***Apache Hadoop Tutorial based Research***

*(Research in association with Sahu Technologies by Harshil Patel(TY KJSCE Student))*



**PREFACE**

This word document explains the various aspects of Apache Hadoop such as installation, how it works, uses and much more. It is presented carefully in this document and makes use of images of installation and other processes to help the viewer get a better understanding.

**ACKNOWLEDGEMENTS**

This research would like to acknowledge and thank Mr. Jitendra Sahu of Sahu Technologies for acting as a mentor and helping us out in creating this tutorial for Apache Hadoop.

The **Apache™ Hadoop®** project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

**CONTENTS OF THIS TUTORIAL**

1. **Hadoop – Home**
2. **Hadoop Overview, Ecosystem and Version**
3. **Big Data Overview**
4. **Big Data Solutions**
5. **Hadoop - Introduction**
6. **Environment Setup**
7. **HDFS Overview**
8. **HDFS Operations**
9. **Command Reference**
10. **MapReduce**
11. **Streaming**
12. **Multi-Node Cluster**
13. **Hadoop Useful Resources and some applications**
14. **Hadoop - Questions and Answers**
15. **When to use Hadoop and when not to use it?**
16. **Hadoop and CERN case study**

**1.Hadoop – Home**

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

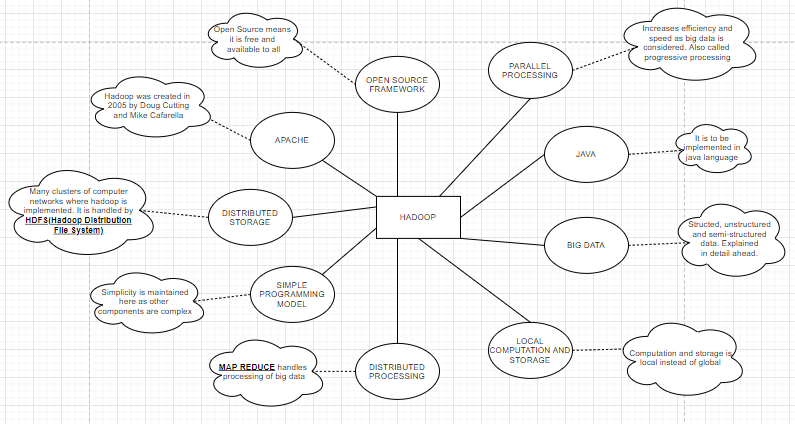
The latest version of Hadoop is the 2.10.0. It is used by a variety of companies and organisations for both research and production purposes.

**Modules**

* **Hadoop Common**: The common utilities that support the other Hadoop modules.
* **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
* **Hadoop YARN**: A framework for job scheduling and cluster resource management.
* **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.
* [**Hadoop Ozone**](https://hadoop.apache.org/ozone/): An object store for Hadoop.

**2.Hadoop Overview and Ecosystem**

Here is a look at some of the prominent features of Apache Hadoop:



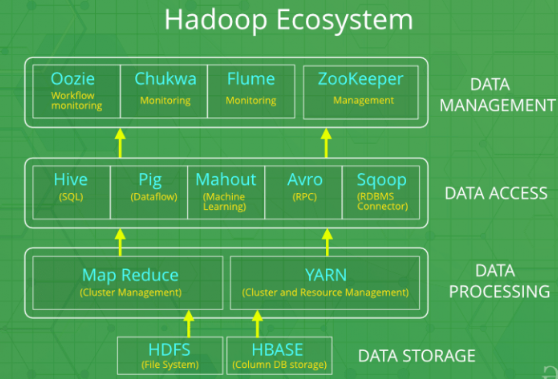
These features are explained in detail with help of table.

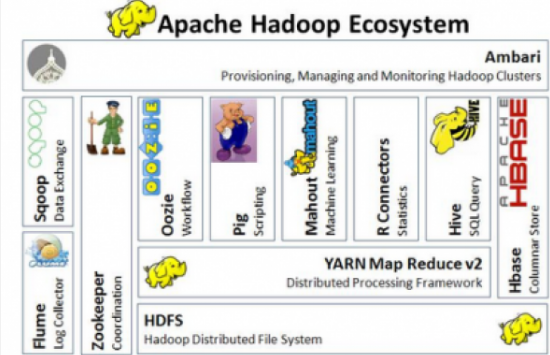
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  | This table helps us understand how Hadoop works and how various modules are integrated together to make it work in an efficient manner. |

