts the steps involved in YARN architecture. It is as follows:



1. A client program submits the application which includes the necessary specifications to launch the application-specific ApplicationMaster itself.

2. The ResourceManager launches the ApplicationMaster by assigning some container.

3. The ApplicationMaster, on boot-up, registers with the ResourceManager. This helps the client program to query the ResourceManager directly for the details.

4. During the normal course, ApplicationMaster negotiates appropriate resource containers via the resource-request protocol.

5. On successful container allocations, the ApplicationMaster launches the container by providing the container launch specification to the NodeManager.

6. The NodeManager executes the application code and provides necessary information such as progress status, etc. to its ApplicationMaster via an application-specific protocol.

7. During the application execution, the client that submitted the job directly communicates with the ApplicationMaster to get status, progress updates, etc. via an application-specific protocol.

8. Once the application has been processed completely, ApplicationMaster deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed.

Questions on Unit-2

1. **Explain the key features and advantages of Hadoop.**
2. **What is the Hadoop ecosystem? Discuss its major components.**

- Explain the Hadoop ecosystem and describe its key components such as HDFS (Hadoop Distributed File System), YARN (Yet Another Resource Negotiator), MapReduce, HBase, Hive, Pig, Sqoop, Flume, and Zookeeper, among others.

3. **Compare Hadoop with traditional SQL databases (Hadoop vs SQL).**
4. **Discuss the history of Hadoop and how it became a dominant framework in big data processing.**

- Provide an overview of Hadoop's origin, starting from its roots in Google's MapReduce and GFS (Google File System) papers. Trace its development by the Apache Foundation and explain how it evolved into the core framework for big data.

5. **Explain the architecture and working of HDFS. How does it ensure fault tolerance and data reliability?**

- Describe the architecture of HDFS, focusing on NameNodes, DataNodes, replication of data blocks, and the concept of data locality. Explain how HDFS ensures fault tolerance through replication and heartbeat mechanisms.

6. **What is Hadoop YARN, and how does it manage resources and applications in a Hadoop cluster?**

- Provide a detailed explanation of YARN's architecture, including the ResourceManager, NodeManagers, and ApplicationMaster. Explain how YARN allows multiple applications to share the cluster's resources and how it improves resource utilization over Hadoop's previous MapReduce-only framework.

7. **Explain the architecture of MapReduce with an example.**

- Provide a detailed explanation of the MapReduce architecture, including the roles of the JobTracker, TaskTracker, Mapper, Reducer, and the input/output phases. Use an example, such as word count, to illustrate how MapReduce works from the map phase to the reduce phase.

8. **Describe the working of the Map and Reduce phases in the MapReduce framework.**

- Discuss in detail the Map phase (input splitting, key-value pairs generation), the Shuffle and Sort phase, and the Reduce phase (aggregation of key-value pairs) in MapReduce. Use diagrams or examples to explain the flow of data.

9. **Write a Java program to implement the word count problem using MapReduce. Explain the Mapper, Reducer, and Driver classes in detail.**

- In this question, students are expected to write a Java program that performs word count using MapReduce. They should explain:

1. The Mapper class that splits input data into words and outputs intermediate key-value pairs.
2. The Reducer class that aggregates these pairs to count occurrences of each word.
3. The Driver class that configures and runs the MapReduce job.

10. **Explain the structure and functionality of the Mapper class in a Java-based MapReduce program. Provide a code snippet as an example.**

Students should explain the structure of the Mapper class, including the key-value pair input and output types, and how the map() method works. They should provide a code example illustrating the map() method, with an explanation of how data is processed.

11. How is the Reducer class implemented in a MapReduce Java program? Explain with a code example.

Students need to write and explain the Reducer class, particularly focusing on the reduce() method, which aggregates values associated with each key. A code snippet should illustrate the process of reducing and outputting final results.

12. Differentiate between

a) HDFS vs HBase
b) Hadoop vs RDBMS
c) Hive vs HBase
d) Hadoop vs SQL.
e) MapReducs Vs SQL

13. Explain in brief the core components of Hadoop ecosystem.

- Explain the Hadoop ecosystem and describe its key components such as HDFS (Hadoop Distributed File System), YARN (Yet Another Resource Negotiator), MapReduce, HBase, Hive, Pig, Sqoop, Flume, and Zookeeper, among others.

