Topic 4: HDFS (Hadoop Distributed File System)

1. HDFS Overview and Design Goals

What HDFS is Designed For

1. **Very large files**
    - Files from gigabytes to terabytes
    - Optimized for large sequential reads/writes
2. **Streaming data access**
    - Write-once, read-many access pattern
    - Optimized for high throughput, not low latency
3. **Commodity hardware**
    - Designed to run on inexpensive, standard hardware
    - Assumes hardware failures will occur

What HDFS is NOT Designed For

1. **Low-latency data access**
    - Not suitable for real-time applications
    - Millisecond-level response not supported
2. **Lots of small files**
    - Each file creates metadata overhead on NameNode
    - Memory limitation on NameNode for metadata
3. **Multiple writers or arbitrary file modifications**
    - No concurrent write access
    - No random write operations
    - Append-only after initial write

2. HDFS Architecture

Key Components

1. NameNode (HDFS Master)

Role: Manages the filesystem metadata

Functions:

- Maintains the **filesystem metadata**
    - File hierarchy structure

    - Block locations for each file
    - File permissions and ownership
- Manages **user access** to data files
- Maps **data blocks to DataNodes** in the cluster
- Performs **filesystem operations**
    - Opening and closing files
    - Creating and deleting directories
- Provides **registration services** for DataNodes
- Receives **periodic heartbeats** from DataNodes
- **Does NOT store actual file data**

Critical Note: Without NameNode, the entire filesystem is inaccessible!

2. DataNode (HDFS Slave)

Role: Stores actual data blocks

Functions:

- Provides **block storage** on local filesystem
- Stores blocks on **native filesystem** (ext3, ext4, xfs)
- Fulfills **read/write requests** from clients
- Performs **data replication** across cluster
- Sends **periodic block reports** to NameNode
    - Lists all blocks being managed
- Sends **heartbeats** to NameNode
    - Confirms DataNode is alive and healthy
- Performs block operations (creation, deletion, replication)

3. SecondaryNameNode

Role: Backup and recovery support

Functions:

- **Periodically backs up** metadata from primary NameNode
- Creates **checkpoints** of filesystem metadata
- **Not a hot standby** - for recovery only
- Helps expedite filesystem metadata recovery
- Reduces recovery time after NameNode failure

4. Standby NameNode (Optional)

Role: High availability

Functions:

- Runs together with primary NameNode
- **Hot standby** that can take over immediately
- Provides **high availability** (HA)

- Maintains up-to-date metadata
- Enables zero-downtime failover

3. HDFS Data Storage Model

Virtual Filesystem Concept

- HDFS appears as **one filesystem to clients**
- Data actually stored in **multiple different locations**
- Built **on top of native filesystems** (ext3, ext4, xfs in Linux)
- Abstracts physical storage complexity

Block-Based Storage

Block Characteristics

- Each file divided into **blocks**
- **Default block size**: 128 MB (configurable)
- **Default replication factor**: 3 copies (configurable)
- Much larger than traditional filesystem blocks (typically 4KB)

Why Large Blocks?

1. **Reduces metadata overhead**: Fewer blocks = less metadata
2. **Optimizes sequential I/O**: Better for large files
3. **Minimizes seek time**: Reduces disk head movement
4. **Network efficiency**: Fewer network transfers

Data Replication Strategy

Default Replication (Factor = 3)

Block 1: Rack 1 Node 1, Rack 1 Node 2, Rack 2 Node 1 **Block 2**: Rack 1 Node 2, Rack 2 Node 1, Rack 2 Node 2

Rack Awareness

- First replica: Same node as client (or random if client not in cluster)
- Second replica: Different node in **same rack**
- Third replica: Different node in **different rack**
- Balances reliability and network bandwidth

Benefits of Replication

1. **Fault tolerance**: Data survives node failures
2. **High availability**: Multiple copies accessible
3. **Load balancing**: Read from nearest replica
4. **Data locality**: Process data where it's stored

4. Logical vs Physical View

Logical View (Client Perspective)

```
/user/bigdata/data.txt (500 MB)
- Single file
- Simple path
```

Physical Implementation

```
/user/bigdata/data.txt
├─ Block 1 (128 MB) → DataNode 1, DataNode 3, DataNode 5
├─ Block 2 (128 MB) → DataNode 2, DataNode 4, DataNode 6
├─ Block 3 (128 MB) → DataNode 1, DataNode 4, DataNode 7
└─ Block 4 (116 MB) → DataNode 2, DataNode 3, DataNode 8

Each block replicated 3 times across different nodes/racks
```

5. HDFS Read Operation (Detailed)

Step-by-Step Process

Step 1: Client opens file

- Calls `open()` on **DistributedFileSystem** object
- FileSystem object created for HDFS

Step 2: NameNode query

- DistributedFileSystem calls NameNode via **RPC**
- Requests locations of first few blocks
- NameNode returns list of DataNodes per block
- Sorted by proximity to client

Step 3: FSDataInputStream returned

- Client receives **FSDataInputStream** object
- Client calls `read()` on stream
- Stream manages DataNode connections

Step 4: Stream to DataNode

- FSDataInputStream connects to **first (nearest) DataNode**
- Data streamed from DataNode to client
- Client calls `read()` repeatedly

Step 5: Block completion

- When block ends, FSDataInputStream closes connection
- Finds best DataNode for **next block**

- May reuse same DataNode if it has next block

Step 6: File completion

- Client calls `close()` on FSDataInputStream
- All resources released

Read Optimization

- **Data locality**: Read from nearest replica
- **Rack awareness**: Prefer same rack to reduce network traffic
- **Automatic failover**: Switch to another replica if DataNode fails
- **Parallel reads**: Can read multiple blocks simultaneously

6. HDFS Write Operation (Detailed)

Step-by-Step Process

Step 1: Client creates file

- Calls `create()` on **DistributedFileSystem**
- Initiates file creation process

Step 2: NameNode operations

- DistributedFileSystem makes **RPC call** to NameNode
- NameNode creates new file in namespace
- Checks permissions and existing files
- Returns **FSDataOutputStream** to client

Step 3: Client writes data

- Client writes data to **FSDataOutputStream**
- Data buffered internally

Step 4: Data packaging

- FSDataOutputStream splits data into **packets**
- Packets queued in internal **data queue**
- Typical packet size: 64KB

Step 5: Pipeline creation

- Packets sent to **first DataNode** in pipeline
- First DataNode forwards to **second DataNode**
- Second DataNode forwards to **third DataNode**
- Creates write pipeline (usually 3 nodes)

Step 6: Acknowledgment

- Each DataNode sends **ack** back up pipeline
- When all acks received, packet removed from queue

- FSDataOutputStream maintains **ack queue** for reliability

Step 7: File completion

- Client calls `close()` on stream
- Flushes remaining packets
- Waits for acknowledgments
- Signals NameNode that write is complete

Write Reliability

- **Packet-level acknowledgment**: Ensures data written successfully
- **Ack queue**: Maintains packets until confirmed
- **Pipeline replication**: All replicas written simultaneously
- **Atomic completion**: All-or-nothing guarantee

7. HDFS Fault Tolerance Mechanisms

DataNode Failure Handling

1. **Heartbeat monitoring**
    - DataNodes send heartbeats every 3 seconds (default)
    - NameNode marks node as dead after 10 minutes no heartbeat
2. **Block re-replication**
    - NameNode detects under-replicated blocks
    - Instructs other DataNodes to create new replicas
    - Restores replication factor
3. **Read failover**
    - If DataNode fails during read, client switches to another replica
    - Transparent to application
4. **Write pipeline recovery**
    - If DataNode fails during write, pipeline reconstructed
    - Write continues with remaining nodes

NameNode Failure (High Availability)

1. **Without HA**
    - SecondaryNameNode checkpoint used for recovery
    - Manual intervention required
    - Downtime during recovery
2. **With HA (Standby NameNode)**
    - Automatic failover to Standby NameNode

- Zero or minimal downtime
- Shared storage for edit logs (NFS or QJM)

Key Points for Exam

Critical Concepts

1. **Block-based storage**: Default 128MB, replication factor 3
2. **Write-once-read-many**: Append-only after initial write
3. **NameNode**: Metadata manager (NOT data storage)
4. **DataNode**: Actual data storage
5. **Data locality**: Move code to data, not data to code

HDFS Architecture Components

Master (NameNode):

- Manages metadata
- Coordinates DataNodes
- Single point of management

Slaves (DataNodes):

- Store actual blocks
- Handle I/O operations
- Report to NameNode

Backup (SecondaryNameNode):

- Checkpoints metadata
- Recovery support
- NOT a hot standby

HA (Standby NameNode):

- Hot standby
- Automatic failover
- Zero downtime

Read Process (6 Steps)

1. Open file → DistributedFileSystem
2. Query NameNode → Get block locations
3. Receive FSDataInputStream
4. Connect to DataNode → Stream data
5. Move to next block when complete
6. Close stream when done

Write Process (7 Steps)

1. Create file → DistributedFileSystem
2. NameNode creates entry → Returns FSDataOutputStream
3. Write data to stream
4. Split into packets → Queue packets
5. Create pipeline → Forward to DataNodes
6. Receive acknowledgments
7. Close stream → Signal completion

Block Replication Strategy

- **1st replica**: Same node as client (or random)
- **2nd replica**: Different node, same rack
- **3rd replica**: Different node, different rack
- **Reason**: Balance reliability and network bandwidth

Design Goals Remember

Optimized for:

- Large files (GB to TB)
- Sequential access
- High throughput
- Commodity hardware

NOT optimized for:

- Small files
- Low latency
- Random writes
- Concurrent writers

Important Defaults

- **Block size**: 128 MB
- **Replication factor**: 3
- **Heartbeat interval**: 3 seconds
- **DataNode timeout**: 10 minutes (default)
- **Max versions**: 3 (for versioned files)

Key Features

1. **Fault tolerance**: Through replication
2. **Scalability**: Add more DataNodes
3. **Data locality**: Process where data resides
4. **High throughput**: Optimized for batch processing
5. **Rack awareness**: Intelligent replica placement