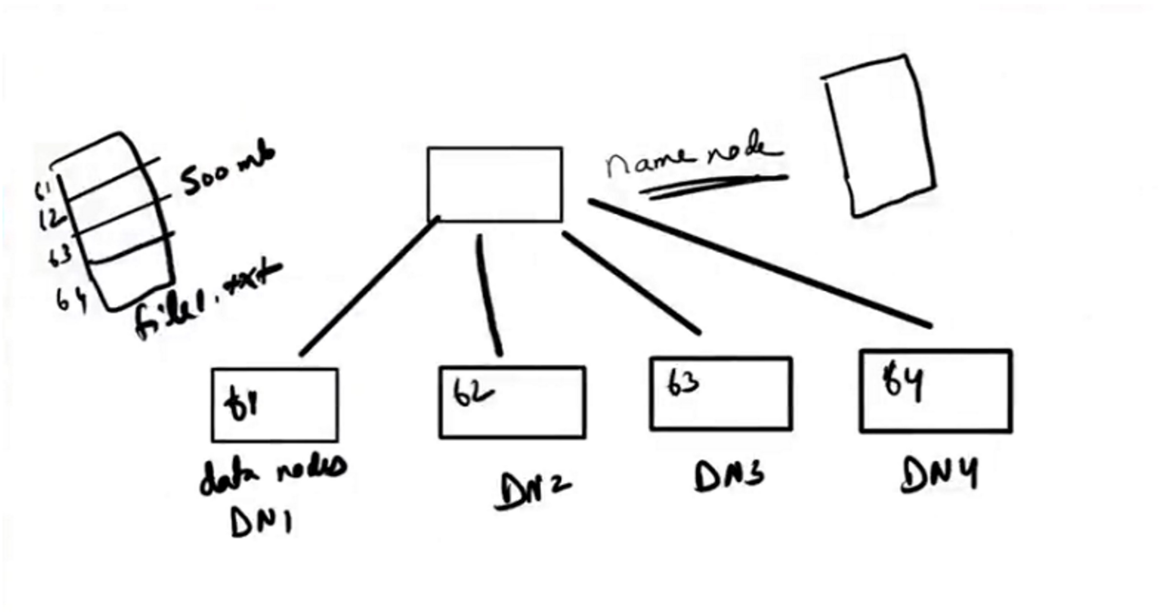
Monday, January 13, 2025 11:51 PM

HDFS (Hadoop Distributed File System): HDFS is a distributed file system designed to store and manage large datasets across multiple machines reliably. It's built to handle files that are too large to store on a single computer by breaking them down into smaller blocks and distributing them across different nodes in a cluster.



Key characteristics of HDFS:

1. Block-based storage: As shown in your example, files are split into fixed-size blocks (default 128MB). A 500MB file would be split into:
   - Block 1: 128MB
   - Block 2: 128MB
   - Block 3: 128MB
   - Block 4: 116MB (remaining data)

2. **Replication:** For fault tolerance, each block is typically replicated across multiple DataNodes (usually 3 replicas by default).

3. Write-once-read-many: HDFS is optimized for sequential writes and reads, not random access.

NameNode in Detail: The NameNode is the "brain" of the HDFS system. It maintains the filesystem namespace and metadata for all files and directories. Let's break down its key responsibilities:

1. Metadata Management:
   - Maintains the filesystem tree and metadata for all files and directories
   - Stores information about block locations, file permissions, and access times
   - Keeps track of which blocks make up each file
   - Records which DataNodes contain which blocks

2. Block Management:
    - Maintains the mapping of file blocks to DataNodes
    - Makes decisions about block replication
    - Monitors DataNode health through heartbeat signals

3. Namespace Operations:
    - Handles operations like opening, closing, and renaming files and directories
    - Manages file system namespace operations like creating and deleting files

4. Client Coordination:
    - When a client wants to read a file, it contacts the NameNode first
    - NameNode provides the client with block locations
    - For writes, NameNode allocates new blocks and decides their placement

Example of NameNode Metadata (based on your example):

Copy

```
File Metadata Table:
File: file1
Total Size: 500MB
Block Distribution:
- Block1 (128MB) → DN1
- Block2 (128MB) → DN2
- Block3 (128MB) → DN3
- Block4 (116MB) → DN4
```

Additional Important Aspects:

1. Single Point of Failure Protection:
    - Secondary NameNode: Takes periodic checkpoints of the namespace
    - EditLog: Records all filesystem changes
    - FSImage: Represents a snapshot of the filesystem metadata

2. Memory Management:
    - All metadata is kept in memory for fast access
    - Periodic checkpoints are written to disk for persistence
    - EditLog records all changes between checkpoints

3. Block Report Processing:
    - Receives block reports from DataNodes
    - Validates block replicas
    - Maintains the overall health of the file system

4. Load Balancing:
    - Ensures even distribution of blocks across DataNodes
    - Manages replication factor requirements
    - Handles DataNode failures and recovery

client wants to read file1 from hdfs the request will go to the namenode name node will tell from where to read the blocks consider like a 1000 page book you have an index page at the start of the book 1000 data pages your index page is more like your name node and your actual pages are more like your data nodes

HDFS --> storage
Spark --> processing
YARN --> Resource manager

All about Block size
==================

why the block size is 128 mb?

what happens if we reduce the block size

64 mb

1 GB file - default block size is 128 mb - 8
1 GB file - default block size is 64 mb - 16
1 GB file - default block size is 32 mb - 32

if we reduce the block size leads to more number of blocks

1000 node cluster

if you choose a block size which is very low…

lets say block size is 1 mb you have a file of 10 gb

10240 blocks

your name node has to keep the metadata of all of it in its table..

your namenode will get overburdened with these many entries..

what if we increase the block size…

1 GB file - default block size is 128 mb - 8
1 GB file - default block size is 256 mb - 4
1 GB file - default block size is 512 mb - 2
128 mb is a good number…

if you data grows then you can add more data nodes when your datanodes increase to a certain level then your metadata also would have grown up..

Block Size Implications:

1. Default Block Size (128 MB): For a 1 GB file:
- Number of blocks = 1024 MB ÷ 128 MB = 8 blocks
- This is considered optimal because:
    - Manageable number of metadata entries
    - Good balance between seek time and data transfer
    - Efficient for most MapReduce jobs

2. Smaller Block Size (64 MB): For a 1 GB file:
- Number of blocks = 1024 MB ÷ 64 MB = 16 blocks Issues:
- Doubles the metadata entries in NameNode
- Increases memory usage in NameNode
- More block reports to process
- Higher network overhead for block management

3. Very Small Block Size (32 MB or less): For a 1 GB file:
- At 32 MB: 1024 MB ÷ 32 MB = 32 blocks
- At 1 MB: 1024 MB ÷ 1 MB = 1024 blocks

Problems with Very Small Blocks:

1. NameNode Overhead:
    - Each block requires metadata storage
    - More blocks = More memory consumption
    - Higher processing overhead for block reports
    - Increased load on NameNode's memory and CPU

2. Network Impact:
    - More blocks mean more block reports
    - Increased network traffic for metadata management
    - Higher overhead in client-NameNode communication

Example of Small Block Impact: For a 10 GB file with 1 MB blocks:

- Number of blocks = 10240 blocks
- Each block needs metadata entry

- NameNode must track:
    - Block locations
    - Replication status
    - Block health
    - Permissions
    - Timestamps

Larger Block Sizes:

1. 256 MB blocks:
    - 1 GB file = 4 blocks
    - Pros:
        - Less metadata
        - Reduced NameNode overhead
    - Cons:
        - Less parallelism for processing
        - Higher memory requirements for processing

2. 512 MB blocks:
    - 1 GB file = 2 blocks
    - Similar trade-offs but more extreme

Why 128 MB is Optimal:

1. Scalability:
    - Balanced growth of metadata with data
    - Manageable even when adding more DataNodes
    - Supports cluster expansion without overwhelming NameNode

2. Processing Efficiency:
    - Good size for most MapReduce jobs
    - Efficient data transfer vs. seek time ratio
    - Suitable for most common file sizes

3. Resource Usage:
    - Reasonable memory requirements
    - Efficient network utilization
    - Balanced storage distribution

4. Future Growth:
    - When data grows, you can add more DataNodes
    - Metadata growth remains proportional
    - NameNode can handle the increased load efficiently

Best Practices:

1. Block Size Considerations:
    - Consider average file size in your system
    - Evaluate processing patterns
    - Account for hardware capabilities
    - Plan for future growth

