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.

## Assumptions and Goals

### Hardware Failure

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system’s data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

### Streaming Data Access

Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

### Large Data Sets

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

### Simple Coherency Model

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

### “Moving Computation is Cheaper than Moving Data”

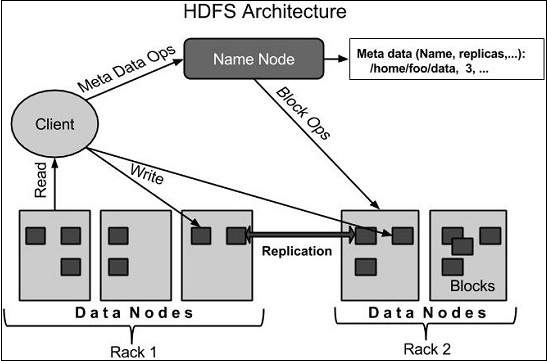
A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

### Portability Across Heterogeneous Hardware and Software Platforms

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

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.

## 

Hadoop, an open-source framework for distributed storage and processing of large data sets, comes with its own file system known as Hadoop Distributed File System (HDFS). HDFS is designed to store and manage vast amounts of data across multiple machines in a distributed and fault-tolerant manner. There are several interfaces and tools available for interacting with HDFS:

1. **Command-Line Interface (CLI):**
   * **hadoop fs**: The Hadoop command-line interface provides commands to interact with HDFS. For example:
     + **hadoop fs -ls**: List files and directories in HDFS.
     + **hadoop fs -mkdir**: Create a directory in HDFS.
     + **hadoop fs -copyToLocal**: Copy files from HDFS to the local file system.
     + **hadoop fs -copyFromLocal**: Copy files from the local file system to HDFS.
2. **Java API:**
   * Hadoop provides a Java API for developers to interact with HDFS programmatically. The key classes include **FileSystem** and **Path**. This API allows developers to read, write, and manipulate data stored in HDFS.
3. **Web UI:**
   * Hadoop comes with a web-based user interface that provides information about the Hadoop cluster and allows users to interact with HDFS through a web browser. The web UI is accessible by default at **http://<namenode>:50070**.
4. **Hadoop File System Shell:**
   * The Hadoop File System Shell (**hadoop fs -shell**) is an interactive shell that allows users to execute Hadoop file system commands interactively.
5. **REST API:**
   * Hadoop also provides a REST API for interacting with HDFS. This allows users to perform various operations using HTTP requests. The REST API can be useful for integration with other systems or for building custom applications.
6. **Apache Hadoop Archives (HAR):**
   * Hadoop Archives provide a way to bundle a large number of small files into a single archive file for more efficient storage and retrieval. Users can create and manipulate HAR files using the **har** command.
7. **Hadoop File System Shell (HDFS Shell):**
   * The HDFS Shell is another command-line interface specifically designed for interacting with HDFS. It includes commands similar to the standard Unix commands and provides additional commands for HDFS-specific operations.

These interfaces and tools provide different ways for users and developers to interact with HDFS, allowing them to manage, process, and analyze large datasets efficiently in a distributed environment.

Top of Form

## 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.