02 HDFS

Why we need a file system?

Without a file system, information placed in a storage area would be one large body of data with no way to tell where one piece of information stops and the next begins.

Functions of a file system

- Control how data is stored and retrieved
- Metadata about the files and folders
- Permissions and security
- Manage storage space efficiently

DIFFERENT FILE SYSTEMS

Microsoft
FAT32 - 4 GB File limit 32 GB Volume limit
NTFS - 16 EB File limit 16 EB Volume limit

HFS - 2 GB File limit 2 TB Volume limit
HFS+ - 8 EB File limit 8 EB Volume limit

Linux
ext3 - 2 TB File limit 32 TB Volume limit
ext4 - 16 TB File limit 1 EB Volume limit
XFS - 8 EB File limit 8 EB Volume limit

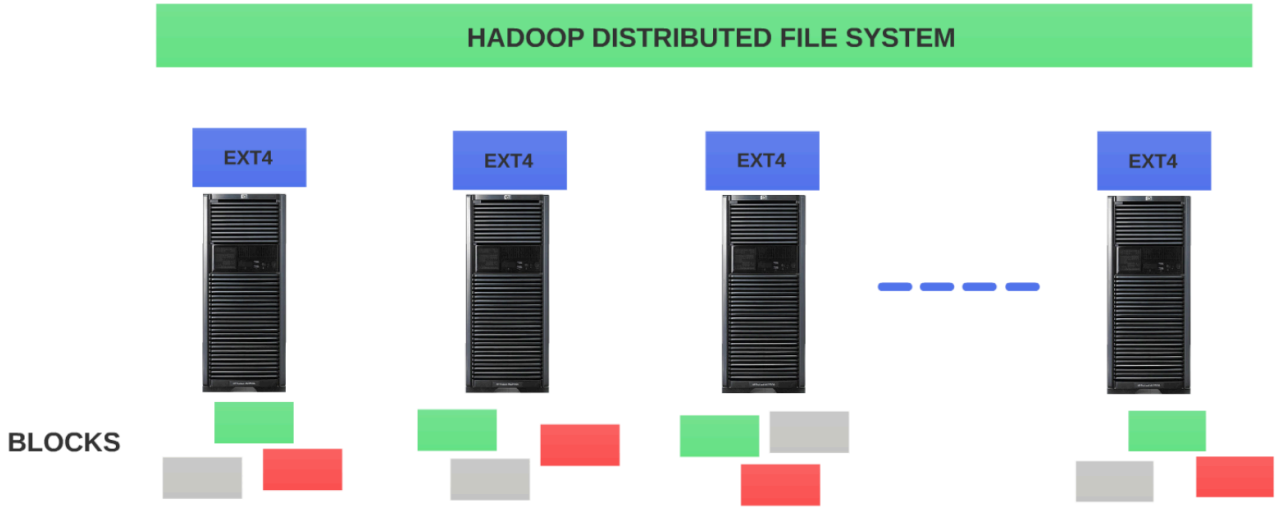
Why another file system ?

Why HDFS?

- **Scalability**:
  - Traditional file systems are designed for single-machine storage, making it challenging to scale out as data grows.
  - HDFS is designed to handle vast amounts of data by distributing it across many machines, providing horizontal scalability.
- **Fault Tolerance**:
  - In ext4, XFS, and similar file systems, data loss can occur if a disk fails.
  - HDFS replicates data across multiple nodes, ensuring that even if one node fails, the data remains accessible.
- **Data Locality**:
  - Traditional file systems do not optimize for data locality, leading to potential bottlenecks in data processing.
  - HDFS moves computation to where the data is stored, reducing network congestion and improving processing efficiency.
- **Throughput Optimization**:
  - ext4 and XFS are optimized for low-latency access, which is important for general-purpose file systems.
  - HDFS is optimized for high-throughput access, suitable for batch processing large datasets, as seen in big data applications.
- **Support for Large Files**:
  - File systems like ext4 have limitations on file sizes and the number of files they can handle efficiently.
  - HDFS is designed to store and manage very large files (terabytes or more) and a massive number of files seamlessly.
- **Streaming Data Access**:
  - Traditional file systems focus on random access patterns, which are less efficient for big data analytics.
  - HDFS is optimized for streaming data access, making it ideal for applications requiring sequential data processing.
- **Integration with Big Data Ecosystem**:
  - HDFS is tightly integrated with the Hadoop ecosystem, enabling seamless interaction with other big data tools like MapReduce, Hive, and Spark.
  - ext4, XFS, and others lack this integration, making them less suitable for big data processing pipelines.
- **Economical Storage**:
  - HDFS is designed to run on commodity hardware, reducing costs compared to the more specialized hardware often required for traditional file systems.

