Unit-2:

Introduction to Hadoop:-Introducing Hadoop, why Hadoop, why not RDBMS, RDBMS versus Hadoop, History of Hadoop, Hadoop overview, use case of Hadoop, Hadoop distributors, HDFS, Processing data with Hadoop, NoSQL, Hadoop-Features of Hadoop.

Introducing to hadoop:

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed *storage* and *computation* across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.OR. Hadoop is an open-source platform for storage and processing of diverse data types that enables data-driven enterprises to rapidly derive the complete value from all their data.

Hadoop has the capability to handle different modes of data such as structured, unstructured and semi-structured data. It gives us the flexibility to collect, process, and analyze data that our old data warehouses failed to do.

Why Hadoop?

Ever wondered why Hadoop has been and is one of the most wanted technology, the key consideration is ***Its capability to handle massive amount of data, different categories of data-fairly quickly***.

The other considerations are (key Attributes)

1. Flexible:

As it is a known fact that only 20% of data in organizations is structured, and the rest is all unstructured, it is very crucial to manage unstructured data which goes unattended. Hadoop manages different types of Big Data, whether structured or unstructured, encoded or formatted, or any other type of data and makes it useful for decision making process. Moreover, Hadoop is simple, relevant and schema-less! Though Hadoop generally supports **Java Programming**, any programming language can be used in Hadoop with the help of the **MapReduce technique**. Though Hadoop works best on **Windows** and **Linux**, it can also work on other operating systems like **BSD** and **OS X**.

2. Scalable

Hadoop is a scalable platform, in the sense that new nodes can be easily added in the system as and when required without altering the data formats, how data is loaded, how programs are written, or even without modifying the existing applications. Hadoop is an open source platform and runs on industry-standard hardware. Moreover, Hadoop is also fault tolerant –

this means, even if a node gets lost or goes out of service, the system automatically reallocates work to another location of the data and continues processing as if nothing had happened!

3. Building more efficient data economy:

Hadoop has revolutionized the processing and analysis of big data world across. Till now, organizations were worrying about how to manage the non-stop data overflowing in their systems. Hadoop is more like a "Dam", which is harnessing the flow of unlimited amount of data and generating a lot of power in the form of relevant information. Hadoop has changed the economics of storing and evaluating data entirely!

4. Robust Ecosystem:

Hadoop has a very robust and a rich ecosystem that is well suited to meet the analytical needs of developers, web start-ups and other organizations. Hadoop Ecosystem consists of various related projects such as MapReduce, Hive, HBase, Zookeeper, HCatalog, Apache Pig, which make Hadoop very competent to deliver a broad spectrum of services.

Hadoop is getting more "Real-Time"!

Did you ever wonder how to stream information into a cluster and analyze it in real time? Hadoop has the answer for it. Yes, Hadoop's competencies are getting more and more real-time. Hadoop also provides a standard approach to a wide set of APIs for big data analytics comprising MapReduce, query languages and database access, and so on.

5. Cost Effective:

Loaded with such great features, the icing on the cake is that Hadoop generates cost benefits by bringing massively parallel computing to commodity servers, resulting in a substantial reduction in the cost per terabyte of storage, which in turn makes it reasonable to model all your data. The basic idea behind Hadoop is to perform cost-effective data analysis present across world wide web!

6. Upcoming Technologies using Hadoop:

With reinforcing its capabilities, Hadoop is leading to phenomenal technical advancements. For instance, HBase will soon become a vital Platform for Blob Stores (Binary Large Objects) and for Lightweight OLTP (Online Transaction Processing). Hadoop has also begun serving as a strong foundation for new-school graph and NoSQL databases, and better versions of relational databases.

7. Hadoop is getting cloudy!

Hadoop is getting cloudier! In fact, cloud computing and Hadoop are synchronizing in several organizations to manage Big Data. Hadoop will become one of the most required apps for cloud computing. This is evident from the number of Hadoop clusters offered by cloud vendors in various businesses. Thus, Hadoop will reside in the cloud soon!

8.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 another functional and available nodes and ensures that distributed computing does not fail.

Why not RDBMS?

RDBMS is not suitable for storing and processing large files ,images and videos.RDBMS is not a good choice when it comes to advanced analytics involving machine learning.

RDBMS versus HADOOP

Difference between RDBMS and Hadoop

parameters	RDBMS	Hadoop
system	Relational database management system	Node based flat structure
data	Suitable for structured data	Suitable for structured and un- structured
processing	OLTP	Analytical and Big data Processing
Choice	When data need consistent relationship	Big data Processing, which does not require any consistent relationship between data.
processor	Needs High end processors to store huge volumes of data	It requires only a processor, a network card and few hard drives
cost	$10000-$14000 per terabytes of storage	-$4000 per terabytes of storage

Distributed Computing Challenges

There are many challenges with distributed computing

Scalability: The system must remain effective when there is a significant increase in either number of resources or number of users. The architecture and algorithms must be efficiently used under these circumstances.

Security: Security in terms of confidentiality, integrity and availability must be provided in DSS. The probable threats are information leakage, integrity violation, denial of services and illegitimate usage.

Design challenges: design of DSS must take care of responsiveness, throughput, load sharing and load balancing of the tasks. Concurrency: Shared access to resources must be made available to the required processes.

Openness and Extensibility: Interfaces should be separated and publicly available to enable easy extensions to existing components and add new components [16]. Migration and load balancing: Task must be allowed to move within the system without affecting the operation of users or applications and load must be distributed among the available resources for better performance.

Security: Access to resources should be secured to ensure only known users are able to perform allowed operations.

Two major challenges

1.hardware failure

When failure happens in distributed system,how does one retriew the data that was stored in the system?hadoop has an answer to this problem in **replication factor**. replication factor can notes the number of data copies of a given data item across the network.

2.How to process this Gigantic store of data?

A key challenge is to integrate the data available on several machines prior to processing it. hadoop solve this problem by using **MapReduce** programming.its a programming model to process the data.

History of Hadoop

History of Hadoop had started in the year 2002 with the project Apache Nutch. Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine which itself is a part of Lucene Project.

2002 – 2004

Apache Nutch was started in the year 2002 by Doug Cutting which is an effort to build an open source web search engine based on Lucene and Java for the search and index component. Nutch was based on sort/merge processing. In June 2003, it was successfully demonstrated on 4 nodes by crawling 100 million pages. However they realised that their architecture wouldn't scale to billions of pages on web. There comes the help with the publication of a paper in 2003 that described the architecture of the Google's Distributed Filesystem, called GFS which has been used in production at Google which would solve their storage needs for the very large files generated as part of the web crawling and indexing process.

2004 – 2006

In the year 2004, they started writing the open source implementation called Nutch Distributed Filesystem (NDFS). In the same year Google published a paper that introduces MapReduce to the world. Early in the year 2005, the Nutch developers had a working MapReduce Implementation in Nutch and by the middle of that year all the major Nutch algorithms had been ported using the MapReduce and NDFS (Nutch Distributed FileSystem). *In Febraury, 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop.*

- 2004 : Initial versions of what is now Hadoop Distributed FileSystem and MapReduce implemented by Doug Cutting and Mike Cafarella.
- December 2005 : Nutch ported to a new framework. Hadoop runs reliably on 20 nodes.

2006 – 2008

Doug Cutting joined Yahoo! in the year 2006, which provided him the dedicated team and resources to turn Hadoop in to a system that ran at web scale. Hadoop was made Apache's top level project in the year 2008.

- February 2006 : Apache Hadoop project officially started to support the standalone development of MapReduce and HDFS.
- February 2006 : Adoption of Hadoop by Yahoo! Grid Team.
- April 2006 : Sort benchmark (10 GB/node) run on 188 nodes in 47.9.
- May 2006 : Yahoo! set up a Hadoop 300 nodes research cluster.
- May 2006 : Sort benchmark run on 500 nodes in 42 hours (better hardware than April benchmark)
- October 2006 : Research cluster reaches 600 nodes.
- December 2006 : Sort benchmark run on 20 nodes in 1.8 hours, 100 nodes in 3.3 hours, 500 nodes in 5.2 hours, 900 nodes in 7.8 hours.
- January 2007 : Research cluster reaches 900 nodes.
- April 2007 : Research clusters – two cluster of 1000 nodes.
- April 2008 : Won 1 Terabyte sort benchmark in 208 seconds on 990 nodes.
- October 2008 : Loading 10 Terabytes of data per day into research clusters.