**Hadoop Ecosystem**

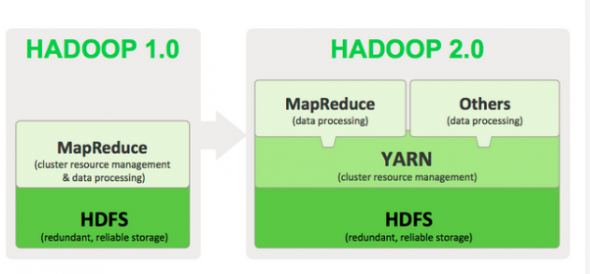
*Hadoop Ecosystem*is a platform or a suite which provides various services to solve the big data problems. It includes Apache projects and various commercial tools and solutions. There are *four major elements of Hadoop* i.e. **HDFS, MapReduce, YARN, and Hadoop Common**. Most of the tools or solutions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

* **HDFS:**Hadoop Distributed File System
* **YARN:** Yet Another Resource Negotiator
* **MapReduce:** Programming based Data Processing
* **Spark:** In-Memory data processing
* **PIG, HIVE:** Query based processing of data services
* **HBase:**NoSQL Database
* **Mahout, Spark MLLib:** [Machine Learning](https://www.geeksforgeeks.org/machine-learning/)algorithm libraries
* **Solar, Lucene:** Searching and Indexing
* **Zookeeper:** Managing cluster
* **Oozie:** Job Scheduling





**Versions of Hadoop**



**3.Big Data Overview**

**What is Big Data?**

Big data is a collection of large datasets that cannot be processed using traditional computing techniques. It is not a single technique or a tool, rather it has become a complete subject, which involves various tools, techniques and frameworks.

Due to the advent of new technologies, devices, and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. The amount of data produced by us from the beginning of time till 2003 was 5 billion gigabytes. If you pile up the data in the form of disks it may fill an entire football field. The same amount was created in every two days in 2011, and in every ten minutes in 2013. This rate is still growing enormously. Though all this information produced is meaningful and can be useful when processed, it is being neglected.

Big Data includes huge volume, high velocity, and extensible variety of data. The data in it will be of three types:

* **Structured data** − Relational data.
* **Semi Structured data** − XML data.
* **Unstructured data** − Word, PDF, Text, Media Logs.

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data:

* **Black Box Data** − It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
* **Social Media Data** − Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
* **Stock Exchange Data** − The stock exchange data holds information about the ‘buy’ and ‘sell’ decisions made on a share of different companies made by the customers.
* **Power Grid Data** − The power grid data holds information consumed by a particular node with respect to a base station.
* **Transport Data** − Transport data includes model, capacity, distance and availability of a vehicle.
* **Search Engine Data** − Search engines retrieve lots of data from different databases.

**Benefits of Big Data**

* Using the information kept in the social network like Facebook, the marketing agencies are learning about the response for their campaigns, promotions, and other advertising mediums.
* Using the information in the social media like preferences and product perception of their consumers, product companies and retail organizations are planning their production.
* Using the data regarding the previous medical history of patients, hospitals are providing better and quick service.

**Big Data Technologies**

Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business.

To harness the power of big data, you would require an infrastructure that can manage and process huge volumes of structured and unstructured data in real-time and can protect data privacy and security.

There are various technologies in the market from different vendors including Amazon, IBM, Microsoft, etc., to handle big data. While looking into the technologies that handle big data, we examine the following two classes of technology −

**Operational Big Data**

This include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored. NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement.

Some NoSQL systems can provide insights into patterns and trends based on real-time data with minimal coding and without the need for data scientists and additional infrastructure.

**Analytical Big Data**

These includes systems like Massively Parallel Processing (MPP) database systems and MapReduce that provide analytical capabilities for retrospective and complex analysis that may touch most or all of the data. MapReduce provides a new method of analysing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low-end machines.

These two classes of technology are complementary and frequently deployed together.

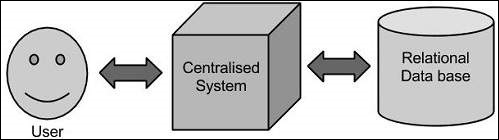
**Big Data Challenges**

* Capturing data
* Curation
* Storage
* Searching
* Sharing
* Transfer
* Analysis
* Presentation

**4.BIG DATA SOLUTIONS**

**Traditional Approach**

In this approach, an enterprise will have a computer to store and process big data. For storage purpose, the programmers will take the help of their choice of database vendors such as Oracle, IBM, etc. In this approach, the user interacts with the application, which in turn handles the part of data storage and analysis.

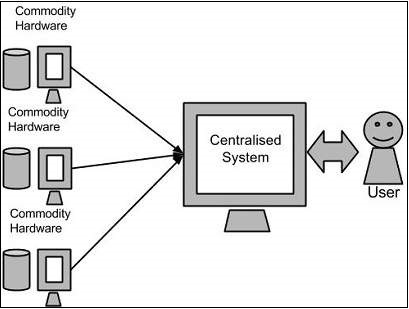


**Limitation**

This approach works fine with those applications that process less voluminous data that can be accommodated by standard database servers, or up to the limit of the processor that is processing the data. But when it comes to dealing with huge amounts of scalable data, it is a hectic task to process such data through a single database bottleneck.

