Google File System:

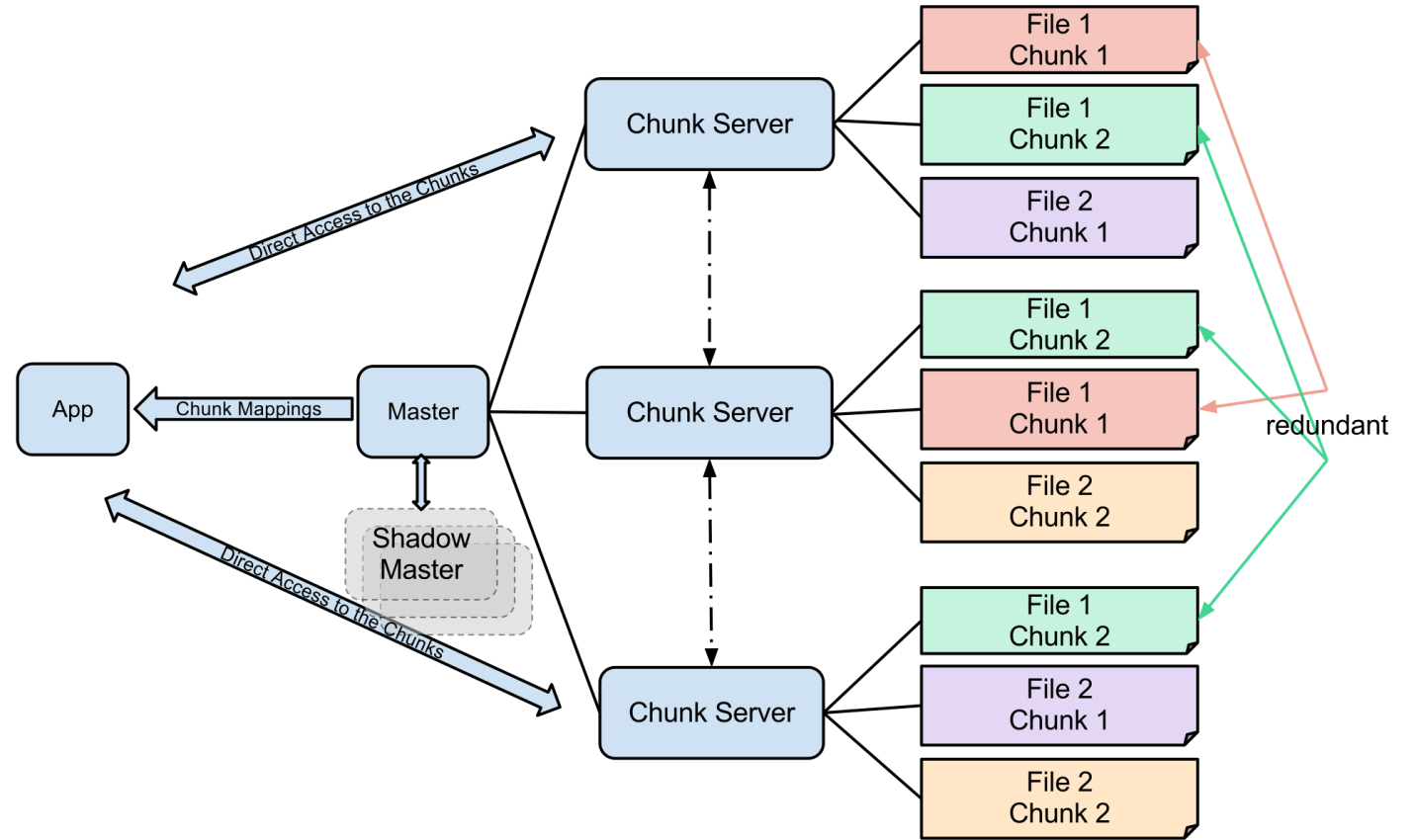
**Outline**

* Motivation
* Assumptions
* Architecture
* Implementation
* Current Status
* Measurements
* Benefits/Limitations
* Conclusion

**Motivation**

* Need for a scalable DFS
* Large distributed data-intensive applications
* High data processing needs
* Performance, Reliability, Scalability and Availability
* More than traditional DFS

Google File System (**GFS** or GoogleFS) is a proprietary **distributed file system** developed by Google for its own use. It is designed to provide efficient, reliable access to data using large clusters of commodity hardware. A new version of the Google File System is codenamed Colossus which was released in 2010.