2008 – now

After 2008 there is a full time development that is going on. There are many releases of Hadoop, you can find them here.

- March 2009 : 17 clusters with a total of 24,000 nodes.
- April 2009 : Won the minute sort by sorting 500 GB in 59 seconds on 1,400 nodes and 100 TB sort in 173 minutes on 3,400 nodes.
- 2011 : Yahoo was running its search engine across 42,000 nodes.
- July 2013 : Gray sort by sorting at a rate of 1.42 Terabytes per minute.

Hadoop Overview

Basically Hadoop accomplishes two tasks:

1. Massive data storage

2. Faster data processing.

Key Aspects of Hadoop

Open source software: it is free to download

Framework: everything is provided –programs ,tools etc.

Distributed: Divides and stores data across multiple computers.

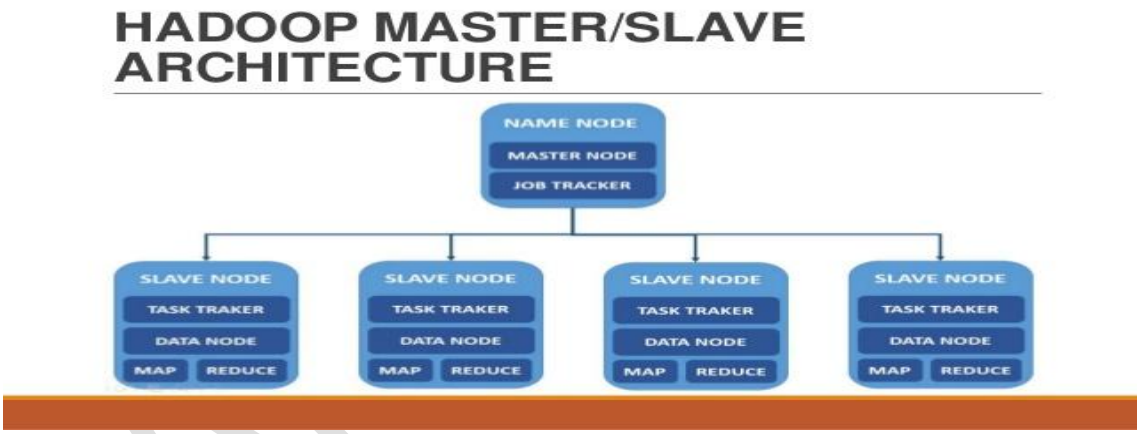
Massive data storage: stores huge amount of data across nodes of low-cost commodity hardware.

Faster data processing: large amount of data is processed in parallel.

Hadoop Conceptual Layer

It is conceptually divided into **Data Storage Layer** which stores huge volume of data and **Data Processing Layer** which processes data in parallel to extract richer and meaningful insights from data.

High-Level Architecture of Hadoop or master slave architecture



In master slave architecture master node is known as **NameNode** and slave nodes are known as **Data nodes.**

Components of master node

1.Master HDFS:its main responsibilities is partitioning the data storage across the slave nodes.iy also keep track of locations of data on DataNodes.

2.Master MapReduce:it decides and schedules computation task on slave nodes.

Hadoop Distributed File System (HDFS) is a distributed file system which is designed to run on commodity hardware. Commodity hardware is cheaper in cost. Since Hadoop requires processing power of multiple machines and since it is expensive to deploy costly hardware,

we use commodity hardware. When commodity hardware is used, failures are more common rather than an exception. HDFS is highly fault-tolerant and is designed to run on commodity hardware. HDFS provides high throughput access to the data stored. So it is extremely useful when you want to build applications which require large data sets. HDFS was originally built as infrastructure layer for Apache Nutch. Architectural diagram is shown bellow.

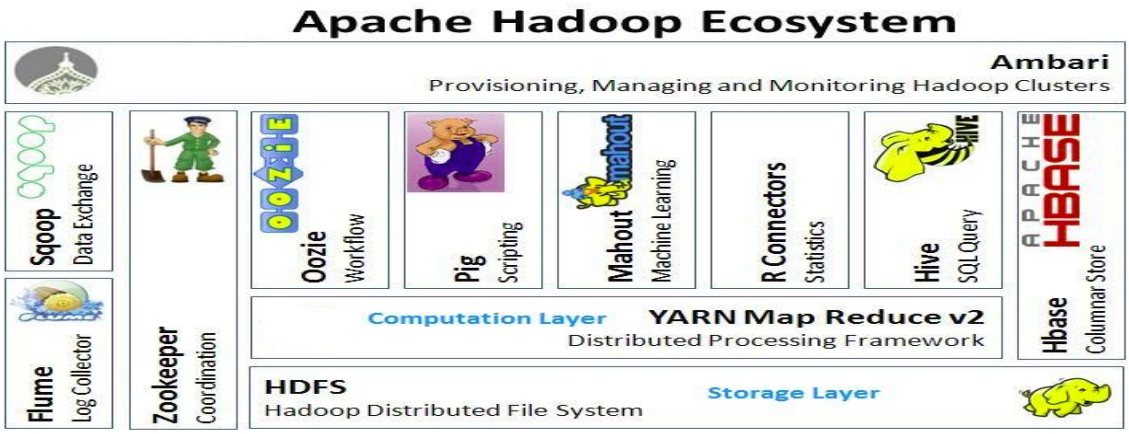
HDFS has master/slave architecture. In this architecture one of the machines will be designated as a master node (or name node). Every other machine would be acting as slave (or data node). NameNode/DataNode are java processes that run on the machines when Hadoop software is installed. NameNode is responsible for managing the metadata about the HDFS Files. This metadata includes various information about the HDFS File such as Name of the file, File Permissions, FileSize, Blocks etc. It is also responsible for performing various namespace operations like opening, closing, renaming the files or directories.

Whenever a file is to be stored in HDFS, it is divided into blocks. By default, blocksize is 64MB (Configurable). These blocks are replicated (default is 3) and stored across various datanodes to take care of hardware failures and for faster data transfers. NameNode maintains a mapping of blocks to DataNodes. DataNodes serves the read and write requests from HDFS file system clients. They are also responsible for creation of block replicas and for checking if blocks are corrupted or not. It sends the ping messages to the NameNode in the form of block mappings.

Key points of HDFS

1. Storage components of Hadoop
2. Distributed file system
3. Modelled after Google file system
4. Optimized for high throughput .
5. Replicate a file for a configured no of times.
6. Re-replicate data blocks automatically on nodes that have failed

Hadoop ecosystem



Hadoop ecosystem is a platform or framework which helps in solving the big data problems. It comprises of different components and services (ingesting, storing, analyzing, and maintaining) inside of it. Most of the services available in the Hadoop ecosystem are to supplement the main four core components of Hadoop which include HDFS, YARN, MapReduce and Common.

Hadoop ecosystem includes both Apache Open Source projects and other wide variety of commercial tools and solutions. Some of the well known open source examples include Spark, Hive, Pig, Sqoop and Oozie.

As we have got some idea about what is Hadoop ecosystem, what it does, and what are its components, let's discuss each concept in detail.

YARN:

YARN (Yet Another Resource Negotiator) acts as a brain of the Hadoop ecosystem. It takes responsibility in providing the computational resources needed for the application executions

YARN consists of two essential components. They are Resource Manager and Node Manager

Resource Manager

- It works at the cluster level and takes responsibility oforrunning the master machine.
- It stores the track of heartbeats from the Node manager.
- It takes the job submissions and negotiates the first container for executing an application.
- It consists of two components: Application manager and Scheduler.

Node manager:

- It works on node level component and runs on every slave machine.
- It is responsible for monitoring resource utilization in each container and managing containers.
- It also keeps track of log management and node health.
- It maintains continuous communication with a resource manager to give updates.

MapReduce

MapReduce acts as a core component in Hadoop Ecosystem as it facilitates the logic of processing. To make it simple, MapReduce is a software framework which enables us in writing applications that process large data sets using distributed and parallel algorithms in a Hadoop environment.