**Google’s Solution**

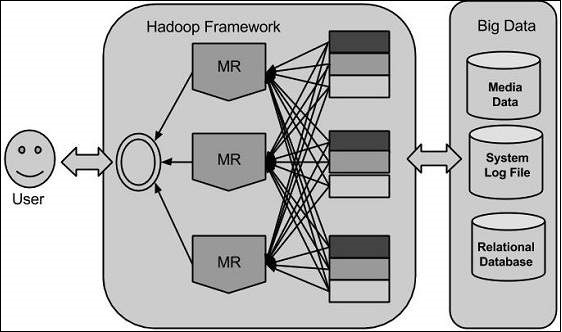
Google solved this problem using an algorithm called MapReduce. This algorithm divides the task into small parts and assigns them to many computers, and collects the results from them which when integrated, form the result dataset.



**Hadoop**

Using the solution provided by Google, **Doug Cutting** and his team developed an Open Source Project called***HADOOP***.

Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel with others. In short, Hadoop is used to develop applications that could perform complete statistical analysis on huge amounts of data.



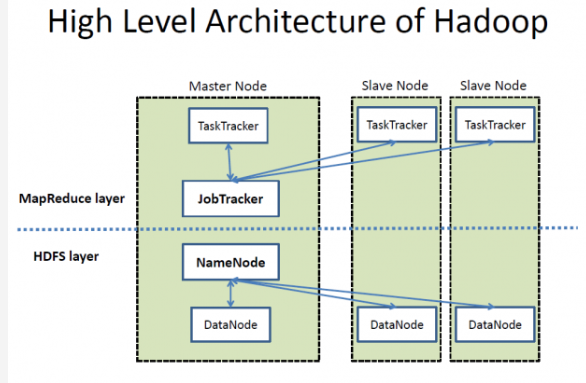
**5.Hadoop Introduction**

**Hadoop Architecture**

At its core, Hadoop has two major layers namely −

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





**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.

**6.Environmental Setup**

**Visit** <https://hadoop.apache.org/>  **for download and get started or follow the steps shown below.**

Hadoop is supported by GNU/Linux platform and its flavours. Therefore, we have to install a Linux operating system for setting up Hadoop environment. In case you have an OS other than Linux, you can install a Virtual box software in it and have Linux inside the Virtual box.



**Pre-installation Setup**

Before installing Hadoop into the Linux environment, we need to set up Linux using **ssh** (Secure Shell).

Follow the steps given below for setting up the Linux environment:

**Creating a User**

At the beginning, it is recommended to create a separate user for Hadoop to isolate Hadoop file system from Unix file system.

Follow the steps given below to create a user:

* Open the root using the command “su”.
* Create a user from the root account using the command “useradd username”.
* Now you can open an existing user account using the command “su username”.

Open the Linux terminal and type the following commands to create a user.

$ su

password:

# useradd hadoop

# passwd hadoop

New passwd:

Retype new passwd

**SSH Setup and Key Generation**

SSH setup is required to do different operations on a cluster such as starting, stopping, distributed daemon shell operations. To authenticate different users of Hadoop, it is required to provide public/private key pair for a Hadoop user and share it with different users.

The following commands are used for generating a key value pair using SSH. Copy the public keys form id\_rsa.pub to authorized\_keys, and provide the owner with read and write permissions to authorized\_keys file respectively.

$ ssh-keygen -t rsa

$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

$ chmod 0600 ~/.ssh/authorized\_keys

**Installing Java**

Java is the main prerequisite for Hadoop. First of all, you should verify the existence of java in your system using the command “java -version”. The syntax of java version command is *$ java -version*

|  |
| --- |
| Input: $ java -version |
| Output:  java version "1.7.0\_71"  Java(TM) SE Runtime Environment (build 1.7.0\_71-b13)  Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode) |

If java is not installed then download:

**jdk-7u71-linux-x64.tar.gz** from oracle.com



**Downloading Hadoop**

Download and extract Hadoop from Apache software foundation using the following commands:

$ su

password:

# cd /usr/local

# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/

hadoop-2.4.1.tar.gz

# tar xzf hadoop-2.4.1.tar.gz

# mv hadoop-2.4.1/\* to hadoop/

# exit

**Hadoop Operation Modes**

Once you have downloaded Hadoop, you can operate your Hadoop cluster in one of the three supported modes −

* **Local/Standalone Mode** − After downloading Hadoop in your system, by default, it is configured in a standalone mode and can be run as a single java process.
* **Pseudo Distributed Mode** − It is a distributed simulation on single machine. Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate java process. This mode is useful for development.
* **Fully Distributed Mode** − This mode is fully distributed with minimum two or more machines as a cluster. We will come across this mode in detail in the coming chapters.

**(A)Installing Hadoop in Standalone Mode**

There are no daemons running and everything runs in a single JVM. Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.