HDFS

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.

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

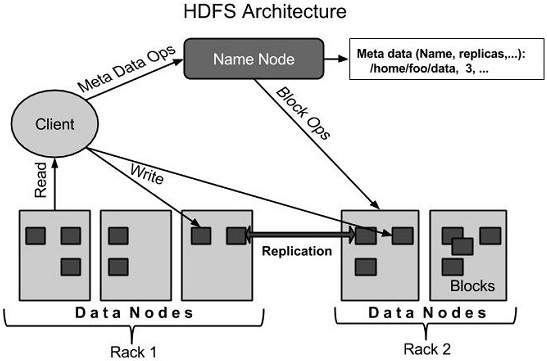
HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

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

Given below is the architecture of a Hadoop File System.



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

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.

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.

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

* **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.
* **Huge datasets** : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
* **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.
* Starting HDFS
* Initially you have to format the configured HDFS file system, open namenode (HDFS server), and execute the following command.
* $ hadoop 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
* 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>
* 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. Follow the steps given below to insert the required file in the Hadoop file system.
* Step 1
* You have to create an input directory.
* $ $HADOOP\_HOME/bin/hadoop fs -mkdir /user/input
* Step 2
* 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
* Step 3
* You can verify the file using ls command.
* $ $HADOOP\_HOME/bin/hadoop fs -ls /user/input
* Retrieving 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.
* Step 1
* Initially, view the data from HDFS using cat command.
* $ $HADOOP\_HOME/bin/hadoop fs -cat /user/output/outfile
* Step 2
* Get the file from HDFS to the local file system using get command.
* $ $HADOOP\_HOME/bin/hadoop fs -get /user/output/ /home/hadoop\_tp/
* Shutting Down the HDFS
* You can shut down the HDFS by using the following command.$ stop-dfs.sh

## Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

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 large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single**NameNode** that manages the file system metadata and one or more slave**DataNodes** that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

HDFS provides a shell like any other file system and a list of commands are available to interact with the file system. These shell commands will be covered in a separate chapter along with appropriate examples.

## How Does Hadoop Work?

### Stage 1

A user/application can submit a job to the Hadoop (a hadoop job client) for required process by specifying the following items:

1. The location of the input and output files in the distributed file system.
2. The java classes in the form of jar file containing the implementation of map and reduce functions.
3. The job configuration by setting different parameters specific to the job.

### Stage 2

The Hadoop job client then submits the job (jar/executable etc) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

### Stage 3

The TaskTrackers on different nodes execute the task as per MapReduce implementation and output of the reduce function is stored into the output files on the file system.

## 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 - Enviornment Setup

Hadoop is supported by GNU/Linux platform and its flavors. 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 Virtualbox software in it and have Linux inside the Virtualbox.

## 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 given below.

$ java -version

If everything is in order, it will give you the following 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 in your system, then follow the steps given below for installing java.

### Step 1

Download java (JDK <latest version> - X64.tar.gz) by visiting the following link[http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads1880260.html.](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads1880260.html)

Then jdk-7u71-linux-x64.tar.gz will be downloaded into your system.

### Step 2

Generally you will find the downloaded java file in Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

$ cd Downloads/

$ ls

jdk-7u71-linux-x64.gz

$ tar zxf jdk-7u71-linux-x64.gz

$ ls

jdk1.7.0\_71 jdk-7u71-linux-x64.gz

### Step 3

To make java available to all the users, you have to move it to the location “/usr/local/”. Open root, and type the following commands.

$ su

password:

# mv jdk1.7.0\_71 /usr/local/

# exit

### Step 4

For setting up PATH and JAVA\_HOME variables, add the following commands to ~/.bashrc file.

export JAVA\_HOME=/usr/local/jdk1.7.0\_71

export PATH=$PATH:$JAVA\_HOME/bin

Now apply all the changes into the current running system.

$ source ~/.bashrc

### Step 5

Use the following commands to configure java alternatives:

# alternatives --install /usr/bin/java java usr/local/java/bin/java 2

# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2

# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2

# alternatives --set java usr/local/java/bin/java

# alternatives --set javac usr/local/java/bin/javac

# alternatives --set jar usr/local/java/bin/jar

Now verify the java -version command from the terminal as explained above.

## Downloading Hadoop

Download and extract Hadoop 2.4.1 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.

## Installing Hadoop in Standalone Mode

Here we will discuss the installation of **Hadoop 2.4.1** 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

Before proceeding further, you need to make sure that Hadoop is working fine. Just issue the following command:

$ hadoop version

If everything is fine with your setup, then you should see the following result:

Hadoop 2.4.1

Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768

Compiled by hortonmu on 2013-10-07T06:28Z

Compiled with protoc 2.5.0

From source with checksum 79e53ce7994d1628b240f09af91e1af4

It means your Hadoop's standalone mode setup is working fine. By default, Hadoop is configured to run in a non-distributed mode on a single machine.

### Example

Let's check a simple example of Hadoop. Hadoop installation delivers the following example MapReduce jar file, which provides basic functionality of MapReduce and can be used for calculating, like Pi value, word counts in a given list of files, etc.

$HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar

Let's have an input directory where we will push a few files and our requirement is to count the total number of words in those files. To calculate the total number of words, we do not need to write our MapReduce, provided the .jar file contains the implementation for word count. You can try other examples using the same .jar file; just issue the following commands to check supported MapReduce functional programs by hadoop-mapreduce-examples-2.2.0.jar file.

$ hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar

### Step 1

Create temporary content files in the input directory. You can create this input directory anywhere you would like to work.

$ mkdir input

$ cp $HADOOP\_HOME/\*.txt input

$ ls -l input

It will give the following files in your input directory:

total 24

-rw-r--r-- 1 root root 15164 Feb 21 10:14 LICENSE.txt

-rw-r--r-- 1 root root 101 Feb 21 10:14 NOTICE.txt

-rw-r--r-- 1 root root 1366 Feb 21 10:14 README.txt

These files have been copied from the Hadoop installation home directory. For your experiment, you can have different and large sets of files.

### Step 2

Let's start the Hadoop process to count the total number of words in all the files available in the input directory, as follows:

$ hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar wordcount input output

### Step 3

Step-2 will do the required processing and save the output in output/part-r00000 file, which you can check by using:

$cat output/\*

It will list down all the words along with their total counts available in all the files available in the input directory.

"AS 4

"Contribution" 1

"Contributor" 1

"Derivative 1

"Legal 1

"License" 1

"License"); 1

"Licensor" 1

"NOTICE” 1

"Not 1

"Object" 1

"Source” 1

"Work” 1

"You" 1

"Your") 1

"[]" 1

"control" 1

"printed 1

"submitted" 1

(50%) 1

(BIS), 1

(C) 1

(Don't) 1

(ECCN) 1

(INCLUDING 2

(INCLUDING, 2

.............

## Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 2.4.1 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

Now apply all the changes into the current running system.

$ source ~/.bashrc

### 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.

$ cd $HADOOP\_HOME/etc/hadoop

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.

export JAVA\_HOME=/usr/local/jdk1.7.0\_71

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

**core-site.xml**

The **core-site.xml** file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and size of Read/Write buffers.

Open the core-site.xml and add the following properties in between <configuration>, </configuration> tags.

<configuration>

<property>

<name>fs.default.name </name>

<value> hdfs://localhost:9000 </value>

</property>

</configuration>

**hdfs-site.xml**

The **hdfs-site.xml** file contains information such as the value of replication data, namenode path, and datanode paths of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

dfs.replication (data replication value) = 1

(In the below given path /hadoop/ is the user name.

hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)

namenode path = //home/hadoop/hadoopinfra/hdfs/namenode

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)

datanode path = //home/hadoop/hadoopinfra/hdfs/datanode

Open this file and add the following properties in between the <configuration> </configuration> tags in this file.

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>

</property>

<property>

<name>dfs.data.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>

</property>

</configuration>

**Note:** In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

**yarn-site.xml**

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

**mapred-site.xml**

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, it is required to copy the file from **mapred-site,xml.template** to **mapred-site.xml** file using the following command.

