<u>Hadoop Operation modes</u>

Hadoop is an open-source framework which is mainly used for storage purpose and maintaining and analysing a large amount of data or datasets on the clusters of commodity hardware.

3 different Modes:

A. Standalone Mode/Local Mode

B. Pseudo-distributed Mode

C. Fully-Distributed Mode

<u>A.Local Mode or Standalone Mode</u>

Standalone mode is the default mode in which Hadoop run. Standalone mode is mainly used for debugging where you don't really use HDFS. Input and output both as a local file system in standalone mode.

And no need to do any custom configuration in the files- mapred-site.xml, core-site.xml, hdfs-site.xml.

Standalone mode is usually the fastest Hadoop modes as it uses the local file system for all the input and output.

Here is the summarized view of the standalone mode-

- Used for debugging purpose
- HDFS is not being used
- Uses local file system for input and output
- No need to change any configuration files
- Default Hadoop Modes

B. Pseudo Distributed Mode (Single Node Cluster)

In Pseudo-distributed Mode we also use only a single node, but the main thing is that the cluster is simulated, which means that all the processes inside the cluster will run independently to each other. All the daemons that are Namenode, Datanode, Secondary Name node, Resource Manager, Node Manager, etc. will be running as a separate process on separate JVM(Java Virtual Machine).

Namenode and Resource Manager are used as Master and Datanode and Node Manager is used as a slave. A secondary name node is also used as a Master. The purpose of the Secondary Name node is to just keep the hourly based backup of the Name node.

- Hadoop is used for development and for debugging purposes both.
- We need to change the configuration files mapred-site.xml, core-site.xml, hdfs-site.xml for setting up the environment.

C.Fully Distributed Mode (Multi-Node Cluster)

This is the production mode of Hadoop where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node.

Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.

• Production phase of Hadoop

• Separate nodes for master and slave daemons

• Data are used and distributed across multiple nodes

Setting up a Hadoop Cluster: Hadoop Configuration, Security in Hadoop, Administering

Setting up a Hadoop cluster involves several steps. Here is a general guide to setting up a Hadoop cluster:

1. Hardware Requirements:

Each machine should have adequate CPU, RAM, and storage to meet your processing and data storage needs.

High-speed network connectivity between the nodes is essential for efficient data transfer.

2. Software Requirements:

Install an operating system (e.g., Linux/Window) on each machine in the cluster.

Install Java Development Kit (JDK) on each machine, as Hadoop is a Java-based framework.

Download and install the Hadoop distribution that you plan to use.

3. Configuration:

Configure the hosts file on each machine to resolve hostnames to IP addresses. This helps Hadoop nodes identify each other.

4. Hadoop Configuration:

Customize Hadoop configuration files, including hadoop-env.sh, core-site.xml, hdfs-site.xml, and yarn-site.xml.

5. HDFS Setup:

Format the Hadoop Distributed File System (HDFS) on the Name Node. This initializes the file system and creates the necessary metadata.

Start the HDFS daemons on the master and slave nodes. The Name Node and Data Nodes handle data storage and replication.

6. YARN Setup:

Start the Resource Manager on the master node and Node Managers on the slave nodes. YARN is responsible for resource management and job scheduling.

7. Start Hadoop Services:

Start the Hadoop services, including HDFS and YARN components, on the respective nodes.

8. Data Ingestion:

Load data into HDFS using tools like hadoop fs or Hadoop-compatible distributed file systems.

9. Running Jobs:

Submit Map Reduce or other Hadoop-based jobs to process data on the cluster.

10. Monitoring and Management:

Utilize Hadoop's web-based user interfaces, like the Hadoop Resource Manager and Name Node web interfaces, to monitor the cluster's status.

11. Troubleshooting and Optimization:

Monitor cluster performance and troubleshoot issues that may arise, such as underutilization, data skew, or resource bottlenecks.

12. Security:

Securing access to the cluster with authentication and authorization, encrypting data in transit and at rest, and setting up firewalls.

Cluster specification

Cluster is a collection of something, a simple computer cluster is a group of various computers that are connected with each other through LAN (Local Area Network), and the nodes in a cluster share the data.

Hadoop cluster is also a collection of various commodity hardware. Hardware components work together as a single unit. In the Hadoop cluster, there are lots of nodes (can be computer and servers) contains Master and Slaves, the Name node and Resource Manager works as Master and data node, and Node Manager works as a Slave.