**Setting Up Hadoop**

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

*export HADOOP\_HOME=/usr/local/hadoop*

Check version using *$ Hadoop version*

By default, Hadoop is configured to run in a non-distributed mode on a single machine.

**(B)Installing Hadoop in Pseudo Distributed Mode**

**Step 1 − Setting Up Hadoop**

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

export HADOOP\_HOME=/usr/local/hadoop

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin

export HADOOP\_INSTALL=$HADOOP\_HOME

**Step 2 − Hadoop Configuration**

You can find all the Hadoop configuration files in the location “$HADOOP\_HOME/etc/hadoop”. It is required to make changes in those configuration files according to your Hadoop infrastructure.

In order to develop Hadoop programs in java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA\_HOME** value with the location of java in your system.

The following are the list of files that you have to edit to configure Hadoop:

**1.core-site.xml**

**2.hdfs-site.xml**

**3.yarn-site.xml**

**4.mapred-site.xml**

These changes are explained further in the table:

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

**Verifying Hadoop Installation**

The following steps are used to verify the Hadoop installation:

**Step 1 − Name Node Setup**

Set up the namenode using the command “hdfs namenode -format” as follows.

|  |
| --- |
| Input:  $ cd ~  $ hdfs namenode -format |
| Output:  10/24/14 21:30:55 INFO namenode.NameNode: STARTUP\_MSG:  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  STARTUP\_MSG: Starting NameNode  STARTUP\_MSG: host = localhost/192.168.1.11  STARTUP\_MSG: args = [-format]  STARTUP\_MSG: version = 2.4.1  ...  ...  10/24/14 21:30:56 INFO common.Storage: Storage directory  /home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.  10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to  retain 1 images with txid >= 0  10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0  10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN\_MSG:  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SHUTDOWN\_MSG: Shutting down NameNode at localhost/192.168.1.11  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**Step 2 − Verifying Hadoop dfs**

|  |
| --- |
| Input:  $ start-dfs.sh |
| Output:  10/24/14 21:37:56  Starting namenodes on [localhost]  localhost: starting namenode, logging to /home/hadoop/hadoop  2.4.1/logs/hadoop-hadoop-namenode-localhost.out  localhost: starting datanode, logging to /home/hadoop/hadoop  2.4.1/logs/hadoop-hadoop-datanode-localhost.out  Starting secondary namenodes [0.0.0.0] |

**Step 3 − Verifying Yarn Script**

|  |
| --- |
| Input:  $ start-yarn.sh |
| Output:  starting yarn daemons  starting resourcemanager, logging to /home/hadoop/hadoop  2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out  localhost: starting nodemanager, logging to /home/hadoop/hadoop  2.4.1/logs/yarn-hadoop-nodemanager-localhost.out |

**Step 4 − Accessing Hadoop on Browser**

|  |  |
| --- | --- |
| Input: http://localhost:50070/ | Output: |

**Step 5 − Verify All Applications for Cluster**

|  |  |
| --- | --- |
| Input: http://localhost:8088/ | Output: |

**7.HDFS Overview**

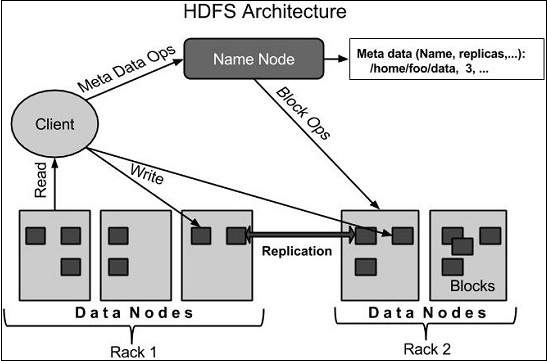
Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

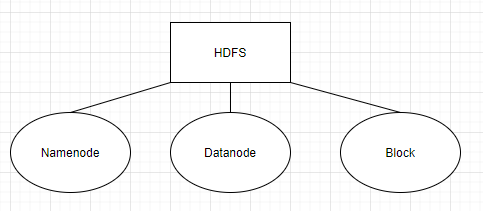
**Features of HDFS**

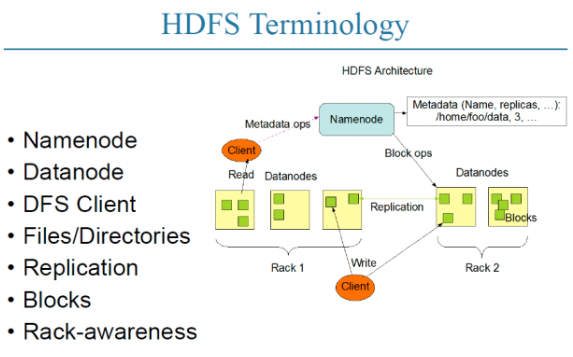
* It is suitable for the distributed storage and processing.
* Hadoop provides a command interface to interact with HDFS.
* The built-in servers of namenode and datanode help users to easily check the status of cluster.
* Streaming access to file system data.
* HDFS provides file permissions and authentication.