$ cp mapred-site.xml.template mapred-site.xml

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration>tags in this file.

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

## 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.

$ cd ~

$ hdfs namenode -format

The expected result is as follows.

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

The following command is used to start dfs. Executing this command will start your Hadoop file system.

$ start-dfs.sh

The expected output is as follows:

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

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

$ start-yarn.sh

The expected output as follows:

starting yarn daemons

starting resourcemanager, logging to /home/hadoop/hadoop

2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out

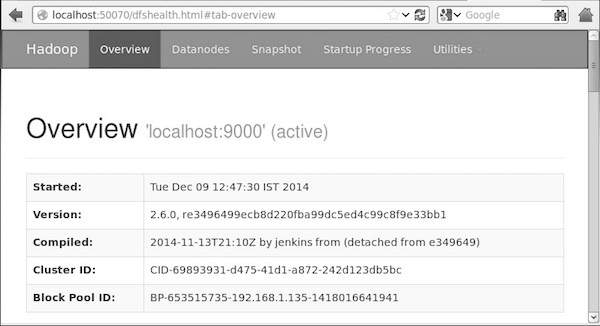
localhost: starting nodemanager, logging to /home/hadoop/hadoop

2.4.1/logs/yarn-hadoop-nodemanager-localhost.out

### Step 4: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following url to get Hadoop services on browser.

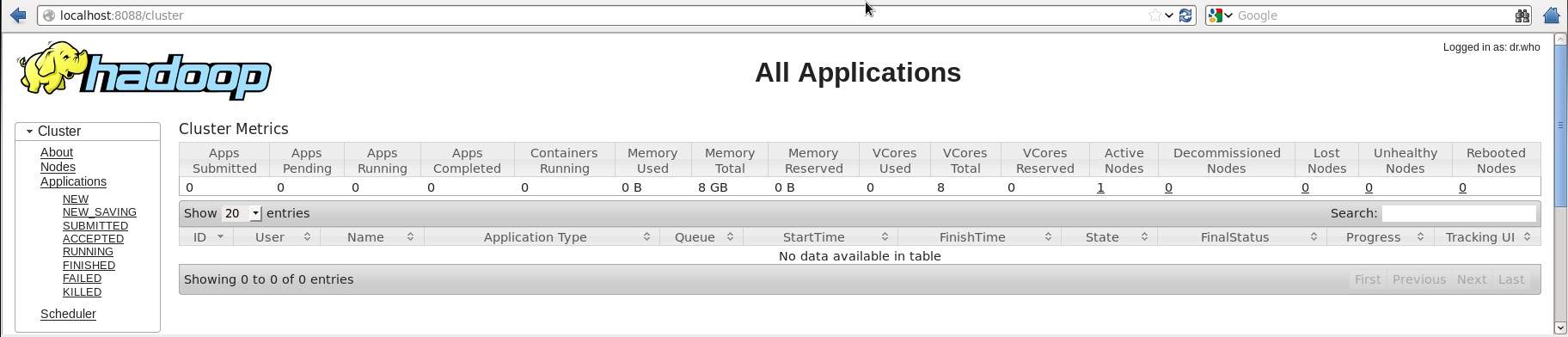
http://localhost:50070/



### Step 5: Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

http://localhost:8088/



# Hadoop - 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 faulttolerant and designed using low-cost hardware.

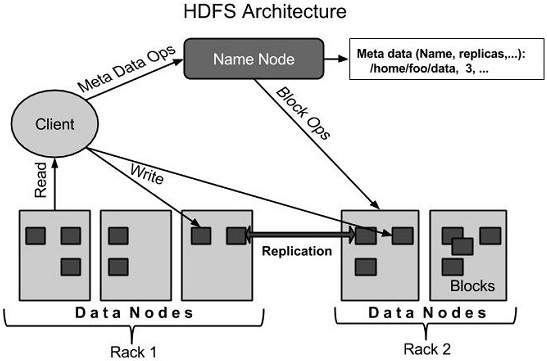
HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

## 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

Given below is the architecture of a Hadoop File System.



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

### 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.

### 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.

### 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

* **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.
* **Huge datasets** : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
* **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.