The Hadoop cluster stores different types of data and processes them.

Structured-Data: The data which is well structured like Mysql.

Semi-Structured Data: The data which has the structure but not the data type like XML, Json (Javascript object notation).

Unstructured Data: The data that doesn’t have any structure like audio, video.

<u>**Hadoop Clusters Properties**</u>

1. **Scalability:** Hadoop clusters are very much capable of scaling-up and scaling-down the number of nodes i.e. servers or commodity hardware.
2. **Flexibility:** Hadoop cluster is very much Flexible and they can handle any type of data irrespective of its type and structure. With the help of this property, Hadoop can process any type of data from online web platforms.
3. **Speed**: Hadoop clusters are very much efficient to work with a very fast speed because the data is distributed among the cluster and also because of its data mapping capability’s i.e. the Map Reduce architecture which works on the Master-Slave phenomena.
4. **No Data-loss**: There is no chance of loss of data from any node in a Hadoop cluster because Hadoop clusters have the ability to replicate the data in some other node.
5. **Economical:** The data is distributed in a cluster among all the nodes. So in the case to increase the storage we only need to add one more another hardware storage which is not that much costliest.

<u>**Hadoop in the cloud.**</u>

Running Hadoop in the cloud is a popular choice for organizations that want to take advantage of Hadoop's data processing capabilities without the overhead of managing physical hardware and infrastructure. Cloud-based Hadoop solutions offer flexibility, scalability, and ease of deployment.

1. Choose a Cloud Provider:

Select a cloud service provider that offers Hadoop services or infrastructure. Some popular options include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), and others. Each cloud provider has its own Hadoop-related services.

2. Choose the Hadoop Distribution:

Most cloud providers offer managed Hadoop distributions, such as Amazon EMR (Elastic MapReduce), Azure HDInsight, and Google Cloud Dataproc.

3. Define Your Use Case:

Clearly define your use case and data processing requirements. Determine whether you need Hadoop for batch processing, real-time processing, data warehousing, or a combination of these.

4. Configure Your Cluster:

Depending on the cloud provider and Hadoop distribution you choose, you can configure your cluster using a web-based console, command-line tools, or Infrastructure as Code (IaC) scripts. Specify the number and type of nodes, storage options, and other cluster settings.

5. Data Storage:

Cloud providers offer various storage options for Hadoop data, such as cloud object storage (e.g., Amazon S3, Azure Blob Storage, Google Cloud Storage). You can also integrate Hadoop with cloud-based databases or data warehouses.

6. Data Ingestion:

Transfer your data to the cloud storage solution or your Hadoop cluster. Many cloud providers offer data transfer services and tools for data ingestion.

7. Security and Access Control:

Implement security measures, including access control, encryption, and authentication, to protect your Hadoop data and cluster resources in the cloud.

8. Cluster Management:

Monitor and manage your Hadoop cluster using cloud provider-specific tools or third-party monitoring solutions. You can scale the cluster up or down based on your processing needs.

9. Data Processing:

Run Hadoop jobs and data processing tasks using the tools and frameworks provided by your Hadoop distribution. For example, you can use Apache Spark, Apache Hive, Apache Pig, or MapReduce for batch processing.

10. Cost Management:

Keep an eye on cloud costs, as cloud services are billed based on usage. Optimize your cluster for cost efficiency by adjusting the number of nodes and services as needed.

11. Backup and Disaster Recovery:

Implement backup and disaster recovery strategies to protect your data and ensure business continuity.

12. Integration and Ecosystem:

Explore the integration options with other cloud services and ecosystem tools. Cloud providers offer a wide range of services for analytics, machine learning, and more.

