HDFS Notes

1. Problems with Big Data

- **Storing and processing large datasets** using traditional systems is inefficient.
- **Storage issues**: Traditional storage systems are not scalable enough to store massive volumes of data.
- **Processing issues**: Processing large datasets in a single machine or centralized system becomes slow and resource-intensive.

- **Hadoop Distributed File System (HDFS)** addresses these challenges by distributing both data storage and processing across many machines.

---

2. Hadoop Components

Hadoop has two key functionalities:

1. **HDFS (Hadoop Distributed File System)**:
- Used for **storing large datasets** across multiple machines.
- Provides high **fault tolerance** and **scalability**.
- Ensures data is **distributed** across a network of machines.

2. **YARN and MapReduce**:
- **YARN (Yet Another Resource Negotiator)** manages resources and schedules tasks across the cluster.
- **MapReduce** handles data **processing** through parallel computation.

---

3. HDFS Daemons (Background Services)

HDFS operates using two main daemons:

- **NameNode (Master)**:
- Responsible for **managing metadata** such as file names, directories, and block locations.
- Does not store actual data, only maintains information on which blocks are stored on which DataNodes.

- **DataNode (Slave)**:

- Stores the **actual data** in blocks.

- Handles **read and write** requests as per instructions from the NameNode.

4. YARN Daemons

- **Resource Manager (Master)**:

- Manages **resources** across all the nodes in the cluster.

- Allocates resources for processing jobs.

- **Node Manager (Slave)**:

- Runs on each DataNode and manages **resource usage** (CPU, memory) for processing tasks on that specific node.

---

5. HDFS Architecture

The architecture follows a **master-slave** design.

- **NameNode**:

- **Master daemon** that stores metadata and manages the HDFS namespace.

- Keeps track of which DataNode holds which block of data.

- If the NameNode fails, HDFS becomes unavailable unless High Availability (HA) is configured.

- **DataNodes**:

- **Slave daemons** that store the actual blocks of data.

- Periodically send **heartbeat signals** to the NameNode to confirm they are functioning.

- If a DataNode fails, the NameNode replicates the lost data to other DataNodes based on the replication factor.

---

6. Blocks in Hadoop

- Data in HDFS is stored in **blocks**. This allows large files to be split and stored across multiple nodes.
- The default block size in Hadoop is **128 MB**, though it can be configured based on requirements.
- Example: A **248 MB** file will be stored as:
- **Block A**: 128 MB
- **Block B**: 120 MB

- **Advantages of block storage**:
- **Parallel processing**: Different blocks can be processed simultaneously.
- **Fault tolerance**: Replication of blocks across nodes ensures data redundancy.

---

7. Block Replication
- Hadoop uses **block replication** to ensure data reliability and fault tolerance.
- By default, each block is **replicated 3 times** and stored on different DataNodes.
- The replication factor can be **customized** at the file or directory level.

Benefits:
- **Fault tolerance**: If one DataNode fails, copies of the data still exist on other nodes.
- **Data locality**: Hadoop tries to store replicas on nodes that are geographically and logically close to reduce network traffic during read operations.

---

8. Rack Awareness
- Hadoop follows a **Rack Awareness Algorithm** to store data intelligently across racks in the data center.
- A **rack** is a group of DataNodes that are physically located close to each other.

How it works:
- The **NameNode** uses rack awareness to distribute block replicas across different racks.
- Default replication:
- One replica is stored on the same rack as the original block.

- Two replicas are stored on different racks.
- This minimizes data loss and **network congestion** when transferring data between nodes.

Advantages:
- Provides **high availability** even if a complete rack fails.
- Improves **network performance** by reducing cross-rack data traffic.

---

9. HDFS Read/Write Mechanism

Write Mechanism:
1. **Client sends a write request** to the NameNode.
2. The NameNode **allocates DataNodes** to store the blocks and provides their locations to the client.
3. The client starts writing data to the first DataNode, which in turn replicates the data to other assigned DataNodes.
4. Each DataNode sends **acknowledgments** back to the client after successfully storing the block.
5. Once replication is complete, the client informs the NameNode that the **write is successful**.

Read Mechanism:
1. **Client sends a read request** to the NameNode.
2. The NameNode provides the **locations (IP addresses)** of the DataNodes holding the requested blocks.
3. The client directly connects to the closest DataNode and **reads the data** from there.
4. If a DataNode is down, the client retrieves the data from one of the replicated DataNodes.

---

10. Additional HDFS Commands

Basic Commands:
- **hadoop version**: Displays the installed Hadoop version.
- **jps**: Shows the list of currently running Java processes on the node.

File Operations:

- **hdfs dfs -ls /**: Lists all files and directories in HDFS.
- **hdfs dfs -put \<local file> \<HDFS path>**: Uploads a file from the local filesystem to HDFS.
- Example: `hdfs dfs -put abc.txt /` (uploads `abc.txt` to the HDFS root directory).
- **hdfs dfs -get \<HDFS file> \<local path>**: Downloads a file from HDFS to the local file system.
- Example: `hdfs dfs -get /abc.txt ./` (downloads `abc.txt` from HDFS to the current local directory).
- **hdfs dfs -cp \<source> \<destination>**: Copies files within HDFS.
- Example: `hdfs dfs -cp /file1 /file2`

File Permissions:

- **hdfs dfs -chmod [permissions] [file/directory]**: Changes the permissions of a file or directory.
- Example: `hdfs dfs -chmod 755 /abc.txt`
- **hdfs dfs -chown [owner] [file/directory]**: Changes the owner of a file or directory.
- Example: `hdfs dfs -chown user1 /abc.txt`

Filesystem Health & Integrity:

- **hadoop fsck /**: Checks the health of the HDFS filesystem.
- Example: `hadoop fsck /` (checks for corrupt files, missing blocks, etc.).
- **hdfs dfsadmin -report**: Displays a report on HDFS, showing details about storage capacity, block count, replication status, and more.

Deleting Files:

- **hdfs dfs -rm \<file>**: Deletes a file from HDFS.
- Example: `hdfs dfs -rm /abc.txt`
- **hdfs dfs -rm -r \<directory>**: Recursively deletes a directory and its contents.
- Example: `hdfs dfs -rm -r /myfolder`

Miscellaneous:

- **hdfs dfs -du [directory]**: Displays the disk usage of a directory and its contents.
- Example: `hdfs dfs -du /myfolder`
- **hdfs dfs -tail [file]**: Displays the last few lines of a file.
- Example: `hdfs dfs -tail /abc.txt`

- **hdfs dfs -help**: Displays help information for HDFS commands.

II. Summary of Key Concepts

- **HDFS**: A distributed file system designed to store large datasets reliably.
- **NameNode**: Manages metadata and coordinates data storage.
- **DataNode**: Stores actual data in blocks.
- **Block Size**: Default block size is 128 MB.
- **Replication**: Default replication factor is 3 for high availability.
- **Rack Awareness**: Ensures data is distributed across racks to prevent data loss.

[redacted]