- This cost efficiency makes HDFS appealing for large-scale data storage and processing needs.

- **Handling of Metadata**:
  - In traditional file systems, metadata management can become a bottleneck as the system scales.
  - HDFS uses a dedicated NameNode to manage metadata efficiently, even in large clusters.

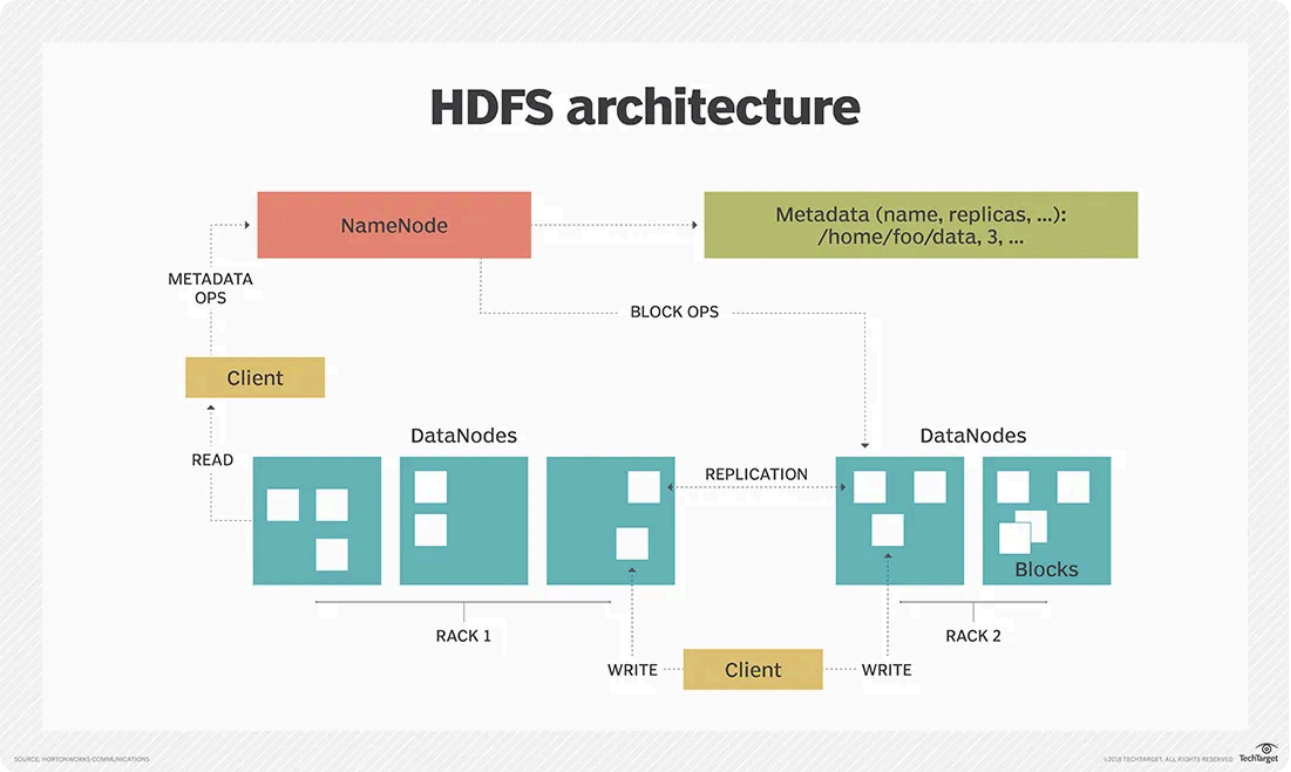


HDFS

Hadoop Distributed File System(HDFS) is a distributed file system designed for handling and storing very large files running on clusters of commodity hardware. It is highly fault tolerant, designed to be deployed on low-cost, commodity hardware and It provides a very high throughput by providing parallel data access.

HDFS instance may consists of hundreds of server machines each storing part of file system data, hence failure of at least one server is inevitable. HDFS has been built to detect these failures and automatically recover them quickly.

HDFS follows master/slave architecture with **NameNode** as master and **DataNode** as slave. Each cluster comprises a **single master node** and **multiple slave nodes**. Internally the files get divided into one or more ***blocks***, and each block is stored on different slave machines depending on the ***replication factor***.



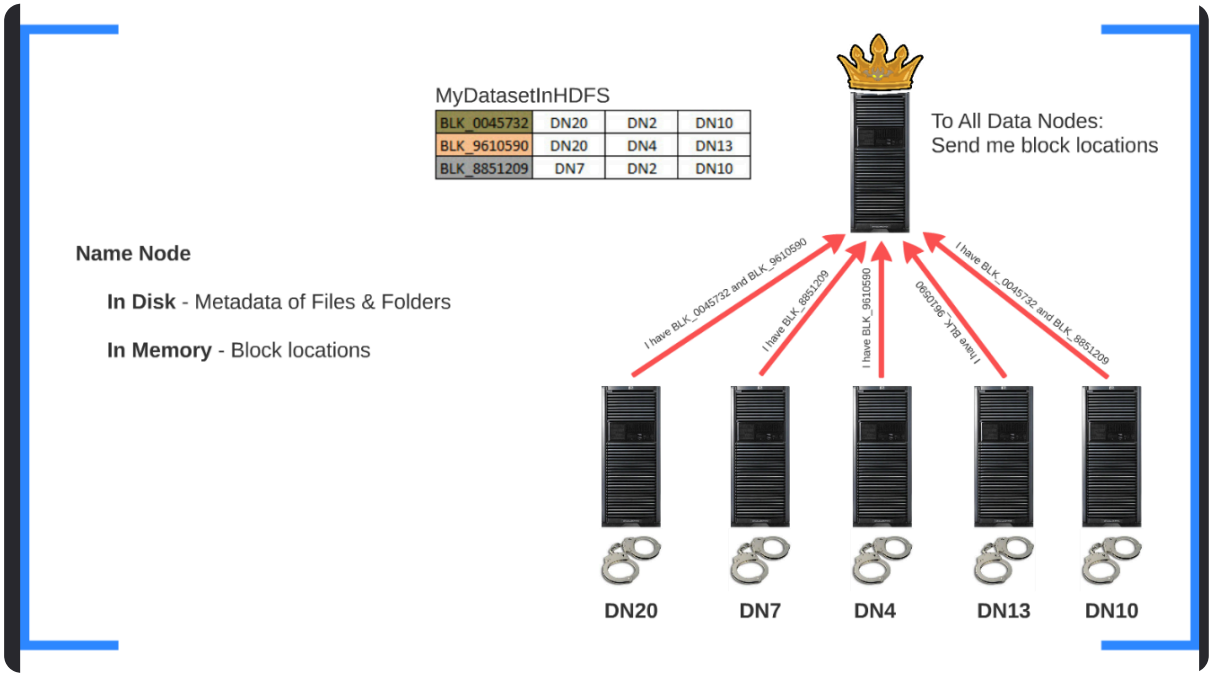
NameNode

- It is the hardware that contains the operating system(GNU/Linux) and the NameNode software. It is responsible for serving the client's read/write requests.
- It Stores metadata such as number of blocks, their location, permission, replicas and other details on the local disk in the form of two files:
  - **FSImage (File System Image)**: It contains the complete namespace of the Hadoop file system since the NameNode creation.
  - **Edit log**: It contains all the recent changes performed to the file system namespace to the most recent **FSImage**.
- NameNode maintains and manages the slave nodes, and assigns tasks to them. It also keeps the status of data node and make sure that it is alive.
- It can manage the files, control a client's access to files, and overseas file operating processes such as renaming, opening, and closing files.

NAMENODE

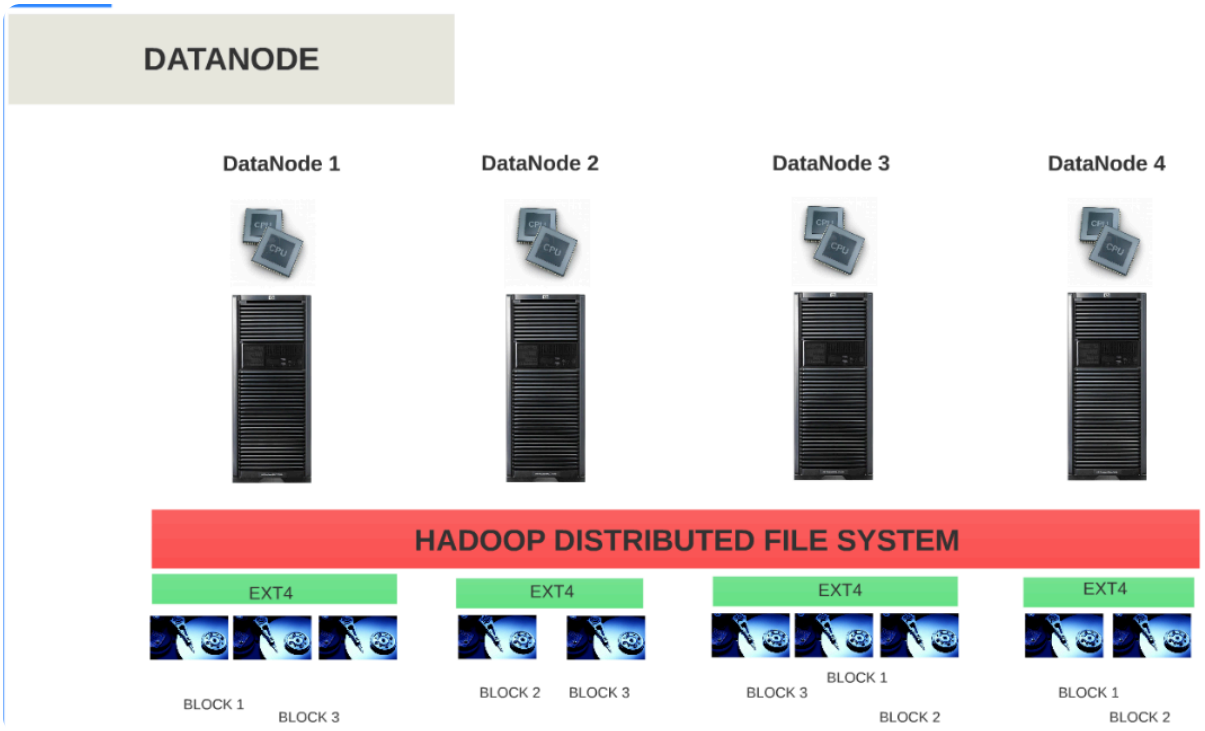
aka Master

File Name - MyDatasetInHDFS
File Size - 350 MB
Replication Factor - 3
Blocks - BLK_0045732, BLK_9610590, BLK_8851209
Block Locations
Permissions
Created By, Created On
Lasted Modified By, Last Modified On

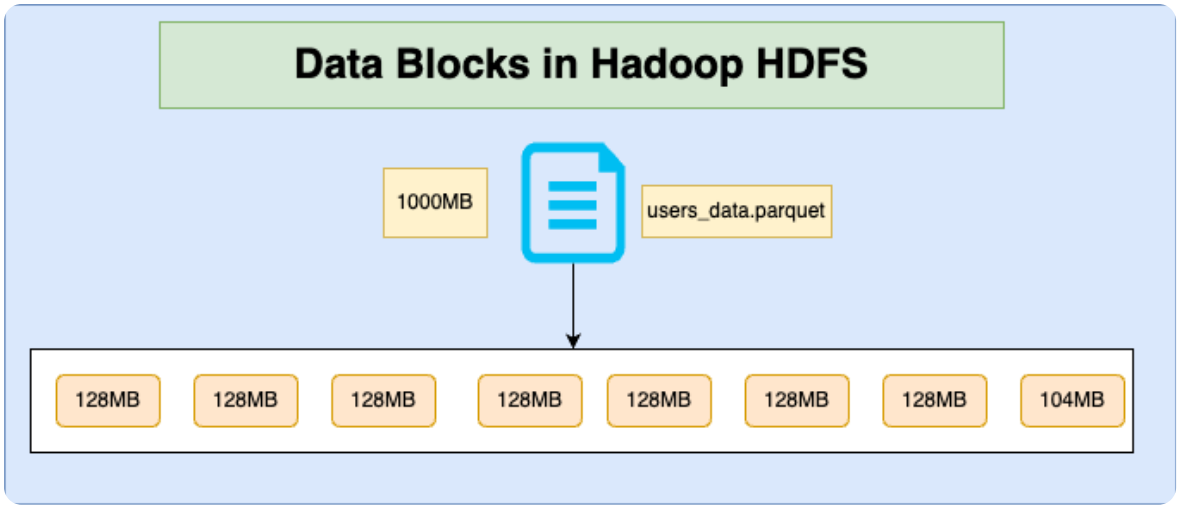


DataNode

- For every node in the HDFS cluster we locate a DataNode which is hardware consisting operating system(GNU/Linux) and a DataNode software which help to control the data storage of their system as they can perform operations on the file systems if the client requests.
- It can also create, replicate, and block files when the NameNode instructs.
- Sends heartbeat and block report to the NameNode to report its health and the list of block it contains respectively.



Blocks



Data Blocks in Hadoop HDFS

Internally HDFS split the file into multiple blocks with the default value of 128MB called block. (64MB in Version 1)

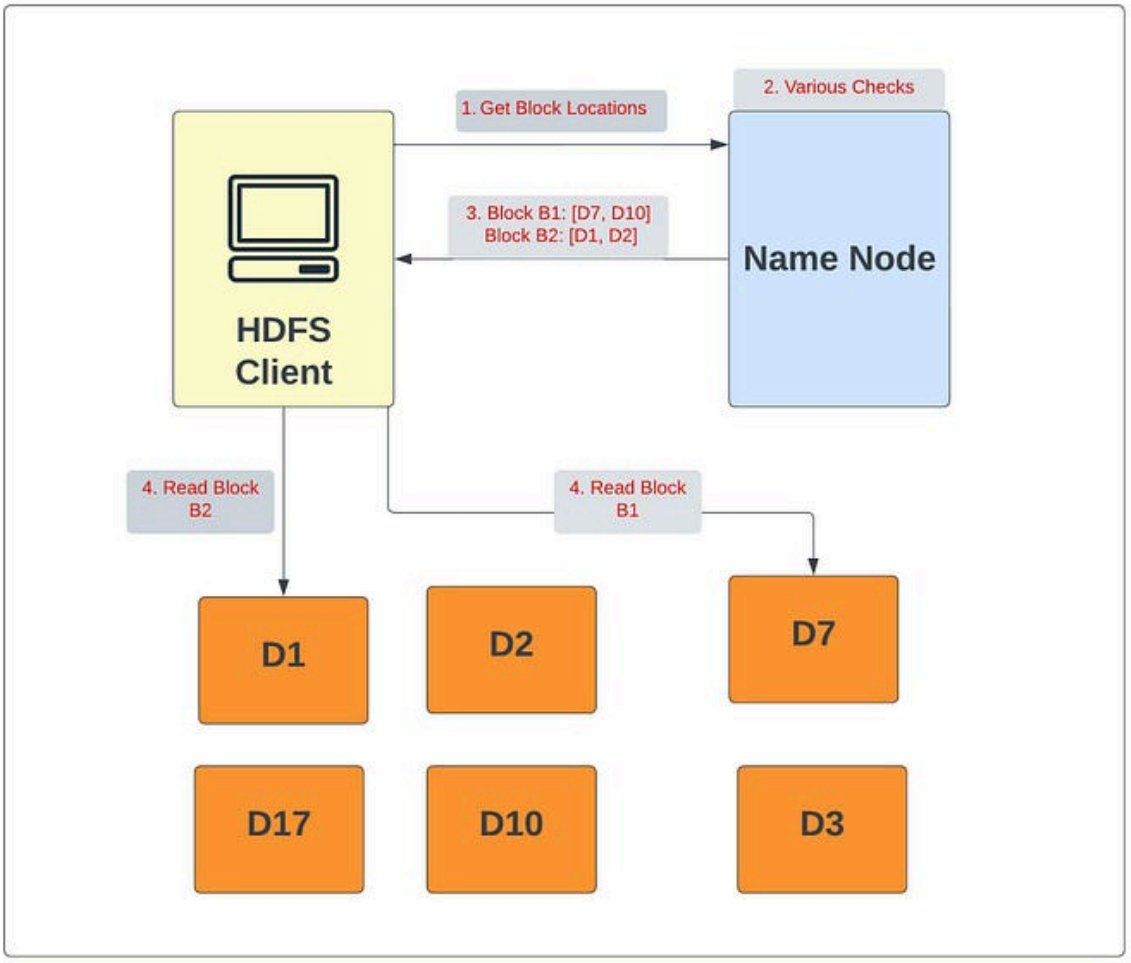
Rack

Rack is the collection of around 40-50 machines (DataNodes) connected using the same network switch. If the network goes down, the whole rack will be unavailable.

Rack Awareness in Hadoop is the concept that chooses DataNodes based on the rack information in the large Hadoop cluster, to improve the network traffic while reading/writing the HDFS file and to store replicas and provide latency and fault tolerance. For the default replication factor of 3, rank awareness algorithm will first store the replica on the local rack, while the second replica will get stored on DataNode in same rank and the third replica will get stored in different rack.

Understanding HDFS Read and Write Operation

Read Operation



Block Diagram for HDFS read Operation

To read from HDFS, the client first communicates with the NameNode for metadata. The NameNode responds with the locations of DataNodes containing blocks. After receiving the DataNodes locations, the client then directly interacts with the DataNodes.

The client starts reading data in parallel from the DataNodes based on the information received from the NameNode. The data will flow directly from the DataNode to the client.

In the above image we can see that first client interact with Name Node to get the location of DataNode containing blocks B1 and B2. NameNode returns a list of DataNodes for each block. For Block B1 if DataNode D7 fails or data block is corrupted then next node in the list D10 will be picked up. Similarly for Block B2, if DataNode D1 fails or if data blocks are corrupted, then D2 will be picked up.

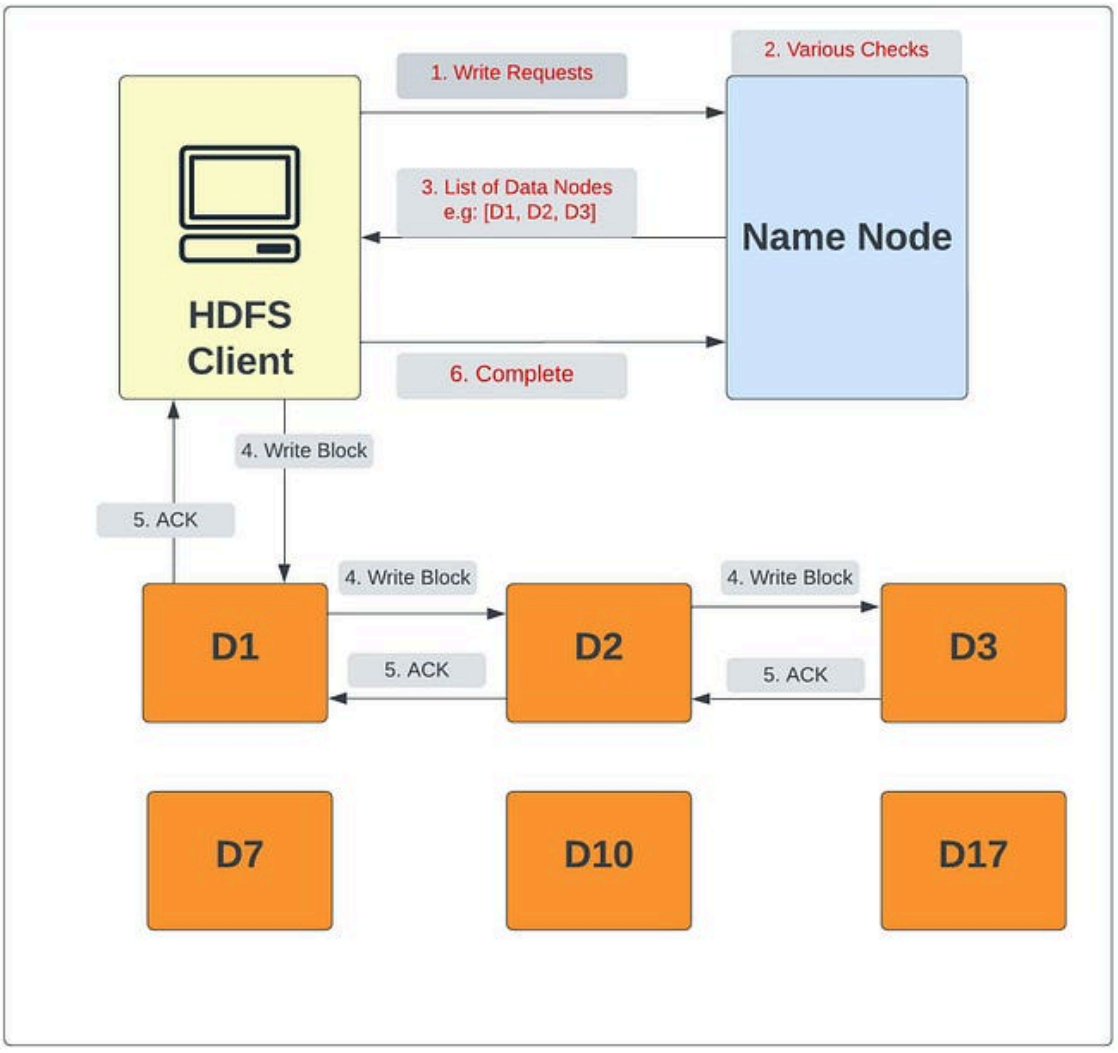
Failure Cases can be summarised as follows:

- Data block is corrupted:
  - Next node in the list is picked up.
- Data Node fails:

- Next node in the list is picked up.
- That node is not tried for the later blocks.

When a client or application receives all the blocks of the file, it combines these blocks into the form of an original file.

Write Operation



Block Diagram for HDFS write Operation

When a client wants to write a file to HDFS, it communicates to the NameNode for metadata. Name Node checks whether file is available or not as well as whether client is authorised or not (performs various checks) and then the NameNode responds with a number of blocks, their location, replicas, and other details. Based on information from NameNode, the client directly interacts with the DataNode.

In the above image we see that once client get the list of DataNode, it interact directly with them. Since the default replication factor for a block is 3, NameNode provides 3 DataNodes D1, D2 and D3 for the write request from the client. Data block is written and replicated in these 3 DataNodes and step 3, 4 and 5 will be repeated until whole file is written on HDFS. In case 1 of the Data Node fail, the data is written to the remaining 2 nodes and NameNode notices under-replication

and arrange extra node for the replication. Once required replicas are created acknowledgement to the client is sent.

DataNode Failure

- In Hadoop, HDFS sends heartbeat signals to NameNode every 3 seconds.
- Every 3 seconds, datanode sends a signal to the namenode, which is called as `heart beat`.
- The signal includes the details about the node.
- If the signal is not received within 3 seconds, then the namenode will wait for 10 minutes. If the signal is not received in 10 minutes, then it declares the datanode as deadnode.

Secondary Namenode

- The secondary NameNode merges the fsimage and edits log files periodically and keeps the edits log size within a limit. It is usually run on a different machine than the primary NameNode since its memory requirements (hardware configuration) are in the same order as the primary NameNode.

NameNode Failure in HDFS

Overview of NameNode in HDFS

- The NameNode is the critical component of the Hadoop Distributed File System (HDFS) responsible for managing the file system namespace and metadata, including file locations and permissions.
- The NameNode does not store the actual data but tracks where data blocks are stored across the DataNodes.
- Because of its central role, the failure of the NameNode can significantly impact the availability and functionality of the HDFS.