Parallel processing feature of MapReduce plays a crucial role in Hadoop ecosystem. It helps in performing Big data analysis using multiple machines in the same cluster.

How does MapReduce work

In the MapReduce program, we have two Functions; one is Map, and the other is Reduce.

Map function: It converts one set of data into another, where individual elements are broken down into tuples. (key /value pairs).

Reduce function: It takes data from the Map function as an input. Reduce function aggregates & summarizes the results produced by Map function.

Apache Spark:

Apache Spark is an essential product from the Apache software foundation, and it is considered as a powerful data processing engine. Spark is empowering the big data applications around the world. It all started with the increasing needs of enterprises and where MapReduce is unable to handle them.

The growth of large unstructured amounts of data increased need for speed and to fulfill the real-time analytics led to the invention of Apache Spark.

Spark Features:

- It is a framework for real-time analytics in a distributed computing environment.
- It acts as an executor of in-memory computations which results in increased speed of data processing compared to MapReduce.
- It is 100X faster than Hadoop while processing data with its exceptional in-memory execution ability and other optimization features.

Spark is equipped with high-level libraries, which support R, Python, Scala, Java etc. These standard libraries make the data processing seamless and highly reliable. Spark can process the enormous amounts of data with ease and Hadoop was designed to store the unstructured data which must be processed. When we combine these two, we get the desired results.

Hive:

Apache Hive is a data warehouse open source software built on Apache Hadoop for performing data query and analysis. Hive mainly does three functions; data summarization, query, and analysis. Hive uses a language called HiveQL(HQL), which is similar to SQL. Hive QL works as a translator which translates the SQL queries into MapReduce Jobs, which will be executed on Hadoop.

Main components of Hive are:

Metastore- It serves as a storage device for the metadata. This metadata holds the information of each table such as location and schema. Metadata keeps track of data and replicates it, and acts as a backup store in case of data loss.

Driver- Driver receives the HiveQL instructions and acts as a Controller. It observes the progress and life cycle of various executions by creating sessions. Whenever HiveQL executes a statement, driver stores the metadata generated out of that action.

Compiler- The compiler is allocated with the task of converting the HiveQL query into MapReduce input. A compiler is designed with the process to execute the steps and functions needed to enable the HiveQL output, as required by the MapReduce.

H Base:

Hbase is considered as a Hadoop database, because it is scalable, distributed, and because NoSQL database that runs on top of Hadoop. Apache HBase is designed to store the structured data on table format which has millions of columns and billions of rows. HBase gives access to get the real-time data to read or write on HDFS.

HBase features:

- HBase is an open source, NoSQL database.
- It is featured after Google's big table, which is considered as a distributed storage system designed to handle big data sets.
- It has a unique feature to support all types of data. With this feature, it plays a crucial role in handling various types of data in Hadoop.
- The HBase is originally written in Java, and its applications can be written in Avro, REST, and Thrift APIs.

Components of HBase:

There are majorly two components in HBase. They are HBase master and Regional server.

a) HBase master: It is not part of the actual data storage, but it manages load balancing activities across all RegionServers.

- It controls the failovers.
- Performs administration activities which provide an interface for creating, updating and deleting tables.
- Handles DDL operations.
- It maintains and monitors the Hadoop cluster.

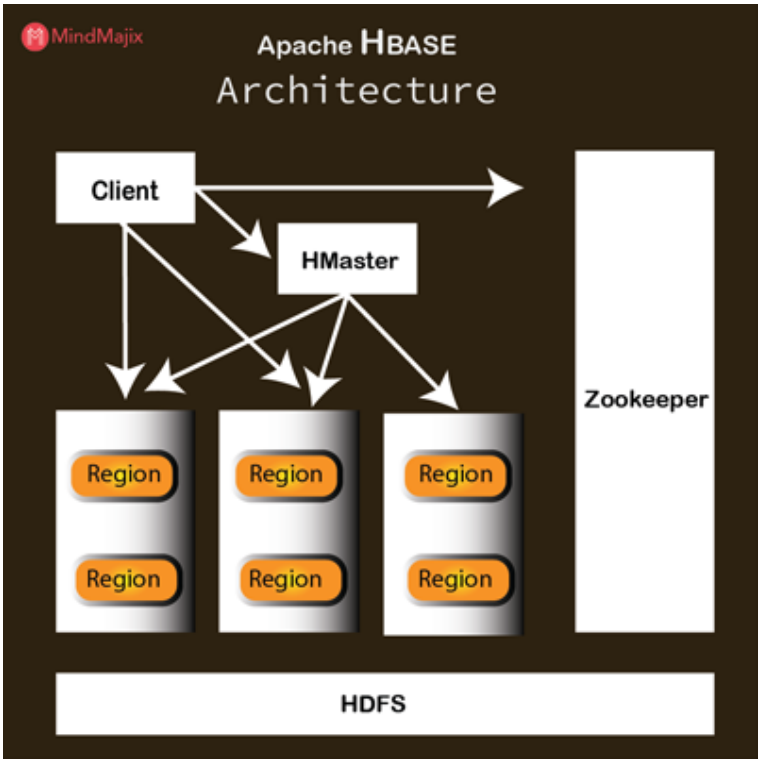
b) Regional server: It is a worker node. It reads, writes, and deletes request from Clients. Region server runs on every node of Hadoop cluster. Its server runs on HDFS data nodes.

H Catalogue:

H Catalogue is a table and storage management tool for Hadoop. It exposes the tabular metadata stored in the hive to all other applications of Hadoop. H Catalogue accepts all kinds of components available in Hadoop such as Hive, Pig, and MapReduce to quickly read and write data from the cluster. H Catalogue is a crucial feature of Hive which allows users to store their data in any format and structure.

H Catalogue defaulted supports CSV, JSON, RCFile,ORC file from and sequenceFile formats.

Benefits of H Catalogue:

- It assists the integration with the other Hadoop tools and provides read data from a Hadoop cluster or write data into a Hadoop cluster. It allows notifications of data availability.
- It enables APIs and web servers to access the metadata from hive metastore.
- It gives visibility for data archiving and data cleaning tools.
- 

Apache Pig:

Apache Pig is a high-level language platform for analyzing and querying large data sets that are stored in HDFS. Pig works as an alternative language to Java programming for MapReduce and generates MapReduce functions automatically. Pig included with Pig Latin, which is a scripting language. Pig can translate the Pig Latin scripts into MapReduce which can run on YARN and process data in HDFS cluster.

Pig is best suitable for solving complex use cases that require multiple data operations. It is more like a processing language than a query language (ex:Java, SQL). Pig is considered as a highly customized one , because the users have a choice to write their functions by using their preferred scripting language.

How does Pig work?

We use 'load' command to load the data in the pig. Then, we can perform various functions such as grouping data, filtering, joining, sorting etc. At last, you can dump the data on a screen, or you can store the result back in HDFS according to your requirement.

Apache Sqoop:

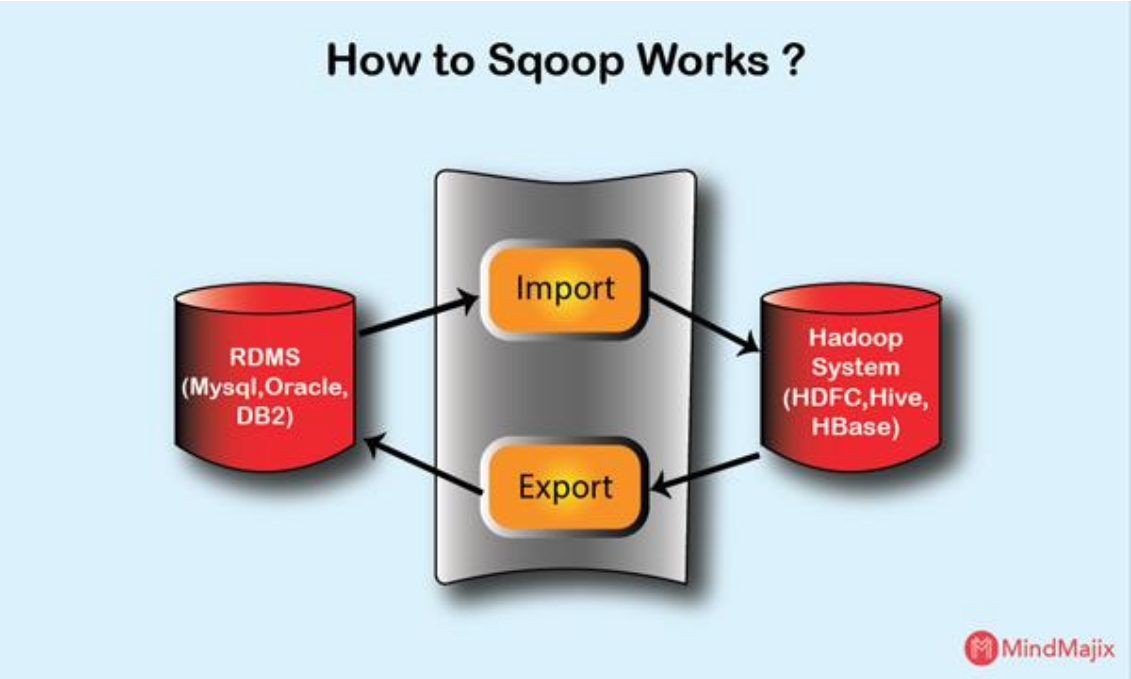
Sqoop works as a front-end loader of Big data. Sqoop is a front-end interface that enables in moving bulk data from Hadoop to relational databases and into variously structured data marts.

Sqoop replaces the function called 'developing scripts' to import and export data. It mainly helps in moving data from an enterprise database to Hadoop cluster to performing the ETL process.

What Sqoop does:

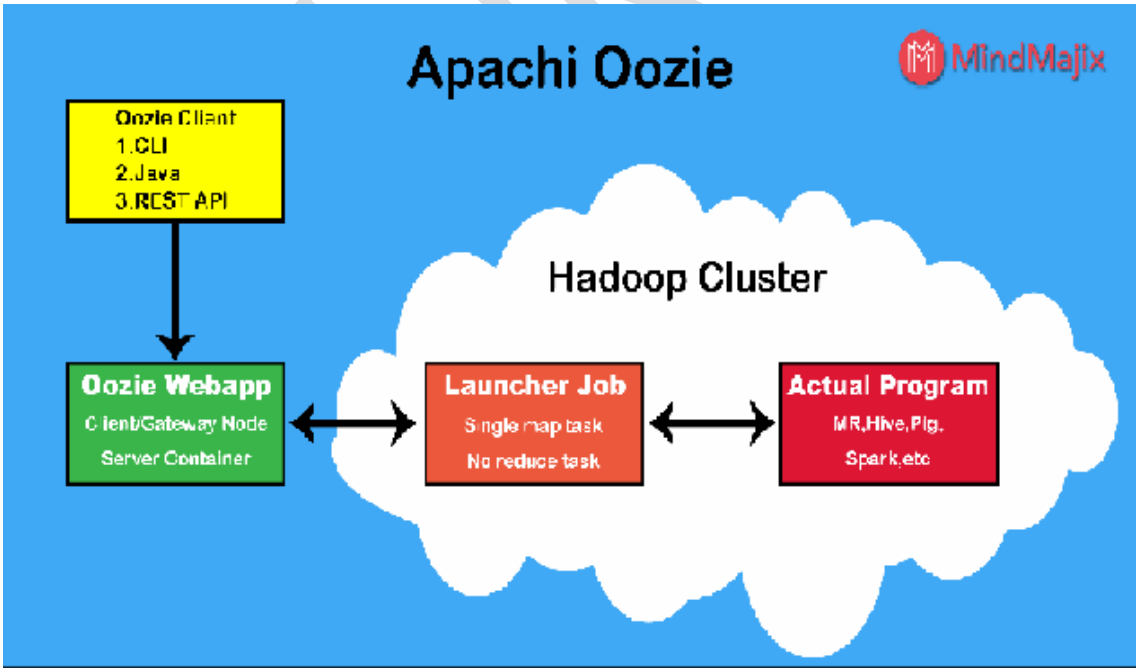
Apache Sqoop undertakes the following tasks to integrate bulk data movement between Hadoop and structured databases.

- Sqoop fulfills the growing need to transfer data from the mainframe to HDFS.
- Sqoop helps in achieving improved compression and light-weight indexing for advanced query performance.
- It facilitates feature to transfer data parallelly for effective performance and optimal system utilization.
- Sqoop creates fast data copies from an external source into Hadoop.
- It acts as a load balancer by mitigating extra storage and processing loads to other devices.



Oozie:

Apache Ooze is a tool in which all sort of programs can be pipelined in a required manner to work in Hadoop's distributed environment. Oozie works as a scheduler system to run and manage Hadoop jobs.



Oozie allows combining multiple complex jobs to be run in a sequential order to achieve the desired output. It is strongly integrated with Hadoop stack supporting various jobs like Pig,

Hive, Sqoop, and system-specific jobs like Java, and Shell. Oozie is an open source Java web application.

Oozie consists of two jobs:

1. Oozie workflow: It is a collection of actions arranged to perform the jobs one after another. It is just like a relay race where one has to start right after one finish, to complete the race.

2. Oozie Coordinator: It runs workflow jobs based on the availability of data and predefined schedules.

Avro:

Apache Avro is a part of the Hadoop ecosystem, and it works as a data serialization system. It is an open source project which helps Hadoop in data serialization and data exchange. Avro enables big data in exchanging programs written in different languages. It serializes data into files or messages.

Avro Schema: Schema helps Avaro in serialization and deserialization process without code generation. Avro needs a schema for data to read and write. Whenever we store data in a file it's schema also stored along with it, with this the files may be processed later by any program.

Dynamic typing: it means serializing and deserializing data without generating any code. It replaces the code generation process with its statistically typed language as an optional optimization.

Avro features:

- Avro makes Fast, compact, dynamic data formats.
- It has Container file to store continuous data format.
- It helps in creating efficient data structures.

Apache Drill :

The primary purpose of Hadoop ecosystem is to process the large sets of data either it is structured or unstructured. Apache Drill is the low latency distributed query engine which is designed to measure several thousands of nodes and query pet bytes of data. The drill has a specialized skill to eliminate cache data and releases space.

Features of Drill:

- It gives an extensible architecture at all layers.
- Drill provides data in a hierarchical format which is easy to process and understandable.

- The drill does not require centralized metadata, and the user doesn't need to create and manage tables in metadata to query data.

Apache Zookeeper:

Apache Zookeeper is an open source project designed to coordinate multiple services in the Hadoop ecosystem. Organizing and maintaining a service in a distributed environment is a complicated task. Zookeeper solves this problem with its simple APIs and Architecture. Zookeeper allows developers to focus on core application instead of concentrating on a distributed environment of the application.

Features of Zookeeper:

- Zookeeper acts fast enough with workloads where reads to data are more common than writes.
- Zookeeper acts as a disciplined one because it maintains a record of all transactions.

Apache Flume:

Flume collects, aggregates and moves large sets of data from its origin and send it back to HDFS. It works as a fault tolerant mechanism. It helps in transmitting data from a source into a Hadoop environment. Flume enables its users in getting the data from multiple servers immediately into Hadoop.

Apache Ambari:

Ambari is an open source software of Apache software foundation. It makes Hadoop manageable. It consists of software which is capable of provisioning, managing, and monitoring of Apache Hadoop clusters. Let's discuss each concept.

Hadoop cluster provisioning: It guides us with a step-by-step procedure on how to install Hadoop services across many hosts. Ambari handles configuration of Hadoop services across all clusters.

Hadoop Cluster management: It acts as a central management system for starting, stopping and reconfiguring of Hadoop services across all clusters.

Hadoop cluster monitoring: Ambari provides us with a dashboard for monitoring health and status.

The Ambari framework acts as an alarming system to notify when anything goes wrong. For example, if a node goes down or low disk space on node etc, it intimates us through notification.

Use CASE of Hadoop

clickStream Data(mouse clicks):helps to understand the purchasing behaviour of customers. clickStreamalso helps to online marketers to optimize their product web pages,promotional contents etc.

clickStream analysis using Hadoop provides following 3 benefits.

1.Hadoop helps to join clickStream data with other data sources such as customer relationship data.this additional data often provides the much needed information to understand customer behaviour.

2.hadoop scalability property helps you to store years of data without ample increment cost.

3.business analysis can use **Apache pig** or **Apache Hive** for website analysis.with these tools we can organize clickStream data by user session ,refine it ,and feed it to visualization or analytics tools.

Hadoop common distributors:

The following companies provides products that include apache Hadoop,commercial support,tools and utilities related to hadoop.

Cloudera CDH:4.0/5.0

MAPR:M3,M5,M8

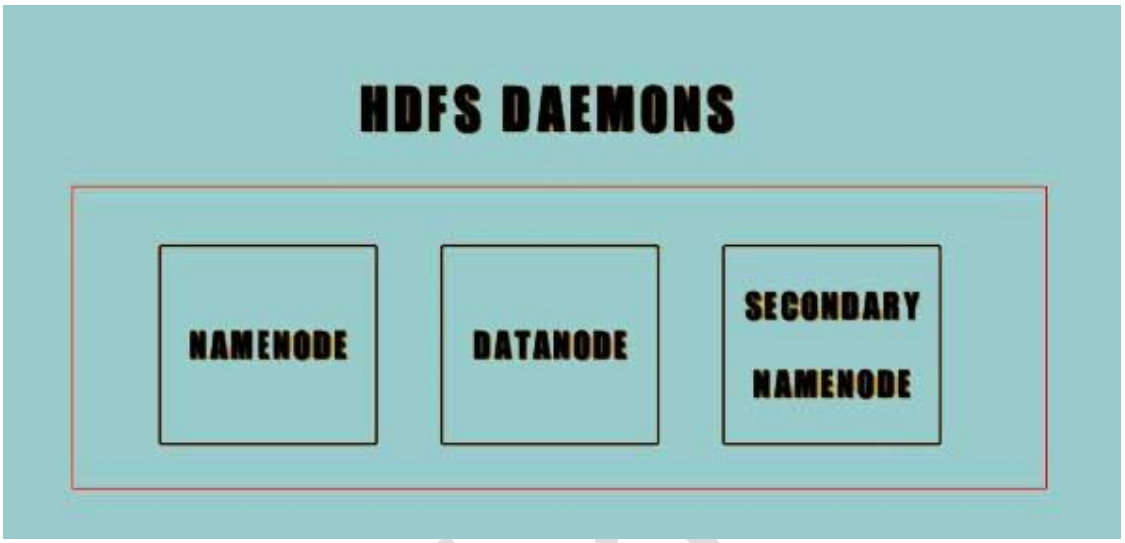
Apache Hadoop: Hadoop 1.0/2.0

Hadoop Distributed file System

1. Key points of Hadoop Distributed File System are as follws.
2. Storage components of Hadoop
3. Distributed File System
4. Modeled after google File System
5. Optimized for high throughput
6. You can replicate a file for a configured no of times,which is tolerant in terms of both software and hardware
7. Re-replicate data block automatically on nodes that have Failed.
8. You can realize the power of HDFS when you perform read or write on large files

This HDFS consists of three Daemons which are:-

- Namenode
- Datanode
- Secondary Namenode.



NameNode:

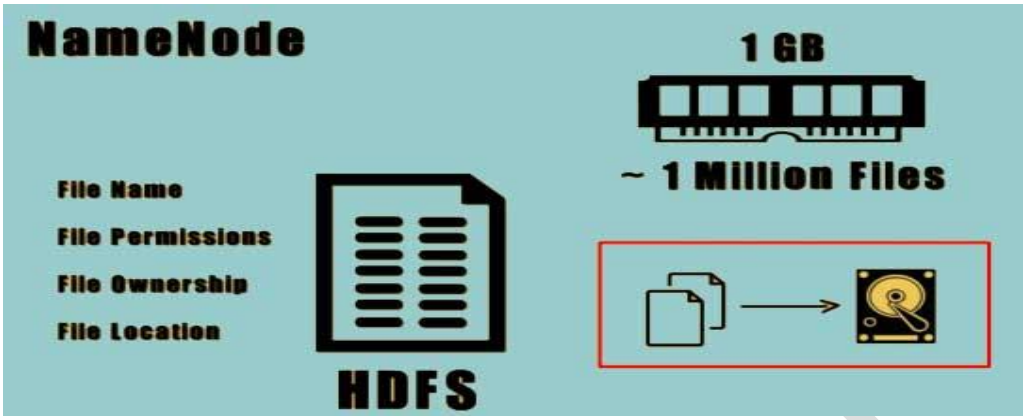
- NameNode is a daemon which maintains and operates all DATA nodes (slave nodes).
- It acts as the recorder of metadata for all blocks in it, and it contains information like size, location, source, and hierarchy, etc.
- It records all changes that happen to metadata.
- If any file gets deleted in the HDFS, the NameNode will automatically record it in EditLog.
- NameNode frequently receives heartbeat and block report from the data nodes in the cluster to ensure they are working and live.

All the nodes work the primary slave architecture.

The Namenode is the master node while the data node is the slave node. Within the HDFS, there is only a single Namenode and multiple Datanodes.

Functionality of Nodes

The Namenode is used for storing the **metadata of the HDFS**. This metadata keeps track and stores information about all the files in the HDFS. All the information is stored in the RAM. Typically the Namenode occupies around 1 GB of space to store around 1 million files.



The Datanodes are responsible for retrieving and storing information as instructed by the Namenode. They periodically report back to the Namenodes about their status and the files they are storing through a heartbeat. The Datanodes stores multiple copies for each file that is present within the Hadoop distributed file system.

DataNode:

- It acts as a slave node daemon which runs on each slave machine.
- The data nodes act as a storage device.
- It takes responsibility to serve read and write request from the user.
- It takes the responsibility to act according to the instructions of NameNode, which includes deleting blocks, adding blocks, and replacing blocks.
- It sends heartbeat reports to the NameNode regularly and the actual time is once in every 3 seconds.

Secondary Name Node?

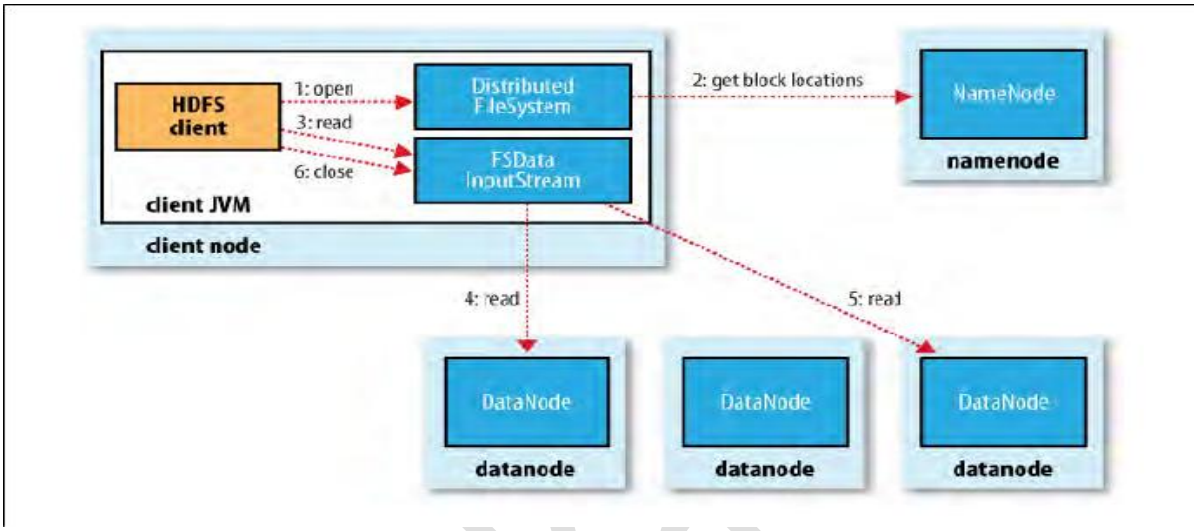
Role of Secondary Namenode in Managing the Filesystem Metadata.Each and every tr In such cases, if the Namenode fails due to corrupted meta data or any other reason than It has to retrieve the FS image from the disk and apply all the transactions to it present within the edit log file. In order to apply all these transactions, the system resources should be available. It also takes lot of time to apply all these transactions. Until all these transactions are not applied the contents of the FS image are inconsistent hence the cluster they cannot be operational. ansaction that occurs on the file system is recorded within the edit log file. At some point of time this file becomes very large.

Secondary Name Node?

Role of Secondary Namenode in Managing the Filesystem Metadata.Each and every tr In such cases, if the Namenode fails due to corrupted meta data or any other reason than It has to retrieve the FS image from the disk and apply all the transactions to it present within the edit log file. In order to apply all these transactions, the system resources should be available. It also takes lot of time to apply all these transactions. Until all these transactions are not applied the contents of the FS image are inconsistent hence the cluster they cannot be operational. ansaction that occurs on the file system is recorded within the edit log file. At some point of time this file becomes very large.

Anatomy of a File Read

To get an idea of how data flows between the client interacting with HDFS, the namenode and the datanodes, consider Figure, which shows the main sequence of events when reading a file.



The client opens the file it wishes to read by calling open() on the FileSystem object, which for HDFS is an instance of distributedFileSystem (step 1 in Figure). DistributedFileSystem calls the namenode, using RPC, to determine the locations ofthe blocks for the first few blocks in the file (step 2). For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the datanodes are sorted according to their proximity to the client (according to the topologyof the cluster's network; see "Network Topology and Hadoop"). If the client is itself a datanode (in the case of a MapReduce task, for instance), then it will read from the local datanode, if it hosts a copy of the block.

The DistributedFileSystem returns an FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from. FSDataInputStream in turn wrapsatanode and namenode I/O.

The client then calls read() on the stream (step 3). DFSInputStream, which has stored the first block in the file. Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream (step 4). When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block (step 5).

This happens transparently to the client, which from its point of view is just reading a continuous stream. Blocks are read in order with the DFSInputStream opening new connections to datanodes as the client reads through the stream. It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls close() on the FSDataInputStream (step 6).

During reading, if the DFSInputStream encounters an error while communicating with a datanode, then it will try the next closest one for that block. It will also rememberoesn't needlessly retry them for later blocks. The DFSInputStream also verifies checksums for the data transferred to it from the datanode. If a corrupted block is found, it is reported to the namenode before the DFSInput Stream attempts to read a replica of the block from another datanode.

One important aspect of this design is that the client contacts datanodes directly to retrieve data and is guided by the namenode to the best datanode for each block. This design allows HDFS to scale to a large number of concurrent clients, since the data traffic is spread across all the datanodes in the cluster. The namenode meanwhile merely has to service block location requests (which it stores in memory, making them very efficient) and does not, for example, serve data, which would quickly become a bottleneck as the number of clients grew.

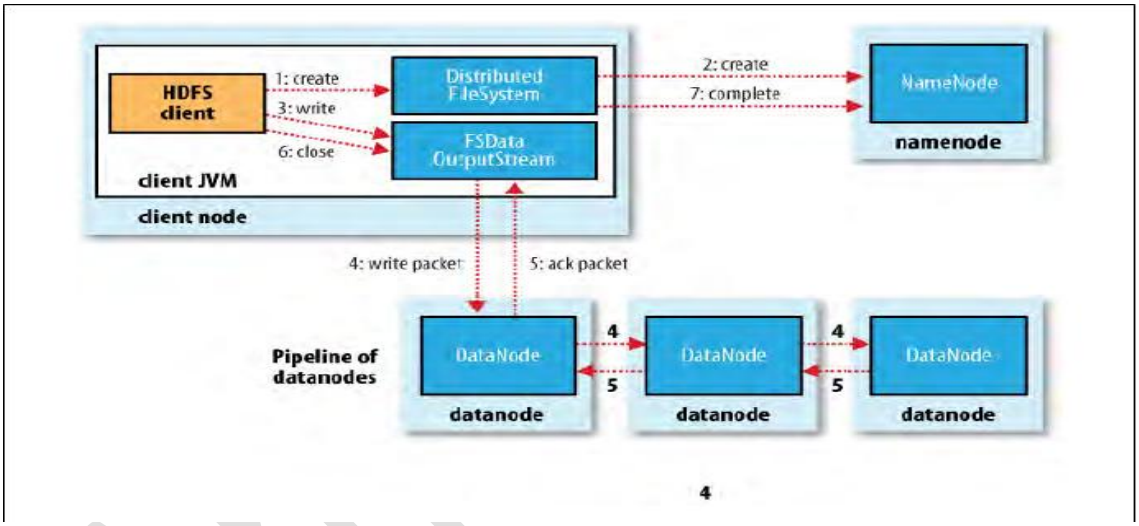
Anatomy of a File Write

Next we'll look at how files are written to HDFS. Although quite detailed, it is instructive to understand the data flow since it clarifies HDFS's coherency model. In case of creating a new file, writing data to it, then closing the file.

The client creates the file by calling create() on DistributedFileSystem (step 1 in Figure). DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it (step 2). The namenode performs various checks to make sure the file doesn't already exist, and that the client has the right permissions to create the file. If these checks pass, the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an IOException. The DistributedFileSystem returns an FSDataOutputStream for the client to start writing data to.

Just as in the read case, SDataOutputStream wraps a DFSOutput Stream, which handles communication with the datanodes and namenode.

As the client writes data (step 3), DFSOutputStream splits it into packets, which it writes to an internal queue, called the data queue. The data queue is consumed by the Datanode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline we'll assume the replication level is three, so there are three nodes in the pipeline. The DataStreamer streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline (step 4).



DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the ack queue. A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline (step 5). If a datanode fails while data is being written to it, then the following actions are taken, which are transparent to the client writing the data. First the pipeline is closed, and any packets in the ack queue are added to the front of the data queue so that datanodes that are downstream from the failed node will not miss any packets. The current block on the good datanodes is given a new identity, which is communicated to the namenode, so that the partial block on the failed datanode will be deleted if the failed datanode recovers later on. The failed datanode is

removed from the pipeline and the remainder of the block's data is written to the two good datanodes in the pipeline. The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node. Subsequent blocks are then treated as normal.

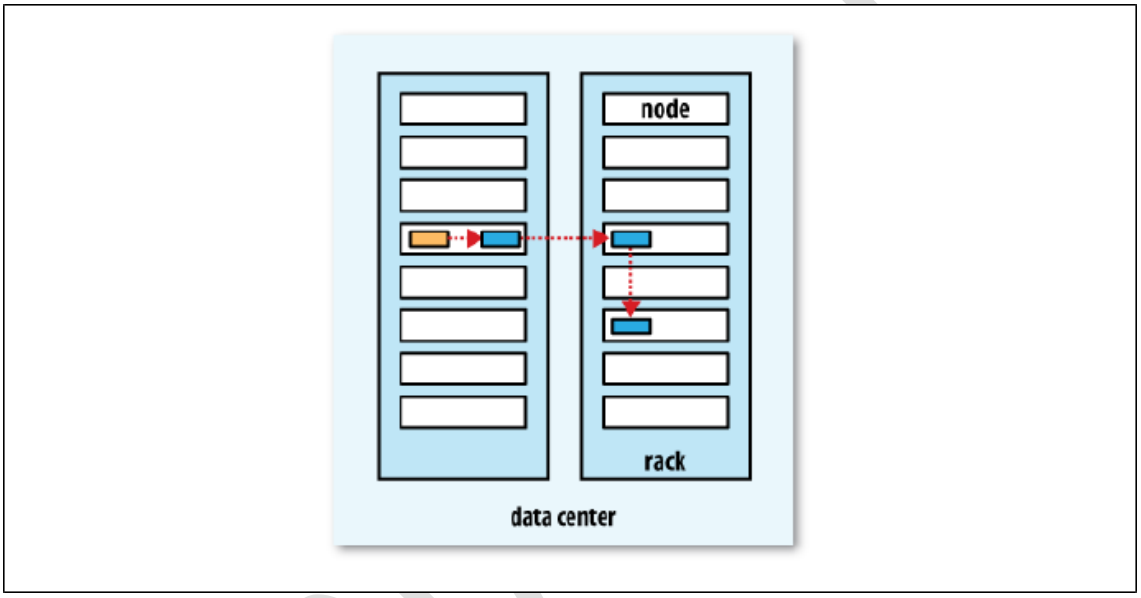
It's possible, but unlikely, that multiple datanodes fail while a block is being written. As long as dfs.replication.min replicas (default one) are written, the write will succeed, and the block will be asynchronously replicated across the cluster until its target replicationfactor is reached (dfs.replication, which defaults to three). When the client has finished writing data, it calls close() on the stream (step 6). This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete (step 7). The namenode already knows which blocks the file is made up of (via Data Streamer asking for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully.

