Hadoop Distributed File System (HDFS)

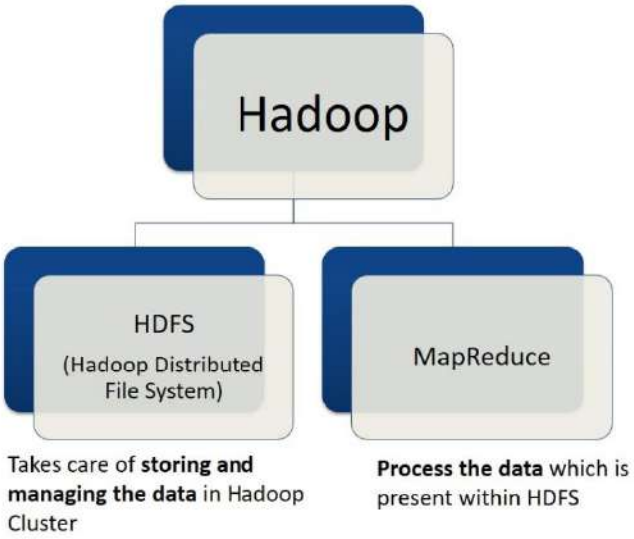
1. **What are the core components of Hadoop? Explain in brief its each of its components.**

Hadoop is an **open-source framework** designed to store and process large sets of data in a distributed computing environment. It's primarily used for big data applications where traditional databases or processing systems might struggle due to the volume, variety, and velocity of data.

At its core, Hadoop consists of two main components:

Hadoop Distributed File System (HDFS): HDFS is a distributed file system that stores data across multiple machines in a Hadoop cluster. It breaks large files into smaller blocks and distributes them across the cluster for efficient storage and retrieval. HDFS ensures data reliability by replicating each block across multiple nodes in the cluster, thereby providing fault tolerance.

MapReduce: MapReduce is a programming model and processing engine for distributed data processing. It allows users to write parallelizable algorithms to process large datasets across the Hadoop cluster. MapReduce jobs typically consist of two phases: the map phase, where data is processed in parallel across multiple nodes, and the reduce phase, where the results from the map phase are aggregated and processed to produce the final output.



In addition to these core components, the Hadoop ecosystem includes various other tools and projects that extend its capabilities and make it easier to work with big data. Some popular components of the Hadoop ecosystem include:

- **Hadoop YARN (Yet Another Resource Negotiator):** YARN is a resource management layer that allows different processing frameworks, such as MapReduce, and Apache Spark, to run on the same Hadoop cluster, enabling more flexible and efficient resource utilization.

- **Apache Hive:** Hive is a data warehouse infrastructure built on top of Hadoop that provides a SQL-like query language, called HiveQL, for querying and analyzing data stored in HDFS.

- **Apache Pig:** Pig is a high-level platform for creating MapReduce programs using a scripting language called Pig Latin. It abstracts away the complexities of writing MapReduce programs directly, making it easier to work with large datasets.

- **Apache HBase:** HBase is a distributed, scalable, and non-relational database that runs on top of Hadoop. It provides real-time read/write access to large datasets, making it suitable for use cases requiring low-latency data access.

Overall, Hadoop has become a cornerstone technology for organizations dealing with big data, enabling them to store, process, and analyze massive datasets in a cost-effective and scalable manner.

