SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY

Big data Analytics

R18 REGULATION

BTech III-II Semester

Unit 4 Notes

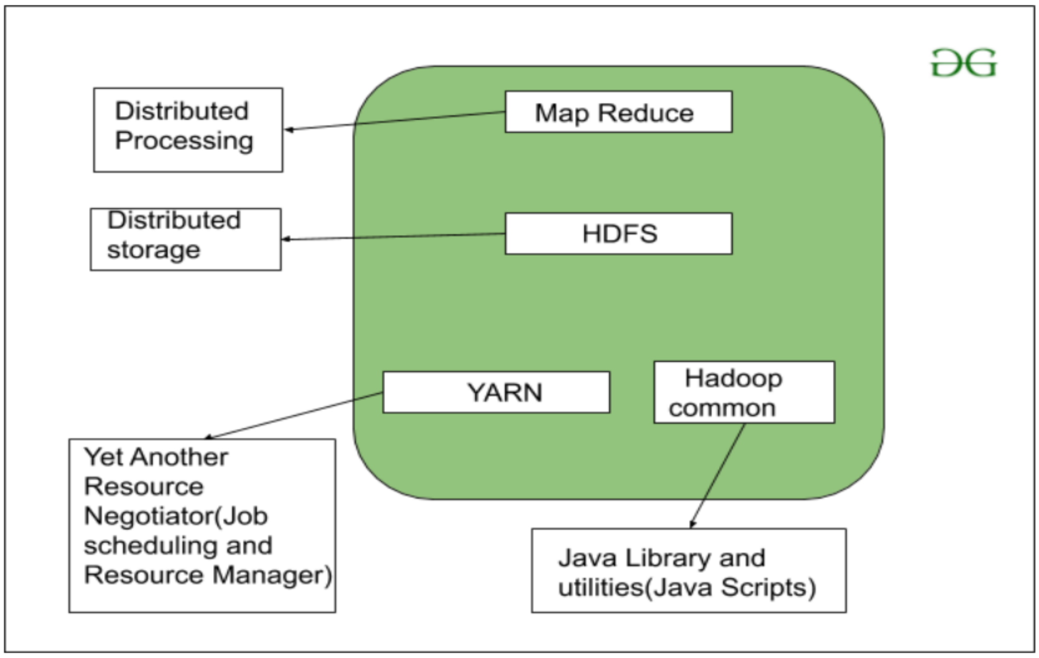
UNIT IV
Hadoop Architecture: Hadoop: RDBMS Vs Hadoop, Hadoop Overview, Hadoop distributors, HDFS, HDFS Daemons, Anatomy of File Write and Read., Name Node, Secondary Name Node, and Data Node, HDFS Architecture, Hadoop Configuration, Map Reduce Framework, Role of HBase in Big Data processing, HIVE, PIG.

Hadoop Architecture: Hadoop: RDBMS Vs Hadoop

Hadoop Architecture:

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

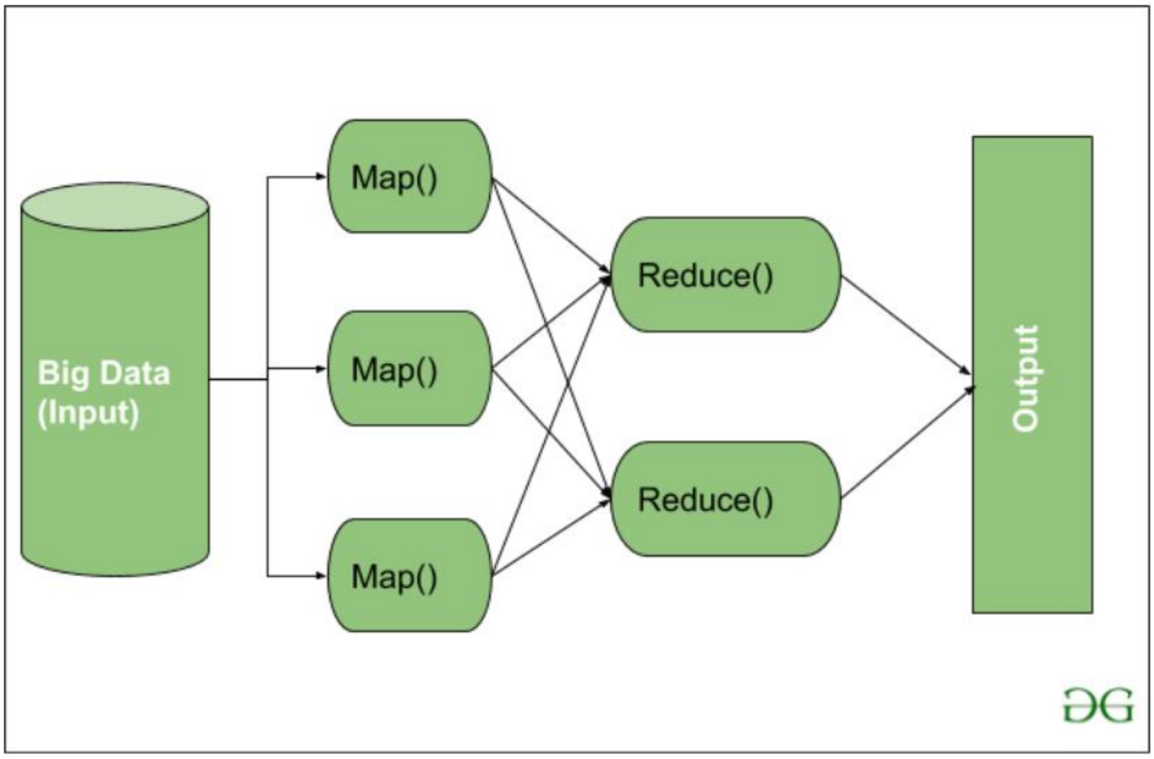
- MapReduce
- HDFS(Hadoop Distributed File System)
- YARN(Yet Another Resource Negotiator)
- Common Utilities or Hadoop Common


Distributed Processing
Map Reduce
Distributed storage
HDFS
YARN
Hadoop common
Yet Another Resource Negotiator(Job scheduling and Resource Manager)
Java Library and utilities(Java Scripts)

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

1. MapReduce

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



Here, we can see that the *Input* is provided to the Map() function then it's *output* is used as an input to the Reduce function and after that, we receive our final output. Let's understand What this Map() and Reduce() does.
As we can see that an Input is provided to the Map(), now as we are using Big Data. The Input is a set of Data. The Map() function here breaks this DataBlocks into **Tuples** that are nothing but a key-value pair. These key-value pairs are now sent as input to the Reduce(). The Reduce() function then combines this broken Tuples or key-value pair based on its Key value and form set of Tuples, and perform some operation like sorting, summation type job, etc. which is then sent to the final Output Node. Finally, the Output is Obtained.
The data processing is always done in Reducer depending upon the business requirement of that industry. This is How First Map() and then Reduce is utilized one by one.

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

- **RecordReader** The purpose of *recordreader* is to break the records. It is responsible for providing key-value pairs in a Map()

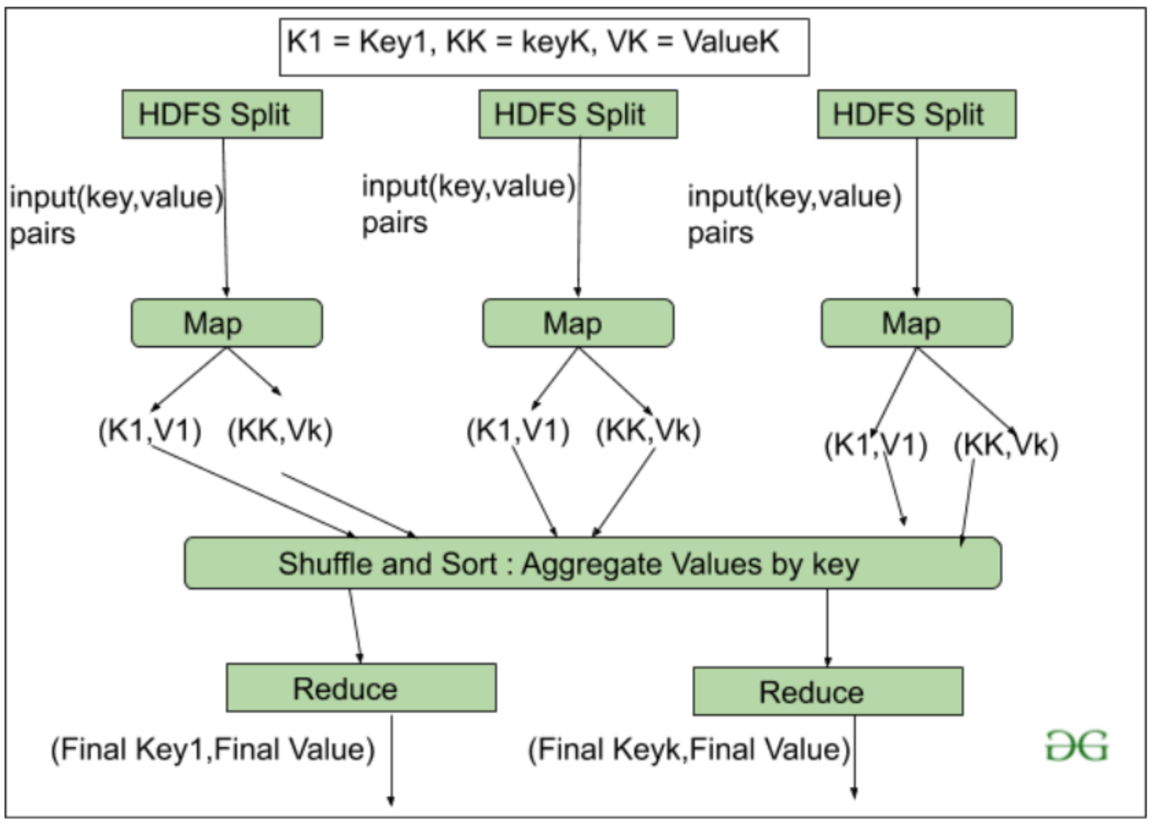
function. The key is actually is its locational information and value is the data associated with it.

- **Map:** A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.
- **Combiner:** Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The intermediate key-value that are generated in the Map is combined with the help of this combiner. Using a combiner is not necessary as it is optional.
- **Partitionar:** Partitional is responsible for fetching key-value pairs generated in the Mapper Phases. The partitioner generates the shards corresponding to each reducer. Hashcode of each key is also fetched by this partition. Then partitioner performs it's(Hashcode) modulus with the number of reducers(*key.hashcode()%(number of reducers)*).

Reduce Task

- **Shuffle and Sort:** The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as *Shuffling*. Using the Shuffling process the system can sort the data using its key value.
  Once some of the Mapping tasks are done Shuffling begins that is why it is a faster process and does not wait for the completion of the task performed by Mapper.

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



2. HDFS

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

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

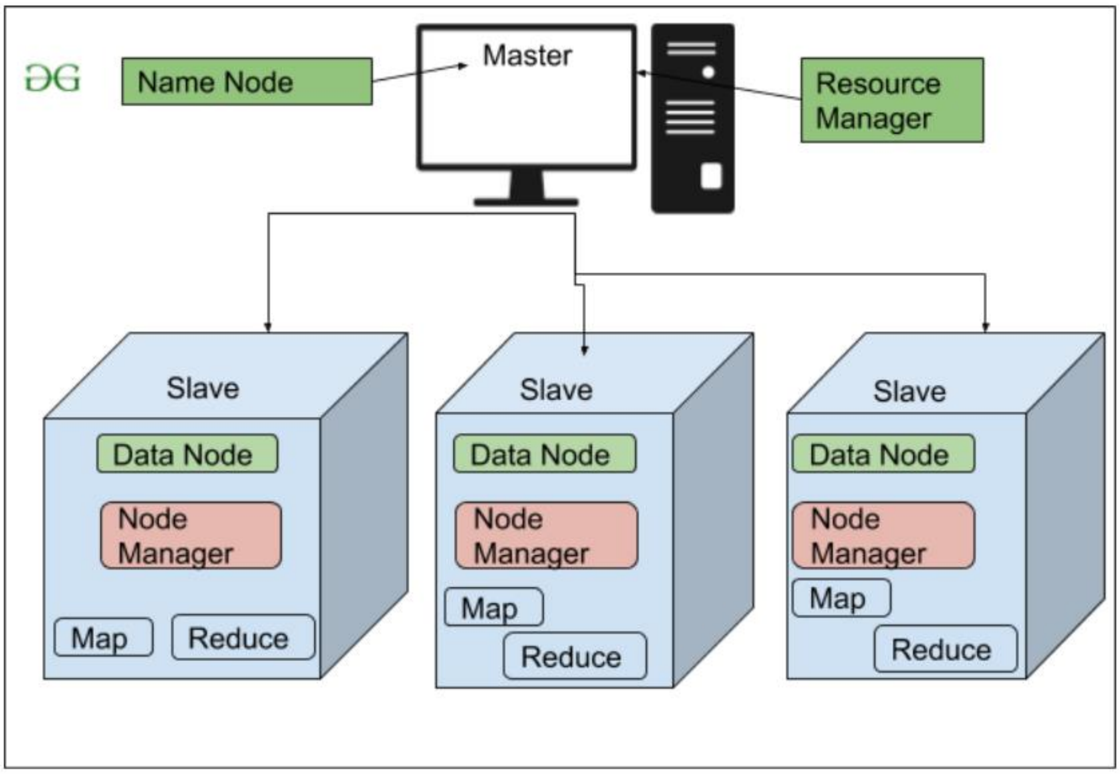
- NameNode(Master)
- DataNode(Slave)

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

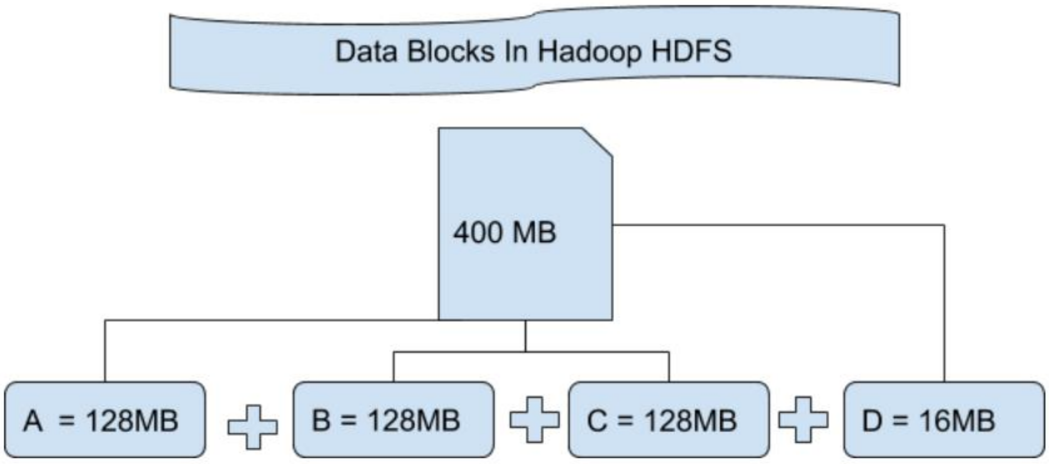
Meta Data can also be the name of the file, size, and the information about the location(Block number, Block ids) of Datanode that Namenode stores to find the closest DataNode for Faster Communication. Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.

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

High Level Architecture Of Hadoop



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



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

Replication In HDFS Replication ensures the availability of the data. Replication is making a copy of something and the number of times you make a copy of that particular thing can be expressed as it's Replication Factor.

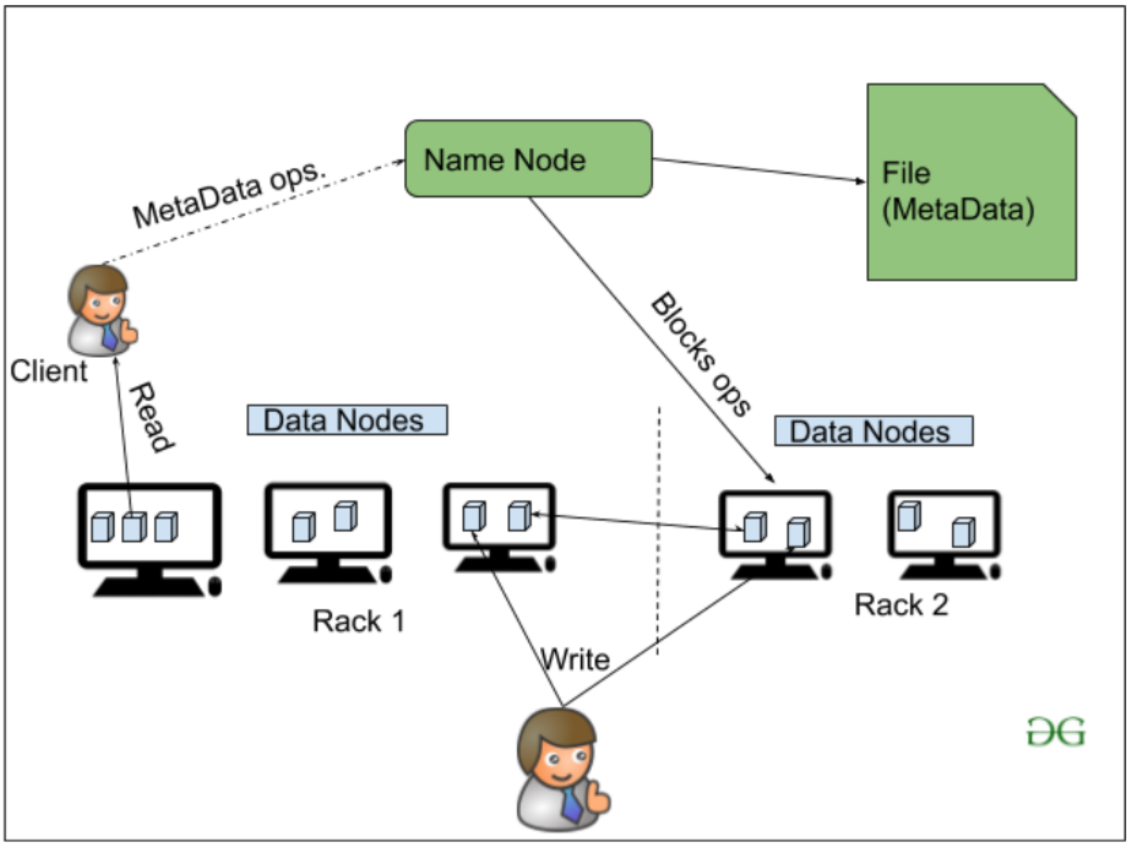
As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks.

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

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

Now one thing we also need to notice that after making so many replica's of our file blocks we are wasting so much of our storage but for the big brand organization the data is very much important than the storage so nobody cares for this extra storage. You can configure the Replication factor in your *hdfs-site.xml* file.
Rack Awareness The rack is nothing but just the physical collection of nodes in our Hadoop cluster (maybe 30 to 40). A large Hadoop cluster is consists of so many Racks . with the help of this Racks information Namenode chooses the closest Datanode to achieve the maximum performance while performing the read/write information which reduces the Network Traffic.

HDFS Architecture



3. YARN(Yet Another Resource Negotiator)

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

Features of YARN

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

4. Hadoop common or Common Utilities

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

RDMS (Relational Database Management System): RDBMS is an information management system, which is based on a data model.In RDBMS tables are used for information storage. Each row of the table represents a record and column represents an attribute of data. Organization of data and their manipulation processes are different in RDBMS from other databases. RDBMS ensures ACID (atomicity, consistency, integrity, durability) properties required for designing a database. The purpose of RDBMS is to store, manage, and retrieve data as quickly and reliably as possible.

Hadoop: It is an open-source software framework used for storing data and running applications on a group of commodity hardware. It has large storage capacity and high processing power. It can manage multiple concurrent processes at the same time. It is used in predictive analysis, data mining and machine learning. It can handle both structured and unstructured form of data. It is more flexible in storing, processing, and managing data than traditional RDBMS. Unlike traditional systems, Hadoop enables multiple analytical processes on the same data at the same time. It supports scalability very flexibly.
Below is a table of differences between RDBMS and Hadoop:

S.No.	RDBMS	Hadoop
1.	Traditional row-column based databases, basically used for data storage, manipulation and retrieval.	An open-source software used for storing data and running applications or processes concurrently.
2.	In this structured data is mostly processed.	In this both structured and unstructured data is processed.
3.	It is best suited for OLTP environment.	It is best suited for BIG data.
4.	It is less scalable than Hadoop.	It is highly scalable.
5.	Data normalization is required in RDBMS.	Data normalization is not required in Hadoop.
6.	It stores transformed and aggregated data.	It stores huge volume of data.
7.	It has no latency in response.	It has some latency in response.
8.	The data schema of RDBMS is static type.	The data schema of Hadoop is dynamic type.

S.No.	RDBMS	Hadoop
9.	High data integrity available.	Low data integrity available than RDBMS.
10.	Cost is applicable for licensed software.	Free of cost, as it is an open source software.

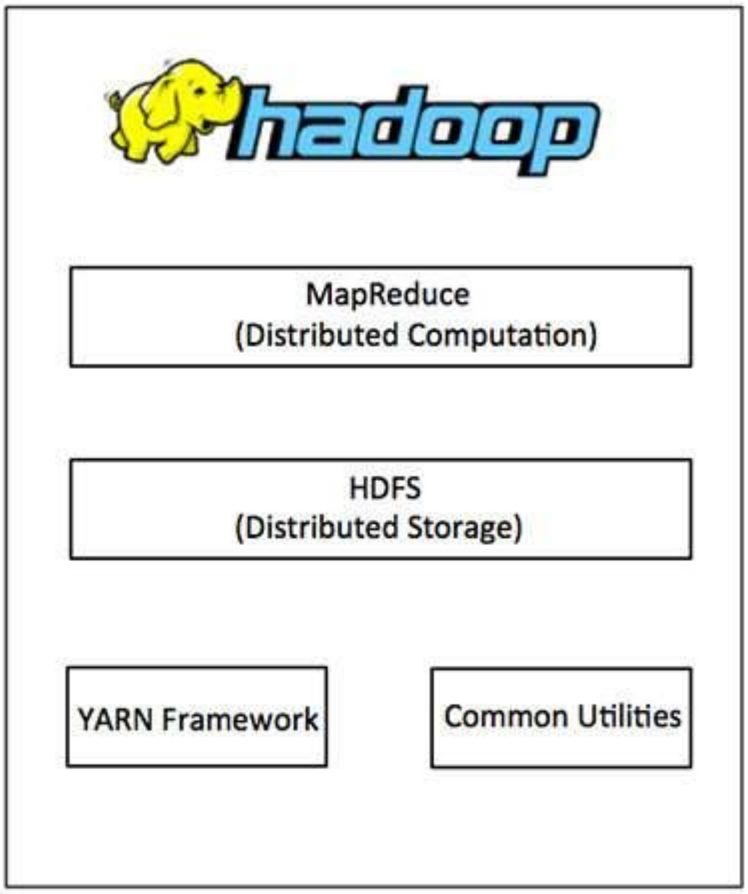
Hadoop Overview

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.

Hadoop Architecture

At its core, Hadoop has two major layers namely –

- Processing/Computation layer (MapReduce), and
- Storage layer (Hadoop Distributed File System).


hadoop
MapReduce (Distributed Computation)
HDFS (Distributed Storage)
YARN Framework
Common Utilities

MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules –

- **Hadoop Common** – These are Java libraries and utilities required by other Hadoop modules.
- **Hadoop YARN** – This is a framework for job scheduling and cluster resource management.

How Does Hadoop Work?

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers with single-CPU, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs –

- Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Advantages of Hadoop

- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Hadoop distributors

Top Commercial Hadoop Vendors

Here is a list of top Hadoop Vendors who will play a key role in big data market growth for the coming years-

1) Amazon Elastic MapReduce

2) Cloudera CDH Hadoop Distribution

3) Hortonworks Data Platform (HDP)

4) MapR Hadoop Distribution

5) IBM Open Platform

6) Microsoft Azure's HDInsight -Cloud based Hadoop Distrbution

7) Pivotal Big Data Suite

8) Datameer Professional

9) Datastax Enterprise Analytics

10) Dell- Cloudera Apache Hadoop Solution.

1) Amazon Web Services Elastic MapReduce Hadoop Distribution

Amazon EMR (previously known as Amazon Elastic MapReduce) is an Amazon Web Services (AWS) tool for big data processing and analysis. Amazon markets EMR as an expandable, low-configuration service that provides an alternative to running on-premises cluster computing.

Amazon EMR is based on Apache Hadoop, a Java-based programming framework that supports the processing of large data sets in a distributed computing environment. Using MapReduce, a core component of the Hadoop software framework, developers can write programs that process massive amounts of unstructured data across a distributed cluster of processors or standalone computers. It was developed by Google for indexing webpages and replaced its original indexing algorithms and heuristics in 2004.

Amazon EMR processes big data across a Hadoop cluster of virtual servers on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3). The *Elastic* in EMR's name refers to its dynamic resizing ability, which enables administrators to increase or reduce resources, depending on their current needs.

Amazon EMR is used for data analysis in log analysis, web indexing, data warehousing, machine learning (ML), financial analysis, scientific simulation and bioinformatics. It also supports workloads based on Apache Spark, Apache Hive, Presto and Apache HBase -- the latter of which integrates with Hive and Pig, which are open source data warehouse tools for Hadoop. Hive uses queries and analyzes data, and Pig offers a high-level mechanism for programming MapReduce jobs to be executed in Hadoop.

Amazon EMR use cases

There are several ways enterprises can use Amazon EMR, including:

- **Machine learning.** EMR's built-in ML tools use the Hadoop framework to create a variety of algorithms to support decision-making, including decision trees, random forests, support-vector machines and logistic regression.
- **Extract, transform and load.** ETL is the process of moving data from one or more data stores to another. Data transformations -- such as sorting, aggregating and joining -- can be done using EMR.
- **Clickstream analysis.** Clickstream data from Amazon S3 can be analysed with Apache Spark and Apache Hive. Apache Spark is an open source data processing tool that can help make data easy to manage and analyze. Spark uses a framework that enables jobs to run across large clusters of computers and can process data in

parallel. Apache Hive is a data warehouse infrastructure built on top of Hadoop that provides tools for working with data that Spark can analyze. Clickstream analysis can help organizations understand customer behaviors, find ways to improve a website layout, discover which keywords people are using in search engines and see which word combinations lead to sales.

- **Real-time streaming.** Users can analyze events using streaming data sources in real time with Apache Spark Streaming and Apache Flink. This enables streaming data pipelines to be created on EMR.
- **Interactive analytics.** EMR Notebooks are a managed service that provide a secure, scalable and reliable environment for data analytics. Using Jupyter Notebook -- an open source web application data scientists can use to create and share live code and equations -- data can be prepared and visualized to perform interactive analytics.
- **Genomics.** Organizations can use EMR to process genomic data to make data processing and analysis scalable for industries including medicine and telecommunications.

2) Hortonworks Hadoop Distribution

Hortonworks is a pure play Hadoop company that drives open source Hadoop distributions in the IT market.

The main goal of Hortonworks is to drive all its innovations through the Hadoop open data platform and build an ecosystem of partners that speeds up the process of Hadoop adoption amongst enterprises.

Apache Ambari is an example of Hadoop cluster management console developed by Hortonworks Hadoop vendor for provision, managing and monitoring Hadoop clusters.

The Hortonworks Hadoop vendor is reported to attract 60 new customers every quarter with some giant accounts like Samsung, Spotify, Bloomberg and eBay.

Hortonworks has grown its revenue at a rapid pace.

However, the professional services revenue generated by Hortonworks Hadoop vendor increases at a faster pace when compared to support and subscription services revenue.

3) Cloudera Hadoop Distribution

Hadoop is an Apache open-source framework that store and process Big Data in a distributed environment across the cluster using simple programming models. Hadoop provides parallel computation on top of distributed storage.

Since Apache Hadoop is open source, many companies have developed distributions that go beyond the original open source code. This is very akin to Linux distributions such as RedHat, Fedora, and Ubuntu. Each of the Linux distributions supports its own functionalities and features like user-friendly GUI in Ubuntu. Similarly, **Red Hat** is popular within enterprises because it offers support and also provides ideology to make changes to any part of the system at will. Red Hat relieves you from software compatibility problems. This is usually a big issue for users who are transitioning from Windows.

Likewise, there are 3 main types of Hadoop distributions which have its own set of functionalities and features and are built under the base HDFS.

Cloudera is the market trend in Hadoop space and is the first one to release commercial Hadoop distribution. It offers consulting services to bridge the gap between – “what does Apache Hadoop provides” and “what organizations need”.

Cloudera Distribution is:

- **Fast for business**: From analytics to data science and everything in between, Cloudera delivers the performance you need to unlock the potential of unlimited data.
- **Makes Hadoop easy to manage**: With Cloudera Manager, automated wizards let you quickly deploy your cluster, irrespective of the scale or deployment environment.
- **Secure without compromise:** Meets stringent data security and compliance needs without sacrificing business agility. Cloudera provides an integrated approach to data security and governance.

4) MapR Hadoop Distribution

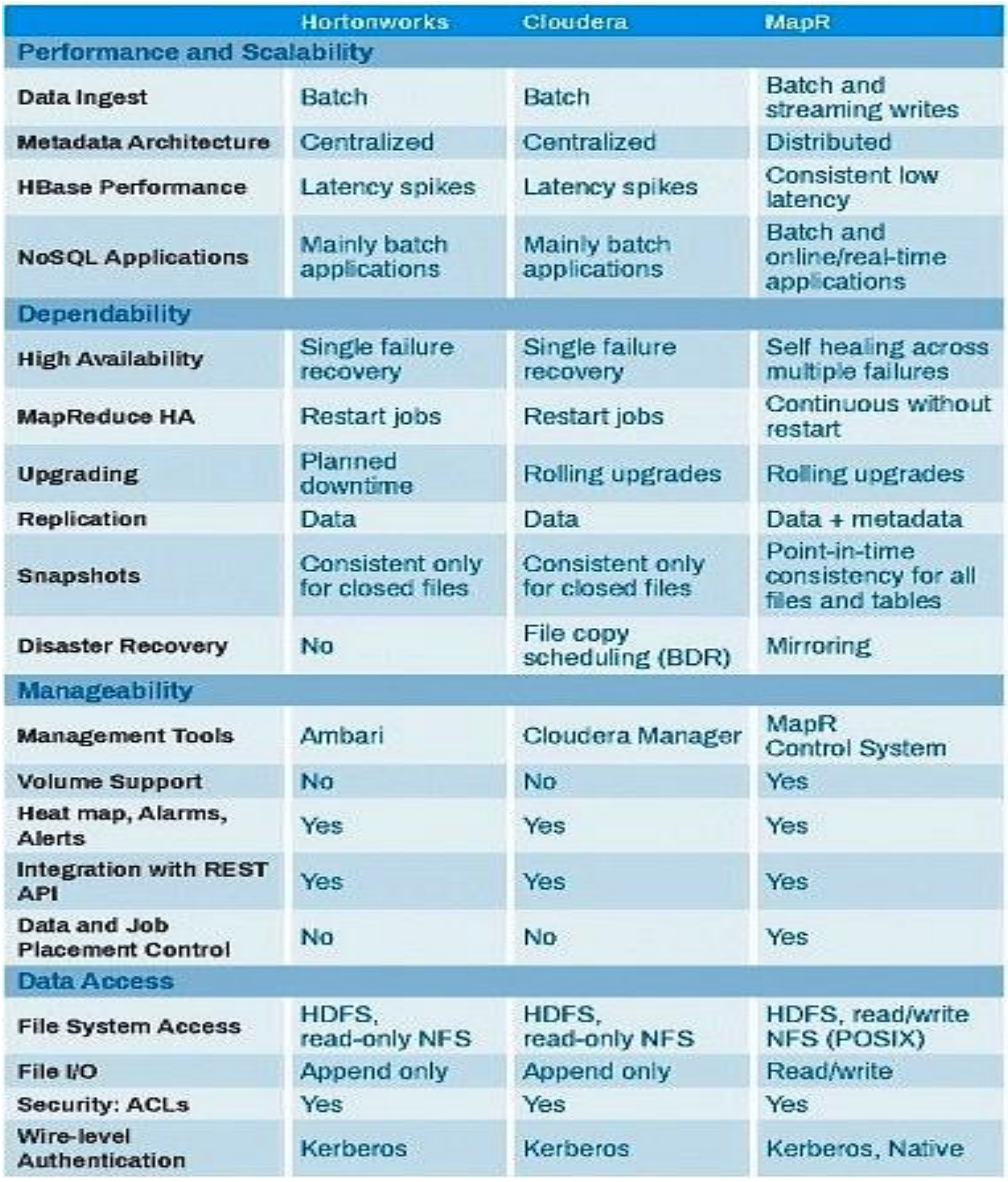
The MapR Converged Data Platform supports big data storage and processing through the Apache collection of Hadoop products, as well as its added-value components. These components from MapR Technologies provide several enterprise-grade proprietary tools to better manage and ensure the resiliency and reliability of data in the Hadoop cluster.

These platform components include MapR File System (MapR-FS); MapReduce; and MapR Control System, the product's user interface. The MapR Hadoop distribution includes a complete implementation of the Hadoop APIs, enabling the product to be fully compatible with the Hadoop ecosystem.

Unlike HDFS, which follows the write-once-read-many paradigm, MapR-FS is a fully read/write Portable Operating System Interface-compliant file system.

By supporting industry-standard NFS, users can easily mount a MapR cluster and execute any file-based application directly on the data residing in the cluster. This enables data from nearly any source to be processed and allows for standard tools to be used to directly access data in the cluster without any modifications.

Additionally, unlike other Hadoop distributions, MapR can process distributed files, database tables and event streams all in the same cluster of nodes. This lets organizations run operational tools such as Apache HBase and analytic tools such as Hive or Impala on one cluster, reducing hardware and operational costs.



	Hortonworks	Cloudera	MapR
Performance and Scalability			
Data Ingest	Batch	Batch	Batch and streaming writes
Metadata Architecture	Centralized	Centralized	Distributed
HBase Performance	Latency spikes	Latency spikes	Consistent low latency
NoSQL Applications	Mainly batch applications	Mainly batch applications	Batch and online/real-time applications
Dependability			
High Availability	Single failure recovery	Single failure recovery	Self healing across multiple failures
MapReduce HA	Restart jobs	Restart jobs	Continuous without restart
Upgrading	Planned downtime	Rolling upgrades	Rolling upgrades
Replication	Data	Data	Data + metadata
Snapshots	Consistent only for closed files	Consistent only for closed files	Point-in-time consistency for all files and tables
Disaster Recovery	No	File copy scheduling (BDR)	Mirroring
Manageability			
Management Tools	Ambari	Cloudera Manager	MapR Control System
Volume Support	No	No	Yes
Heat map, Alarms, Alerts	Yes	Yes	Yes
Integration with REST API	Yes	Yes	Yes
Data and Job Placement Control	No	No	Yes
Data Access			
File System Access	HDFS, read-only NFS	HDFS, read-only NFS	HDFS, read/write NFS (POSIX)
File I/O	Append only	Append only	Read/write
Security: ACLs	Yes	Yes	Yes
Wire-level Authentication	Kerberos	Kerberos	Kerberos, Native

5) IBM Infosphere BigInsights Hadoop Distribution

IBM is an analytics platform that enables analysis of a wide variety of unconventional information types and formats at large-scale volumes, without up-front preprocessing. IBM BigInsights V2.0:

- Delivers enterprise Hadoop capabilities with easy-to-use administration and management capabilities, rich developer tools, powerful analytic functions, and the latest versions of Apache Hadoop and associated projects.
- Provides extensive new capabilities by offering enhanced big data tools for visualization, monitoring, development, enhanced functionality for better enterprise integration with other IBM products, and additional platform support for Cloudera.

- Introduces two new Application Accelerators that may help you to a faster ROI: The Social Data Analytics Accelerator and the Machine Data Analytics Accelerator.
- Provides analytics and enterprise functionality on top of Apache Hadoop technology to meet big data enterprise requirements.
- Lets you run with the IBM-provided Apache Hadoop distribution or deploy to a Cloudera cluster.
- Includes data discovery to enable exploratory analysis and modeling of unconventional data types.

New capabilities in IBM InfoSphere BigInsights Enterprise Edition V2.0:

- **Enhanced big data tools: Visualization, monitoring, development**
    - With a consistent, unified, and extensible user interface, the big data tools bring big data collaboration to the enterprise and can help users unlock the value within data by enabling various roles of an organization to collaboratively leverage, discover, and explore large amounts of data. In this release, BigInsights adds the following new features for four major roles:
    - **Business analysts and business users**
        - A centralized dashboard to visualize analytic results, including new charts, BigSheets workbooks, data operations, and to visualize metrics from the monitoring service.
        - The ability to view BigSheets data flows between and across data sets to quickly navigate and relate analysis and charts.
        - BigSheets usability enhancements, including inner outer joins, enhanced filters for BigSheets columns, column data-type mapping for collections, application of analytics to BigSheets columns, data preparation enabling users to define and update schemas for data sources to improve error checking and analysis, and additional charting capabilities.
        - Workflow composition that enables users to compose new applications from existing applications and BigSheets, and to invoke analytics applications from the web console, including integration within BigSheets.
        - New Apps providing enhanced data import capability: a new REST data source App that enables users to load data from any data source supporting REST APIs into BigInsights , including popular social media services; a new Sampling App that enables users to sample data for analysis; and a new Subsetting App that enables users to subset data for data analysis.
    - **Data scientists**
        - A unified tooling environment that supports the data analytics lifecycle by enabling users to sample data and define, test, and deploy analytics applications from the BigInsights Eclipse tools, and to administer, execute, and monitor the deployed applications from the BigInsights Web Console.
        - Integration with R with an App that allows users to execute R scripts directly from the BigInsights Web Console.
        - Extended text analytics capability that performs global analysis to address key use cases such as customer profiling and lead generation.

    - **Administrators**
        - New monitoring capabilities that provide a centralized dashboard view to visualize key performance indicators including CPU, disk, and memory and network usage for the cluster, data services such as HDFS, HBase, Zookeeper, and Flume, and application services including MapReduce, Hive, and Oozie.
        - Enhanced status information and control over the major cluster capabilities, building on the existing server management capabilities.
        - Usability improvements for application permissions and deployment.
        - New capability to view and control all applications from a single page.
    - **Developers**
        - A workflow editor that greatly simplified the creation of complex Oozie workflows with a consumable interface.
        - A Pig editor with content assist and syntax highlighting that enables users to create and execute new applications using Pig in local or cluster mode from the Eclipse IDE.
        - Usability improvements to the Jaql Editor, such as extended support for Jaql syntax, extended content assist, and improved execution feedback.
        - Enablement of BigSheets macro and BigSheets reader development.
        - Enhanced Text Analytics development, including the support for modular rule sets, populating external artifacts, and providing workflow UI extensions for domain and language extensions.
        - Enhanced scope of the development artifacts during the deployment phase, including artifacts for Text Analytics, scripts for Jaql, Hive SQL, Pig, and Derby, and BigSheets macros and readers.
- **Enhanced enterprise integration**
    - **InfoSphere Data Explorer:** InfoSphere BigInsights includes a limited-use license to the included IBM InfoSphere Data Explorer program, which helps organizations discover, navigate, and visualize vast amounts of structured and unstructured information across many enterprise systems and data repositories. It also provides a cost-effective and efficient entry point for exploring the value of big data technologies through a powerful framework for developing applications that leverage existing enterprise data.
    - **InfoSphere Streams:** InfoSphere BigInsights includes a limited-use license to the included InfoSphere Streams program, which enables real-time, continuous analysis of data on the fly. Streams is an enterprise-class stream processing system that can be used to extract actionable insights from data as it arrives in the enterprise while transforming data and ingesting it into BigInsights at high speeds.

- **Netezza Data Warehouse Appliances:** New UDFs to connect SQL workloads from Netezza with the power of InfoSphere BigInsights . Now you can control the data read and written from or to HDFS via UDFs.
- **Rational® and Data Studio:** By installing the InfoSphere BigInsights Eclipse development tools into your existing Rational Data Studio (RAD), Rational Team Concert™ , or Data Studio environment, Infosphere BigInsights provides the functionality to allow for integration with IBM Rational Application Developer V8.5, Rational Team Concert and Data Studio, enabling more collaborative enterprise big data application development.
- **InfoSphere Guardium® :** InfoSphere BigInsights can be used with InfoSphere Guardium to audit the Web Console and HBase and various other open source logs for added protection and audit purposes.
- **IBM Cognos® Business Intelligence:** InfoSphere BigInsights includes a limited-use license to the included IBM Cognos Business Intelligence program, which enables business users to access and analyze the information they need to improve decision making, gain better insight, and manage performance. IBM Cognos Business Intelligence includes software for query, reporting, analysis, and dashboards, as well as software to gather and organize information from multiple sources.

6) Microsoft Azure's HDInsight Cloud based Hadoop Distribution

Forrester rates Microsoft Hadoop Distribution as 4/5- based on the Big Data Vendor’s current Hadoop Distributions, market presence and strategy - with Cloudera and Hortonworks scoring 5/5

Microsoft is an IT organization not known for embracing open source software solutions, but it has made efforts to run this open data platform software on Windows. Hadoop as a service offering by Microsoft’s big data solution is best leveraged through its public cloud product -Windows Azure’s HDInsight particularly developed to run on Azure. There is another production ready feature of Microsoft named Polybase that lets the users search for information available on SQL Server during the execution of Hadoop queries. Microsoft has great significance in delivering a growing Hadoop Stack to its customers.Microsoft Azure’s HDInsight is public-cloud only based product and customers can not run on their own hardware with this.

According to analyst Mike Gualtieri at Forrester: “Hadoop’s momentum is unstoppable as its open source roots grow wildly into enterprises. Its refreshingly unique approach to data management is transforming how companies store, process, analyze, and share big data”.

Commercial Hadoop Vendors continue to mature overtime with increased worldwide adoption of Big Data technologies and growing vendor revenue. There are several top Hadoop vendors namely Hortonworks, Cloudera, Microsoft and IBM. These Hadoop vendors are facing a tough competition in the open data platform. With the war heating up amongst big data vendors, nobody is sure as to who will top the list of commercial Hadoop vendors. With Hadoop buying cycle on the upswing, Hadoop vendors must capture the market share at a rapid pace to make the venture investors happy.

7) Pivotal Big Data Suite

Pivotal's Hadoop distribution, Pivotal HD is 100 percent Apache compliant, uses other Apache components, and is based on the Open Data Platform. Pivotal GemFire is a distributed data management platform designed for diverse data management situations, but is optimized for high volume, latency-sensitive, mission-critical, transactional systems.

The Pivotal GreenPlum Database is a shared-nothing, massively parallel processing (MPP) database used for business intelligence processing as well as for advanced analytics. Pivotal's HAWQ is an ANSI compliant SQL dialect that supports application portability and the use of data visualization tools such as SAS and Tableau.

8) Datameer Professional

Datameer Professional allows you to ingest, analyze, and visualize terabytes of structured and unstructured data from more than 60 different sources including social media, mobile data, web, machine data, marketing information, CRM data, demographics, and databases to name a few. Datameer also offers you 270 pre-built analytic functions to combine and analyze your unstructured and structured data after ingest.

Datameer focuses on big data analytics in a single application built on top of Hadoop. Datameer features a wizard-based data integration tool, iterative point-and-click analytics, drag-and-drop visualizations, and scales from a single workstation up to thousands of nodes. Datameer is available for all major Hadoop distributions.

9) Datastax Enterprise Analytics

DataStax delivers powerful integrated analytics to 20 of the Fortune 100 companies and well-known companies such as eBay and Netflix. DataStax is built on open source software technology for its primary services: Apache Hadoop (analytics0, Apache Cassandra (NoSQL distributed database), and Apache Solr (enterprise search).

DataStax made the choice to use Apache Cassandra, which provides an "always-on" capability for DataStax Enterprise (DSE) Analytics. DataStax OpsCenter also offers a web-based visual management system for DSE that allows cluster management, point-and-click provisioning and administration, secured administration, smart data protection, and visual monitoring and tuning.

10) Dell- Cloudera Apache Hadoop Solution.

Dell's Statistica Big Data Analytics is an integrated, configurable, cloud-enabled software platform that you can easily deploy in minutes. You can harvest sentiments from social media and the web and combine that data to better understand market traction and trends. Dell leverages Hadoop, Lucene/Solr search, and Mahout machine learning to bring you a highly scalable analytic solution running on Dell PowerEdge servers.

Dell summarizes its hardware software requirements for your Hadoop cluster simply as, 2 – 100 Linux servers for Hadoop Cluster, 6GB RAM, 2+ Core, 1TB HDD per server. The point is that entry into a Hadoop solution is simple and inexpensive. And as Dell puts it, "Gain robust big data analytics on an open and easily deployed platform."

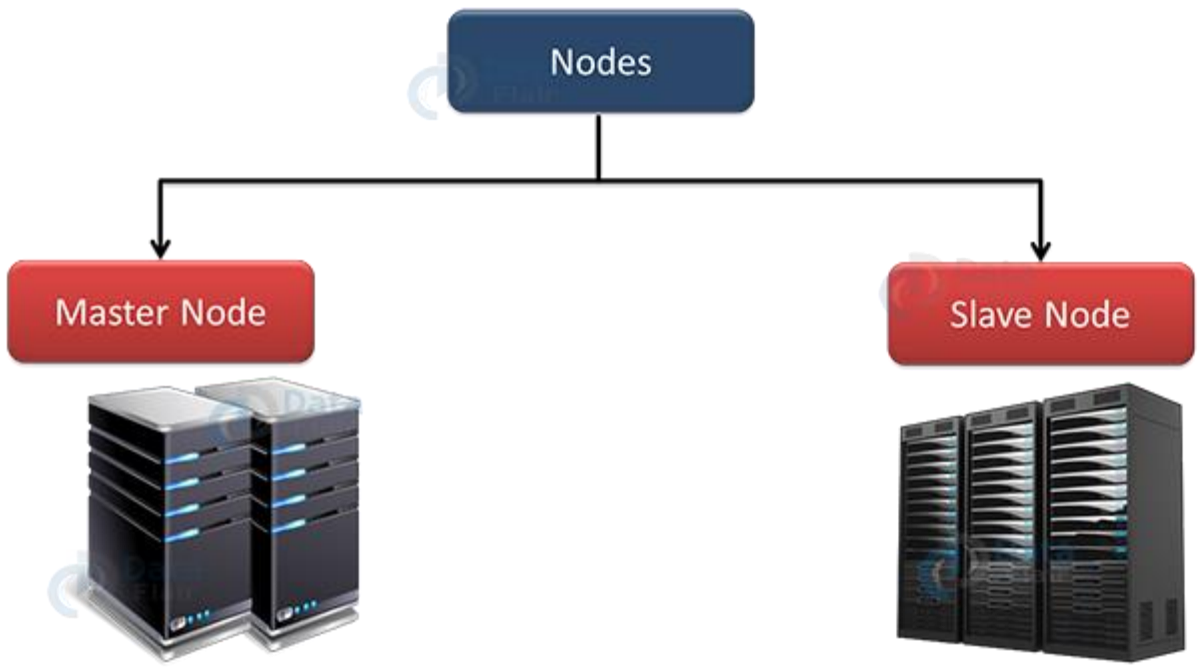
HDFS

Hadoop Distributed File system – HDFS is the world's most reliable storage system. HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. It is designed on the principle of storage of less number of large files rather than the huge number of small files.

Hadoop HDFS provides a fault-tolerant storage layer for Hadoop and its other components. HDFS Replication of data helps us to attain this feature. It stores data reliably, even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel.

HDFS Nodes

As we know, Hadoop works in **master-slave** fashion, HDFS also has two types of nodes that work in the same manner. These are the **NameNode(s)** and the **DataNodes**.



1. HDFS Master (Namenode)

NameNode regulates file access to the clients. It maintains and manages the slave nodes and assigns tasks to them. NameNode executes file system namespace operations like opening, closing, and renaming files and directories.

NameNode runs on the high configuration hardware.

2. HDFS Slave (Datanode)

There are n number of slaves (where n can be up to 1000) or DataNodes in the Hadoop Distributed File System that manages storage of data. These slave nodes are the actual worker nodes that do the tasks and serve read and write requests from the file system's clients.

They perform block creation, deletion, and replication upon instruction from the NameNode. Once a block is written on a DataNode, it replicates it to other DataNode, and the process continues until creating the required number of replicas.

DataNodes runs on commodity hardware having an average configuration.

4. Hadoop HDFS Daemons

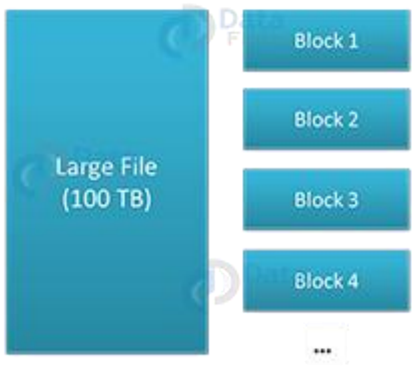
There are two daemons which run on HDFS for data storage:

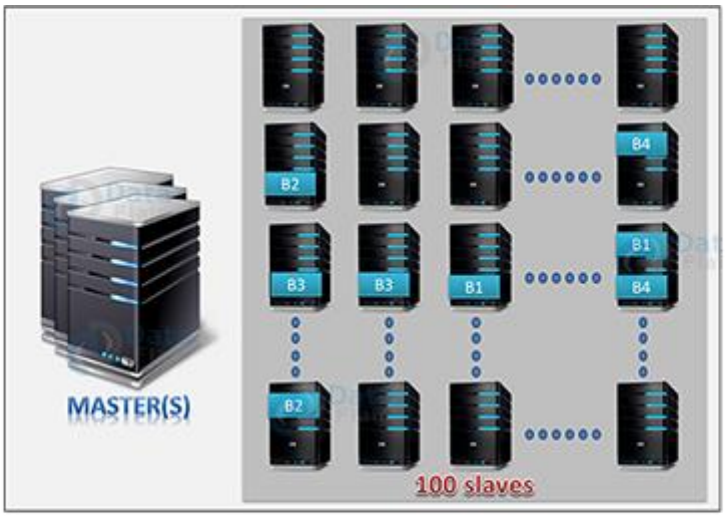
- **Namenode:** This is the daemon that runs on all the masters. NameNode stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs, etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of the metadata is available for persistence. So NameNode memory should be high as per the requirement.

- **Datanode:** This is the daemon that runs on the slave. These are actual worker nodes that store the data.

Data storage in HDFS

Hadoop HDFS broke the files into small pieces of data known as blocks. The default block size in HDFS is 128 MB. We can configure the size of the block as per the requirements. These blocks are stored in the cluster in a distributed manner on different nodes. This provides a mechanism for MapReduce to process the data in parallel in the cluster.





HDFS stores multiple copies of each block across the cluster on different nodes. This is a replication of data. By default, the HDFS replication factor is
3. [Hadoop HDFS provides high availability](), fault tolerance, and reliability. HDFS splits a large file into n number of small blocks and stores them on different DataNodes in the cluster in a distributed manner. It replicates each block and stored them across different DataNodes in the cluster.

Rack Awareness in Hadoop HDFS

Hadoop runs on a cluster of computers spread commonly across many racks.

Name Node places replicas of a block on multiple racks for improved fault tolerance.

Name Node tries to place at least one replica of a block in a different rack so that if a complete rack goes down, then also the system will be highly available.

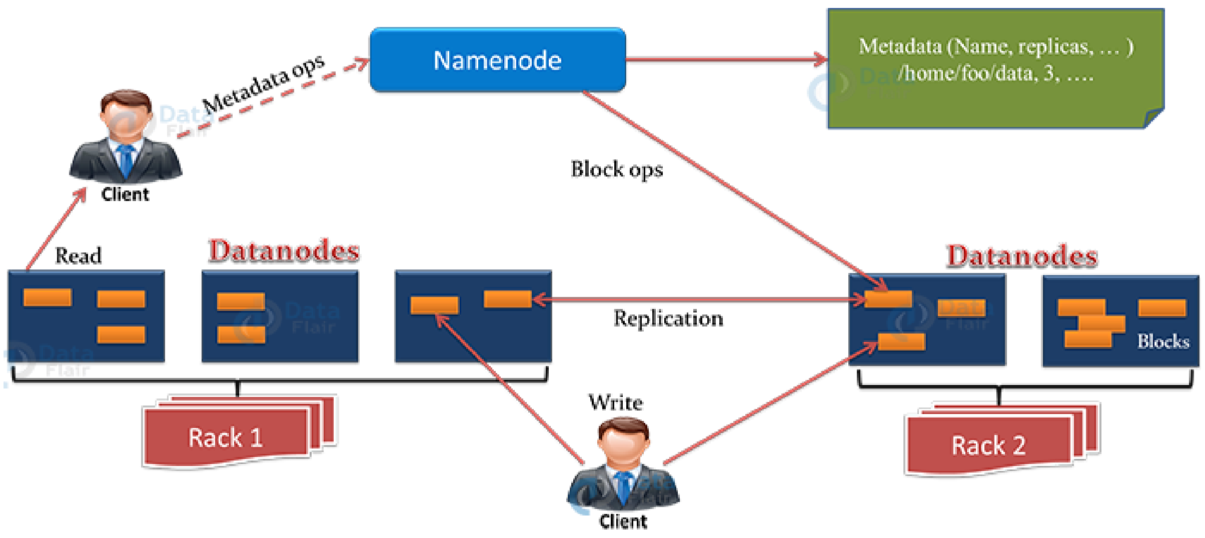
Optimize replica placement distinguishes HDFS from other distributed file systems. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization.

For more understanding, learn Rack Awareness in detail.

HDFS Architecture

This architecture gives you a complete picture of the Hadoop Distributed File System. There is a single Name Node that stores metadata, and there are multiple Data Nodes that do actual storage work. Nodes are arranged in racks, and replicas of data blocks are stored on different racks in the cluster to provide fault tolerance.

In the remaining section of this tutorial, we will see how read and write operations are performed in HDFS? To read or write a file in HDFS, the client needs to interact with Name Node. HDFS applications need a write-once-read-many access model for files. A file, once created and written, cannot be edited.



Name Node stores metadata, and Data Node stores actual data. The client interacts with Name Node for performing any tasks, as Name Node is the center piece in the cluster.

There are several Data Nodes in the cluster which store HDFS data in the local disk. Data Node sends a heartbeat message to Name Node periodically to indicate that it is alive. Also, it replicates data to other Data Node as per the replication factor.

Hadoop HDFS Features

a. Distributed Storage

HDFS stores data in a distributed manner. It divides the data into small pieces and stores it on different DataNodes in the cluster. In this manner, the Hadoop Distributed File System provides a way to MapReduce to process a subset of large data sets broken into blocks, parallelly on several nodes. MapReduce is the heart of Hadoop, but HDFS is the one who provides it all these capabilities.

b. Blocks

HDFS splits huge files into small chunks known as blocks. Block is the smallest unit of data in a filesystem. We (client and admin) do not have any control on the block like block location. NameNode decides all such things.

HDFS default block size is 128 MB. We can increase or decrease the block size as per our need. This is unlike the OS filesystem, where the block size is 4 KB.

If the data size is less than the block size of HDFS, then block size will be equal to the data size.

For example, if the file size is 129 MB, then 2 blocks will be created for it. One block will be of default size 128 MB, and the other will be 1 MB only and not 128 MB as it will waste the space (here block size is equal to data size). Hadoop is intelligent enough not to waste the rest of 127 MB. So it is allocating 1 MB block only for 1 MB data.

The major advantage of storing data in such block size is that it saves disk seek time and another advantage is in the case of processing as **mapper** processes 1 block at a time. So 1 mapper processes large data at a time.

c. Replication

Hadoop HDFS creates duplicate copies of each block. This is known as replication. All blocks are replicated and stored on different DataNodes across the cluster. It tries to put at least 1 replica in a different rack.

HDFS Daemons:

Daemons mean **Process**. Hadoop Daemons are a set of processes that run on Hadoop. Hadoop is a framework written in Java, so all these processes are Java Processes.
Apache Hadoop 2 consists of the following Daemons:

- NameNode
- DataNode
- Secondary Name Node
- Resource Manager
- Node Manager

Namenode, Secondary NameNode, and Resource Manager work on a Master System while the Node Manager and DataNode work on the Slave machine.

1. NameNode

NameNode works on the Master System. The primary purpose of Namenode is to manage all the MetaData. Metadata is the list of files stored in HDFS(Hadoop Distributed File System). As we know the data is stored in the form of blocks in a Hadoop cluster. So the DataNode on which or the location at which that block of the file is stored is mentioned in MetaData. All information regarding the logs of the transactions happening in a Hadoop cluster (when or who read/wrote the data) will be stored in MetaData. MetaData is stored in the memory.

Features:

- It never stores the data that is present in the file.
- As Namenode works on the Master System, the Master system should have good processing power and more RAM than Slaves.
- It stores the information of DataNode such as their Block id's and Number of Blocks

How to start Name Node?

```
hadoop-daemon.sh start namenode
```

How to stop Name Node?

```
hadoop-daemon.sh stop namenode
```

The **namenode** daemon is a master daemon and is responsible for storing all the location information of the files present in HDFS. The actual data is never stored on a namenode. In other words, it holds the metadata of the files in HDFS.

The name node maintains the entire metadata in RAM, which helps clients receive quick responses to read requests. Therefore, it is important to run name node from a machine that has lots of RAM at its disposal. The higher the number of files in HDFS, the higher the consumption of RAM. The name node daemon also maintains a persistent checkpoint of the metadata in a file stored on the disk called the `fsimage` file.

Whenever a file is placed/deleted/updated in the cluster, an entry of this action is updated in a file called the `edits` logfile. After updating the `edits` log, the metadata present in-memory is also updated accordingly. It is important to note that the `fsimage` file is not updated for every write operation.

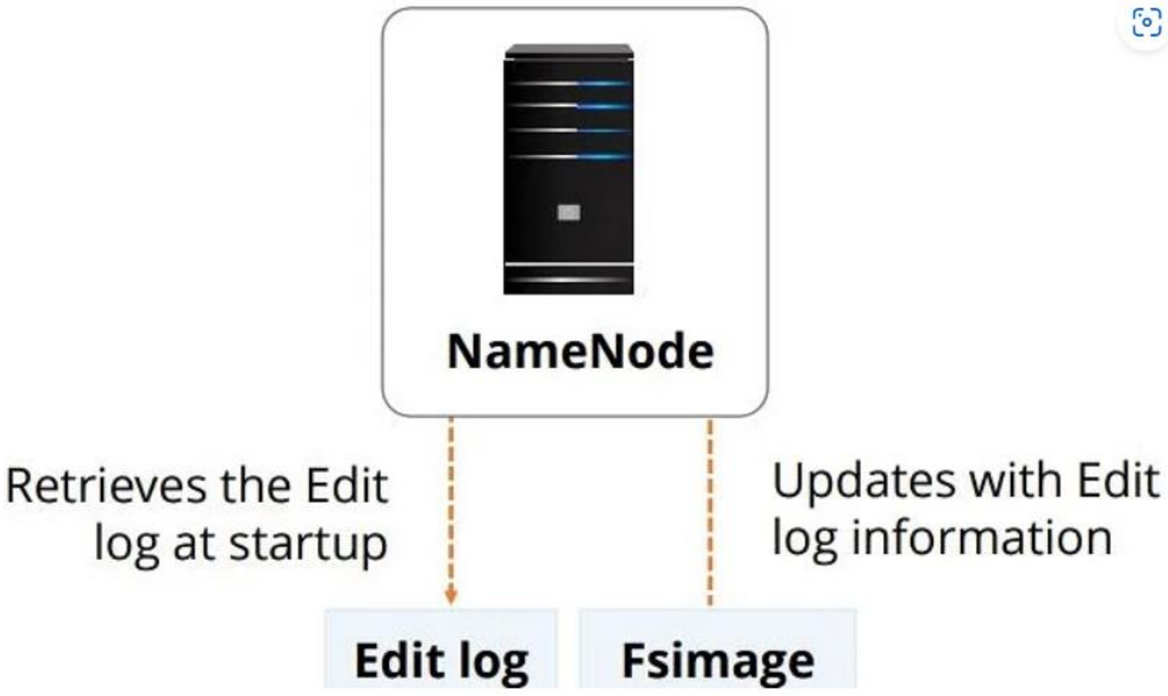
In case the name node daemon is restarted, the following sequence of events occur at name node boot up:

1. Read the `fsimage` file from the disk and load it into memory (RAM).
2. Read the actions that are present in the `edits` log and apply each action to the in-memory representation of the `fsimage` file.
3. Write the modified in-memory representation to the `fsimage` file on the disk.

The preceding steps make sure that the in-memory representation is up to date.

The namenode daemon is a single point of failure in Hadoop 1.x, which means that if the node hosting the namenode daemon fails, the filesystem becomes unusable. To handle this, the administrator has to configure the namenode to write the fsimage file to the local disk as well as a remote disk on the network. This backup on the remote disk can be used to restore the namenode on a freshly installed server. Newer versions of Apache Hadoop (2.x) now support **High Availability** (**HA**), which deploys two namenodes in an active/passive configuration, wherein if the active namenode fails, the control falls onto the passive namenode, making it active. This configuration reduces the downtime in case of a namenode failure.

Since the fsimage file is not updated for every operation, it is possible the edits logfile would grow to a very large file. The restart of namenode service would become very slow because all the actions in the large edits logfile will have to be applied on the `fsimage` file. The slow boot up time could be avoided using the secondary namenode daemon.


NameNode
Retrieves the Edit log at startup
Updates with Edit log information
Edit log
Fsimage

The namespace image and the edit log stores information of the data and the metadata. NameNode also determines the linking of blocks to DataNodes. Furthermore, the NameNode is a single point of failure. The DataNode is a multiple instance server. There can be several numbers of DataNode servers. The number depends on the type of network and the storage system.

The DataNode servers, stores, and maintains the data blocks. The NameNode Server provisions the data blocks on the basis of the type of job submitted by the client.

DataNode also stores and retrieves the blocks when asked by clients or the NameNode. Furthermore, it reads/writes requests and performs block creation, deletion, and replication of instruction from the NameNode. There can be only one Secondary NameNode server in a cluster. Note that you cannot treat the Secondary NameNode server as a disaster recovery server. However, it partially restores the NameNode server in case of a failure.

2. DataNode

DataNode works on the Slave system. The NameNode always instructs DataNode for storing the Data. DataNode is a program that runs on the slave system that serves the read/write request from the client. As the data is stored in this DataNode, they should possess high memory to store more Data.

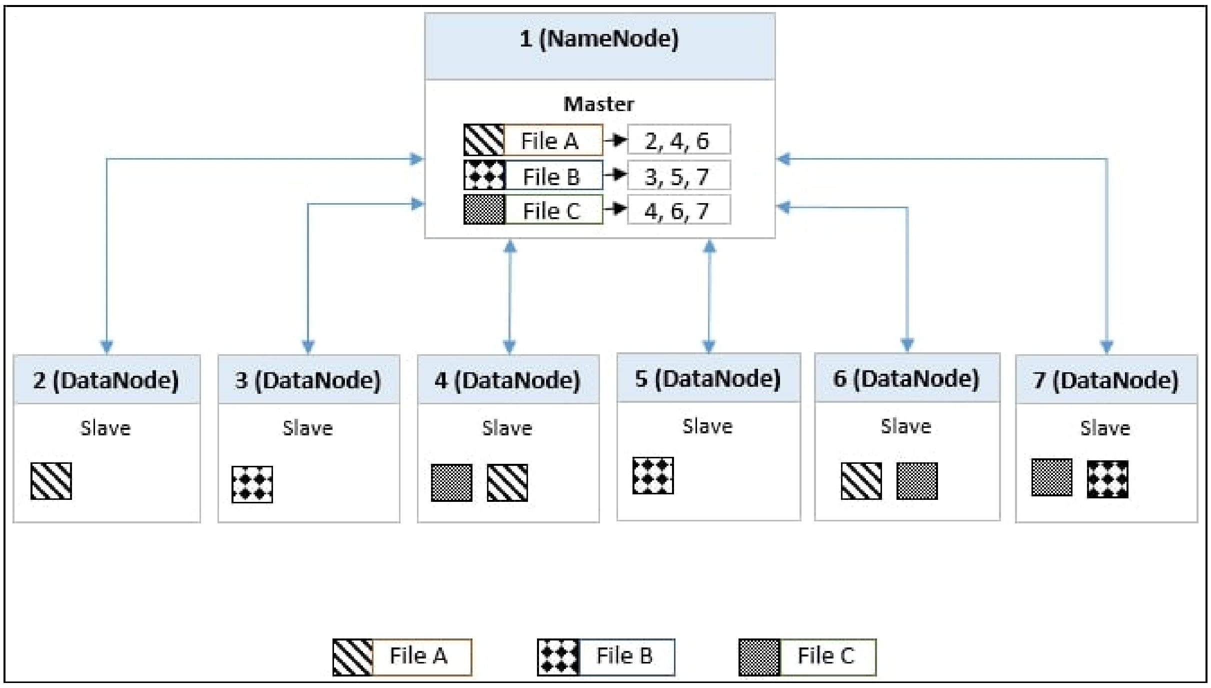
How to start Data Node?

```
hadoop-daemon.sh start datanode
```

How to stop Data Node?

```
hadoop-daemon.sh stop datanode
```

The **datanode** daemon acts as a slave node and is responsible for storing the actual files in HDFS. The files are split as data blocks across the cluster. The blocks are typically 64 MB to 128 MB size blocks. The block size is a configurable parameter. The file blocks in a Hadoop cluster also replicate themselves to other datanodes for redundancy so that no data is lost in case a datanode daemon fails. The datanode daemon sends information to the namenode daemon about the files and blocks stored in that node and responds to the namenode daemon for all filesystem operations. The following diagram shows how files are stored in the cluster:



File blocks of files A, B, and C are replicated across multiple nodes of the cluster for redundancy. This ensures availability of data even if one of the nodes fail.

You can also see that blocks of file A are present on nodes 2, 4, and 6; blocks of file B are present on nodes 3, 5, and 7; and blocks of file C are present on 4, 6, and 7. The replication factor configured for this cluster is 3, which signifies that each file block is replicated three times across the cluster. It is the responsibility of the namenode daemon to maintain a list of the files and their corresponding locations on the cluster. Whenever a client needs to access a file, the namenode daemon provides the location of the file to client and the client, then accesses the file directly from the data node daemon.

3. secondary Name Node is used for taking the hourly backup of the data.

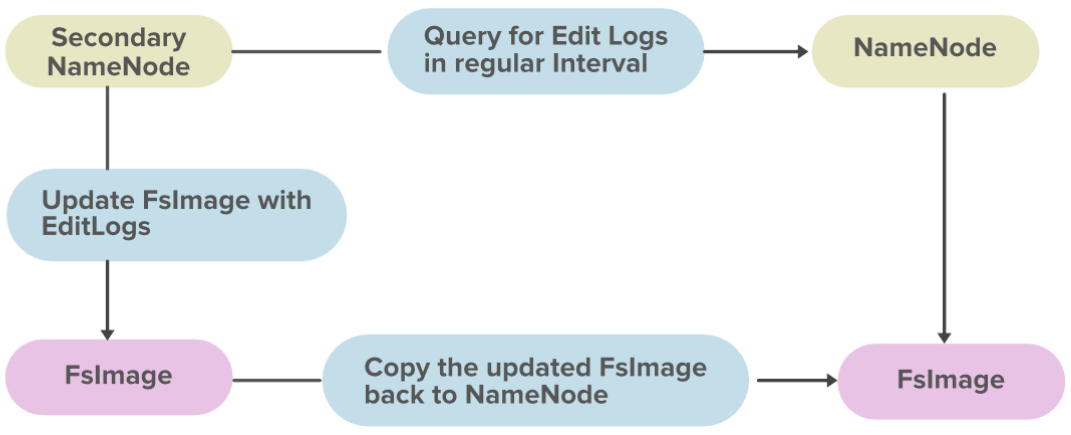
In case the Hadoop cluster fails, or crashes, the secondary Namenode will take the hourly backup or checkpoints of that data and store this data into a file name *fsimage*. This file then gets transferred to a new system.

A new Meta Data is assigned to that new system and a new Master is created with this Meta Data, and the cluster is made to run again correctly. This is the benefit of Secondary Name Node.

Now in Hadoop2, we have High-Availability and Federation features that minimize the importance of this Secondary Name Node in Hadoop2.
Major Function Of Secondary NameNode:

- It groups the Edit logs and Fsimage from NameNode together.
- It continuously reads the MetaData from the RAM of NameNode and writes into the Hard Disk.

As secondary NameNode keeps track of checkpoints in a Hadoop Distributed File System, it is also known as the checkpoint Node.


Secondary NameNode
Query for Edit Logs in regular Interval
NameNode
Update FsImage with EditLogs
FsImage
Copy the updated FsImage back to NameNode
FsImage

The Hadoop Daemon's	**Port**
Name Node	50070
Data Node	50075
Secondary Name Node	50090

These ports can be configured manually in *hdfs-site.xml* and *mapred-site.xml* files.

4. Resource Manager

Resource Manager is also known as the Global Master Daemon that works on the Master System.

The Resource Manager Manages the resources for the applications that are running in a Hadoop Cluster.

The Resource Manager Mainly consists of 2 things.

1. Applications Manager
2. Scheduler

An Application Manager is responsible for accepting the request for a client and also makes a memory resource on the Slaves in a Hadoop cluster to host the *Application Master*.
The scheduler is utilized for providing resources for applications in a Hadoop cluster and for monitoring this application.

How to start ResourceManager?

```
yarn-daemon.sh start resourcemanager
```

How to stop ResourceManager?

```
stop:yarn-daemon.sh stop resoucemnager
```

5. Node Manager

The Node Manager works on the Slaves System that manages the memory resource within the Node and Memory Disk. Each Slave Node in a Hadoop cluster has a single NodeManager Daemon running in it. It also sends this monitoring information to the Resource Manager.

How to start Node Manager?

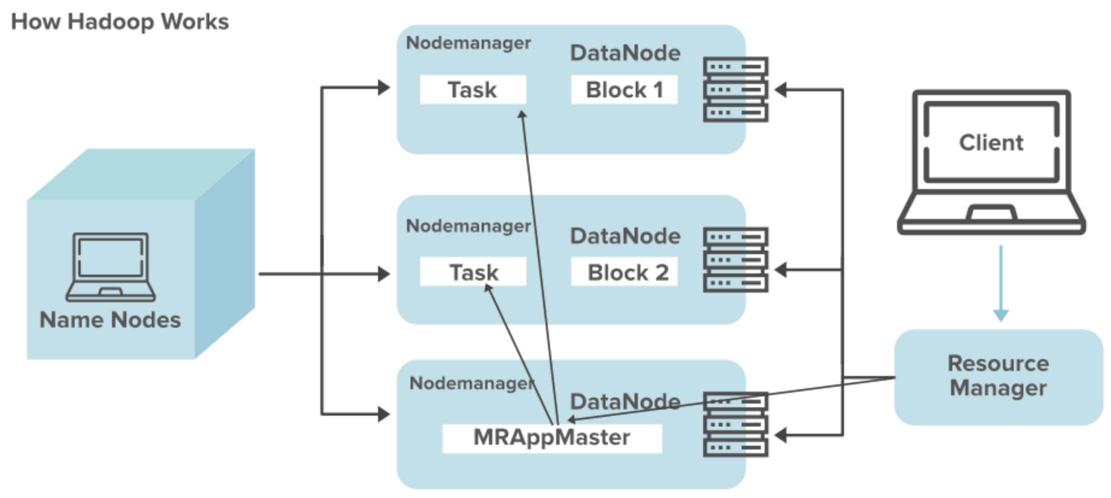
```
yarn-daemon.sh start node manager
```

How to stop Node Manager?

```
yarn-daemon.sh stop nodemanager
```

The Hadoop Daemon's	Port
ResourceManager	**8088**
NodeManager	**8042**

The below diagram shows how Hadoop works.



Anatomy of File Write and Read

Big data is nothing but a collection of data sets that are large, complex, and which are difficult to store and process using available data management tools or traditional data processing applications. Hadoop is a framework (open source) for writing, running, storing, and processing large datasets in a parallel and distributed manner.
It is a solution that is used to overcome the challenges faced by big data.
Hadoop has two components:

- HDFS (Hadoop Distributed File System)
- YARN (Yet Another Resource Negotiator)

We focus on one of the components of Hadoop i.e., HDFS and the anatomy of file reading and file writing in HDFS. HDFS is a file system designed for storing very large files (files that are hundreds of megabytes, gigabytes, or terabytes in size) with streaming data access, running on clusters of commodity hardware(commonly available hardware that can be obtained from various vendors). In simple terms, the storage unit of Hadoop is called HDFS.

Some of the characteristics of HDFS are:

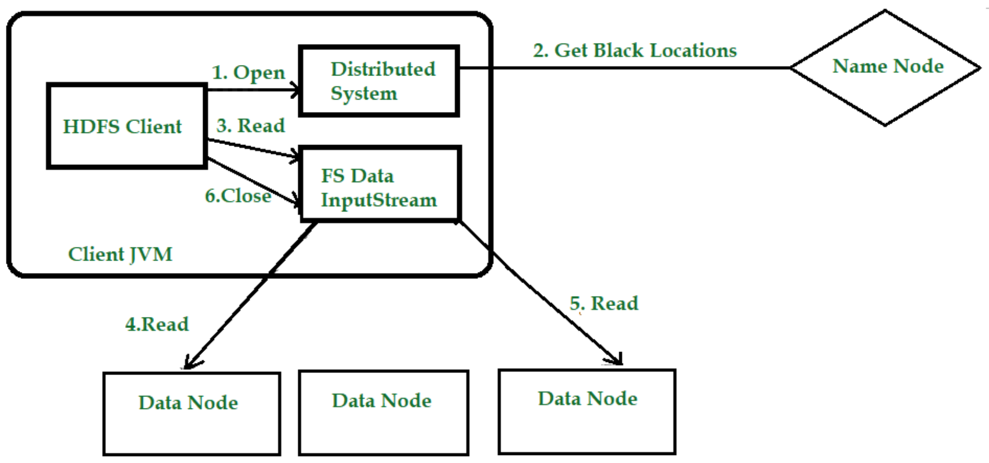
- Fault-Tolerance
- Scalability
- Distributed Storage
- Reliability
- High availability
- Cost-effective
- High throughput

Building Blocks of Hadoop:

Name Node
Data Node
Secondary Name Node (SNN)
Job Tracker
Task Tracker

Anatomy of File Read in HDFS

Let’s get an idea of how data flows between the client interacting with HDFS, the name node, and the data nodes with the help of a diagram. Consider the figure:



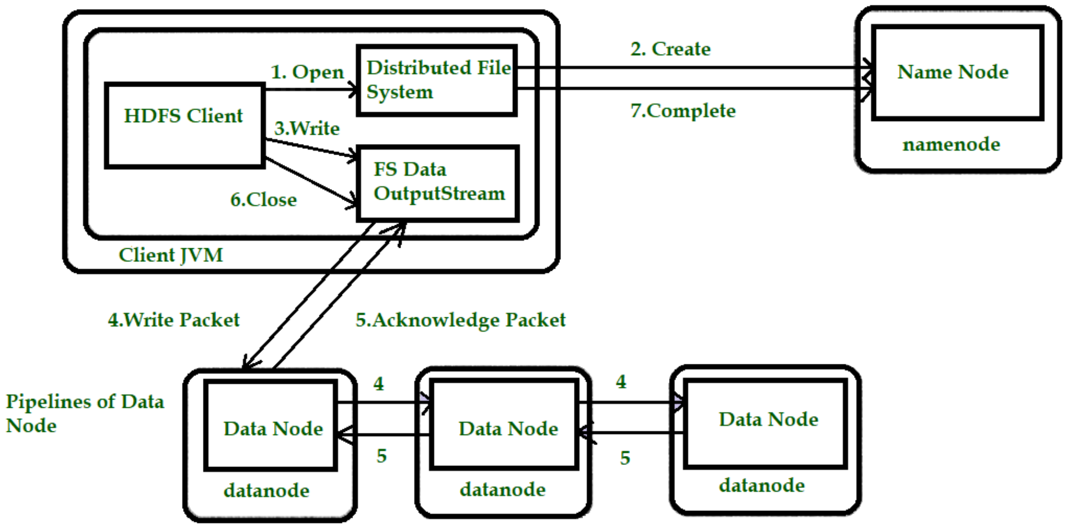
Step 1: The client opens the file it wishes to read by calling open() on the File System Object(which for HDFS is an instance of Distributed File System).
Step 2: Distributed File System (DFS) calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks 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.

Anatomy of File Write in HDFS

Next, we'll check out how files are written to HDFS. Consider figure 1.2 to get a better understanding of the concept.

Note: HDFS follows the Write once Read many times model. In HDFS we cannot edit the files which are already stored in HDFS, but we can append data by reopening the files.



Step 1: The client creates the file by calling create() on DistributedFileSystem(DFS).
Step 2: DFS makes an RPC call to the name node to create a new file in the file system's namespace, with no blocks associated with it. The name node 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 name node prepares a record of the new file; otherwise, the file can't be created and therefore the client is thrown an error i.e. IOException. The DFS returns an FSDataOutputStream for the client to start out writing data to.
Step 3: Because the client writes data, the DFSOutputStream splits it into packets, which it writes to an indoor queue called the info queue. The data queue is consumed by the DataStreamer, which is liable for asking the name

node to allocate new blocks by picking an inventory of suitable data nodes to store the replicas. The list of data nodes forms a pipeline, and here we'll assume the replication level is three, so there are three nodes in the pipeline. The DataStreamer streams the packets to the primary data node within the pipeline, which stores each packet and forwards it to the second data node within the pipeline.
Step 4: Similarly, the second data node stores the packet and forwards it to the third (and last) data node in the pipeline.
Step 5: The DFSOutputStream sustains an internal queue of packets that are waiting to be acknowledged by data nodes, called an "ack queue".
Step 6: This action sends up all the remaining packets to the data node pipeline and waits for acknowledgments before connecting to the name node to signal whether the file is complete or not.
HDFS follows Write Once Read Many models. So, we can't edit files that are already stored in HDFS, but we can include them by again reopening the file. This design allows HDFS to scale to a large number of concurrent clients because the data traffic is spread across all the data nodes in the cluster. Thus, it increases the availability, scalability, and throughput of the system.

Name Node

NameNode works as Master in Hadoop cluster. Below listed are the main function performed by NameNode:
1. Stores metadata of actual data. E.g. Filename, Path, No. of Data Blocks, Block IDs, Block Location, No. of Replicas, Slave related configuration
2. Manages File system namespace.
3. Regulates client access request for actual file data file.
4. Assign work to Slaves(DataNode).
5. Executes file system name space operation like opening/closing files, renaming files and directories.
6. As Name node keep metadata in memory for fast retrieval, the huge amount of memory is required for its operation. This should be hosted on reliable hardware.

1. NameNode is the centerpiece of HDFS.
2. NameNode is also known as the Master
NameNode only stores the metadata of HDFS – the directory tree of all files in the file system, and tracks the files across the cluster.
3. NameNode does not store the actual data or the dataset. The data itself is actually stored in the DataNodes.

4. NameNode knows the list of the Blocks and its location for any given file in HDFS. With this information NameNode knows how to construct the file from blocks.
5. NameNode is so critical to HDFS and when the NameNode is down, HDFS/Hadoop cluster is inaccessible and considered down.
6. NameNode is a single point of failure in Hadoop cluster.
7. NameNode is usually configured with a lot of memory (RAM). Because the block locations are held in main memory

HDFS NameNode
1. NameNode is the main central component of HDFS architecture framework.
2. NameNode is also known as Master node.
3. HDFS Namenode stores meta-data i.e. number of data blocks, file name, path, Block IDs, Block location, no. of replicas, and also Slave related configuration. This meta-data is available in memory in the master for faster retrieval of data.
4. NameNode keeps metadata related to the file system namespace in memory, for quicker response time. Hence, more memory is needed. So NameNode configuration should be deployed on reliable configuration.
5. NameNode maintains and manages the slave nodes, and assigns tasks to them.
6. NameNode has knowledge of all the DataNodes containing data blocks for a given file.
7. NameNode coordinates with hundreds or thousands of data nodes and serves the requests coming from client applications.

Two files ‘FSImage’ and the ‘EditLog’ are used to store metadata information.

FsImage: It is the snapshot the file system when Name Node is started. It is an “Image file”. FsImage contains the entire filesystem namespace and stored as a file in the NameNode’s local file system. It also contains a serialized form of all the directories and file inodes in the filesystem. Each inode is an internal representation of file or directory’s metadata.

EditLogs: It contains all the recent modifications made to the file system on the most recent FsImage. NameNode receives a create/update/delete request from the client. After that this request is first recorded to edits file.

Functions of NameNode in HDFS

1. It is the master daemon that maintains and manages the DataNodes (slave nodes).
2. It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc.
3. It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
4. It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
5. It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
6. The NameNode is also responsible to take care of the replication factor of all the blocks.
7. In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

Secondary Name Node

What is Secondary Name Node?

Role of Secondary Namenode in Managing the Filesystem Metadata.

Each and every transaction that occurs on the file system is recorded within the edit log file. At some point of time this file becomes very large.

Namenode holds the metadata for HDFS like Block information, size etc. This Information is stored in main memory as well as disk for persistence storage . The information is stored in 2 different files .They are

- **Editlogs-** It keeps track of each and every changes to HDFS.
- **Fsimage-** It stores the snapshot of the file system.

Any changes done to HDFS gets noted in the edit logos the file size grows where as the size of fsimage remains same.

This not have any impact until we restart the server. When we restart the server the edit file logs are written into fsimage file and loaded into main memory which takes some time.

If we restart the cluster after a long time there will be a vast down time since the edit log file would have grown. Secondary name node would come into picture in rescue of this problem.

Secondary Name node simply gets edit logs from name node periodically and copies to fsimage. This new fsimage is copied back to namenode. Namenode now, this uses this new fsimage for next restart which reduces the startup time.

It is a helper node to Name node and to precise Secondary Name node whole purpose is to have checkpoint in HDFS, which helps name node to function effectively. Hence, It is also called as **Checkpoint node**.

Now there are two important files which reside in the namenode' s current directory,

1. FsImage file :-This file is the snapshot of the HDFS metadata at a certain point of time .
2. Edits Log file :-This file stores the records for changes that have been made in the HDFS namespace .

The main function of the **Secondary namenode** is to store the latest copy of the FsImage and the Edits Log files.

How does it help?

When the namenode is restarted , the latest copies of the Edits Log files are applied to the FsImage file in order to keep the HDFS metadata latest. So it becomes very important to store a copy of these two files , which is done by secondary namenode.

Now to keep latest versions of these two files, the secondary name node takes the checkpoints at hourly basis which is the default time gap .

Checkpoint:-

A checkpoint is nothing but the updation of the latest FsImage file by applying the latest Edits Log files to it .If the time gap of a checkpoint is large the there will be too many Edits Log files generated and it will be very cumbersome and time consuming to apply them all at once on the latest FsImage file . And this may lead to acute start time for the primary namenode after a reboot .

However, the secondary namenode is just a helper to the primary namenode in a HDFS cluster as it cannot perform all the functions of the primary namenode .

Note:-
There are two options to which can be used along with secondary namenode command

1. -geteditsize:- this option helps to find the current size of the edit_ingress file present in namenode' s current directory.Here edit_ingress file is the ongoing in progress Edits Log file .
2. -checkpoint [force]:- this option forcefully checkpoints the secondary namenode to the latest state of the primary namenode , whatever may the size of the Edits Log file may be. But ideally the size of the Edits Log file should be greater than or equal to the checkpoint file size .

Namenode holds the meta data 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.

fsimage – Its the snapshot of the filesystem when namenode started
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. 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. The following issues we will encounter in this situation.

1.Editlog become very large , which will be challenging to manage it
2.Namenode restart takes long time because lot of changes has to be merged
3.In the case of crash, we will lost huge amount of metadata since fsimage is very old

Secondary Namenode helps to overcome the above issues by taking over responsibility of merging editlogs with fsimage from the 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.

Things have been changed over the years especially with Hadoop 2.x. Now Namenode is highly available with fail over feature. Secondary Namenode is optional now & Standby Namenode has been to used for failover process. **Standby NameNode** will stay up-to-date with all the file system changes the Active NameNode makes .

The **Secondary namenode** is a helper node in hadoop, To understand the functionality of the secondary namenodelet's understand how the namenode works.

Name node stores metadata like file system namespace information, blockinformation etc in the memory.It also stores the persistent copy of the same on the disk. Name node stores information in two files.

fsimage: It's a snapshot of the file system, stores information like modification time access time, permission, replication.

Edit logs: It stores details of all the activities/transactions being performed on the HDFS..

When the namenode is in the active state the edit logs size grows continuously as the edit logs can only be applied to the fsimage at the time of name node restart, to get the latest state of the HDFS. If edit logs grows significantly and name node tries to apply it on fsimage at the time of name node restart, the process can take very long, here secondary node come into the play.
Secondary namenode keeps the checkpoint on the name node, It reads the edit logs from the namenode continuously after a specific interval and applies it to the fsimage copy of secondary name node. In this way the fsimage file will have the most recent state of HDFS.
The secondary namenode copies new fsimage to primary, so fsimage is updated.

Since fsimage is updated, there will be no overhead of copying of edit logs at the moment of restarting the cluster.
Secondary namenode is a helper node and can't replace the name node.

Data Node

HDFS Data Node
1. Data Node is also known as Slave node.
2. In Hadoop HDFS Architecture, Data Node stores actual data in HDFS.
3. Data Nodes responsible for serving, read and write requests for the clients.
4. Data Nodes can deploy on commodity hardware.
5. Data Nodes sends information to the Name Node about the files and blocks stored in that node and responds to the Name Node for all filesystem operations.
6. When a Data Node starts up it announce itself to the Name Node along with the list of blocks it is responsible for.
7. Data Node is usually configured with a lot of hard disk space. Because the actual data is stored in the Data Node.

Functions of Data Node in HDFS
1. These are slave daemons or process which runs on each slave machine.
2. The actual data is stored on Data Nodes.
3. The Data Nodes perform the low-level read and write requests from the file system's clients.
4. Every Data Node sends a heartbeat message to the Name Node every 3 seconds and conveys that it is alive. In the scenario when Name Node does not receive a heartbeat from a Data Node for 10 minutes, the Name Node considers that particular Data Node as dead and starts the process of Block replication on some other Data Node..
5. All Data Nodes are synchronized in the Hadoop cluster in a way that they can communicate with one another and make sure of
i. Balancing the data in the system
ii. Move data for keeping high replication
iii. Copy Data when required

Basic Operations of Datanode:

- Datanodes is responsible of storing actual data.
- Upon instruction from Namenode, it performs operations like creation/replication/deletion of data blocks.
- When one of Datanode gets down then it will not make any effect on Hadoop cluster due to **replication**.
- All Datanodes are synchronized in the Hadoop cluster in a way that they can communicate with each other for various operations.

What happens if one of the Datanodes gets failed in HDFS?

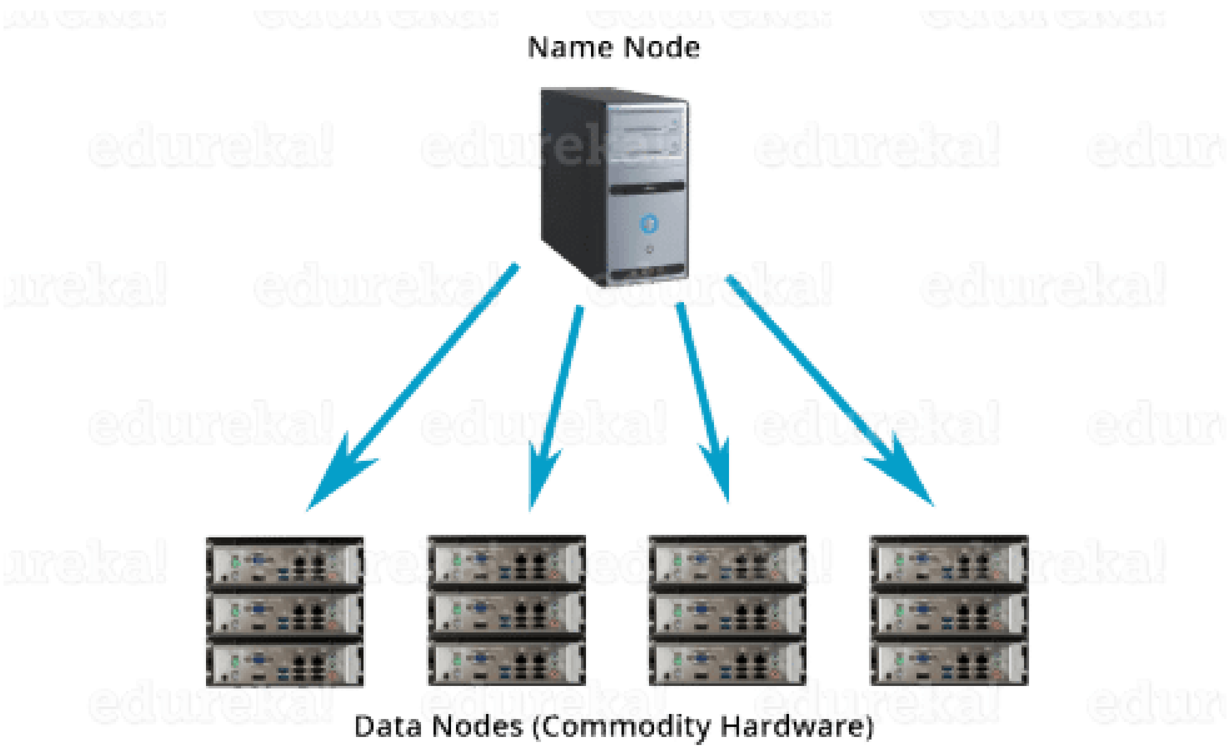
Namenode periodically receives a heartbeat and a Block report from each Datanode in the cluster. Every Datanode sends heartbeat message after every **3 seconds** to Namenode.

The health report is just information about a particular Datanode that is working properly or not. In the other words we can say that particular Datanode is alive or not.

A block report of a particular Data node contains information about all the blocks on that resides on the corresponding Data node. When Name node doesn't receive any heartbeat message for **10 minutes**(ByDefault) from a particular Data node then corresponding Data node is considered Dead or failed by Name node.

Since blocks will be under replicated, the system starts the replication process from one Data node to another by taking all block information from the Block report of corresponding Datanode. The Data for replication transfers directly from one Data node to another without data passing through Name node.

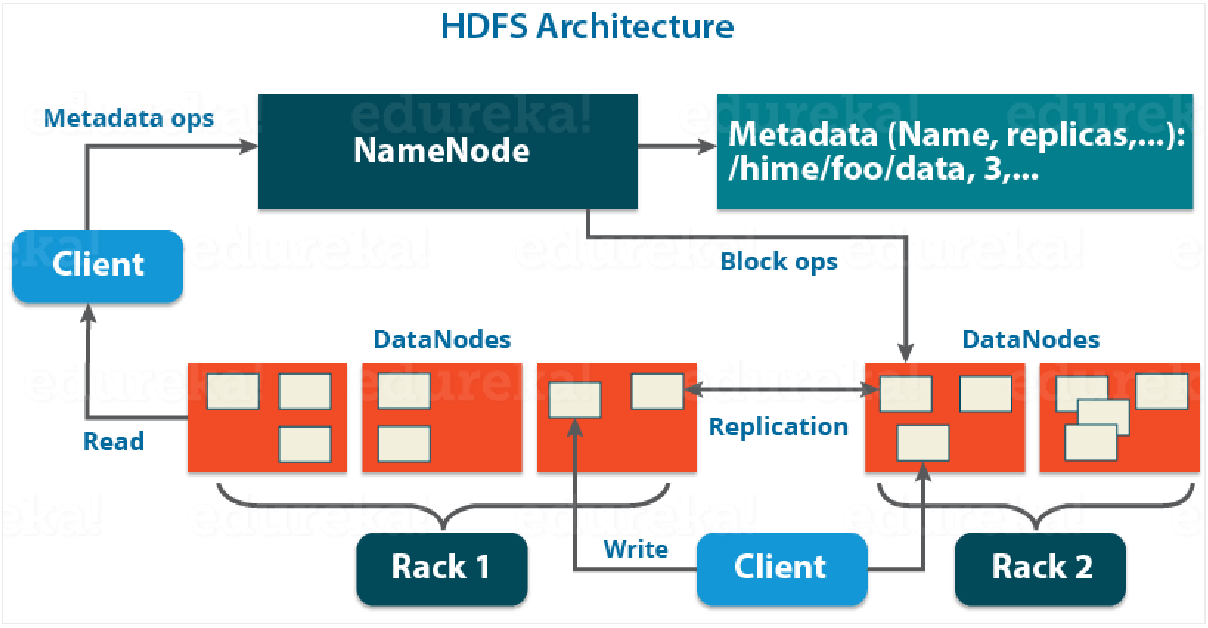
HDFS Architecture



Apache HDFS or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines.

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

NameNode:



NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes). NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients. I will be discussing this High Availability feature of Apache Hadoop HDFS in my next blog. The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

Functions of NameNode:

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
  - **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
  - **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.
- It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.

- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog.
- In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

Understand the various properties of Namenode, Datanode and Secondary Namenode from the Hadoop Administration Course.

DataNode:

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

Functions of DataNode:

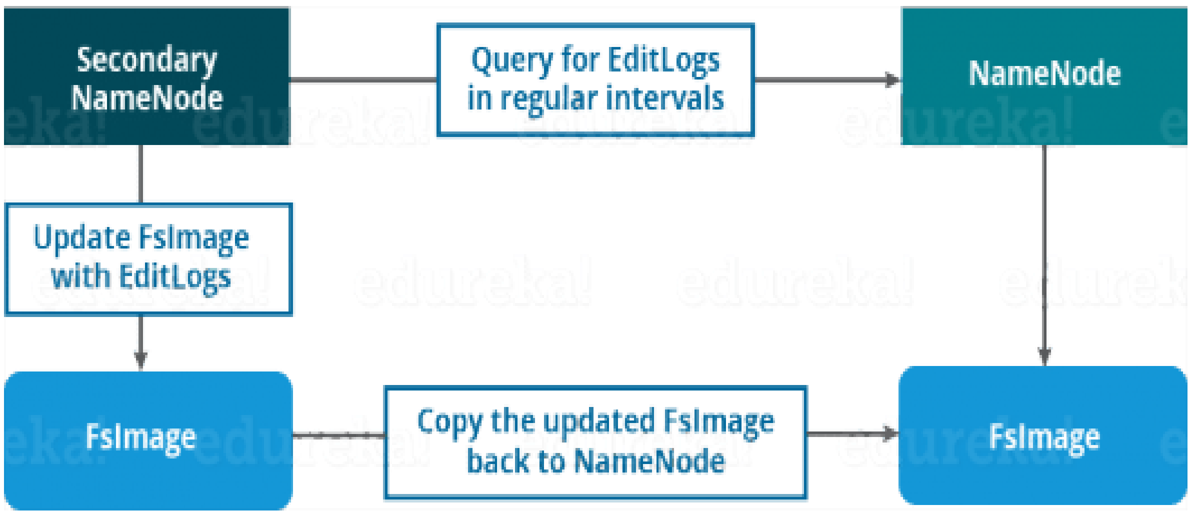
- These are slave daemons or process which runs on each slave machine.
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system's clients.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

Till now, you must have realized that the NameNode is pretty much important to us. If it fails, we are doomed. But don't worry, we will be talking about how Hadoop solved this single point of failure problem in the next Apache Hadoop HDFS Architecture blog. So, just relax for now and let's take one step at a time.

Learn more about Big Data and its applications from the Data Engineer certification.

Secondary NameNode:

Apart from these two daemons, there is a third daemon or a process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a **helper daemon.** And don't be confused about the Secondary NameNode being a **backup NameNode because it is not.**


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

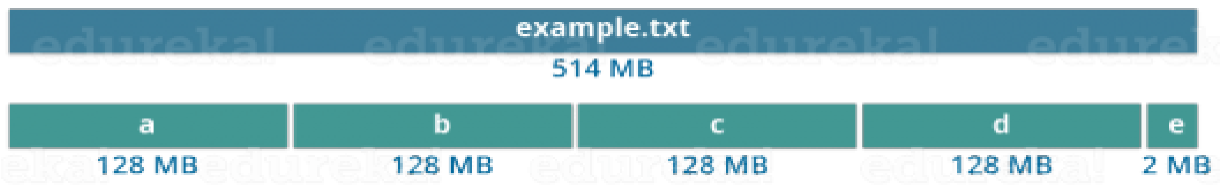
Functions of Secondary NameNode:

- The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system.
- It is responsible for combining the EditLogs with FsImage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.

Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

Blocks:
Now, as we know that the data in HDFS is scattered across the DataNodes as blocks. **Let's have a look at what is a block and how is it formed?**

Blocks are the nothing but the smallest continuous location on your hard drive where data is stored. In general, in any of the File System, you store the data as a collection of blocks. Similarly, HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster. The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement.



It is not necessary that in HDFS, each file is stored in exact multiple of the configured block size (128 MB, 256 MB etc.). Let's take an example where I have a file “example.txt” of size 514 MB as shown in above figure. Suppose that we are using the default configuration of block size, which is 128 MB. Then, how many blocks will be created? 5, Right. The first four blocks will be of 128 MB. But, the last block will be of 2 MB size only.

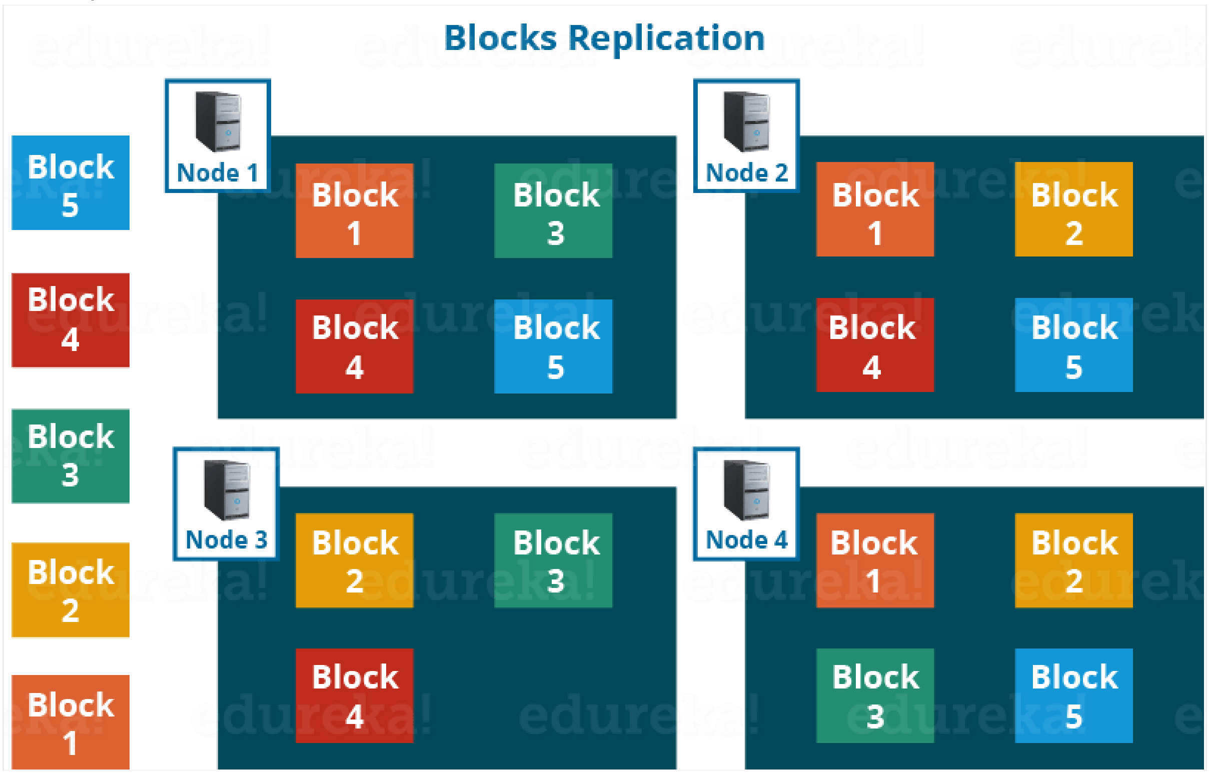
Now, you must be thinking why we need to have such a huge blocks size i.e. 128 MB?

Well, whenever we talk about HDFS, we talk about huge data sets, i.e. Terabytes and Petabytes of data. So, if we had a block size of let’s say of 4 KB, as in Linux file system, we would be having too many blocks and therefore too much of the metadata. So, managing these no. of blocks and metadata will create huge overhead, which is something, we don’t want.

*As you understood **what a block is**, let us understand how the replication of these blocks takes place in the next section of this HDFS Architecture. Meanwhile, you may check out this video tutorial on HDFS Architecture where all the HDFS Architecture concepts has been discussed in detail:*

Replication Management:

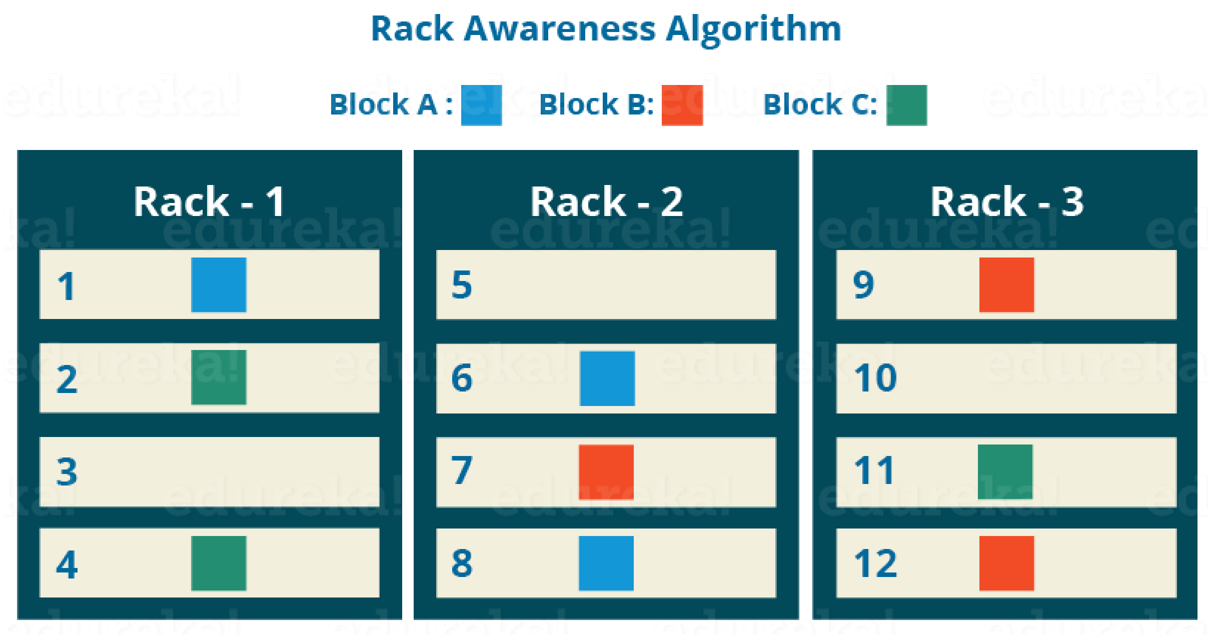
HDFS provides a reliable way to store huge data in a distributed environment as data blocks. The blocks are also replicated to provide fault tolerance. The default replication factor is 3 which is again configurable. So, as you can see in the figure below where each block is replicated three times and stored on different DataNodes (considering the default replication factor):



Therefore, if you are storing a file of 128 MB in HDFS using the default configuration, you will end up occupying a space of 384 MB (3*128 MB) as the blocks will be replicated three times and each replica will be residing on a different DataNode.

Note: The NameNode collects block report from DataNode periodically to maintain the replication factor. Therefore, whenever a block is over-replicated or under-replicated the NameNode deletes or add replicas as needed.

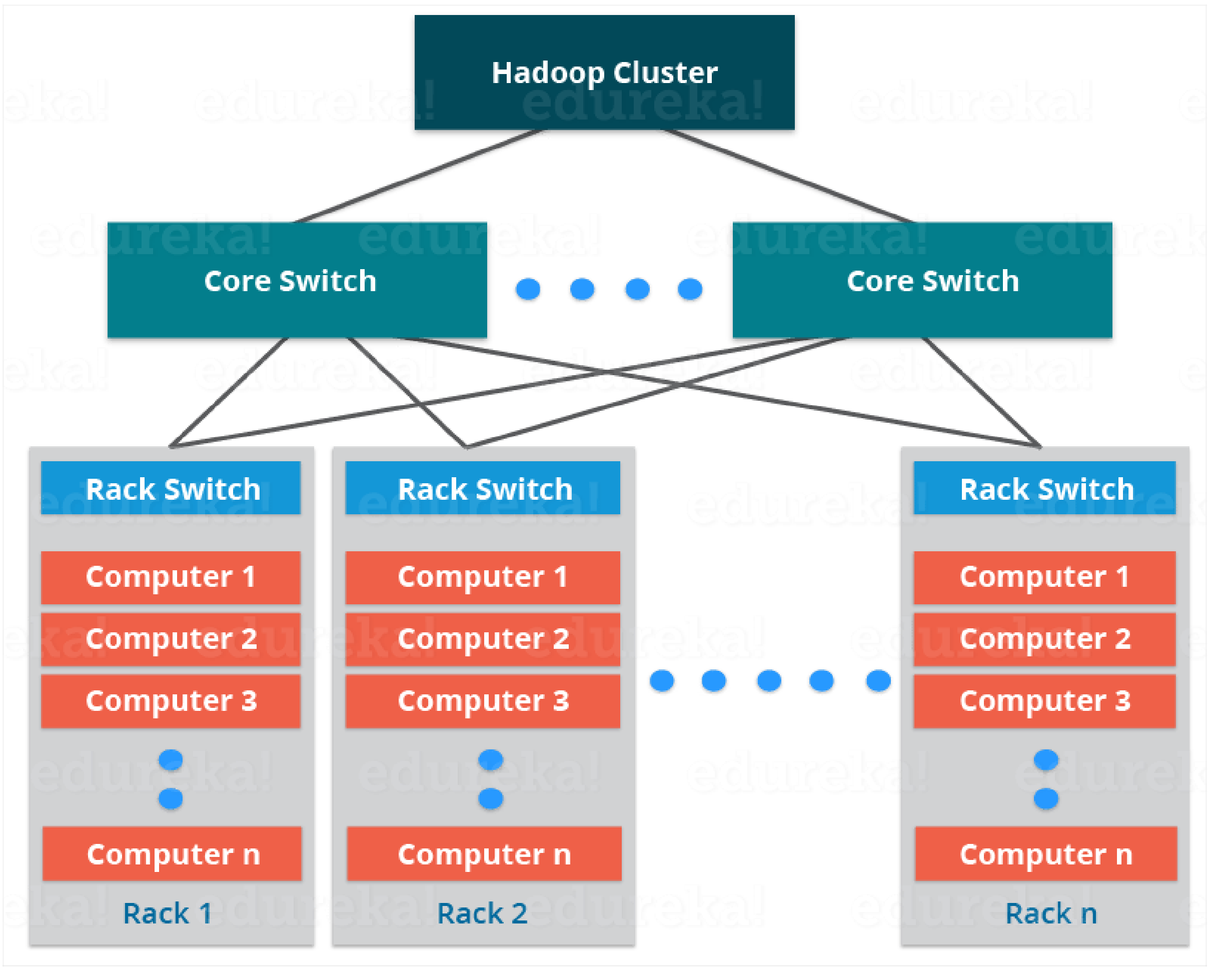
Rack Awareness:



Anyways, moving ahead, let's talk more about how HDFS places replica and what is rack awareness? Again, the NameNode also ensures that all the replicas are not stored on the same rack or a single rack. It follows an in-built Rack Awareness Algorithm to reduce latency as well as provide fault tolerance.

Considering the replication factor is 3, the Rack Awareness Algorithm says that the first replica of a block will be stored on a local rack and the next two replicas will be stored on a different (remote) rack but, on a different DataNode within that (remote) rack as shown in the figure above.

This is how an actual Hadoop production cluster looks like. Here, you have multiple racks populated with DataNodes:



Advantages of Rack Awareness:

So, now you will be thinking why do we need a Rack Awareness algorithm? The reasons are:

- **To improve the network performance:** The communication between nodes residing on different racks is directed via switch. In general, you will find *greater network bandwidth* between machines in the same rack than the machines residing in different rack. So, the Rack Awareness helps you to have reduce write traffic in between different racks and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks.

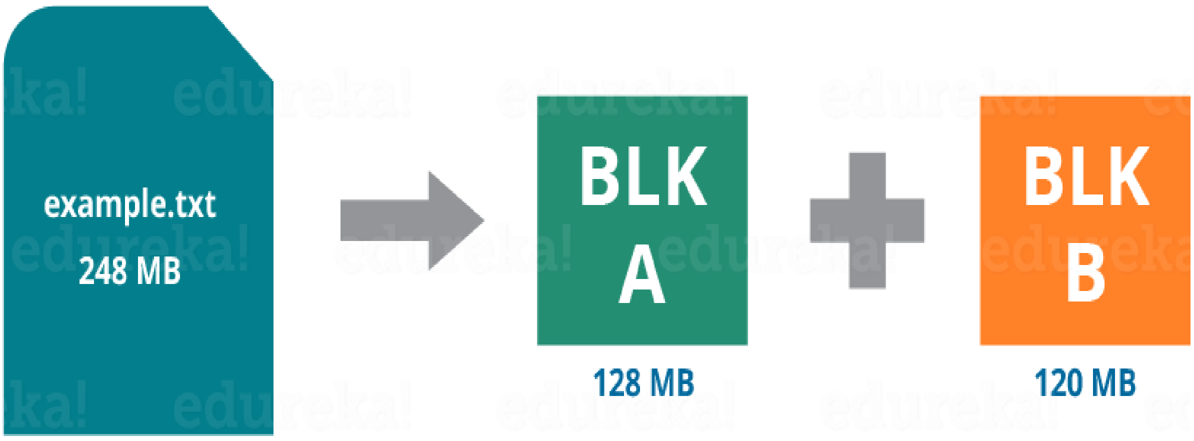
- **To prevent loss of data:** We don't have to worry about the data even if an entire rack fails because of the switch failure or power failure. And if you think about it, it will make sense, as it is said that *never put all your eggs in the same basket.*

HDFS Read/ Write Architecture:

Now let's talk about how the data read/write operations are performed on HDFS. HDFS follows Write Once – Read Many Philosophy. So, you can't edit files already stored in HDFS. But, you can append new data by re-opening the file. Get a better understanding of the Hadoop Clusters, nodes, and architecture from the Hadoop Admin Training in Chennai.

HDFS Write Architecture:

Suppose a situation where an HDFS client, wants to write a file named "example.txt" of size 248 MB.

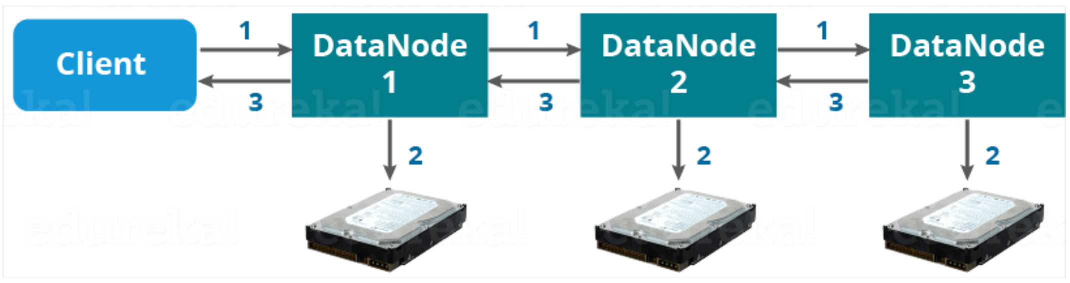


Assume that the system block size is configured for 128 MB (default). So, the client will be dividing the file "example.txt" into 2 blocks – one of 128 MB (Block A) and the other of 120 MB (block B).