2. Cluster Planning:
    - Start with default 128 MB
    - Monitor NameNode memory usage
    - Track metadata growth
    - Adjust based on workload patterns

3. Performance Optimization:
    - Balance between too many and too few blocks
    - Consider network capacity
    - Account for processing requirements
    - Plan for scalability

The 128 MB block size represents a sweet spot that balances:

- Metadata management overhead
- Processing efficiency
- Network utilization
- Storage distribution
- Future scalability needs

Name Node Federation:

NameNode Federation is an architectural enhancement in HDFS that allows multiple independent NameNodes to coexist in a single Hadoop cluster. Each NameNode manages a specific portion of the filesystem namespace, enabling horizontal scaling of the namespace and improving overall availability.

Key Concepts of NameNode Federation:

1. Namespace Volume:

- Each NameNode manages an independent namespace volume
- Contains its own namespace and block pool
- Example structure:

  Copy
  ```
  NameNode1 → /user, /home
  NameNode2 → /data, /analytics
  NameNode3 → /archive, /backup
  ```

2. Block Pool:

- Collection of blocks belonging to a single namespace

- Managed independently by each NameNode
- Block Pool ID (BPID) uniquely identifies each pool
- DataNodes store blocks from multiple block pools

Architecture Components:

1. Multiple Independent NameNodes:

Copy
```
NameNode1:
- Namespace: /user
- Block Pool: BP1
- Metadata: Own FSImage and EditLog
NameNode2:
- Namespace: /data
- Block Pool: BP2
- Metadata: Own FSImage and EditLog
```

1. Block Pool Management:

- Each DataNode registers with all NameNodes
- Stores blocks from multiple block pools
- Reports block status to respective NameNodes

- Example:

  Copy
  ```
  DataNode1:
  - Blocks from BP1
  - Blocks from BP2
  - Blocks from BP3
  ```

Benefits:

1. Scalability:

- Horizontal scaling of namespace
- Better resource utilization
- Increased number of files and blocks
- Example:

  Copy
  ```
  Before Federation:
  - Single NameNode
  - 100 million files limit

  After Federation:
  - 3 NameNodes
  ```

```
- 300 million files possible
```

2. Performance:
- Load distribution across NameNodes
- Improved throughput
- Reduced response time
- Parallel namespace operations
3. Isolation:
- Namespace isolation for different applications
- Independent failure domains
- Separate administration possible
- Example:

  Copy
  ```
  NameNode1: Production data
  NameNode2: Test data
  NameNode3: Archive data
  ```

Implementation Details:
1. Client Interaction:

Copy
```
ViewFs (Client-side mount table):
/user    → nn1:8020/user
/data    → nn2:8020/data
/archive → nn3:8020/archive
```

1. DataNode Registration:

Copy
```
DN Registration Process:
1. Get list of all NameNodes
2. Register with each NameNode
3. Receive block pool IDs
4. Start block reporting
```

1. Block Management:

Copy
```
Block Reporting:
- Per block pool basis
- Independent block reports
- Separate heartbeats
```

Operational Considerations:

1. Configuration:

xml

Copy

```
hdfs-site.xml:
<property>
  <name>dfs.nameservices</name>
  <value>ns1,ns2,ns3</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.ns1</name>
  <value>nn1:8020</value>
</property>
```

1. Administration:

- Each NameNode needs:
  - Separate configuration
  - Independent monitoring
  - Own backup strategy
  - Dedicated resource allocation

2. Recovery Procedures:

Copy

```
Failure Scenarios:
- Single NameNode failure affects only its namespace
- Other NameNodes continue operating
- Independent recovery possible
```

Best Practices:

1. Namespace Planning:

- Logical grouping of data
- Consider access patterns
- Plan for growth
- Example:

  Copy

  ```
  NN1: /user, /home (user data)
  NN2: /data (application data)
  NN3: /tmp, /processing (temporary data)
  ```

2. Resource Allocation:

- Adequate memory per NameNode

- CPU resources
- Network capacity
- Storage for metadata

3. Monitoring:

- Track each NameNode's:
  - Memory usage
  - CPU utilization
  - Namespace size
  - Block report processing time

Federation vs. HA:

- Federation: Horizontal scaling
- HA: Fault tolerance
- Can be used together
- Example:

  Copy
  ```
  NN1: Active + Standby (for /user)
  NN2: Active + