Hadoop Ecosystem Fundamentals of Distributed Systems

In the last lecture, when we learned about Data Platform Architecture, we saw the most widely used Data Lake i.e. HDFS. HDFS is not just any data lake, but it is widely used as a distributed file system using which many other distributed systems like Apache Hive, Apache Spark, etc are built. In this lecture, we are going to deep dive into HDFS architecture, its applications, and practical exposure.

What is Hadoop Distributed File System (HDFS)?

In the last lecture, we discussed a bit of inspiration from HDFS architecture how data is distributed in multiple nodes, and whenever we need this data or want to store the data, we just need to call the cluster to get/store the data

But this raises an important question: How does the system know where a particular piece of data is stored among the 100 nodes or how does it store the data? This is where the architecture of HDFS (Hadoop Distributed File System) comes into play.

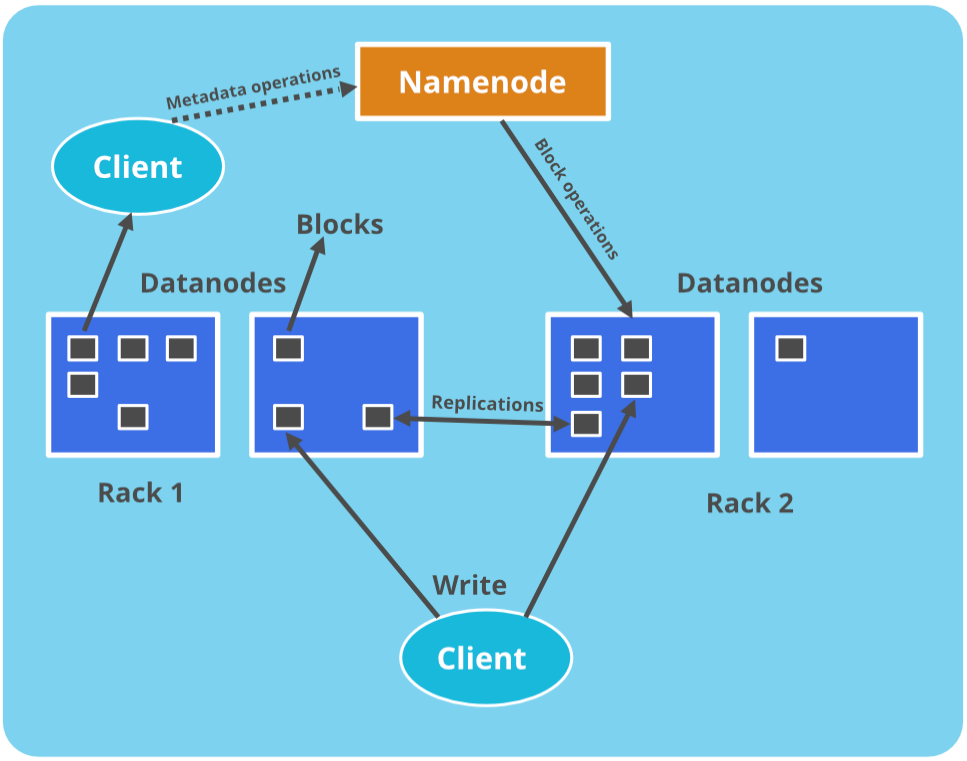
Understanding HDFS (Hadoop Distributed File System)

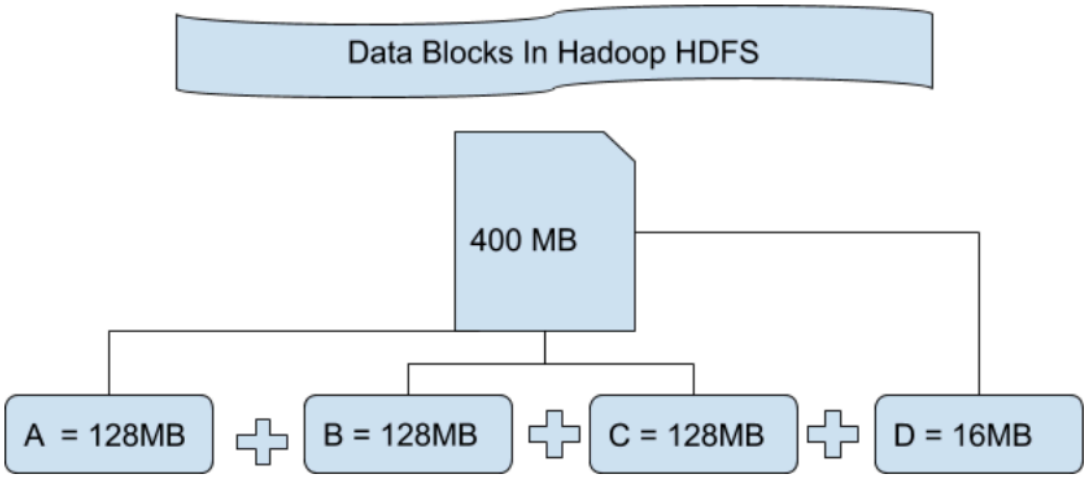
In HDFS, the system is designed to handle large amounts of data distributed across many nodes. To manage and retrieve data efficiently in such a distributed system, HDFS uses two types of nodes:

1. **NameNode**: This is the **master node** in the cluster. The NameNode is responsible for keeping a record of where every piece of data is stored across the cluster. Think of the NameNode as a librarian who knows exactly where every book (or piece of data) is located in a vast library (the cluster).
2. **DataNode**: These are the **worker nodes** that actually store the data. When data is written to the HDFS, the NameNode records which DataNode holds each piece of data. When you request data, the NameNode directs you to the appropriate DataNode to retrieve it.

Architecture of HDFS:

Hadoop 1.x Architecture





HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.

The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata.

Secondary NameNode:
When NameNode runs out of disk space, a secondary NameNode is activated to perform a checkpoint.

DataNode
Every slave machine that contains data organizes a DataNode.DataNodes do the following:

- DataNodes store every data.
- It handles all of the requested operations on files, such as reading file content and creating new data

what if in HDFS the master goes down? in that case how we are doing or how we are achieving high availability plus what is the algorithm that how these replication are being kept is it just a random order is there something behind it will focus in detail on that topic

To overcome this issue researchers come up with the approach that we have to have a high availability model by creating 2 name nodes. One will be in Active mode and one will be in Standby/Passive mode.

Hadoop 2.x Architecture

Problem: As you know in Hadoop 1.x architecture Name Node was a single point of failure, which means if your Name Node daemon is down somehow, you don't have access to your Hadoop Cluster. How to deal with this problem?

Solution: Hadoop 2.x is featured with Name Node HA which is referred as HDFS High Availability (HA).

- Hadoop 2.x supports two Name Nodes at a time one node is active and another is standby node
- Active Name Node handles the client operations in the cluster
- StandBy Name Node manages metadata same as Secondary Name Node in Hadoop 1.x
- When Active Name Node is down, Standby Name Node takes over and will handle the client operations then after
- HDFS HA can be configured by two ways
    a. Using Shared NFS Directory : In this method, a shared Network File System (NFS) directory is used to store the edit logs for the NameNodes. This allows multiple NameNodes to access and write to the same log directory.
    b. Using Quorum Journal Manager : Quorum Journal Manager is a more robust and fault-tolerant solution for HDFS HA. Instead of using a shared NFS directory, QJM utilizes multiple JournalNodes to store edit logs.

Now, how the synchronization between Active Namenode and Standby Namenode is going to happen. Who will tell the Standby namenode that now you are an active node and who will tell the data node to send the message to the latest active namenode?

There free tool available which can manage all that synchronization:

Zookeeper:

Zookeeper is an open-source distributed coordination service that plays a crucial role in maintaining and managing the configuration information, naming, synchronization, and group services over large clusters in distributed

systems. Zookeeper acts as a centralized repository where distributed applications can put their data, which is accessible to other nodes.

It'll also help to elect the leader so inside the zookeeper there is an algorithm which will elect the master node. You have to provide the IP address of all the namenodes present in your cluster to the zookeeper.

How does the Zookeeper elect the master and maintain high availability?

It does this through a simple feedback loop. The Zookeeper, through its Fail-over Controller, monitors the Leader and Follower/Stand-by nodes.

It receives a heartbeat or instant notification of the current health/status of each node.
The moment a leader fails, one of the stand-by nodes is elected as the new leader by Zookeeper almost immediately. The election is completed and the new leader sends a message to the data nodes to report to the new master node.

Heartbeat Mechanism:

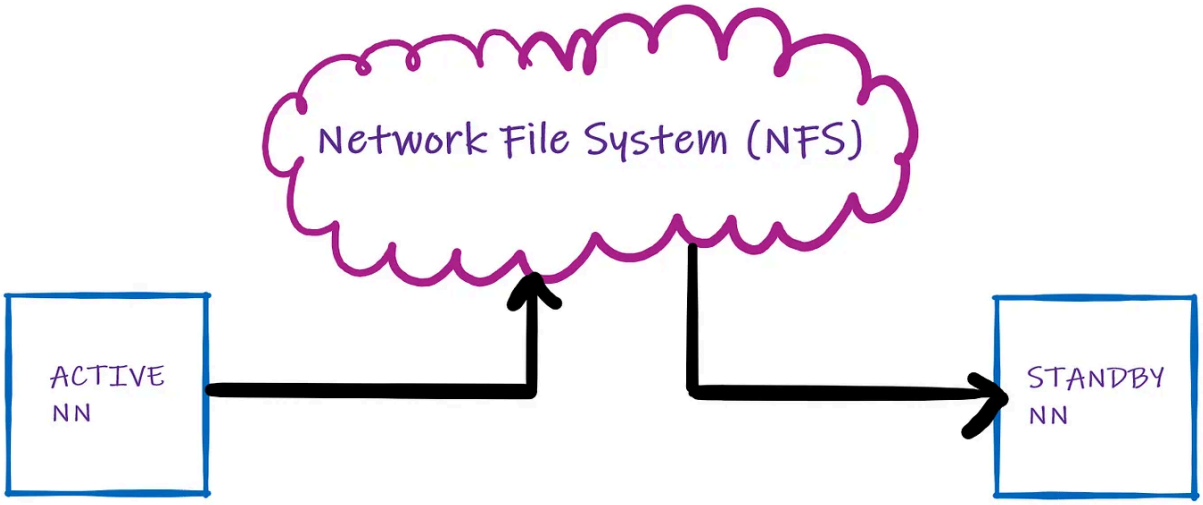
By default every 3 seconds, zookeeper will send a message to the active node and will do it 10 times by default. If the current Active node is not responding then Zookeeper is going to mark that node as a down.

It'll then send the message to the standby node that you are now the Active node.

How does NameNode Metadata Sync?

As you can imagine or see in the image 'HDFS High Availability Architecture', the name node metadata is a single point of failure, hence this metadata is replicated to introduce redundancy and enable high-availability (HA).

1) Shared Storage



We now know that there exists an Active Name Node and a Standby Name Node. Any change in the *active* is synced in real-time to the shared folder/storage i.e. network file system (NFS).

This NFS is accessible to the *standby,* which downloads all of the relevant incremental information in real-time to maintain the sync between the Namenodes.

Thus, in the event of a failure of the *active*, the *standby* name node already has all the relevant information to continue “business as usual” post fail-over. This is not used in the Production environment.

What is Replication in HDFS?

Replication in HDFS refers to the process of creating and storing multiple copies of the same data across different nodes in the cluster. These copies are known as ***replicas****. The purpose of replication is to ensure data reliability, availability, and fault tolerance.*

How Replication Works

Let’s break down the replication process:

1. ***Data Blocks****: In HDFS, files are split into smaller chunks called* ***blocks****. Each block is typically 128 MB by default, but this size can be configured. When a file is uploaded to HDFS, it's divided into these blocks, and each block is then stored across the cluster.*
2. ***Replication Factor****: HDFS allows you to specify a* ***replication factor*** *for each file. The replication factor determines how many copies of each block are created and stored. By default, the replication factor is set to 3, meaning each block will have three copies stored on different DataNodes.*

Now let's take a look at how we can access the data through HDFS

Setup HDFS in the local

Now lets learn about setting up HDFS locally on different operating systems (Unix/Linux, Windows, and macOS). It requires some platform-specific steps that are mentioned below:-