Hadoop Architecture

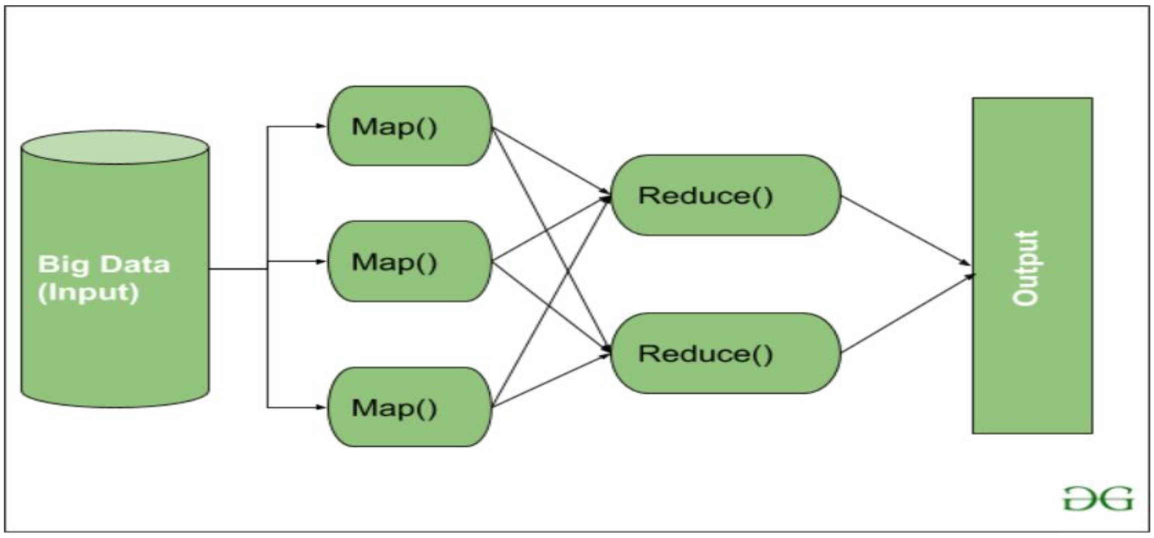
Hadoop is an open source framework from Apache and is used to store process and analyse data which are very huge in volume. Hadoop is written in Java and is not OLAP (online analytical processing). It is used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. Moreover it can be scaled up just by adding nodes in the cluster.

The Hadoop Architecture Mainly consists of 4 components.

1. **MapReduce**
2. **HDFS(Hadoop Distributed File System)**
3. **YARN(Yet Another Resource Negotiator)**
4. **Common Utilities or Hadoop Common**

1. Map Reduce

The major feature of Map Reduce is to perform the distributed processing in parallel in a Hadoop cluster which Makes Hadoop working so fast. When you are dealing with Big Data, serial processing is no more of any use. Map Reduce has mainly 2 tasks which are divided phase-wise:



Map Task:

RecordReader The purpose of *recordreader* is to break the records. It is responsible for providing key-value pairs in a Map() function. The key is actually is its locational information and value is the data associated with it.

- **Map:** A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.

- **Combiner:** Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The intermediate key-value that are generated in the Map is combined with the help of this combiner. Using a combiner is not necessary as it is optional.
- **Partitionar:** Partitional is responsible for fetching key-value pairs generated in the Mapper Phases.

Reduce Task

Shuffle and Sort: The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as *Shuffling*. Using the Shuffling process the system can sort the data using its key value.

Once some of the Mapping tasks are done Shuffling begins that is why it is a faster process and does not wait for the completion of the task performed by Mapper.

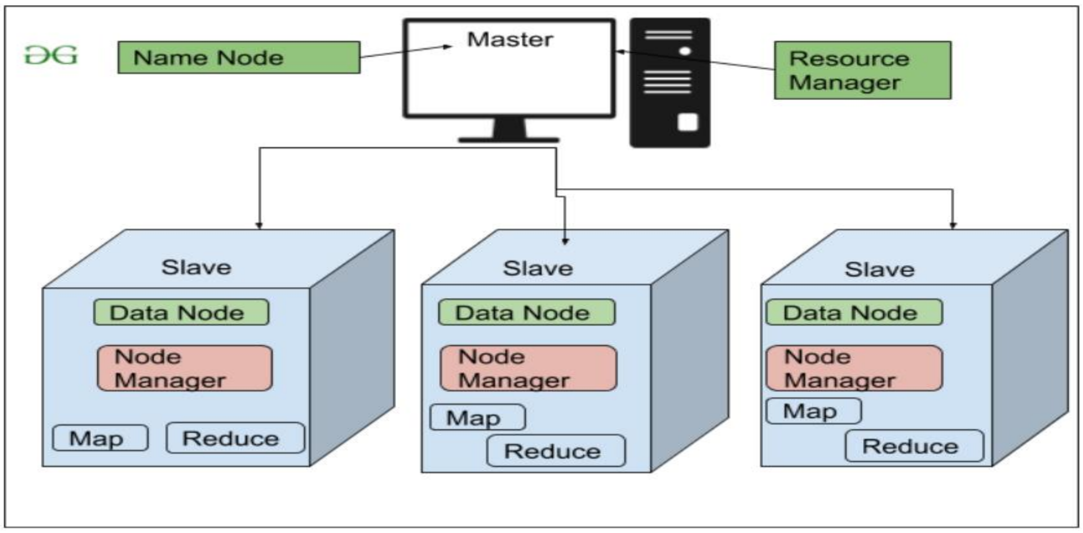
- **Reduce:** The main function or task of the Reduce is to gather the Tuple generated from Map and then perform some sorting and aggregation sort of process on those key-value depending on its key element.
- **OutputFormat:** Once all the operations are performed, the key-value pairs are written into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

2.HDFS

HDFS (Hadoop Distributed File System) is utilized for storage permission.

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

- **NameNode(Master)**
- **DataNode(Slave)**



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

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

3. YARN (Yet Another Resource Negotiator)

YARN is a Framework on which Map Reduce works. YARN performs 2 operations that are Job

A. Scheduling

- The Purpose of **Job scheduler** is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be maximized.
- Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc

B. Resource Management.

Manage all the resources that are made available for running a Hadoop cluster.

Features of YARN

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

Core components in Hadoop

Core components have HDFS, Map Reduce, and YARN, is part of the foundation of Cloudera's platform.

Common Hadoop Shell commands

HDFS is the primary or major component of the Hadoop ecosystem which is responsible for storing large data sets of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files.

To use the HDFS commands, start the Hadoop services using the following command:

ls: This command is used to list all the files. : bin/hdfs dfs -ls /

mkdir: To create a directory: bin/hdfs dfs -mkdir <folder name>

touchz: It creates an empty file:bin/hdfs dfs -touchz <file_path>

Session: 8 HDFS Data Storage Process

o HDFS Data storage process

HDFS divides files into blocks and stores each block on a Data Node. Multiple Data Nodes are linked to the cluster. The Name Node then distributes replicas of these data blocks across the cluster. It processes this huge data in a distributed environment using many Data Nodes.

It also instructs the user or application where to locate wanted information.

Where are the HDFS logs stored?

By default, the logs are written under the /usr/lpp/mmfs/hadoop/logs directory.

Hadoop Distributed File System (HDFS) is a distributed and fault-tolerant file system designed to store and manage large volumes of data across a cluster of computers. The HDFS data storage process involves several key steps:

1. **Data Ingestion:**

Data is ingested into HDFS from various sources, such as local files, data streams, databases, or external data repositories. Data may be generated and collected continuously, batch-loaded, or streamed into the HDFS cluster.

2. **Data Splitting:**

HDFS divides the ingested data into fixed-size blocks, typically 128 MB or 256 MB in size.

3. **Replication:**

Each data block is replicated across multiple DataNodes in the HDFS cluster to ensure fault tolerance and data durability. The default replication factor is usually set to 3, meaning that each block has two additional copies in the cluster. Replication can be adjusted based on requirements.

4. **NameNode and DataNode Architecture:**

HDFS follows a master-slave architecture with two main components:

NameNode: The NameNode is the master server responsible for managing the metadata and namespace of the file system. It keeps track of the structure of the file system, including the directory tree and metadata about data blocks.

DataNodes: DataNodes are slave servers that store and manage the actual data blocks. They periodically send heartbeat signals to the NameNode to report their status.

5. **Block Placement:**

The NameNode decides the initial placement of data blocks within the cluster. It takes into account factors like data locality, data replication, and the available storage capacity on DataNodes.

6. **Data Write Operation:**

When new data is written to HDFS, the client contacts the NameNode to request block locations. The NameNode returns the locations of the target DataNodes.

The data is then sent to the DataNodes, where it is stored as separate replicas. The DataNodes acknowledge the successful write operation.

7. **Data Read Operation:**

To read data, the client contacts the NameNode to obtain the locations of the required data blocks.

The client can read data from the nearest DataNodes to maximize data locality and minimize network transfer.

8. **Data Replication Management:**

The HDFS framework continuously monitors the health and status of DataNodes. If a DataNode fails or becomes unresponsive, HDFS automatically replicates the data blocks stored on that node to other healthy DataNodes to maintain the desired replication factor.

9. **Balancing and Maintenance:**

Periodically, HDFS balances data distribution by moving blocks between DataNodes to ensure that storage is evenly distributed across the cluster.

10. **Data Deletion and Expiration:**

When data is deleted or expires, the client communicates with the NameNode to remove the metadata information associated with the data. The actual data blocks will be marked as available for reuse.

o Anatomy of writing and reading file in HDFS

For each block, the name node returns the addresses of the data nodes that have a copy of that block.

The DFS returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the data node and name node I/O.

Anatomy of File Read in HDFS

Data flows between the client interacting with HDFS, the name node, and the data nodes with the help of a diagram

Step 1: The client opens the file it wishes to read by calling open() on the File System Object.

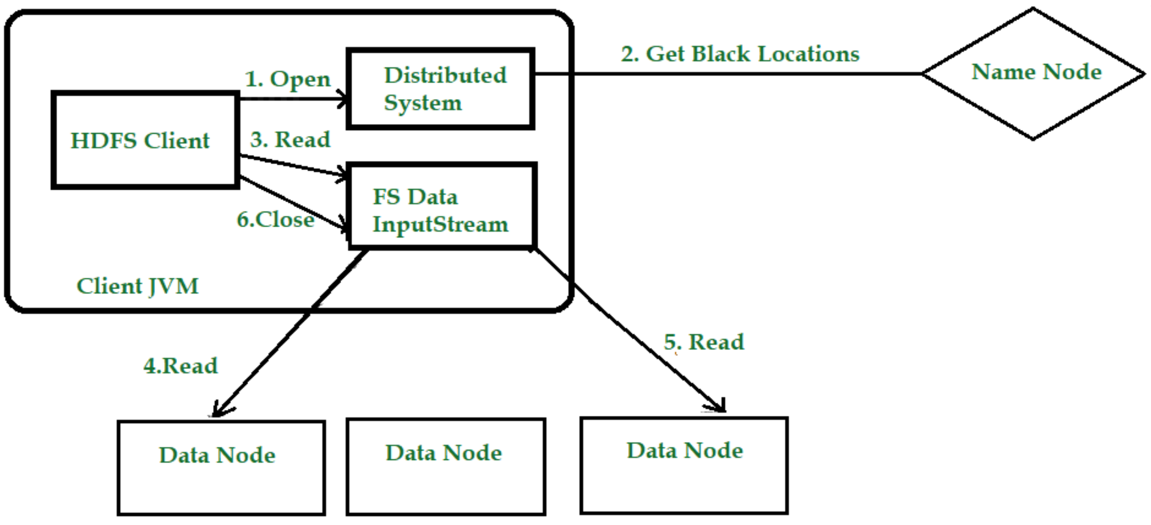
Step 2: Distributed File System(DFS) calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file. For each block, the name node returns the addresses of the data nodes that have a copy of that block. The DFS returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the data node and name node I/O.

Step 3: The client then calls read() on the stream. DFSInputStream, which has stored the info node addresses for the primary few blocks within the file, then connects to the primary (closest) data node for the primary block in the file.

Step 4: Data is streamed from the data node back to the client, which calls read() repeatedly on the stream.

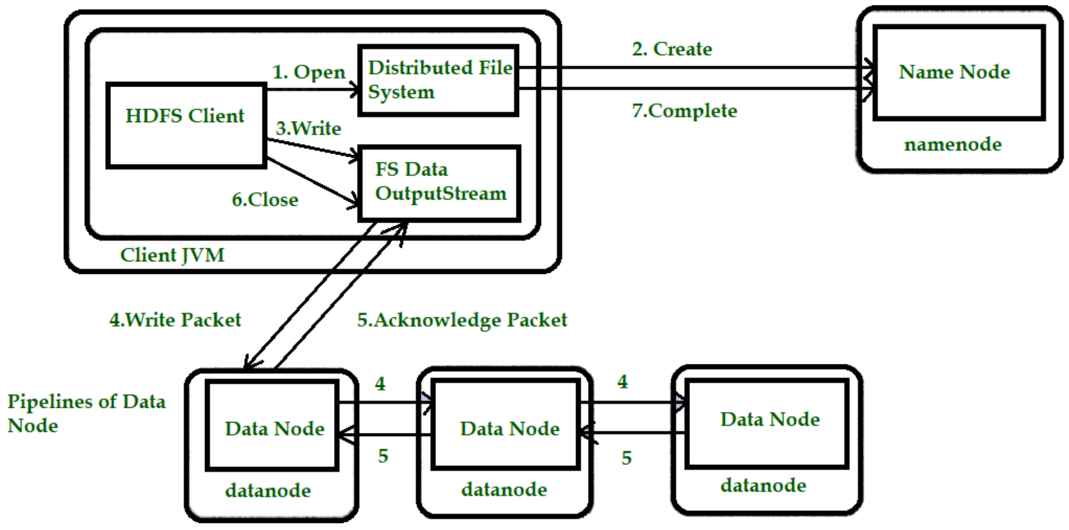
Step 5: When the end of the block is reached, DFSInputStream will close the connection to the data node, then finds the best data node for the next block. This happens transparently to the client, which from its point of view is simply reading an endless stream. Blocks are read as, with the DFSInputStream opening new connections to data nodes because the client reads through the stream. It will also call the name node to retrieve the data node locations for the next batch of blocks as needed.