**HDFS Architecture**



HDFS follows the master-slave architecture and it has the following elements:





**1.Namenode**

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

* Manages the file system namespace.
* Regulates client’s access to files.
* It also executes file system operations such as renaming, closing, and opening files and directories.

**2.Datanode**

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

* Datanodes perform read-write operations on the file systems, as per client request.
* They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

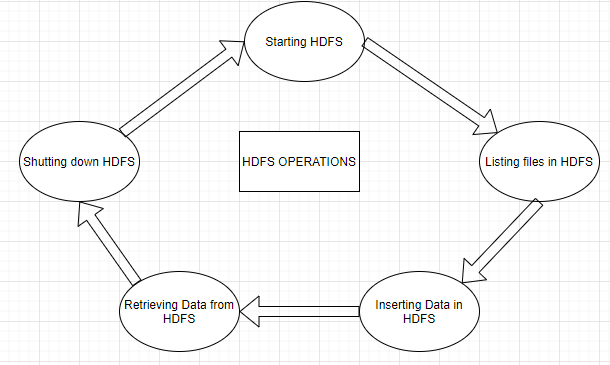
**3.Block**

Generally, the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

**Goals of HDFS**

1. **Fault detection and recovery** − Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore, HDFS should have mechanisms for quick and automatic fault detection and recovery.
2. **Huge datasets** − HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
3. **Hardware at data** − A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

**8.HDFS Operations**



**1.Starting HDFS**

Initially you have to format the configured HDFS file system, open namenode (HDFS server), and execute the following command:

$ adoop namenode -format

After formatting the HDFS, start the distributed file system. The following command will start the namenode as well as the data nodes as cluster.

$ start-dfs.sh

**2.Listing Files in HDFS**

After loading the information in the server, we can find the list of files in a directory, status of a file, using **‘ls’**. Given below is the syntax of **ls** that you can pass to a directory or a filename as an argument.

$ $HADOOP\_HOME/bin/hadoop fs -ls <args>

**3.Inserting Data into HDFS**

Assume we have data in the file called file.txt in the local system which is ought to be saved in the hdfs file system

|  |
| --- |
| Step1: You have to create an input directory  $ $HADOOP\_HOME/bin/hadoop fs -mkdir /user/input |
| Step2: Transfer and store a data file from local systems to the Hadoop file system using the put command.  $ $HADOOP\_HOME/bin/hadoop fs -put /home/file.txt /user/input |
| Step3: You can verify the file using ls command.  $ $HADOOP\_HOME/bin/hadoop fs -ls /user/input |

**4.Retreiving Data from HDFS**

Assume we have a file in HDFS called **outfile**. Given below is a simple demonstration for retrieving the required file from the Hadoop file system.

|  |
| --- |
| Step1: Initially, view the data from HDFS using **cat** command  $ $HADOOP\_HOME/bin/hadoop fs -cat /user/output/outfile |
| Step2: Get the file from HDFS to the local file system using **get** command.  $ $HADOOP\_HOME/bin/hadoop fs -get /user/output/ /home/hadoop\_tp/ |

**5.Shutting down the HDFS**

This can be done using the following command:

$ stop-dfs.sh

**9.Command References**

There are many more commands in **"$HADOOP\_HOME/bin/hadoop fs"** than are demonstrated here, although these basic operations will get you started. Running ./bin/hadoop dfs with no additional arguments will list all the commands that can be run with the FsShell system. **$HADOOP\_HOME/bin/hadoop fs -help** can alsobe used.

|  |  |
| --- | --- |
| **Sr.No** | **Command & Description** |
| 1 | **-ls <path>**  Lists the contents of the directory specified by path, showing the names, permissions, owner, size and modification date for each entry. |
| 2 | **-lsr <path>**  Behaves like -ls, but recursively displays entries in all subdirectories of path. |
| 3 | **-du <path>**  Shows disk usage, in bytes, for all the files which match path; filenames are reported with the full HDFS protocol prefix. |
| 4 | **-dus <path>**  Like -du, but prints a summary of disk usage of all files/directories in the path. |
| 5 | **-mv <src><dest>**  Moves the file or directory indicated by src to dest, within HDFS. |
| 6 | **-cp <src> <dest>**  Copies the file or directory identified by src to dest, within HDFS. |
| 7 | **-rm <path>**  Removes the file or empty directory identified by path. |
| 8 | **-rmr <path>**  Removes the file or directory identified by path. Recursively deletes any child entries (i.e., files or subdirectories of path). |
| 9 | **-put <localSrc> <dest>**  Copies the file or directory from the local file system identified by localSrc to dest within the DFS. |
| 10 | **-copyFromLocal <localSrc> <dest>**  Identical to -put |
| 11 | **-moveFromLocal <localSrc> <dest>**  Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success. |
| 12 | **-get [-crc] <src> <localDest>**  Copies the file or directory in HDFS identified by src to the local file system path identified by localDest. |
| 13 | **-getmerge <src> <localDest>**  Retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest. |
| 14 | **-cat <filen-ame>**  Displays the contents of filename on stdout. |
| 15 | **-copyToLocal <src> <localDest>**  Identical to -get |
| 16 | **-moveToLocal <src> <localDest>**  Works like -get, but deletes the HDFS copy on success. |
| 17 | **-mkdir <path>**  Creates a directory named path in HDFS.  Creates any parent directories in path that are missing (e.g., mkdir -p in Linux). |
| 18 | **-setrep [-R] [-w] rep <path>**  Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time) |
| 19 | **-touchz <path>**  Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0. |
| 20 | **-test -[ezd] <path>**  Returns 1 if path exists; has zero length; or is a directory or 0 otherwise. |
| 21 | **-stat [format] <path>**  Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y). |
| 22 | **-tail [-f] <file2name>**  Shows the last 1KB of file on stdout. |
| 23 | **-chmod [-R] mode,mode,... <path>...**  Changes the file permissions associated with one or more objects identified by path.... Performs changes recursively with R. mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask. |
| 24 | **-chown [-R] [owner][:[group]] <path>...**  Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified. |
| 25 | **-chgrp [-R] group <path>...**  Sets the owning group for files or directories identified by path.... Sets group recursively if -R is specified. |
| 26 | **-help <cmd-name>**  Returns usage information for one of the commands listed above. You must omit the leading '-' character in cmd. |

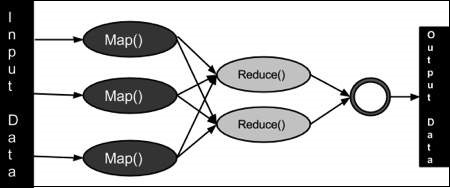
**10.Map Reduce**

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely **Map and Reduce**. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

**The Algorithm**

* Generally MapReduce paradigm is based on sending the computer to where the data resides!
* MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  + **Map stage** − The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  + **Reduce stage** − This stage is the combination of the **Shuffle**stage and the **Reduce** stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
* During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
* The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
* Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
* After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



**Inputs and Outputs (Java Perspective)**

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

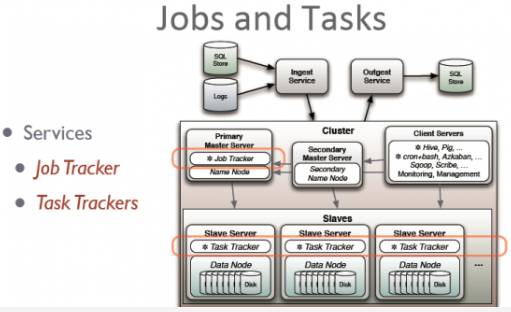
The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a **MapReduce job** − (Input) <k1, v1> → map → <k2, v2> → reduce → <k3, v3>(Output).

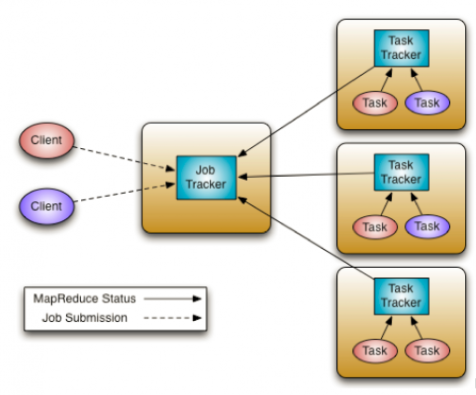
|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| **Map** | <k1, v1> | list (<k2, v2>) |
| **Reduce** | <k2, list(v2)> | list (<k3, v3>) |

**Terminology**

1. **PayLoad** − Applications implement the Map and the Reduce functions, and form the core of the job.
2. **Mapper** − Mapper maps the input key/value pairs to a set of intermediate key/value pair.
3. **NamedNode** − Node that manages the Hadoop Distributed File System (HDFS).
4. **DataNode** − Node where data is presented in advance before any processing takes place.
5. **MasterNode** − Node where JobTracker runs and which accepts job requests from clients.
6. **SlaveNode** − Node where Map and Reduce program runs.
7. **JobTracker** − Schedules jobs and tracks the assign jobs to Task tracker.
8. **Task Tracker** − Tracks the task and reports status to JobTracker.
9. **Job** − A program is an execution of a Mapper and Reducer across a dataset.
10. **Task** − An execution of a Mapper or a Reducer on a slice of data.
11. **Task Attempt** − A particular instance of an attempt to execute a task on a SlaveNode.