Now, the following protocol will be followed whenever the data is written into HDFS:

- At first, the HDFS client will reach out to the NameNode for a Write Request against the two blocks, say, Block A & Block B.
- The NameNode will then grant the client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually.
- The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness that we have discussed earlier.
- Let's say the replication factor is set to default i.e. 3. Therefore, for each block the NameNode will be providing the client a list of (3) IP addresses of DataNodes. The list will be unique for each block.
- Suppose, the NameNode provided following lists of IP addresses to the client:
    - For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}
    - For Block B, set B = {IP of DataNode 3, IP of DataNode 7, IP of DataNode 9}

- Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
- Now the whole data copy process will happen in three stages:

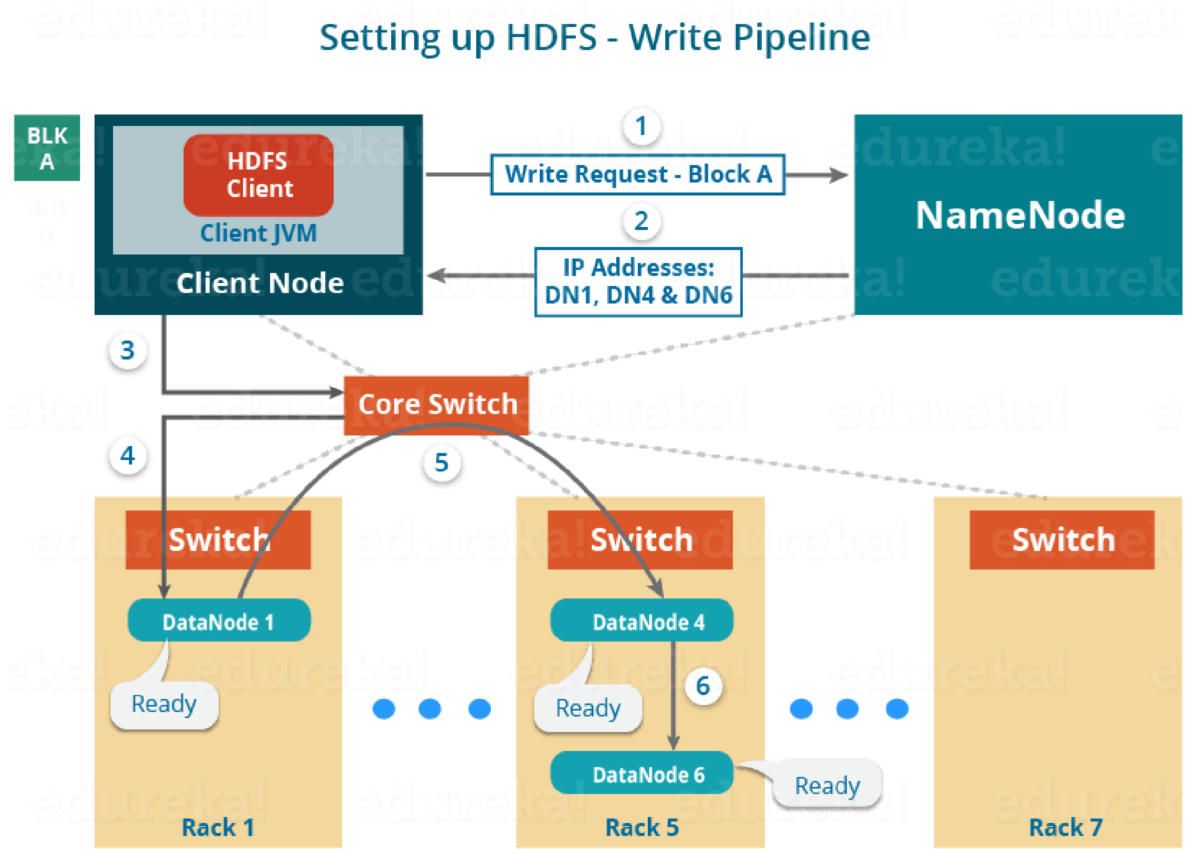


1. Set up of Pipeline
2. Data streaming and replication
3. Shutdown of Pipeline (Acknowledgement stage)

1. Set up of Pipeline:

Before writing the blocks, the client confirms whether the DataNodes, present in each of the list of IPs, are ready to receive the data or not. In doing so, the client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that block. Let us consider Block A. The list of DataNodes provided by the NameNode is:

For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}.

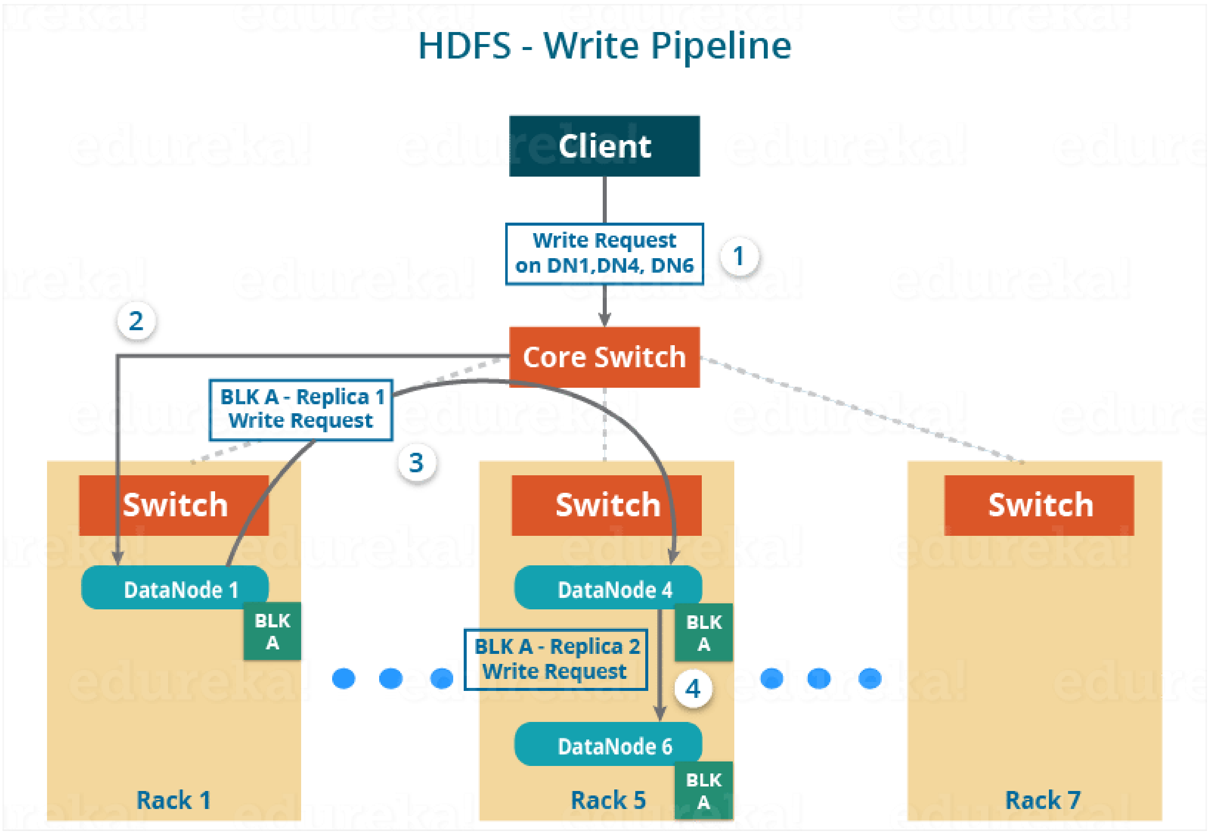


So, for block A, the client will be performing the following steps to create a pipeline:

- The client will choose the first Data Node in the list (Data Node IPs for Block A) which is Data Node 1 and will establish a TCP/IP connection.
- The client will inform Data Node 1 to be ready to receive the block. It will also provide the IPs of next two Data Nodes (4 and 6) to the Data Node 1 where the block is supposed to be replicated.
- The Data Node 1 will connect to Data Node 4. The DataNode 1 will inform Data Node 4 to be ready to receive the block and will give it the IP of DataNode 6. Then, Data Node 4 will tell Data Node 6 to be ready for receiving the data.
- Next, the acknowledgement of readiness will follow the reverse sequence, i.e. From the DataNode 6 to 4 and then to 1.
- At last DataNode 1 will inform the client that all the DataNodes are ready and a pipeline will be formed between the client, DataNode 1, 4 and 6.
- Now pipeline set up is complete and the client will finally begin the data copy or streaming process.

2. Data Streaming:

As the pipeline has been created, the client will push the data into the pipeline. Now, don't forget that in HDFS, data is replicated based on replication factor. So, here Block A will be stored to three DataNodes as the assumed replication factor is 3. Moving ahead, the client will copy the block (A) to DataNode 1 only. The replication is always done by DataNodes sequentially.



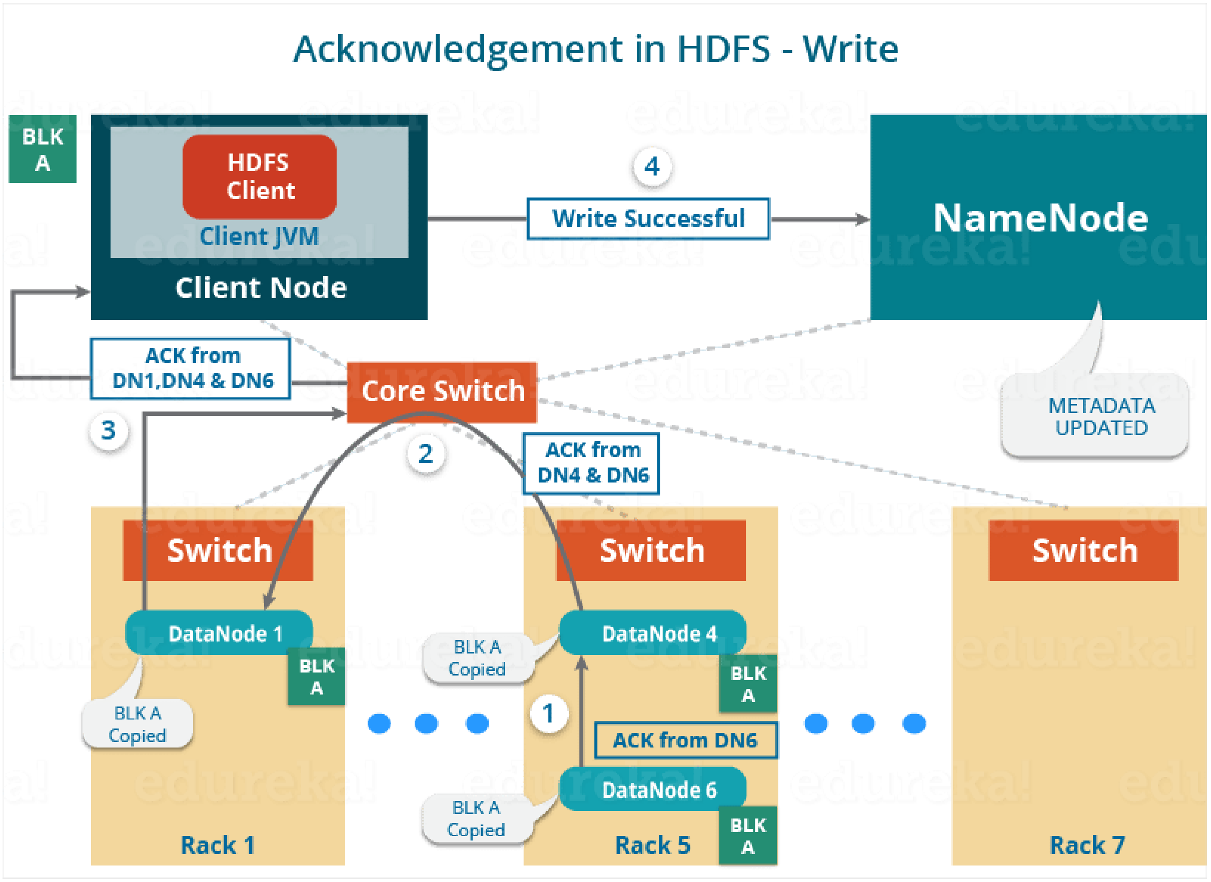
So, the following steps will take place during replication:

- Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
- Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
- Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.

3. Shutdown of Pipeline or Acknowledgement stage:

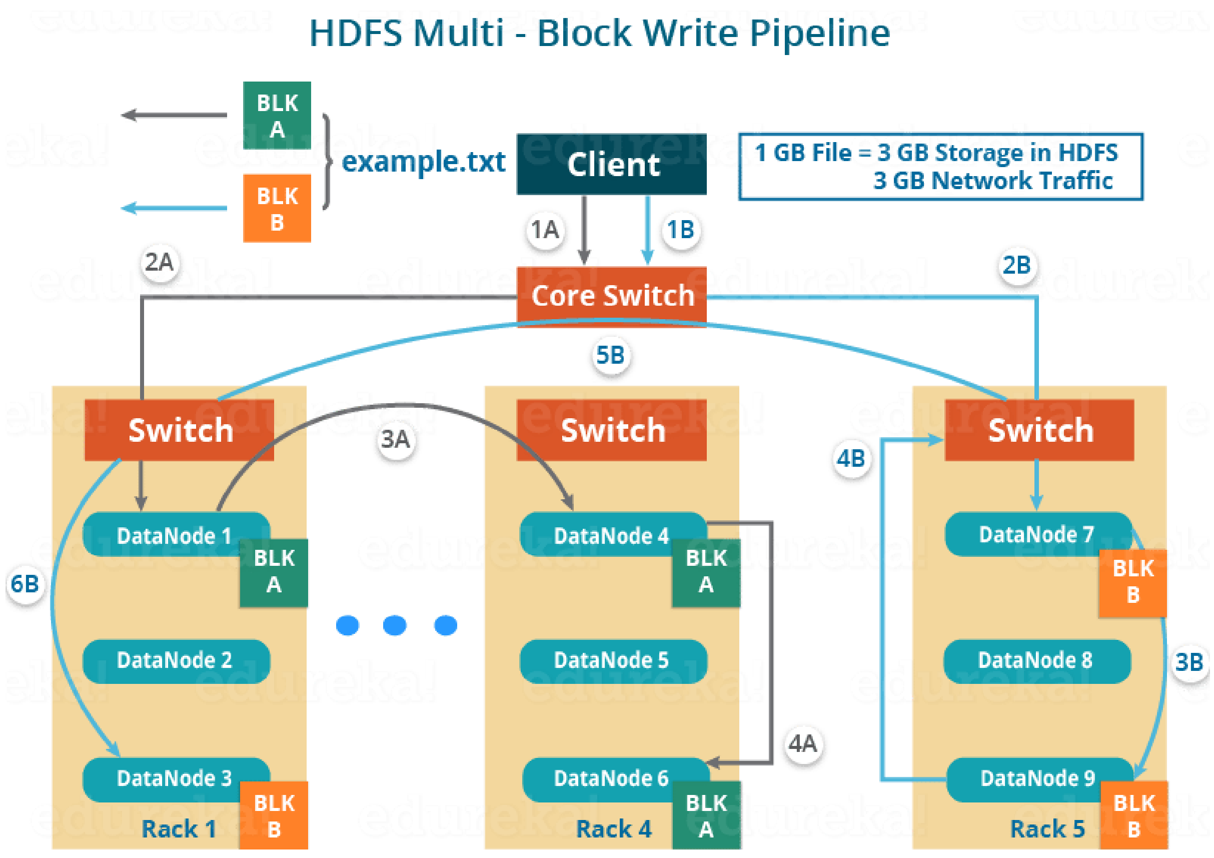
Once the block has been copied into all the three DataNodes, a series of acknowledgements will take place to ensure the client and NameNode that the data has been written successfully. Then, the client will finally close the pipeline to end the TCP session.

As shown in the figure below, the acknowledgement happens in the reverse sequence i.e. from DataNode 6 to 4 and then to 1. Finally, the DataNode 1 will push three acknowledgements (including its own) into the pipeline and send it to the client. The client will inform NameNode that data has been written successfully. The NameNode will update its metadata and the client will shut down the pipeline.



Similarly, Block B will also be copied into the DataNodes in parallel with Block A. So, the following things are to be noticed here:

- The client will copy Block A and Block B to the first DataNode **simultaneously**.
- Therefore, in our case, two pipelines will be formed for each of the block and all the process discussed above will happen in parallel in these two pipelines.
- The client writes the block into the first DataNode and then the DataNodes will be replicating the block sequentially.

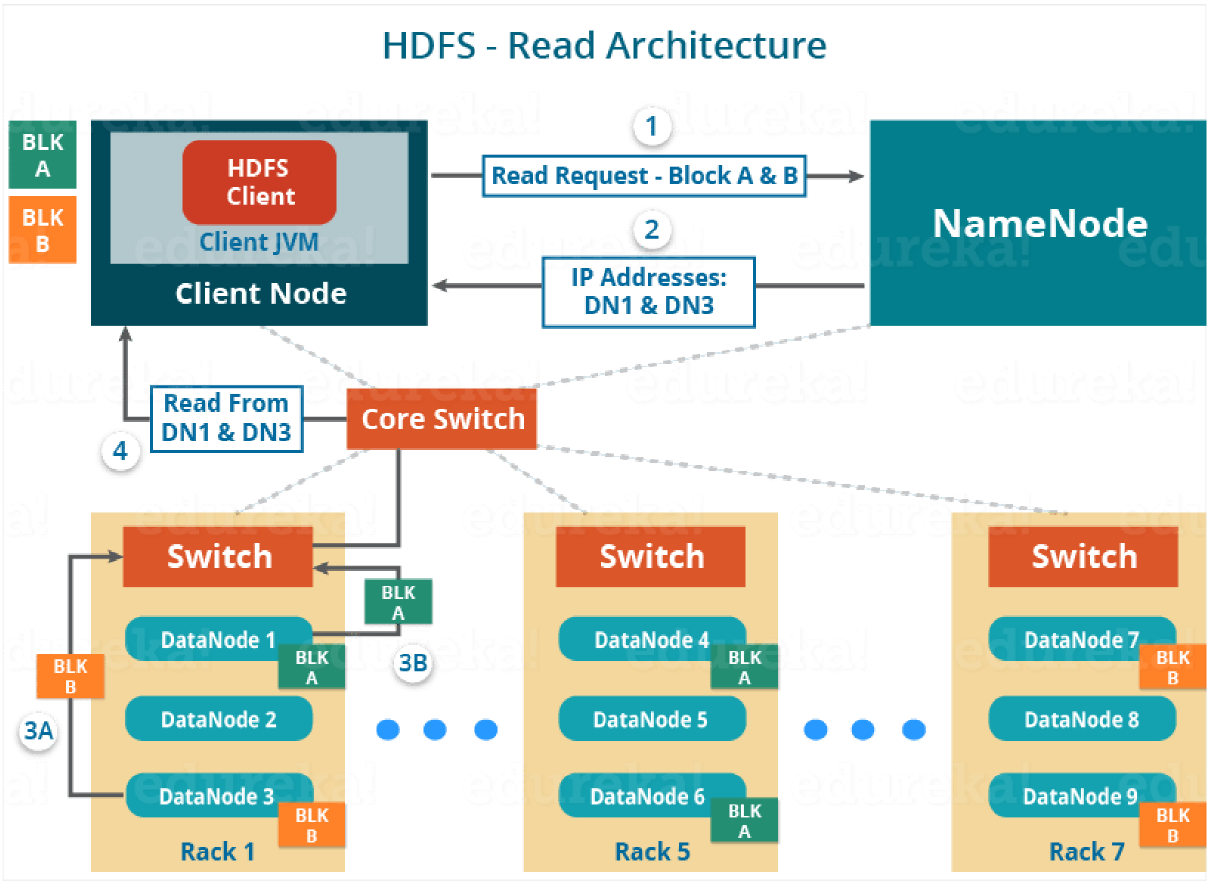


As you can see in the above image, there are two pipelines formed for each block (A and B). Following is the flow of operations that is taking place for each block in their respective pipelines:

- For Block A: 1A -> 2A -> 3A -> 4A
- For Block B: 1B -> 2B -> 3B -> 4B -> 5B -> 6B

HDFS Read Architecture:

HDFS Read architecture is comparatively easy to understand. Let's take the above example again where the HDFS client wants to read the file "example.txt" now.



Now, following steps will be taking place while reading the file:

- The client will reach out to NameNode asking for the block metadata for the file “example.txt”.
- The NameNode will return the list of DataNodes where each block (Block A and B) are stored.
- After that client, will connect to the DataNodes where the blocks are stored.
- The client starts reading data parallel from the DataNodes (Block A from DataNode 1 and Block B from DataNode 3).
- Once the client gets all the required file blocks, it will combine these blocks to form a file.

While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.

Now, you should have a pretty good idea about Apache Hadoop HDFS Architecture. I understand that there is a lot of information here and it may not be easy to get it in one go. I would suggest you to go through it again and I am sure you will find it easier this time. Now, in my next blog, I will be talking about Apache Hadoop HDFS Federation and High Availability Architecture.

Hadoop Configuration

It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce. The hdfs-site. xml file contains the configuration settings for HDFS daemons; the Name Node, the Secondary Name Node, and the Data Nodes. Here, we can configure hdfs-site.

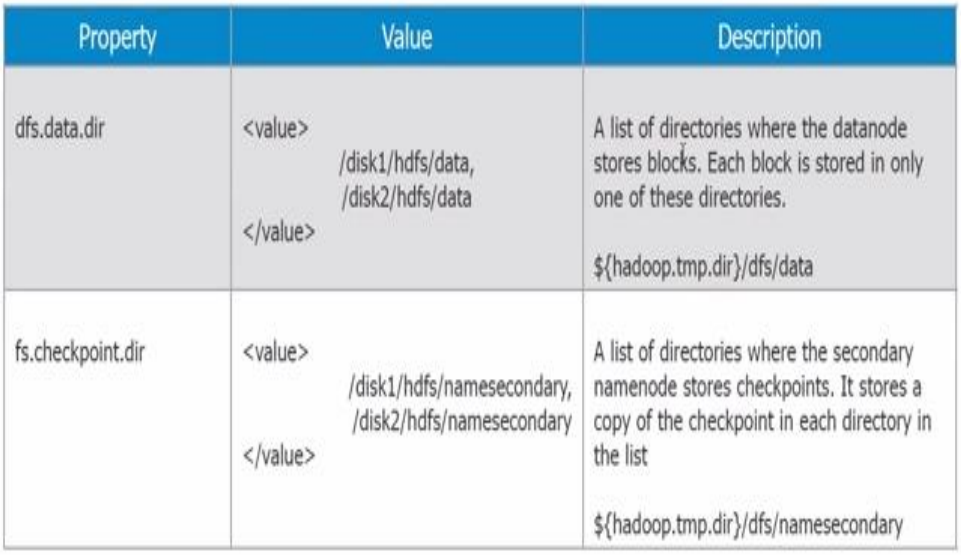
Slaves & Masters:
Slaves contain a list of hosts, one per line, that are needed to host DataNode and TaskTracker servers. The Masters contain a list of hosts, one per line, that are required to host secondary NameNode servers. The Masters file informs about the Secondary NameNode location to Hadoop daemon. The ‘Masters’ **file at Master server contains a hostname, Secondary Name Node servers.**

The Hadoop-env.sh, core-ite.xml, hdfs-site.xml, mapred-site.xml, Masters and Slaves are all available under ‘conf’ directory of Hadoop installation directory.