Replica Placement

How does the namenode choose which datanodes to store replicas on? There's a tradeoff between reliability and write bandwidth and read bandwidth here. For example, placing all replicas on a single node incurs the lowest write bandwidth penalty since the replication pipeline runs on a single node, but this offers no real redundancy (if the node fails, the data for that block is lost). Also, the read bandwidth is high for off-rack reads. At the other extreme, placing replicas in different data centers may maximize redundancy, but at the cost of bandwidth. Even in the same data center (which is what all Hadoop clusters to date have run in), there are a variety of placement strategies. Indeed, Hadoop changed its placement strategy in release 0.17.0 to one that helps keep a fairly even distribution of blocks across the cluster. And from 0.21.0, block placement policies are pluggable.

Hadoop's default strategy is to place the first replica on the same node as the client (for clients running outside the cluster, a node is chosen at random, although the system tries not to pick nodes that are too full or too busy). The second replica is placed on a different rack from the first (off-rack), chosen at random. The third replica is placed on the same rack as the second, but on a different node chosen at random. Further replicas are placed on random

nodes on the cluster, although the system tries to avoid placingtoo many replicas on the same rack.Once the replica locations have been chosen, a pipeline is built, taking network topology into account. For a replication factor of 3, the pipeline might look likeOverall, this strategy gives a good balance among reliability (blocks are stored on two racks), write bandwidth (writes only have to traverse a single network switch), read performance (there's a choice of two racks to read from), and block distribution acrossthe cluster (clients only write a single block on the local rack).



Working with HDFS Commands

To get the list of files and directories at the roots of HDFS

hadoop fs –ls /

To get the list of complete files and directories of HDFS

hadoop fs –ls –R /

To copy file from local file system to HDFS

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

To copy file from HDFS to local file system

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

To copy file from local file system to HDFS via copyFromLocal command

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

To copy file from Hadoop file system to Local file system via copyToLocal command.

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

To display the the contents of an HDFS file on console

hadoop fs –cat/sample/test.txt

To copy a file from one directory to another on HDFS

hadoop fs –cp/sample/test.txt/sample1

To remove a directory from HDFS

hadoop fs –rm-r/sample1

Special Features of HDFS

1.data replication

2.data pipeline

Processing Data with Hadoop

Hadoop MapReduce processes a huge amount of data in parallel by dividing the job into a set of independent tasks (sub-job). In Hadoop, MapReduce works by breaking the processing into phases: **Map tasks** and **Reduce tasks.**

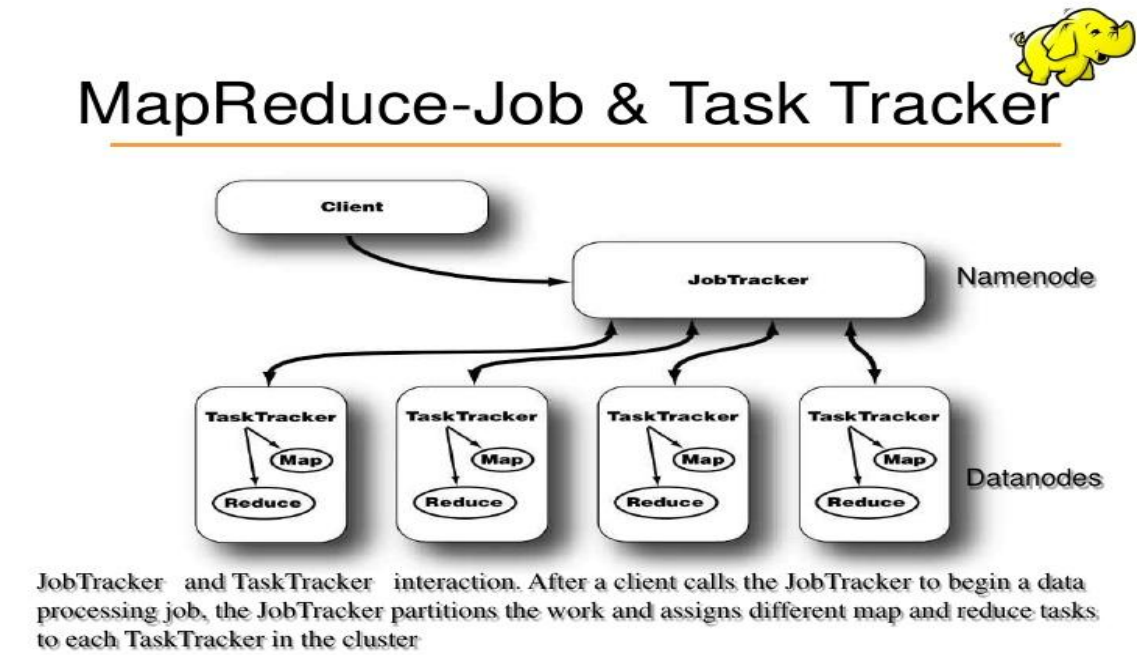
- **Map stage** − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
- **Reduce stage** − This stage is the combination of the **Shuffle**stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

MapReduce Daemons

- **JobTracker** – it provides the connectivity between Hadoop and your Application .it Schedules jobs and tracks the assign jobs to Task tracker.it also monitor all the running tasks.when task fails ,it automatically re-schedule the task to different node after a predefined no of retries.it is a master daemon responsible for executing overall MapReduce job.there is a single Job Tracker per Hadoop cluster.

- **Task Tracker** − Tracks the task and reports status to JobTracker. it is a responsible for executing individual task that is assigned by the JobTracker. There is a single Task Trackerper slave and spawn multiple java Virtual Machines(JVMs)to handle multiple map or reduce task in parallel. Interaction of **job** tracker and **task** tracker is shown in the bellow diagram.

How does Map Reduce works?

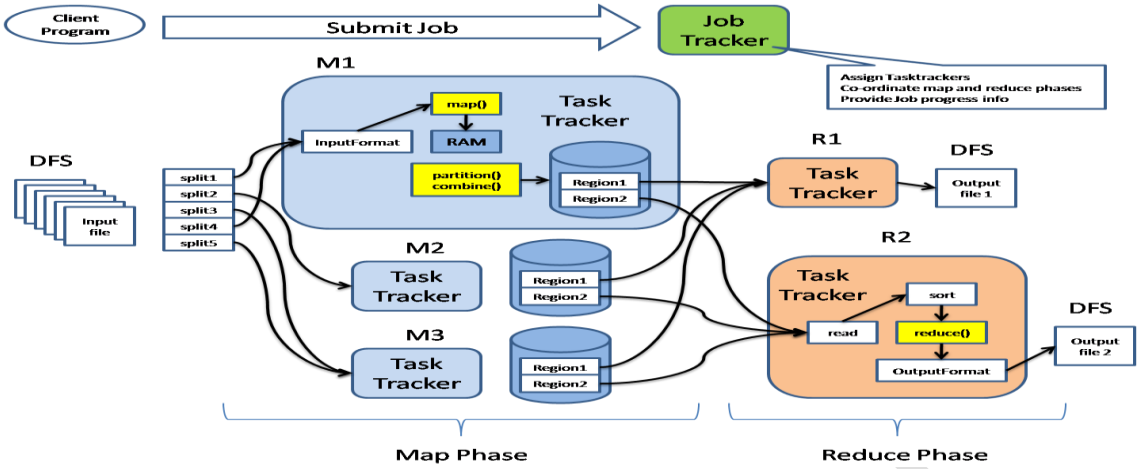


JobTracker and TaskTracker interaction. After a client calls the JobTracker to begin a data processing job, the JobTracker partitions the work and assigns different map and reduce tasks to each TaskTracker in the cluster