Step 6: When the client has finished reading the file, a function is called, close() on the FSDataInputStream.



Anatomy of File Write in HDFS

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



Step 1: The client creates the file by calling create() on DistributedFileSystem(DFS).

Step 2: DFS makes an RPC call to the name node to create a new file in the file system's namespace, with no blocks associated with it. The name node performs various checks to make sure the file doesn't already exist and that the client has the right permissions to create the file. If these checks pass, the name node prepares a record of the new file; otherwise, the file can't be created and therefore the client is thrown an error i.e. IOException. The DFS returns an FSDataOutputStream for the client to start out writing data to.

Step 3: Because the client writes data, the DFSOutputStream splits it into packets, which it writes to an indoor queue called the info queue. The data queue is consumed by the DataStreamer, which is liable for asking the name node to allocate new blocks by picking an inventory of suitable data nodes to store the replicas. The list of data nodes forms a pipeline, and here we'll assume the replication level is three, so there are three nodes in the pipeline. The DataStreamer streams the packets to the primary data node within the pipeline, which stores each packet and forwards it to the second data node within the pipeline.

Step 4: Similarly, the second data node stores the packet and forwards it to the third (and last) data node in the pipeline.

Step 5: The DFSOutputStream sustains an internal queue of packets that are waiting to be acknowledged by data nodes, called an "ack queue".

Step 6: This action sends up all the remaining packets to the data node pipeline and waits for acknowledgments before connecting to the name node to signal whether the file is complete or not.

HDFS follows Write Once Read Many models. So, we can't edit files that are already stored in HDFS, but we can include them by again reopening the file.

o Handling Read/Write failures

Hadoop Distributed File System (HDFS) is designed to handle read and write failures gracefully by using data replication and recovery mechanisms. These mechanisms ensure data durability and fault tolerance. Here's how HDFS handles read and write failures:

Handling Write Failures:

1. **Replication**: When data is written to HDFS, it is split into fixed-size blocks, typically 128 MB or 256 MB. These data blocks are replicated across multiple DataNodes in the cluster. The default replication factor is usually set to 3, which means that each block has three copies. Replication helps in ensuring data durability and fault tolerance.
2. **Acknowledge from DataNodes:** When a client writes data to HDFS, it waits for acknowledgments from the DataNodes where the data blocks are stored. The write operation is considered successful only when a specified number of acknowledgments (based on the replication factor) are received. If a DataNode fails to acknowledge the write, the client can retry the write operation on a different DataNode.
3. **Pipeline Write:** To further improve data durability, HDFS uses a pipeline write mechanism. In a pipeline, data is transferred from one DataNode to another in a sequence. This ensures that multiple copies of the data are stored across different DataNodes before the write operation is considered successful.
4. **Checksums:** HDFS uses checksums to verify data integrity during read and write operations. If data corruption is detected during a read operation, the system can fetch a fresh copy of the data from another DataNode.

Handling Read Failures:

1. **Replication:** Read failures are addressed by the replication mechanism. When a DataNode hosting a particular data block becomes unavailable or unresponsive, HDFS

can retrieve the data from one of the other replicas. The client reads data from an available replica.

2. **Data Locality:** HDFS strives to maximize data locality. When a read operation is requested, the client first tries to read the data from the DataNode that is closest to it in terms of network proximity. This minimizes network transfer and improves read performance.
3. **Parallel Reading:** HDFS supports parallel reading of data blocks. If one replica of a block is slow or unavailable, the client can read the data from the other replicas in parallel, which can speed up the read operation.
4. **Checksums:** Checksums are used to ensure data integrity during read operations. If data corruption is detected, the system can request a fresh copy of the data from another replica.