**About Job Tracker (Point 7) and Task Tracker(Point 10)**





Above the file systems comes the MapReduce engine, which consists of one JobTracker, to which client applications submit MapReduce jobs. The JobTracker pushes work out to available TaskTracker nodes in the cluster, striving to keep the work as close to the data as possible.

With a rack-aware file system, the JobTracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network.

If a TaskTracker fails or times out, that part of the job is rescheduled. The TaskTracker on each node spawns off a separate Java Virtual Machine process to prevent the TaskTracker itself from failing if the running job crashes the JVM. A heartbeat is sent from the TaskTracker to the JobTracker every few minutes to check its status. The Job Tracker and TaskTracker status and information is exposed by Jetty and can be viewed from a web browser.

**Important Commands**

All Hadoop commands are invoked by the **$HADOOP\_HOME/bin/hadoop** command. Running the Hadoop script without any arguments prints the description for all commands.

**Usage** − hadoop [--config confdir] COMMAND

The following table lists the options available and their description.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **namenode -format**  Formats the DFS filesystem. |
| 2 | **secondarynamenode**  Runs the DFS secondary namenode. |
| 3 | **namenode**  Runs the DFS namenode. |
| 4 | **datanode**  Runs a DFS datanode. |
| 5 | **dfsadmin**  Runs a DFS admin client. |
| 6 | **mradmin**  Runs a Map-Reduce admin client. |
| 7 | **fsck**  Runs a DFS filesystem checking utility. |
| 8 | **fs**  Runs a generic filesystem user client. |
| 9 | **balancer**  Runs a cluster balancing utility. |
| 10 | **oiv**  Applies the offline fsimage viewer to an fsimage. |
| 11 | **fetchdt**  Fetches a delegation token from the NameNode. |
| 12 | **jobtracker**  Runs the MapReduce job Tracker node. |
| 13 | **pipes**  Runs a Pipes job. |
| 14 | **tasktracker**  Runs a MapReduce task Tracker node. |
| 15 | **historyserver**  Runs job history servers as a standalone daemon. |
| 16 | **job**  Manipulates the MapReduce jobs. |
| 17 | **queue**  Gets information regarding JobQueues. |
| 18 | **version**  Prints the version. |
| 19 | **jar <jar>**  Runs a jar file. |
| 20 | **distcp <srcurl> <desturl>**  Copies file or directories recursively. |
| 21 | **distcp2 <srcurl> <desturl>**  DistCp version 2. |
| 22 | **archive -archiveName NAME -p <parent path> <src>\* <dest>**  Creates a hadoop archive. |
| 23 | **classpath**  Prints the class path needed to get the Hadoop jar and the required libraries. |
| 24 | **daemonlog**  Get/Set the log level for each daemon |

**How to Interact with MapReduce Jobs**

Usage − hadoop job [GENERIC\_OPTIONS]

The following are the Generic Options available in a Hadoop job.

|  |  |
| --- | --- |
| **Sr.No.** | **GENERIC\_OPTION & Description** |
| 1 | **-submit <job-file>**  Submits the job. |
| 2 | **-status <job-id>**  Prints the map and reduce completion percentage and all job counters. |
| 3 | **-counter <job-id> <group-name> <countername>**  Prints the counter value. |
| 4 | **-kill <job-id>**  Kills the job. |
| 5 | **-events <job-id> <fromevent-#> <#-of-events>**  Prints the events' details received by jobtracker for the given range. |
| 6 | **-history [all] <jobOutputDir> - history < jobOutputDir>**  Prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option. |
| 7 | **-list[all]**  Displays all jobs. -list displays only jobs which are yet to complete. |
| 8 | **-kill-task <task-id>**  Kills the task. Killed tasks are NOT counted against failed attempts. |
| 9 | **-fail-task <task-id>**  Fails the task. Failed tasks are counted against failed attempts. |
| 10 | **-set-priority <job-id> <priority>**  Changes the priority of the job. Allowed priority values are VERY\_HIGH, HIGH, NORMAL, LOW, VERY\_LOW |

**11.Hadoop Streaming**

Hadoop streaming is a utility that comes with the Hadoop distribution. This utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

**How Streaming Works**

In the above example, both the mapper and the reducer are python scripts that read the input from standard input and emit the output to standard output. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

When a script is specified for mappers, each mapper task will launch the script as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the standard input (STDIN) of the process. In the meantime, the mapper collects the line-oriented outputs from the standard output (STDOUT) of the process and converts each line into a key/value pair, which is collected as the output of the mapper. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value. If there is no tab character in the line, then the entire line is considered as the key and the value is null. However, this can be customized, as per one need.

When a script is specified for reducers, each reducer task will launch the script as a separate process, then the reducer is initialized. As the reducer task runs, it converts its input key/values pairs into lines and feeds the lines to the standard input (STDIN) of the process. In the meantime, the reducer collects the line-oriented outputs from the standard output (STDOUT) of the process, converts each line into a key/value pair, which is collected as the output of the reducer. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) is the value. However, this can be customized as per specific requirements.