1. Prerequisites (Common for All OS)

Java Development Kit (JDK)

Ensure that JDK 8 or later is installed on your machine. You can verify by running:
java -version

2. Unix/Linux (Ubuntu)

Step 1: Install JDK

sudo apt-get update
sudo apt-get install openjdk-11-jdk

Step 2: Download Hadoop

Wget
https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
tar -xzvf hadoop-3.3.6.tar.gz
sudo mv hadoop-3.3.6 /usr/local/hadoop

Step 3: Configure Environment Variables

Add the following to your .bashrc file:

Setting up HDFS locally on different operating systems (Unix/Linux, Windows, and macOS) requires some platform-specific steps. Below is a detailed guide for each OS.

Prerequisites (Common for All OS)

Java Development Kit (JDK)

Ensure that JDK 8 or later is installed on your machine. You can verify by running:

```
java -version
```

A) Unix/Linux (Ubuntu)

Step 1: Install JDK

```
sudo apt-get update
sudo apt-get install openjdk-11-jdk
```

Step 2: Download Hadoop

```
wget
https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
tar -xzvf hadoop-3.3.6.tar.gz
sudo mv hadoop-3.3.6 /usr/local/hadoop
```

Step 3: Configure Environment Variables

Add the following to your .bashrc file:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$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_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Apply the changes:

```
source ~/.bashrc
```

Step 4: Configure Hadoop

Modify the following configuration files located in $HADOOP_HOME/etc/hadoop/:

hadoop-env.sh:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

core-site.xml:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```

hdfs-site.xml:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/datanode</value>
</property>
```

Create directories for NameNode and DataNode:

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
```

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

Step 5: Format NameNode

hdfs namenode -format

Step 6: Start HDFS

start-dfs.sh

Step 7: Access HDFS

- **Web Interface:** http://localhost:9870/
- **Command Line:** hdfs dfs -ls /

Step 8: Stop HDFS

stop-dfs.sh

3. Windows

Step 1: Install JDK

1. Download the JDK from Oracle's website and install it.
2. Set JAVA_HOME:
    - Open **System Properties** > **Environment Variables**.
    - Add a new JAVA_HOME variable pointing to your JDK installation directory, e.g., C:\Program Files\Java\jdk-11.

Step 2: Download Hadoop

1. Download Hadoop from Hadoop Releases page.
2. Extract the Hadoop tar file using a tool like 7-Zip.
3. Move the extracted folder to C:\hadoop.

Step 3: Configure Environment Variables

1. Open **System Properties** > **Environment Variables**.
2. Add the following to the **System variables** section:
   HADOOP_HOME=C:\hadoop
   JAVA_HOME=C:\Program Files\Java\jdk-11
   PATH=%PATH%;%HADOOP_HOME%\bin;%HADOOP_HOME%\sbin
3. Add the following in the command prompt:
   set HADOOP_HOME=C:\hadoopset PATH=%PATH%;%HADOOP_HOME%\bin;%HADOOP_HOME%\sbin

Step 4: Configure Hadoop

Edit the following files located in C:\hadoop\etc\hadoop\:

hadoop-env.cmd:

```
set JAVA_HOME=C:\Program Files\Java\jdk-11
```

core-site.xml:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```

hdfs-site.xml:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///c:/hadoop/hadoop_data/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///c:/hadoop/hadoop_data/hdfs/datanode</value>
</property>
```

Create directories for NameNode and DataNode:

```
mkdir C:\hadoop\hadoop_data\hdfs\namenode

mkdir C:\hadoop\hadoop_data\hdfs\datanode
```

Step 5: Format NameNode

```
hdfs namenode -format
```

Step 6: Start HDFS

```
start-dfs.cmd
```

Step 7: Access HDFS

- **Web Interface:** http://localhost:9870/
- **Command Line:** hdfs dfs -ls /

Step 8: Stop HDFS

stop-dfs.cmd

4. macOS

Step 1: Install JDK

1. Install Java using Homebrew:
   brew install openjdk@11
2. Set JAVA_HOME in your .zshrc or .bash_profile:
   export JAVA_HOME=$(/usr/libexec/java_home -v 11)

Step 2: Download Hadoop

wget
https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
tar -xzvf hadoop-3.3.6.tar.gz
sudo mv hadoop-3.3.6 /usr/local/hadoop

Step 3: Configure Environment Variables

Add the following to your .zshrc or .bash_profile:

```
export JAVA_HOME=$(/usr/libexec/java_home -v 11)
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
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_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Apply the changes:

```
source ~/.zshrc
```

Step 4: Configure Hadoop

Modify the following configuration files located in $HADOOP_HOME/etc/hadoop/:

hadoop-env.sh:

```
export JAVA_HOME=$(/usr/libexec/java_home -v 11)
```

core-site.xml:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```

hdfs-site.xml:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/datanode</value>
</property>
```

Create directories for NameNode and DataNode:

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
```

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

Step 5: Format NameNode

```
hdfs namenode -format
```

Step 6: Start HDFS

start-dfs.sh

Step 7: Access HDFS

- **Web Interface:** http://localhost:9870/
- **Command Line:** hdfs dfs -ls /

Step 8: Stop HDFS

stop-dfs.sh

Commands

1. Listing Directory Contents

To view the contents of a directory in HDFS, you can use the hdfs dfs -ls command. This command provides a list of files and subdirectories within the specified directory.

Command:
hdfs dfs -ls /path/to/directory

Explanation: This command lists the contents of the specified directory. If no directory is specified, it lists the contents of the current working directory in HDFS.

2. List All Files Recursively

To recursively list all files in a directory and its subdirectories, use the -R option with the ls command.

Command:
hdfs dfs -ls -R /path/to/directory

Explanation: This command will display the contents of the directory and all of its subdirectories, allowing you to see the entire file hierarchy.

3. Create a Directory

Creating directories in HDFS is straightforward with the mkdir command.

Command:
hdfs dfs -mkdir /path/to/directory

Explanation: This command creates a new directory at the specified path. Directories in HDFS are similar to those in a traditional file system.

4. Remove a Directory

To remove an empty directory in HDFS, use the rmdir command.

Command:
hdfs dfs -rmdir /path/to/directory

Explanation: This command removes the specified directory if it is empty. If the directory contains files or subdirectories, the command will fail.

5. Remove Files and Directories

To delete files or directories (even non-empty ones) in HDFS, use the rm command.

Command to Remove Files:
hdfs dfs -rm /path/to/file
Command to Remove Directories:
hdfs dfs -rm -r /path/to/directory

Explanation:

- The first command removes a single file.
- The second command recursively removes a directory and its contents.

6. Rename or Move a File

The mv command is used to rename or move files and directories within HDFS.

Command:
hdfs dfs -mv /source/path /destination/path

Explanation: This command moves or renames a file or directory from the source path to the destination path.

7. Copy Files from Local Filesystem to HDFS

To upload files from your local filesystem to HDFS, use the put command.

Command:
hdfs dfs -put /local/path /hdfs/path

Explanation: This command copies files from your local file system to the specified path in HDFS.

8. Copy Files from HDFS to Local Filesystem

To download files from HDFS to your local filesystem, use the get command.

Command:
hdfs dfs -get /hdfs/path /local/path

Explanation: This command retrieves files from HDFS and stores them in the specified local directory.

9. Copy Files within HDFS

To copy files within HDFS, use the cp command.

Command:
hdfs dfs -cp /source/path /destination/path
Explanation: This command copies files or directories from one location in HDFS to another.

10. Check File Status

To check the status of a file or directory, use the stat command.

Command:
hdfs dfs -stat /path/to/file

Explanation: This command displays information about the file or directory, such as its size, modification time, and replication factor.

11. Display File Contents

To display the contents of a file, use the cat command.

Command:
hdfs dfs -cat /path/to/file

Explanation: This command outputs the entire contents of the specified file to the console.

12. Display First N Lines of a File

To view the first few lines of a file, use the head command.

Command:
hdfs dfs -head /path/to/file

Explanation: This command displays the first few lines of the file, which is useful for quickly inspecting the beginning of a large file.

13. Display Last N Lines of a File

To view the last few lines of a file, use the tail command.

Command:
hdfs dfs -tail /path/to/file
Explanation: This command displays the last few lines of the file, which can be useful for monitoring logs or checking the most recent entries in a file.