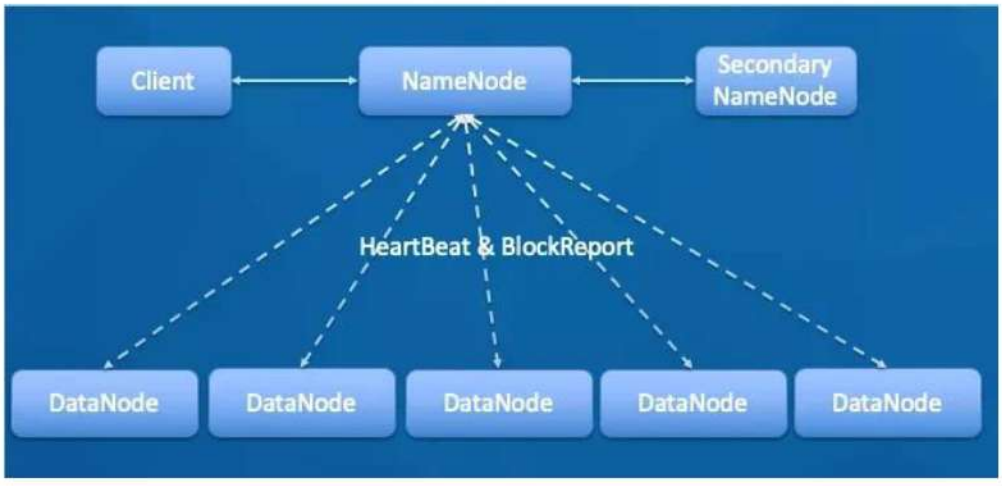
2. What is HDFS?

Hadoop comes with a **distributed file system** called HDFS. In HDFS data is **distributed** over **several machines and replicated** to ensure their durability to failure and high availability to parallel applications. It is cost-effective as it **uses commodity hardware**. It involves the concept of **blocks, data nodes and name node**.

Where to use HDFS

- Very Large Files: Files should be of hundreds of megabytes, gigabytes or more.
- Streaming Data Access: HDFS is built on write-once and read-many-times pattern.
- Commodity Hardware: It works on low cost hardware.

HDFS Architecture



HDFS follows Master/Slave architecture. HDFS consists of four components, namely:

- **HDFS Client**
- **Name Node**
- **Data Node**
- **Secondary Name Node.**

HDFS Client:

- When a file is uploaded to HDFS, the client divides the file into multiple blocks and stores them.
- Interacts with the Name Node to obtain the location information of the file.
- Interact with Data Node to read and write data.
- Client provides some commands to manage and access HDFS, such as start and close HDFS.

Name Node:

The Name Node is a crucial component of the HDFS architecture, responsible for managing the file system namespace and metadata. Its primary functions include:

- **Namespace Management:** The Name Node maintains the file system namespace, which includes information about the directory structure, file names, and attributes such as permissions, replication factor, and block locations.
- **Metadata Management:** It manages the metadata associated with files and directories, including their sizes, timestamps, and block locations. This metadata is stored in memory and periodically persisted to disk to ensure durability.
- **Coordination:** The Name Node coordinates access to data stored in HDFS by handling client requests for file operations such as file creation, deletion, reading, and writing. It also coordinates data replication and block placement decisions across the Hadoop cluster.
- **Heartbeats and Block Reports:** NameNode receives periodic heartbeats and block reports from Data Nodes, which provide information about the health and status of the DataNodes and the data blocks they store. Based on this information, NameNode can detect failures and take corrective actions.

DataNode:

DataNode is another critical component of HDFS responsible for storing and managing the actual data blocks that constitute files. Its main functions include:

- **Storage of Data Blocks**: DataNodes store data blocks on the local file system of each individual node in the Hadoop cluster. These data blocks are replicated across multiple DataNodes to ensure fault tolerance and data durability.
- **Heartbeats and Block Reports**: DataNodes send periodic heartbeats to the NameNode to indicate that they are alive and functioning correctly. They also provide block reports to inform the NameNode about the data blocks they store and their health status.
- **Data Block Replication**: DataNodes replicate data blocks to ensure fault tolerance and high availability. The replication process is managed by the NameNode, which instructs DataNodes to replicate blocks to other nodes in the cluster according to the configured replication factor.