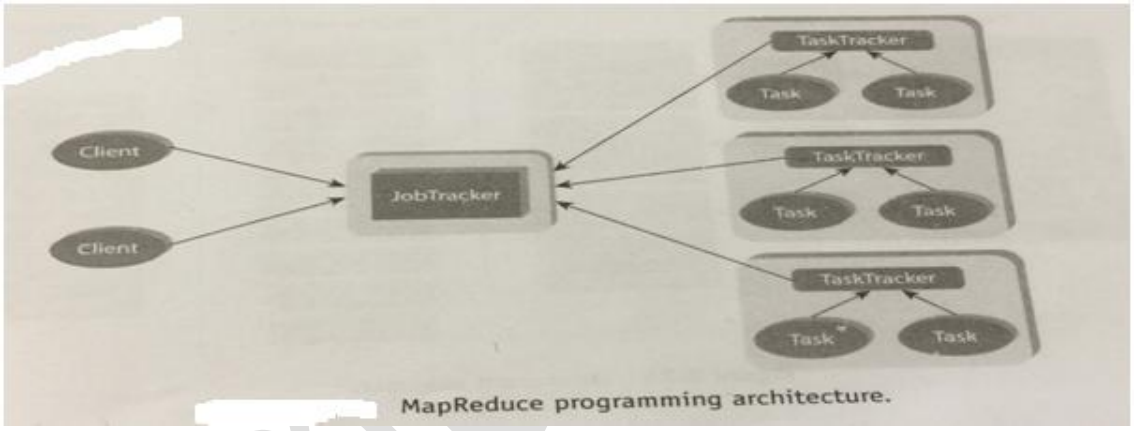
MapReduce programs work in two phases:

1. Map phase
2. Reduce phase.

An input to each phase is **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**. Following MapReduce programming workflow diagrams depicts the working of MapReduce .



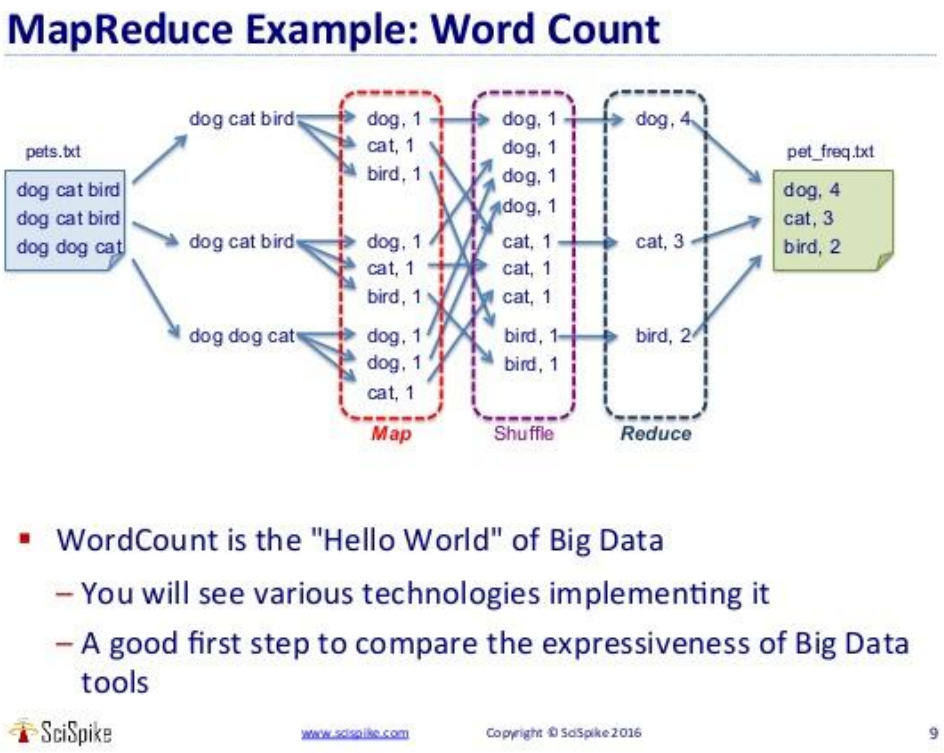
The following steps and diagram describes how MapReduce perform its tasks.



1. The input dataset is split into multiple pieces of data.
2. The framework creates a master and several workers processes and executes the worker processes remotely.
3. Several map tasks works simultaneously and read pieces of data that were assigned to each map task.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 workerd to grt the key/value data for their partition.the data thus received is shuffled and sorted as per the keys.
6. Then it call 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 transfer the control to the user program.

MapReduce Example:



What is a Relational Database?

Traditional RDBMS (relational database management system) have been the de facto standard for database management throughout the age of the internet. The architecture behind RDBMS is such that data is organized in a highly-structured manner, following the relational model. Though, RDBMS is now considered to be a declining database technology. While the precise organization of the data keeps the warehouse very "neat", the need for the data to be well-structured actually becomes a substantial burden at extremely large volumes, resulting in performance declines as size gets bigger. Thus, RDBMS is generally not thought of as a scalable solution to meet the needs of 'big'data.

What is NoSQL?

NoSQL (commonly referred to as "Not Only SQL") represents a completely different framework of databases that allows for high-performance, agile processing of information at massive scale. In other words, it is a database infrastructure that as been very well-adapted to the heavy demands of big data.

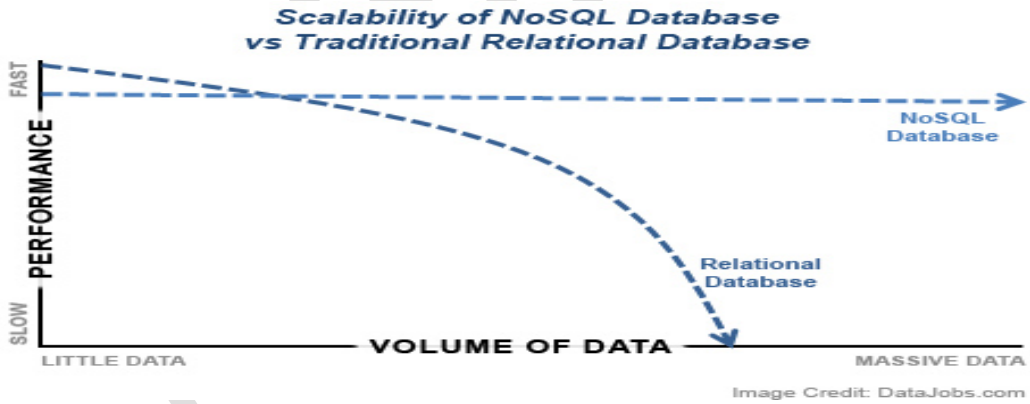
The efficiency of NoSQL can be achieved because unlike relational databases that are highly structured, NoSQL databases are unstructured in nature, trading off stringent consistency requirements for speed and agility. NoSQL centers around the concept of distributed databases, where unstructured data may be stored across multiple processing nodes, and often across multiple servers. This distributed architecture allows NoSQL databases to be horizontally scalable; as data continues to explode, just add more hardware to keep up, with no slowdown in performance. The NoSQL distributed database infrastructure has been the solution to handling some of the biggest data warehouses on the planet – i.e. the likes of Google, Amazon, and the CIA.

1.NoSQL databases are non-relational:they do not adhere to relational data model.In fact,they are either key-value pairs or document-oriented or column oriented or graph based databases.

2.**distributed** :they are distributed meaning the data is distributed across several nodes in a cluster constituted of low-cost commodity hardware.

3.No support for ACID properties.:They do not offer support for ACID properties of transaction.They have adherence to browser CAP(consistency,availability,partition tolerance) theorem.

4.No fixed table schema: They do not mandate for the data to strictly adhere to any schema structure at the time of storage.



Difference between SQL and MapReduce

	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