**Important Commands**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Options** | **Description** |
| -input directory/file-name | Required | Input location for mapper. |
| -output directory-name | Required | Output location for reducer. |
| -mapper executable or script or JavaClassName | Required | Mapper executable. |
| -reducer executable or script or JavaClassName | Required | Reducer executable. |
| -file file-name | Optional | Makes the mapper, reducer, or combiner executable available locally on the compute nodes. |
| -inputformat JavaClassName | Optional | Class you supply should return key/value pairs of Text class. If not specified, TextInputFormat is used as the default. |
| -outputformat JavaClassName | Optional | Class you supply should take key/value pairs of Text class. If not specified, TextOutputformat is used as the default. |
| -partitioner JavaClassName | Optional | Class that determines which reduce a key is sent to. |
| -combiner streamingCommand or JavaClassName | Optional | Combiner executable for map output. |
| -cmdenv name=value | Optional | Passes the environment variable to streaming commands. |
| -inputreader | Optional | For backwards-compatibility: specifies a record reader class (instead of an input format class). |
| -verbose | Optional | Verbose output. |
| -lazyOutput | Optional | Creates output lazily. For example, if the output format is based on FileOutputFormat, the output file is created only on the first call to output.collect (or Context.write). |
| -numReduceTasks | Optional | Specifies the number of reducers. |
| -mapdebug | Optional | Script to call when map task fails. |
| -reducedebug | Optional | Script to call when reduce task fails. |

**12.Hadoop-Multinode Clusters**

Visit Hadoop.apache.org for more details.

**13.Hadoop Useful Resources**

1. <Hadoop>- Introduction to Apache Hadoop. (<https://hadoop.apache.org/>)
2. [Hadoop Wikipedia](https://en.wikipedia.org/wiki/Apache_Hadoop)- A Complete Wikipedia of Hadoop. (<https://en.wikipedia.org/wiki/Apache_Hadoop>)
3. [Hadoop Basics](Hadoop%20Basics)- A Basics of Hadoop. (<https://www.ibm.com/products/software>)
4. Hadoop has many applications, for example, check out <https://www.engpaper.com/hadoop-2018.htm> which shows that hadoop is used for developing stock market prediction system, architectural design, analysing big data health records, parallel image edge detection algorithm, 3D face image identification from video streaming using map reduce, etc.

**14.Hadoop Questions and Answers**

Questions and Answers are available on the internet along with Interview questions, mock tests and online quizzes based on Hadoop.

**15.When and when not to use Hadoop**

Hadoop is used for:

* *Search* – Yahoo, Amazon, Zvents
* *Log processing* – Facebook, Yahoo
* *Data Warehouse* – Facebook, AOL
* *Video and Image Analysis* – New York Times, Eyealike

Scenarios where Hadoop cannot be used:

* *Low Latency data access* : Quick access to small parts of data
* *Multiple data modification* : Hadoop is a better fit only if we are primarily concerned about reading data and not modifying data.
* *Lots of small files* : Hadoop is suitable for scenarios, where we have few but large files.

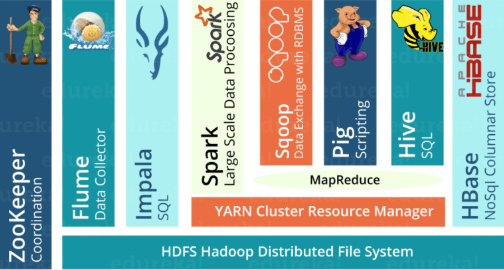
**16.Hadoop and CERN case study**

The *Large Hadron Collider* in Switzerland is one of the largest and most powerful machines in the world. It is equipped with around 150 million sensors, producing a petabyte of data every second, and the data is growing continuously.

CERN researches said that this data has been scaling up in terms of amount and complexity, and one of the important task is to serve these scalable requirements. So, they setup a Hadoop cluster. By using Hadoop, they limited their cost in hardware and complexity in maintenance.

***They integrated Oracle & Hadoop and they got advantages of integrating***.Oracle, optimized their Online Transactional System & Hadoop provided them scalable distributed data processing platform. They designed a hybrid system, and first they moved data from Oracle to Hadoop. Then, they executed query over Hadoop data from Oracle using Oracle APIs. They also used Hadoop data formats like *Avro* & *Parquet* for high performance analytics without need of changing the end-user apps connecting to Oracle.

The main Hadoop components they are using at the CERN-IT Hadoop service:



Based on CERN case study, we can conclude that:

* Hadoop is scalable and excellent for Big Data analytics
* Oracle is proven for concurrent transactional workloads
* Solutions are available to integrate Oracle and Hadoop
* There is a great value in using hybrid systems (Oracle + Hadoop):
  + Oracle APIs for legacy applications and OLTP workloads
  + Scalability on commodity Hardware for analytic workloads