1. Recoverable Failure of NameNode

- **Definition**: A recoverable failure is when the NameNode encounters an issue that can be resolved without losing any metadata or requiring complex recovery procedures.
- **Examples**:
  - **Software Crash**: The NameNode might crash due to software bugs, but the metadata and logs remain intact on disk.
  - **Temporary Network Issues**: The NameNode might lose connectivity with some DataNodes temporarily, which could lead to an inability to respond to client requests.
  - **Out-of-Memory Errors**: Insufficient memory allocation might cause the NameNode to fail.
- **Recovery Process**:

- **Automatic Restart**: HDFS can be configured to automatically restart the NameNode in case of a crash.
- **Manual Intervention**: If the failure is due to configuration or resource limitations, an administrator might need to adjust settings and restart the NameNode.
- **No Data Loss**: In these scenarios, no data loss occurs, and the HDFS can resume normal operations after the NameNode is back online.

2. Non-recoverable Failure of NameNode

- **Definition**: A non-recoverable failure is when the NameNode fails in such a way that it results in a loss of metadata or when the system cannot recover without manual intervention and possible data loss.
- **Examples**:
  - **Disk Failure**: If the disk storing the NameNode's metadata (namespace image and edit logs) fails, it may lead to data loss.
  - **Corruption of Metadata**: If the metadata files become corrupted and no backup is available, it can lead to severe data loss.
  - **Catastrophic Hardware Failure**: Physical damage to the NameNode server, such as from fire or flood, leading to the loss of metadata.
- **Recovery Process**:
  - **Rebuild from Backup**: If backups of the metadata (such as fsimage and edit logs) are available, the NameNode can be rebuilt.
  - **Data Reconstruction**: In some cases, manual reconstruction of the file system may be necessary, which can be complex and may result in partial data loss.
  - **Potential Data Loss**: Non-recoverable failures might lead to data loss, especially if recent changes are not captured in the backups.

Checkpoint Node

- **Role**: The Checkpoint Node is responsible for creating checkpoints by merging the namespace image with the edit logs.
- **Operation**:
  - Periodically, the Checkpoint Node downloads the latest fsimage and edit logs from the NameNode.
  - It merges these into a new fsimage, which is then uploaded back to the NameNode.
- **Benefits**:
  - Reduces the size of the edit logs, improving the start-up time of the NameNode.
  - Helps in maintaining a more up-to-date backup of the file system metadata.

Backup NameNode

- **Role**: The Backup NameNode is similar to the Checkpoint Node but also keeps an in-memory copy of the file system metadata.
- **Operation**:
  - Like the Checkpoint Node, the Backup NameNode merges the fsimage and edit logs.
  - In case of failure, it can provide an additional layer of backup for faster recovery.
- **Difference from Standby NameNode**:
  - The Backup NameNode is not involved in active failover; it is more of a support role, whereas the Standby NameNode is designed to take over immediately during a failure.

3. High Availability (HA) in HDFS

- **Concept**: High Availability (HA) in HDFS ensures that the system can continue functioning even if the NameNode fails. This is achieved by setting up two NameNodes in an active-passive configuration.
- **Active NameNode**: The Active NameNode handles all client requests and manages the metadata.
- **Standby NameNode**: The Standby NameNode is a backup that continuously synchronizes with the Active NameNode to keep an up-to-date copy of the metadata.
- **Automatic Failover**:
  - If the Active NameNode fails, the Standby NameNode automatically takes over with minimal disruption.
  - Tools like **Zookeeper** are used for leader election and to manage the failover process.
- **No Single Point of Failure**: With HA, the HDFS is more resilient as it removes the single point of failure inherent in a single NameNode setup.

Summary

- The NameNode is crucial in HDFS, and its failure can lead to significant disruption.
- Recoverable failures allow for a straightforward restart or intervention without data loss, while non-recoverable failures may result in metadata loss and require more complex recovery.
- High Availability configurations, including Active and Standby NameNodes, enhance HDFS resilience by ensuring continued operation in the event of a NameNode failure.

- The Checkpoint Node and Backup NameNode provide additional layers of protection by maintaining and merging metadata, facilitating quicker and more reliable recovery processes.

Before Hadoop2, NameNode was the single point of failure. **The High Availability Hadoop cluster architecture** introduced in Hadoop 2, allows for two or more NameNodes running in the cluster in a hot standby configuration.