Core-site.xml and hdfs-site.xml:
The core-site.xml file informs Hadoop daemon where NameNode runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.

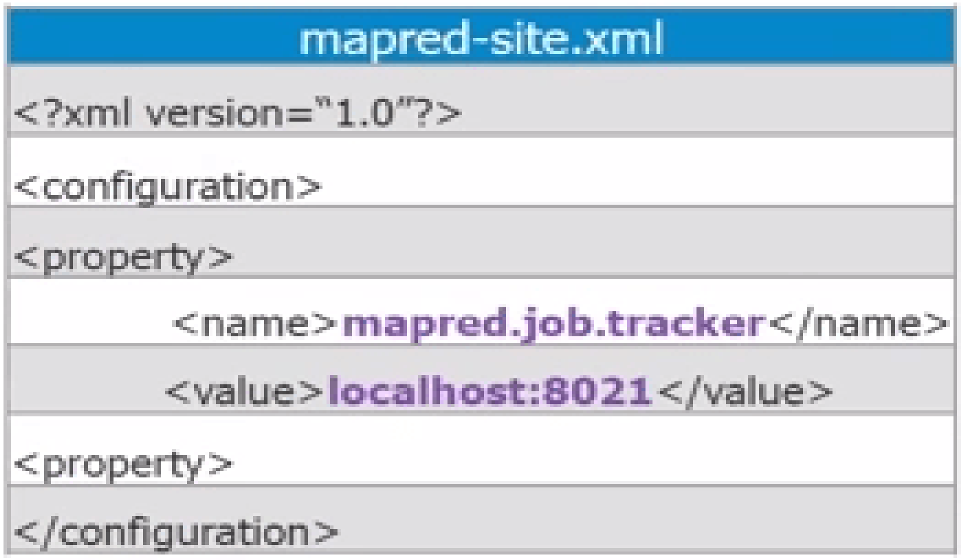
The hdfs-site.xml file contains the configuration settings for HDFS daemons; the NameNode, the Secondary NameNode, and the DataNodes. Here, we can configure hdfs-site.xml to specify default block replication and permission checking on HDFS. The actual number of replications can also be specified when the file is created. The default is used if replication is not specified in create time.

Defining HDFS Details in hdfs-site.xml:



Property	Value	Description
dfs.data.dir	<value> /disk1/hdfs/data, /disk2/hdfs/data </value>	A list of directories where the datanode stores blocks. Each block is stored in only one of these directories. ${hadoop.tmp.dir}/dfs/data
fs.checkpoint.dir	<value> /disk1/hdfs/namesecondary, /disk2/hdfs/namesecondary </value>	A list of directories where the secondary namenode stores checkpoints. It stores a copy of the checkpoint in each directory in the list ${hadoop.tmp.dir}/dfs/namesecondary

Mapred-site.xml:



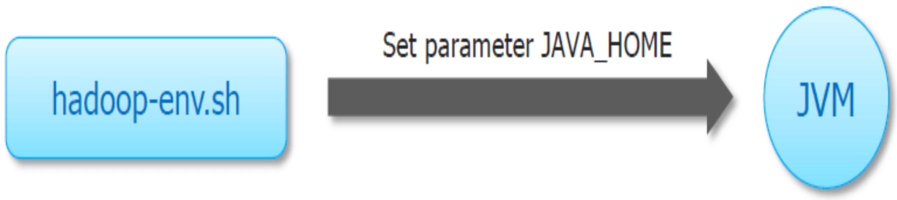
```
mapred-site.xml
<?xml version="1.0"?>
<configuration>
<property>
        <name>mapred.job.tracker</name>
        <value>localhost:8021</value>
<property>
</configuration>
```

The mapred-site.xml file contains the configuration settings for MapReduce daemons; the job tracker and the task-trackers.

Defining mapred-site.xml:

Property	Value	Description
mapred.job.tracker	<value> localhost:8021 </value>	The hostname and the port that the jobtracker RPC server runs on. If set to the default value of local, then the jobtracker runs in-process on demand when you run a MapReduce job.
mapred.local.dir	${hadoop.tmp.dir}/mapred/local	A list of directories where MapReduce stores intermediate data for jobs. The data is cleared out when the job ends.
mapred.system.dir	${hadoop.tmp.dir}/mapred/system	The directory relative to fs.default.name where shared files are stored, during a job run.
mapred.tasktracker.map.tasks.maximum	2	The number of map tasks that may be run on a tasktracker at any one time
mapred.tasktracker.reduce.tasks.maximum	2	The number of reduce tasks tat may be run on a tasktracker at any one time.

Per-Proccess Run Time Environment:



This file offers a way to provide customer parameters for each of the servers. Hadoop-env.sh is sourced by the entire Hadoop core scripts provided in the 'conf/' directory of the installation.

Here are some examples of environment variables than can be specified:

exportHADOOP_DATANODE_HEAPSIZE="128"

exportHADOOP_TASKTRACKER_HEAPSIZE="512"

The 'hadoop-metrics.properties' file controls the reporting and the default condition is set as not to report.

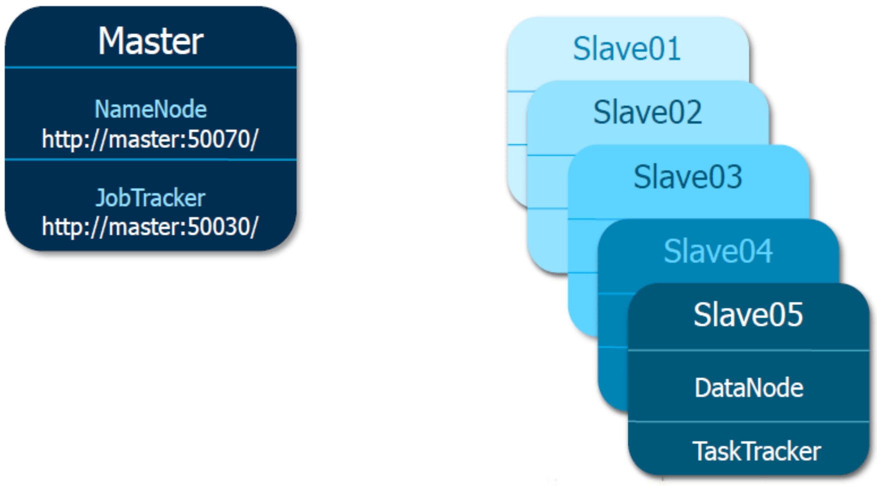
Hadoop Cluster in Facebook:

Facebook uses Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting, analytics and machine learning. Currently, Facebook has two major clusters: A 1100-machine cluster with 800 cores and about 12 PB raw storage. Another one is a 300 machine cluster with

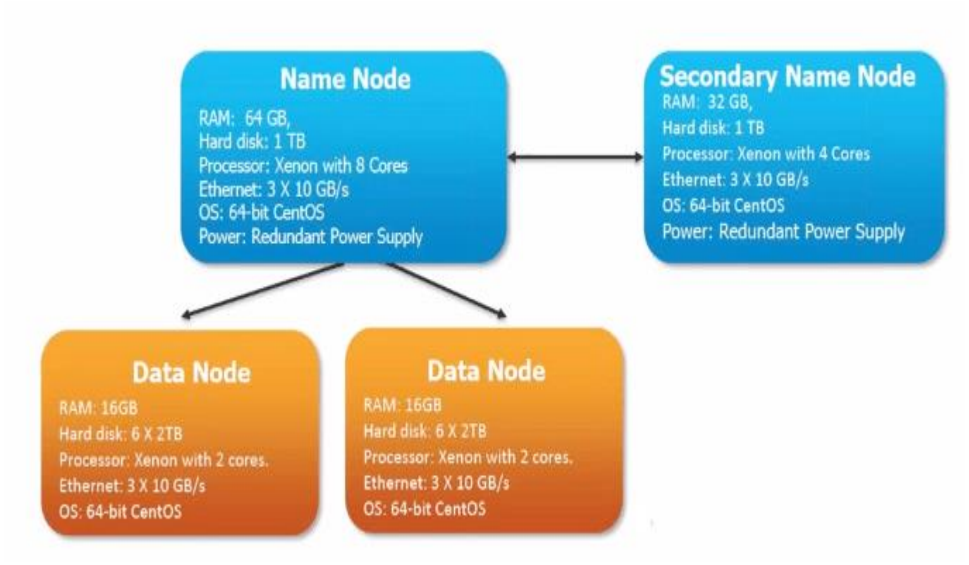
2,400 cores and about 3 PB raw storage. Each of the commodity node has 8 cores and 12 TB storage.

Facebook uses streaming and Java API a lot and have used Hive to build a higher-level data warehousing framework. They have also developed a FUSE application over HDFS.

Sample Cluster Configuration:



Hadoop Cluster – A Typical Use Case:



The above image clearly explains the configuration of each nodes. NameNode has high memory requirement and will have a lot of RAM and does not require a lot of memory on hard disk. The memory requirement for a secondary NameNode is not as high as the primary NameNode. Each DataNode requires 16 GB of memory and are high on hard disk as they are supposed to store data. They have multiple drives as well. Learn more from this Big Data Course about Hadoop Clusters, HDFS, and other important topics to become a Hadoop professional.

Map Reduce Framework:

MapReduce is a software framework and programming model used for processing huge amounts of data. **MapReduce** program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.
Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. The programs of Map Reduce in cloud computing are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

The input to each phase is **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

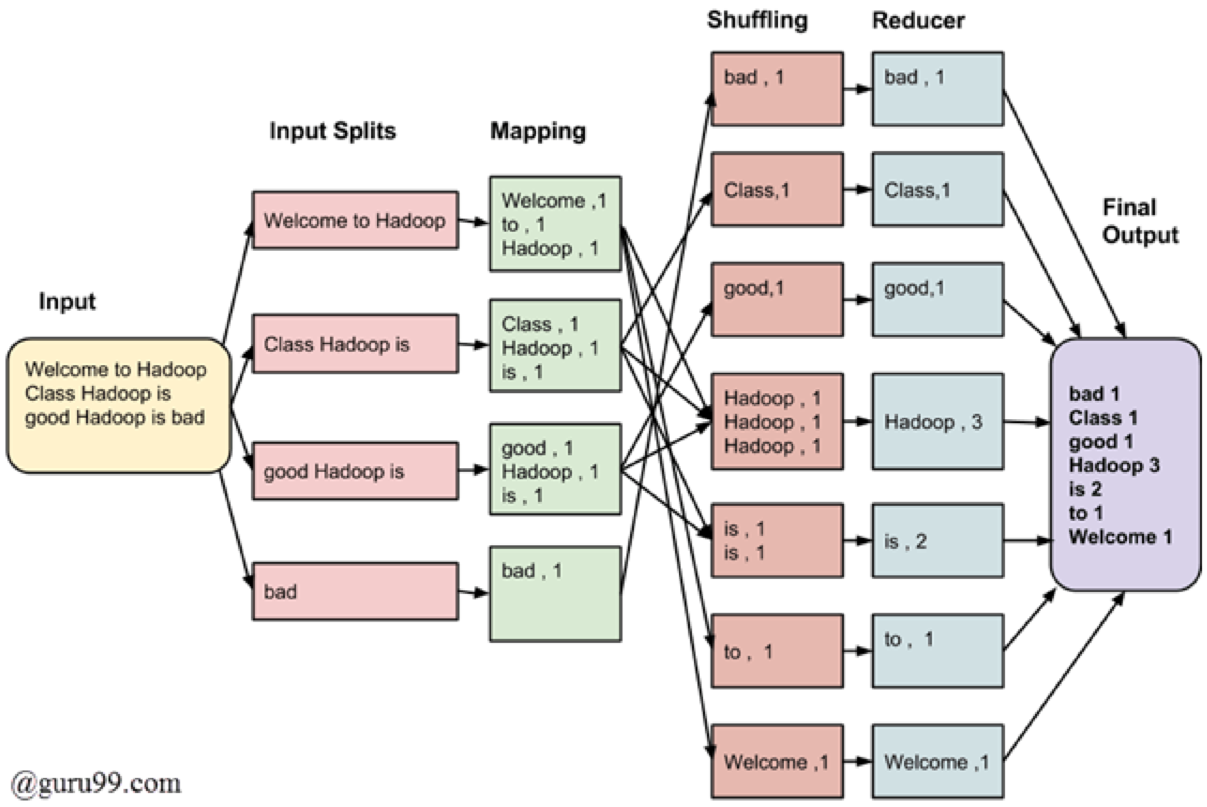
MapReduce Architecture in Big Data explained with Example

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

Now in this MapReduce tutorial, let's understand with a MapReduce example–

Consider you have following input data for your MapReduce in Big data Program

```
Welcome to Hadoop Class
Hadoop is good
Hadoop is bad
```



The final output of the MapReduce task is

bad	
Class	
good	
Hadoop	
is	
to	

Welcome

The data goes through the following phases of MapReduce in Big Data

Input Splits:

An input to a MapReduce in Big Data job is divided into fixed-size pieces called **input splits** Input split is a chunk of the input that is consumed by a single map

Mapping

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

Shuffling

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

Reducing

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

MapReduce Architecture

- One map task is created for each split which then executes map function for each record in the split.

- It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel.
- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default).
- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
- In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
- Reduce task doesn’t work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
- On this machine, the output is merged and then passed to the user-defined reduce function.
- Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

How MapReduce Organizes Work?

Now in this MapReduce tutorial, we will learn how MapReduce works

Hadoop divides the job into tasks. There are two types of tasks:

1. **Map tasks** (Splits & Mapping)

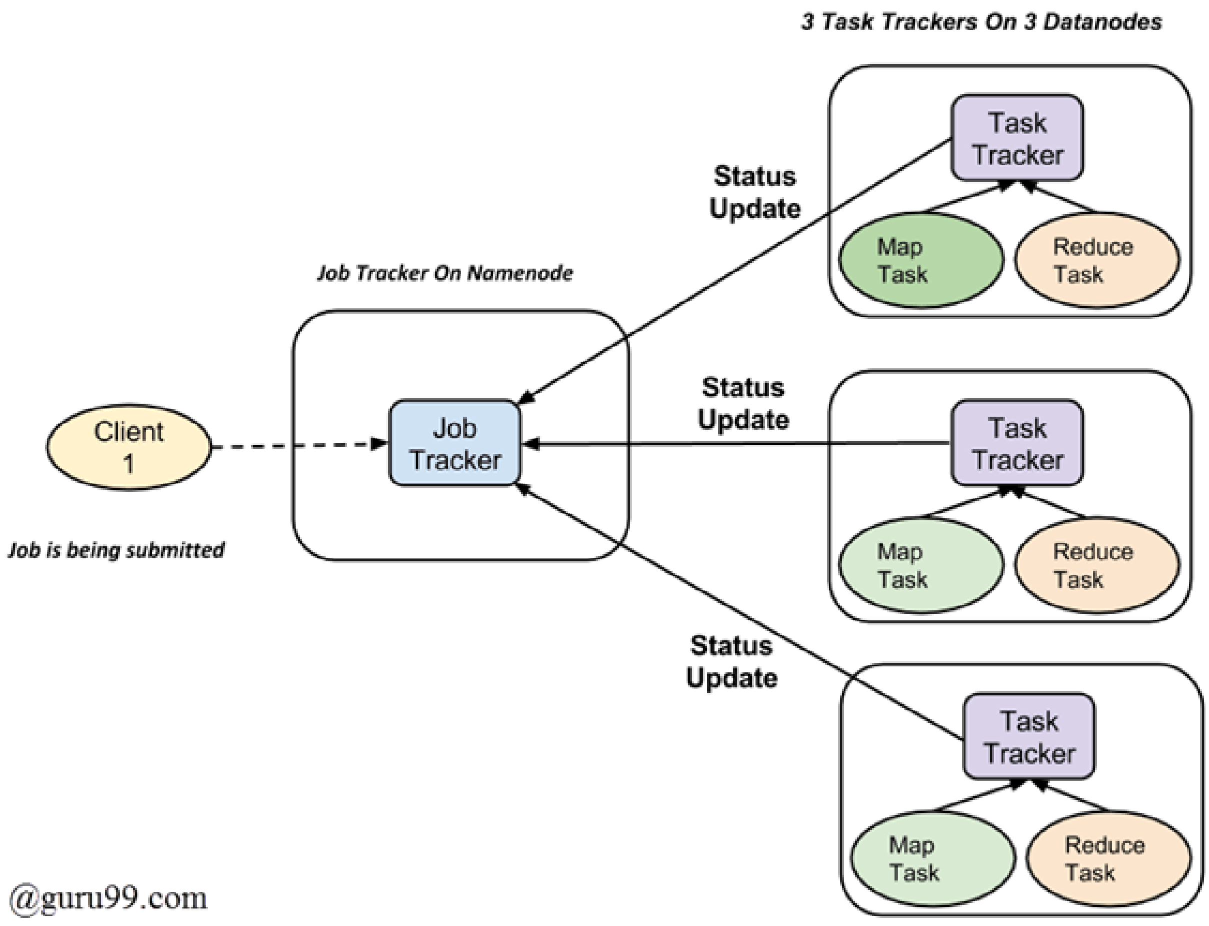
2. **Reduce tasks** (Shuffling, Reducing)

as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

1. **Jobtracker**: Acts like a **master** (responsible for complete execution of submitted job)
2. **Multiple Task Trackers**: Acts like **slaves,** each of them performing the job

For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.



How Hadoop MapReduce Works

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

Role of Hbase in Big Data Processing:

HBase provides low latency random read and write access to petabytes of data by distributing requests from applications across a cluster of hosts. Each host has access to data in HDFS and S3, and serves read and write requests in milliseconds.

Since 1970, RDBMS is the solution for data storage and maintenance related problems. After the advent of big data, companies realized the benefit of processing big data and started opting for solutions like Hadoop.

Hadoop uses distributed file system for storing big data, and MapReduce to process it. Hadoop excels in storing and processing of huge data of various formats such as arbitrary, semi-, or even unstructured.

Limitations of Hadoop

Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.

A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

Applications such as HBase, Cassandra, couchDB, Dynamo, and MongoDB are some of the databases that store huge amounts of data and access the data in a random manner.

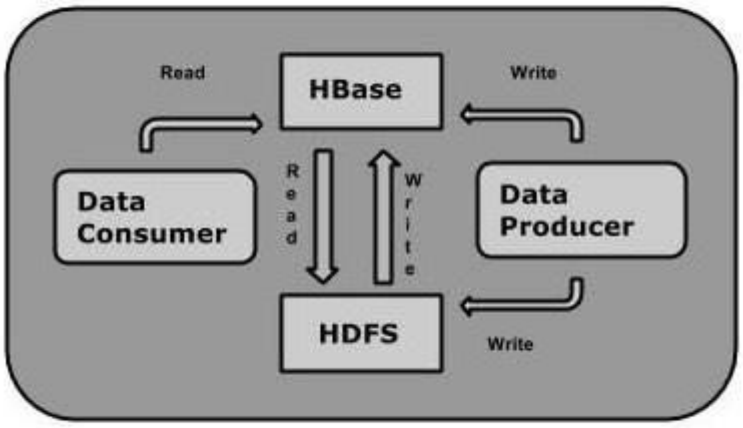
What is HBase?

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.

HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

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.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

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. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Given below is an example schema of table in HBase.

Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	**col2**	**col3**	**col1**	**col2**	**col3**	**col1**	**col2**	**col3**	**col1**	**col2**	**col3**
1												
2												
3												

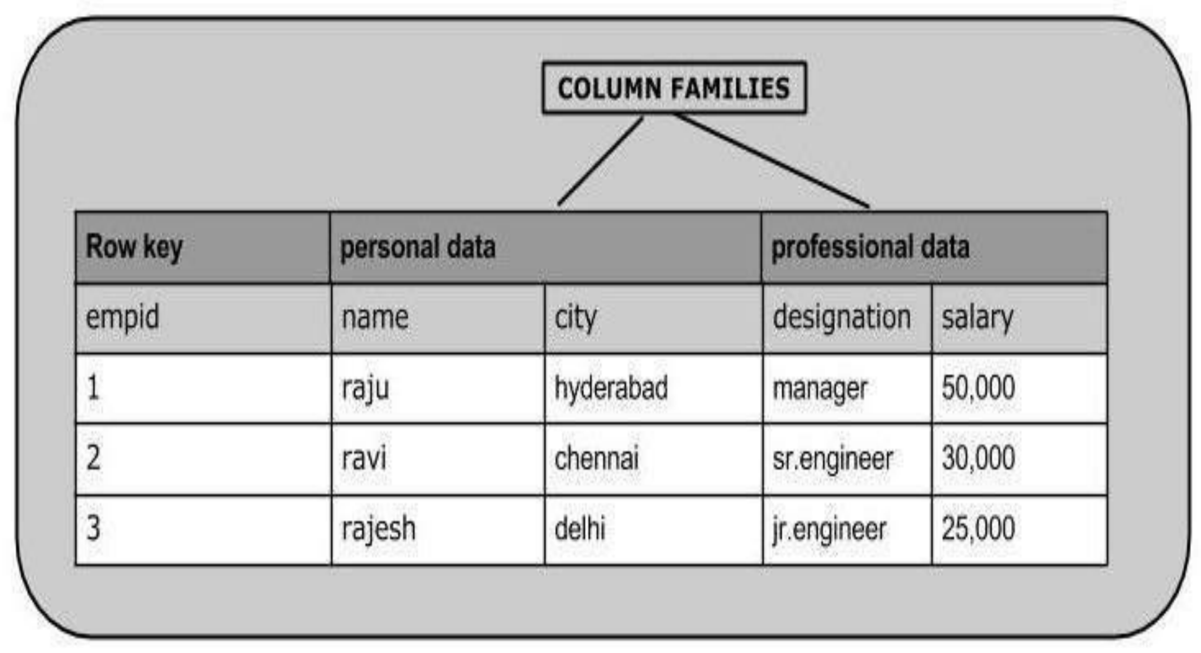
Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).

Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:



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 and RDBMS

HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Where to Use HBase

- Apache HBase is used to have random, real-time read/write access to Big Data.
- It hosts very large tables on top of clusters of commodity hardware.
- Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

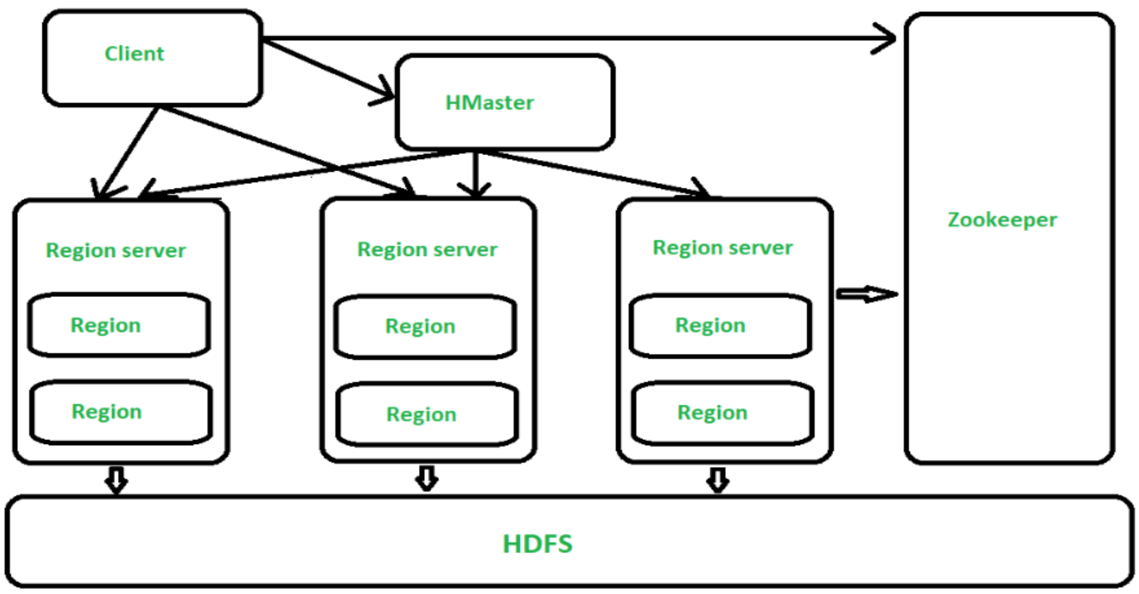
Applications of HBase

- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase - Architecture

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.

Note: The term 'store' is used for regions to explain the storage structure.



HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

MasterServer

The master server -

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

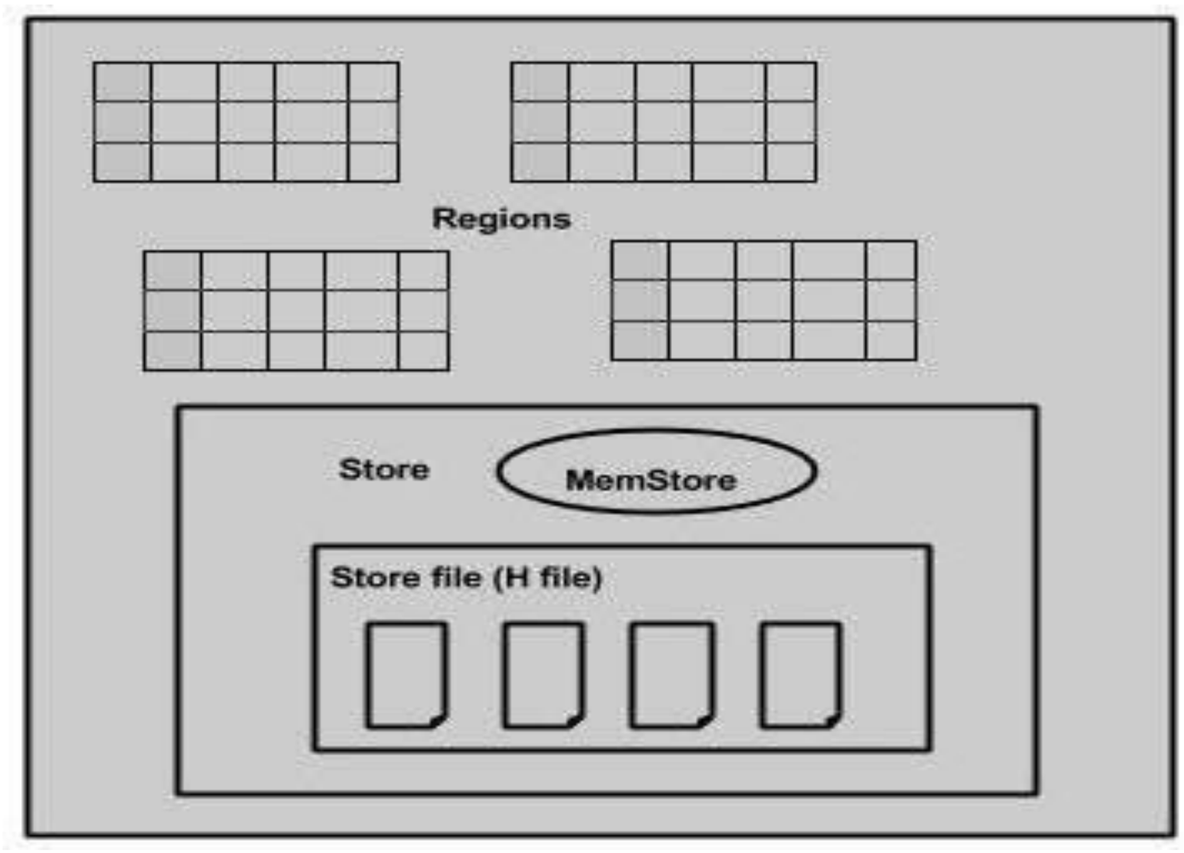
Regions are nothing but tables that are split up and spread across the region servers.

Region server

The region servers have regions that -

- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Decide the size of the region by following the region size thresholds.

When we take a deeper look into the region server, it contain regions and stores as shown below:



The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

Zookeeper

- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.

- In pseudo and standalone modes, HBase itself will take care of zookeeper.

Hive

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

Features of Hive

These are the following features of Hive:

- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile, and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.
- It supports user-defined functions (UDFs) where user can provide its functionality.

Limitations of Hive

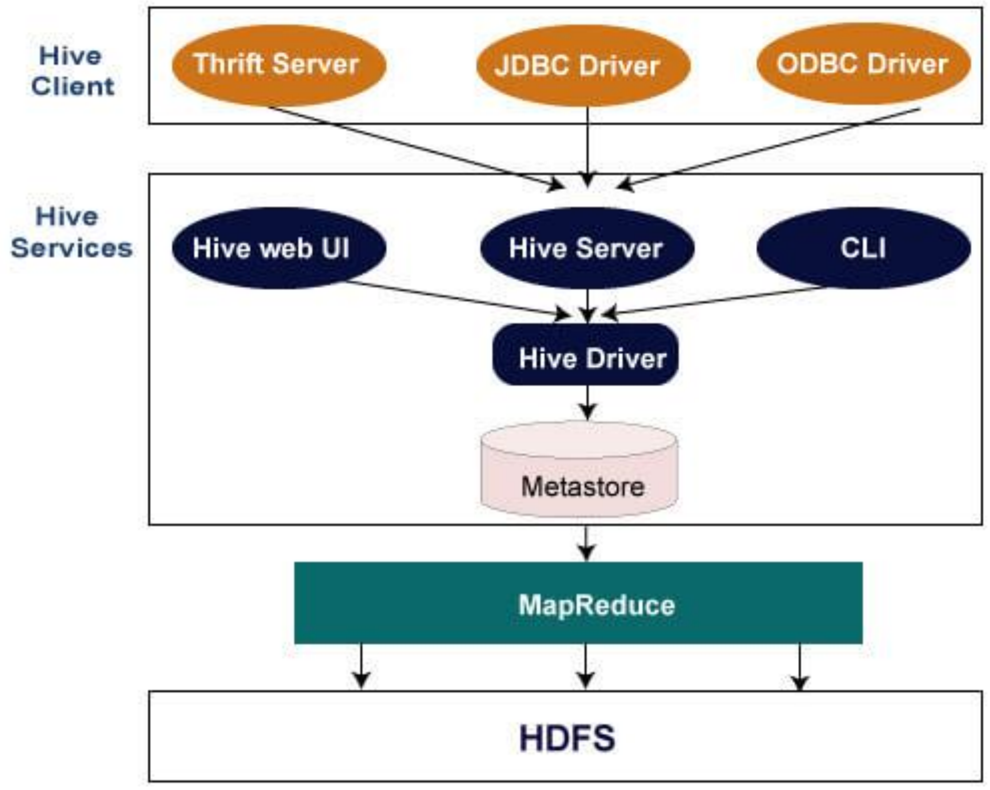
- Hive is not capable of handling real-time data.
- It is not designed for online transaction processing.
- Hive queries contain high latency.

Differences between Hive and Pig

Hive	Pig
Hive is commonly used by Data Analysts.	Pig is commonly used by programmers.
It follows SQL-like queries.	It follows the data-flow language.
It can handle structured data.	It can handle semi-structured data.
It works on server-side of HDFS cluster.	It works on client-side of HDFS cluster.
Hive is slower than Pig.	Pig is comparatively faster than Hive.

Hive Architecture

The following architecture explains the flow of submission of query into Hive.



Hive Client

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

- Thrift Server - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- JDBC Driver - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
- ODBC Driver - It allows the applications that support the ODBC protocol to connect to Hive.

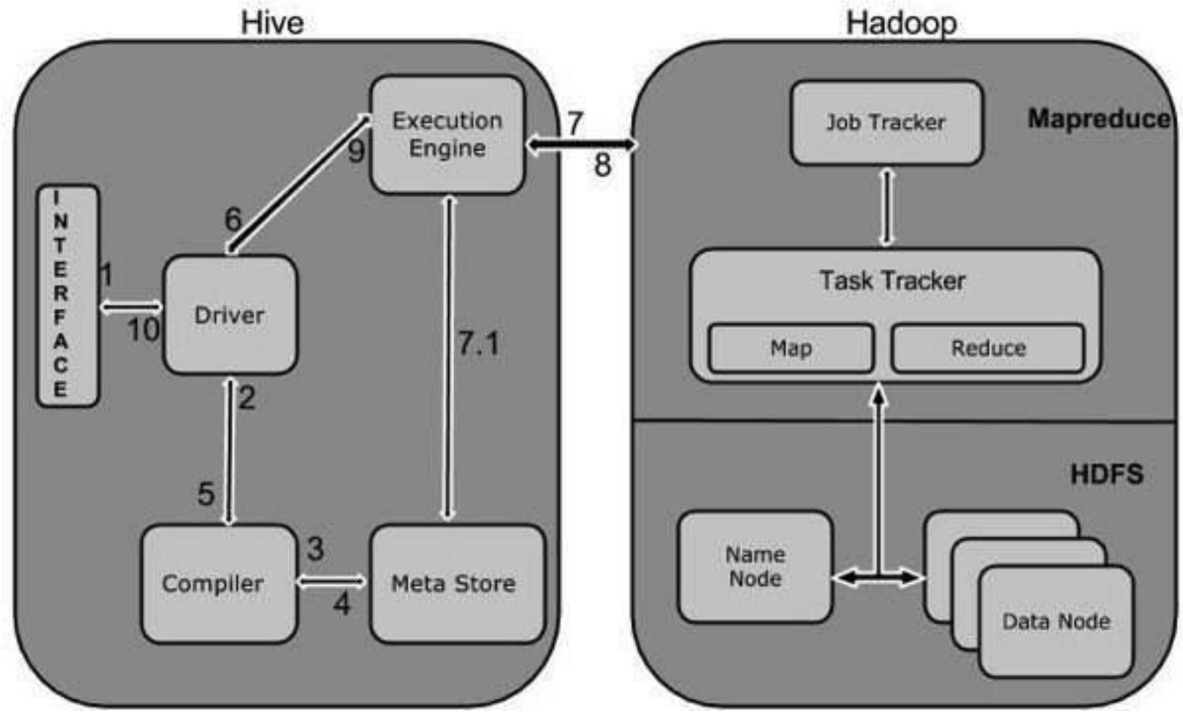
Hive Services

The following are the services provided by Hive:-

- Hive CLI - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- Hive Web User Interface - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- Hive MetaStore - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- Hive Server - It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- Hive Driver - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
- Hive Compiler - The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
- Hive Execution Engine - Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

Step No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.

5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	**Execute Plan** The driver sends the execute plan to the execution engine.
7	**Execute Job** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results** The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

PIG

What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with **Hadoop**; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using **Apache Pig**, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as **Pig Engine** that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

- Using **Pig Latin**, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses **multi-query approach**, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features –

- **Rich set of operators** – It provides many operators to perform operations like join, sort, filer, etc.
- **Ease of programming** – Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- **Optimization opportunities** – The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
- **Extensibility** – Using the existing operators, users can develop their own functions to read, process, and write data.

- **UDF's** – Pig provides the facility to create **User-defined Functions** in other programming languages such as Java and invoke or embed them in Pig Scripts.
- **Handles all kinds of data** – Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Apache Pig Vs MapReduce

Listed below are the major differences between Apache Pig and MapReduce.

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

Apache Pig Vs SQL

Listed below are the major differences between Apache Pig and SQL.

Pig	SQL

Pig Latin is a procedural language.	**SQL is a declarative language.**
In Apache Pig, schema is optional. We can store data without designing a schema (values are stored as $01, $02 etc.)	**Schema is mandatory in SQL.**
The data model in Apache Pig is nested relational.	**The data model used in SQL is flat relational.**
Apache Pig provides limited opportunity for Query optimization.	**There is more opportunity for query optimization in SQL.**

In addition to above differences, Apache Pig Latin –

- Allows splits in the pipeline.
- Allows developers to store data anywhere in the pipeline.
- Declares execution plans.
- Provides operators to perform ETL (Extract, Transform, and Load) functions.

Apache Pig Vs Hive

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

Apache Pig	Hive
Apache Pig uses a language called **Pig Latin**. It was originally created at **Yahoo**.	Hive uses a language called **HiveQL**. It was originally created at **Facebook**.
Pig Latin is a data flow language.	HiveQL is a query processing language.
Pig Latin is a procedural language and it fits in pipeline paradigm.	HiveQL is a declarative language.
Apache Pig can handle structured, unstructured, and semi-structured data.	Hive is mostly for structured data.

Applications of Apache Pig

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used –

- To process huge data sources such as web logs.
- To perform data processing for search platforms.
- To process time sensitive data loads.

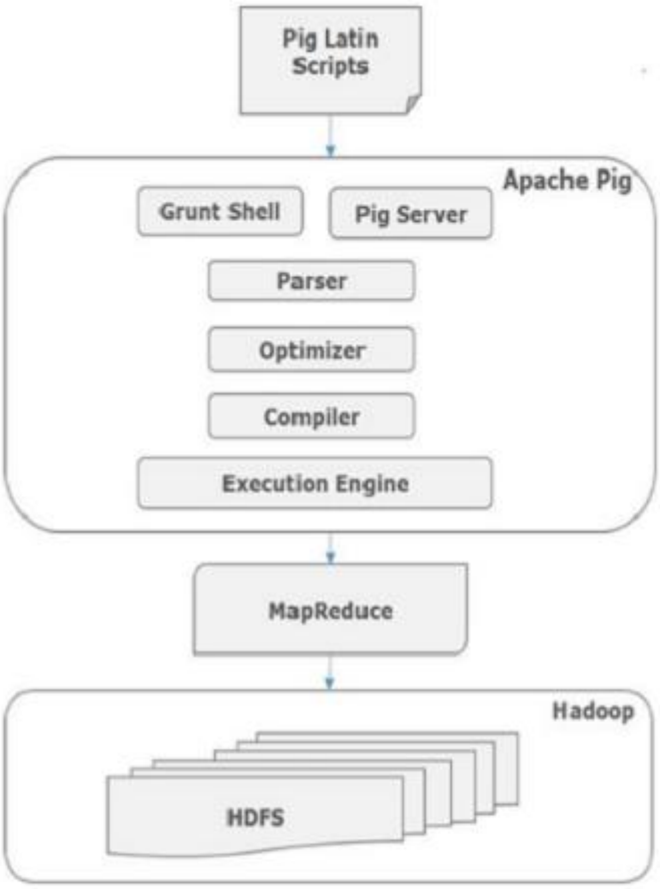
Apache Pig – History

In **2006**, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset. In **2007**, Apache Pig was open sourced via Apache incubator. In **2008**, the first release of Apache Pig came out. In **2010**, Apache Pig graduated as an Apache top-level project.

The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.



Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

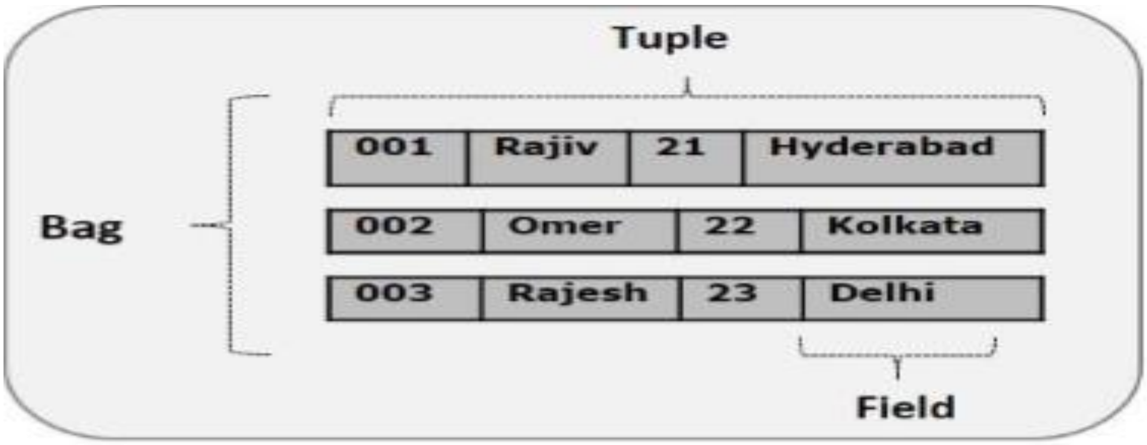
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

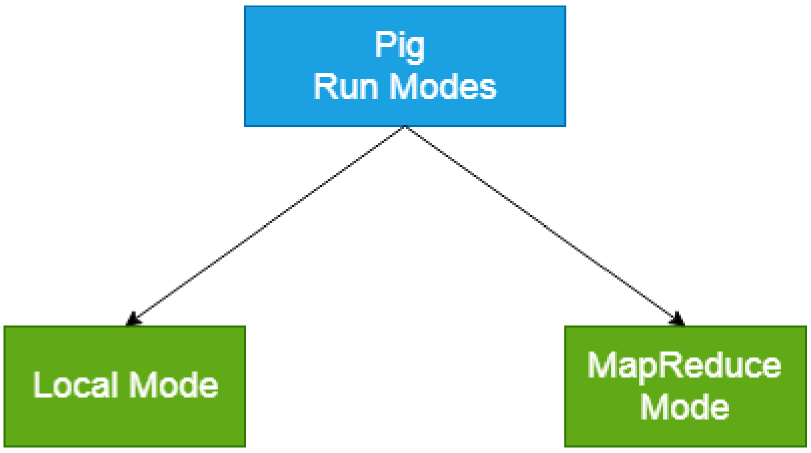
Pig Latin Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**. Given below is the diagrammatical representation of Pig Latin's data model.



Apache Pig Run Modes

Apache Pig executes in two modes: Local Mode and MapReduce Mode.



Local Mode

- It executes in a single JVM and is used for development experimenting and prototyping.
- Here, files are installed and run using localhost.

- The local mode works on a local file system. The input and output data stored in the local file system.

The command for local mode grunt shell:

```
1. $ pig-x local
```

MapReduce Mode

- The MapReduce mode is also known as Hadoop Mode.
- It is the default mode.
- In this Pig renders Pig Latin into MapReduce jobs and executes them on the cluster.
- It can be executed against semi-distributed or fully distributed Hadoop installation.
- Here, the input and output data are present on HDFS.

The command for Map reduce mode:

```
1. $ pig
```

Ways to execute Pig Program

These are the following ways of executing a Pig program on local and MapReduce mode: -

- **Interactive Mode** - In this mode, the Pig is executed in the Grunt shell. To invoke Grunt shell, run the pig command. Once the Grunt mode executes, we can provide Pig Latin statements and command interactively at the command line.
- **Batch Mode** - In this mode, we can run a script file having a .pig extension. These files contain Pig Latin commands.
- **Embedded Mode** - In this mode, we can define our own functions. These functions can be called as UDF (User Defined Functions). Here, we use programming languages like Java and Python.

Pig Latin

The Pig Latin is a data flow language used by Apache Pig to analyze the data in Hadoop. It is a textual language that abstracts the programming from the Java MapReduce idiom into a notation.

Pig Latin Statements

The Pig Latin statements are used to process the data. It is an operator that accepts a relation as an input and generates another relation as an output.

- It can span multiple lines.
- Each statement must end with a semi-colon.
- It may include expression and schemas.
- By default, these statements are processed using multi-query execution

Pig Latin Conventions

Convention	Description
( )	The parenthesis can enclose one or more items. It can also be used to indicate the tuple data type. Example - (10, xyz, (3,6,9))
[ ]	The straight brackets can enclose one or more items. It can also be used to indicate the map data type. Example - [INNER | OUTER]
{ }	The curly brackets enclose two or more items. It can also be used to indicate the bag data type Example - { block | nested_block }
...	The horizontal ellipsis points indicate that you can repeat a portion of the code. Example - cat path [path ...]

Pig Example

Use case: Using Pig find the most occurred start letter.

Solution:

Case 1: Load the data into bag named "lines". The entire line is stuck to element line of type character array.

```
1. grunt> lines  = LOAD "/user/Desktop/data.txt" AS (line: chararray);
```

Case 2: The text in the bag lines needs to be tokenized this produces one word per row.

```
1. grunt>tokens = FOREACH lines GENERATE flatten(TOKENIZE(line))   As token:
    chararray;
```

Case 3: To retain the first letter of each word type the below command .This commands uses substring method to take the first character.

```
1. grunt>letters = FOREACH tokens  GENERATE SUBSTRING(0,1)   as letter : cha
   rarray;
```

Case 4: Create a bag for unique character where the grouped bag will contain the same character for each occurrence of that character.

```
1. grunt>lettergrp = GROUP letters by letter;
```

Case 5: The number of occurrence is counted in each group.

```
1. grunt>countletter  = FOREACH  lettergrp  GENERATE group  , COUNT(letters)
   ;
```

Case 6: Arrange the output according to count in descending order using the commands below.

```
1. grunt>OrderCnt = ORDER countletter  BY  $1  DESC;
```

Case 7: Limit to One to give the result.

```
1. grunt> result  =LIMIT    OrderCnt    1;
```

Case 8: Store the result in HDFS . The result is saved in output directory under sonoo folder.

```
1. grunt> STORE   result   into 'home/sonoo/output';
```