Secondary NameNode:

The Secondary NameNode in HDFS plays a crucial role in assisting the primary NameNode with metadata management and maintaining the overall health of the Hadoop Distributed File System. Despite its name, the Secondary NameNode does not act as a backup or standby for the primary NameNode. Instead, its functions include:

- **Edits Log Processing:** The Secondary NameNode also assists in processing the edits log generated by the primary NameNode. The edits log records all the changes made to the file system metadata since the last checkpoint. The Secondary NameNode downloads the edits log from the primary NameNode and applies these changes to the namespace image during the checkpoint creation process. By periodically processing the edits log, the Secondary NameNode helps in managing the growth of the log and ensures efficient recovery in case of NameNode failure.
- **Metadata Size Management:** Since the primary NameNode stores all the metadata in memory, the size of the metadata can become a bottleneck for large Hadoop clusters with a massive number of files and directories. The Secondary NameNode helps in managing the size of the metadata by periodically creating checkpoints and merging them with the edits log. This process reduces the amount of data that needs to be loaded into memory during NameNode restarts, thereby improving the recovery time.
- **System Health Monitoring:** The Secondary NameNode monitors the health of the HDFS cluster by analyzing various metrics such as the size of the edits log, the time taken to create a checkpoint, and the overall status of the NameNode. It can alert

administrators if there are any issues detected in the file system or if the checkpoint creation process fails.

Overall, the Secondary NameNode plays a critical role in ensuring the reliability, efficiency, and scalability of the Hadoop Distributed File System by assisting the primary NameNode with metadata management and system health monitoring.

3. What is metadata? What information does it provide?

In Hadoop Distributed File System (HDFS), metadata refers to the **information about the file system structure**, such as **file names, directory structure, file permissions, file sizes,** and other attributes associated with files and directories. Metadata is stored separately from the actual data blocks in HDFS. It is managed and coordinated by the NameNode, a crucial component of the Hadoop architecture. The NameNode stores the metadata in memory and persists it to disk to ensure durability.

Metadata in HDFS provides essential information for managing and accessing files stored in the distributed file system. Some of the key information provided by metadata includes:

- **File Names:** Metadata stores the names of files stored in the HDFS. This allows users and applications to refer to files by their names.
- **Directory Structure:** Metadata maintains the hierarchical directory structure of the file system. It records the parent-child relationships between directories and files.
- **File Permissions:** Metadata stores permissions associated with files and directories, including read, write, and execute permissions for users, groups, and others.
- **File Sizes:** Metadata includes information about the sizes of files stored in HDFS. This helps users and applications to determine the amount of data stored in each file.
- **Timestamps:** Metadata records timestamps associated with files and directories, including the creation time, last access time, and last modification time.
- **Replication Factor:** Metadata stores the replication factor for each file, which determines the number of copies of each data block stored across the Hadoop cluster.
- **Block Locations:** Metadata maintains information about the locations of data blocks that constitute each file. This information is crucial for data retrieval and data locality optimizations in HDFS.

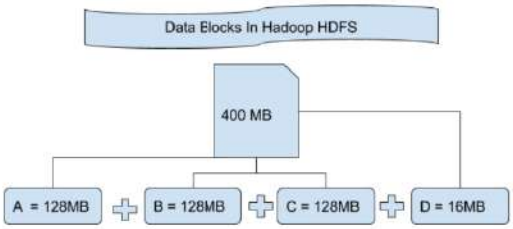
By storing this metadata separately from the data blocks, HDFS ensures efficient management and scalability of large-scale distributed file systems. It allows Hadoop NameNode to efficiently manage and coordinate access to files and directories across the Hadoop cluster.

4. What is a "block" in HDFS?

The **Hadoop Distributed File System (HDFS)** stores files in block-sized chunks called data blocks. These blocks are then stored as independent units and are restricted to 128 MB blocks by default. However, they can be adjusted by the user according to their requirements.

Users can adjust block size through the dfs.block.size in the hdfs-site.xml.

If the file size is not a multiple of 128 MB, the last block may be smaller.



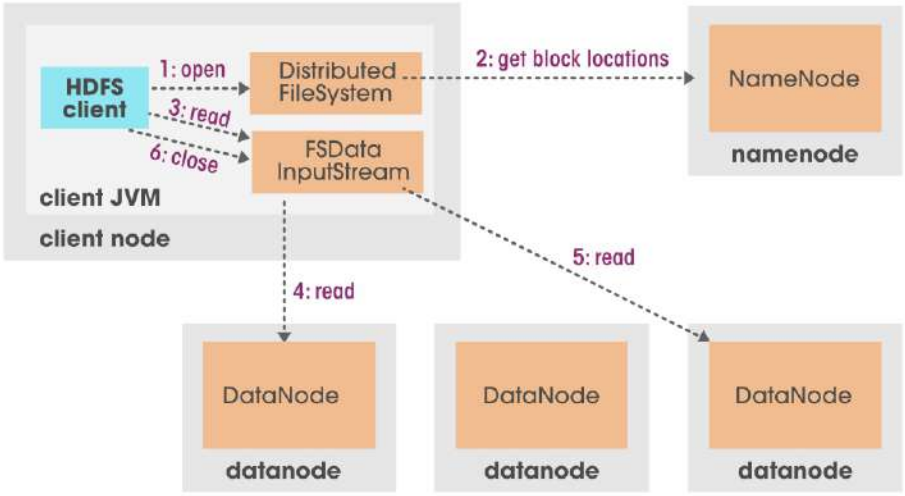
Advantages

1. No limitation on the file size as a file can be larger than any single disk in the network.
2. Since blocks are of a fixed size, we can easily calculate the number of blocks stored on a given disk. This provides simplicity to the storage subsystem.
3. Blocks are easy to replicate between DataNodes and, thus, provide fault tolerance and high availability.
4. Since blocks don't require storing file metadata, another system can separately handle the metadata.

5. Describe the HDFS Architecture.

Follow question 2

6. Describe the working principle of HDFS – Data Read Operation



Syntax For Reading Data From HDFS:

hdfs dfs -get <source-path> <destination-path>

 # here source path is file path on HDFS that we want to read

 # destination path is where we want to store the read file on local machine

Anatomy of File Read in HDFS

Step 1: The client opens the file it wishes to read by calling open() on the File System Object (which for HDFS is an instance of Distributed File System).

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.

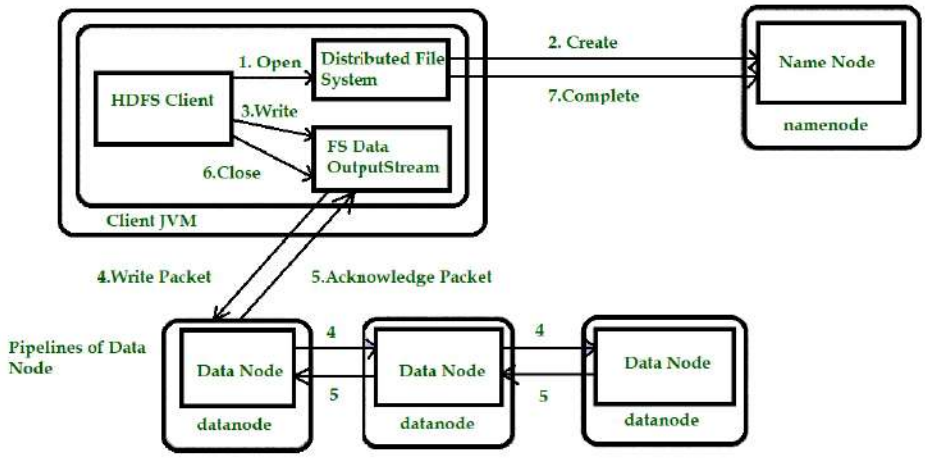
Step 3: The client then calls read() on the stream. DFSInputStream, which has stored the info node addresses for the primary few blocks within the file, then connects to the primary (closest) data node for the primary block in the file.

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.

7. Describe the working principle of HDFS – Data Write Operation



Syntax For Writing Data to HDFS:

hdfs dfs -put <From Path> <To-path>

 # here To-path is the location of HDFS

 # From-path is the path for local machine

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. This design allows HDFS to scale to a large number of concurrent clients because the data traffic is spread across all the data nodes in the cluster. Thus, it increases the availability, scalability, and throughput of the system.

8. What is Hadoop cluster and how many different modes can you configure a Hadoop cluster?

A Hadoop cluster is a distributed computing environment that consists of a collection of interconnected nodes, each running Hadoop software components. These clusters are designed to store and process large volumes of data across multiple machines in a parallel and fault-tolerant manner. The primary components of a Hadoop cluster include the Hadoop Distributed File System (HDFS) for storage and frameworks like MapReduce or Apache Spark for distributed data processing.

Different modes can be configured for setting up a Hadoop cluster, each serving specific purposes and catering to different requirements. The main modes are:

1. **Standalone Mode (Local Mode):**
   - In standalone mode, Hadoop runs on a single node without HDFS, meaning it uses the local file system rather than HDFS for storage.
   - It is primarily used for development, testing, and debugging purposes when dealing with small datasets where the distributed nature of Hadoop is not necessary.

2. **Pseudo-Distributed Mode:**
   - Pseudo-distributed mode simulates a multi-node Hadoop cluster on a single physical machine.
   - Each Hadoop daemon (such as NameNode, DataNode, ResourceManager, NodeManager) runs on a separate JVM (Java Virtual Machine) instance.
   - Although it runs on a single machine, it provides a more realistic environment for testing and development compared to standalone mode.

3. **Fully-Distributed Mode:**
   - Fully-distributed mode is the typical deployment mode for production Hadoop clusters.
   - In this mode, Hadoop components run on multiple nodes in a real network environment, with each node serving a specific role (e.g., NameNode, DataNode, ResourceManager, NodeManager).
   - It offers scalability, fault tolerance, and high availability by distributing data and computation across multiple nodes.

Each of these modes serves different purposes and is suitable for various stages of development, testing, and production deployment. The choice of mode depends on factors such as the scale of data processing, resource availability, and specific requirements of the application or use case.

9. What are the benefits of multiple MasterNodes?

In a Hadoop cluster, having multiple master nodes offers several benefits:

1. High Availability: With multiple master nodes, if one master node fails, another can take over its responsibilities, ensuring uninterrupted operation of the cluster. This high availability architecture enhances the reliability of the Hadoop system.

2. Load Balancing: Distributing the workload across multiple master nodes helps in balancing the processing load. This prevents any single master node from becoming a bottleneck and ensures efficient resource utilization.

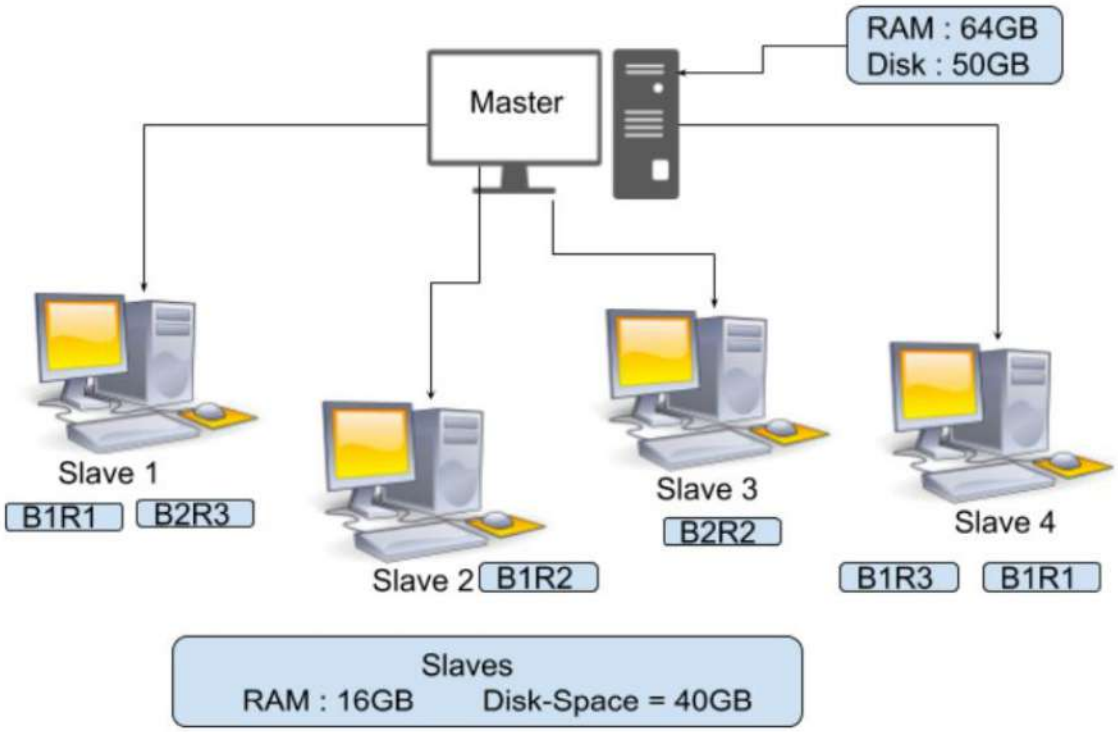
3. Scalability: Adding more master nodes allows the Hadoop cluster to scale horizontally. As the data and processing requirements grow, additional master nodes can be added to accommodate the increased workload without significant performance degradation.

4. Fault Tolerance: Multiple master nodes enhance fault tolerance by providing redundancy. If one master node encounters hardware failure or other issues, the remaining master nodes can continue to operate, preventing data loss or downtime.

5. Improved Performance: With multiple master nodes handling various aspects of cluster management such as resource allocation, job scheduling, and metadata management, the overall performance of the Hadoop cluster can improve due to parallel processing and efficient resource utilization.

Overall, multiple master nodes in a Hadoop cluster contribute to a more robust, scalable, and fault-tolerant infrastructure, which is crucial for handling large-scale data processing and analytics workloads.

10. Explain how replication is performed internally in HDFS.



In the above image, you can see that there is a Master with RAM = 64GB and Disk Space = 50GB and 4 Slaves with RAM = 16GB, and disk Space = 40GB. Here you can observe that RAM for Master is more. It needs to be kept more because your Master is the one who is going to guide this slave so your Master has to process fast. Now suppose you have a file of size 150MB then the total file blocks will be 2 shown below.

128MB = Block 1
22MB = Block 2
As the replication factor by-default is 3 so we have 3 copies of this file block

FileBlock1-Replica1(B1R1) FileBlock2-Replica1(B2R1)
FileBlock1-Replica2(B1R2) FileBlock2-Replica2(B2R2)
FileBlock1-Replica3(B1R3) FileBlock2-Replica3(B2R3)
These blocks are going to be stored in our Slave as shown in the above diagram which means if suppose your Slave 1 crashed then in that case B1R1 and B2R3 get lost. But you can recover the B1 and B2 from other slaves as the Replica of this file blocks is already present in other slaves, similarly, if any other Slave got crashed then we can obtain that file block some other slave. Replication is going to increase our storage but Data is More necessary for us.

11. Explain how can you configure replication factor in HDFS.

The number of times you make a copy of that particular thing can be expressed as its Replication Factor. As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks. By default the Replication Factor for Hadoop is set to 3 which can be configured means you can change it Manually as per your requirement like in above example we have made 4 file blocks which means that 3 Replica or copy of each file block is made means total of 4×3 = 12 blocks are made for the backup purpose.

You can configure the Replication factor in you hdfs-site.xml file.

Here, we have set the replication Factor to one as we have only a single system to work with Hadoop i.e. a single laptop, as we don't have any Cluster with lots of the nodes. You need to simply change the value in dfs.replication property as per your need.

```
20 <property>
21 <name>dfs.replication</name>
22 <value>1</value>
23 </property>
```