14. Illustrate with a word count example the working of map reduce concept.

-Discuss in detail the Map phase (input splitting, key-value pairs generation), the Shuffle and Sort phase, and the Reduce phase (aggregation of key-value pairs) in MapReduce. Use diagrams or examples to explain the flow of data.

15. Illustrate with example how files are stored in HDFS. Also, justify how HDFS is fault tolerant.

- Describe replication of data blocks with neat diagram. Explain how HDFS ensures fault tolerance through replication and heartbeat mechanisms.

16. Explain the hadoop distributed file system architecture with a neat sketch.

- Describe the architecture of HDFS, focusing on NameNodes, DataNodes, Secondary NameNode.

17. Explain the architecture and working of Hadoop YARN. Discuss its main components and how it improves the functionality of Hadoop MapReduce.

SQL vs MapReduce

Parameter	SQL	MapReduce
Access	Interactive and batch	Batch
Structure	Static	Dynamic
Updates	Read and write many times	Write once, read many times
Integrity	High	Low
Scalability	Nonlinear	Linear

Hive vs RDBMS

Hive	RDBMS
Enforces Scheme on Read.	Enforces Scheme on Write.
It uses SQL (Structured Query Language).	It uses HQL (Hive Query Language).
Hive is based on write once and read many times.	RDBMS is designed for read and write many times
Batch-oriented System. Suited for OLAP	Suited for day-to-day transaction data and supports OLTP
Scaling is at very low cost	Scaling is costlier
Parallel Computing	Serial Computing

HDFS Vs HBase

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS is WORM (Write once and read multiple times or many times)	HBase supports real-time random read and write.
HDFS is based on Google File System (GFS)	HBase is based on Google Big Table.
HDFS supports only full table scan or partition table scan	Hbase supports random small range scan or table scan.
HDFS does not support dynamic storage owing to its rigid structure.	HBase supports dynamic storage.
HDFS has high latency operations	HBase has low latency operations.
HDFS is most suitable for batch analytics.	HBase is for real-time analytics.

Hadoop vs RDBMS

Feature	Hadoop	RDBMS
Data Variety	Used for structured, semi-structured and unstructured data. Hadoop supports a variety of data formats in real time such as XML, JSON, and text-based flat file formats.	Used for structured data
Data Storage	Usually datasets of size terabytes, petabytes	Usually datasets of size gigabytes
Querying	HiveQL	SQL
Query Response	In Hadoop, there is latency due to batch processing.	In RDBMS, query response time is immediate.
Schema	Schema required on read	Schema required on write
Speed	Writes are faster compared to reads as there is no adherence to schema required at the time of inserting or writing data. Schema is enforced at read time	Reads are very fast (supported by building indexes on required columns).
	Hadoop is designed for write once read many times. It does not work for random reading and writing of a few records like RDBMS.	RDBMS is designed for read and write many times.
Cost	Apache Hadoop is open-source, large-scale, distributed, scalable, data intensive computing.	Available as proprietary RDBMS such as Oracle, MS SQL Server, IBM DB2, etc. Also open-source RDBMS are available such as MySQL, PostgreSQL, etc.
Use Cases	Analytics, data discovery	OLTP (Online Transaction Processing). Mainly used to store and process day-to-day business
Throughput	High	Low
Scalability	Horizontal (Hadoop scales by adding nodes to a Hadoop cluster of easily available commodity machines).	Vertical: RDBMS scales vertically by increasing the horsepower (CPU, Hard Disk Capacity, RAM, etc.) of the machine.
Hardware	Commodity/Utility Hardware	High End Servers
Integrity	Low	High. Obeys ACID properties A – Atomicity C – Consistency I – Integrity D – Durability

Hadoop vs SQL

Hadoop	SQL
Scale out	Scale up
Key–Value pairs	Relational table
Functional Programming	Declarative Queries
Offline batch processing	Online transaction processing

Hive Vs HBase

Hive	HBASE
Hive is a MapReduce-based SQL engine that runs on top of Hadoop.	HBase is a key-value NoSQL database that runs on top of HDFS.
Hive is a query engine that uses queries that are mostly similar to SQL queries.	It is Data storage, particularly for unstructured data.
It has a schema model.	It is free from the schema model.
Batch-oriented System. Suited for OLAP	Suited for day-to-day transaction data and supports OLTP (for real-time data streaming.)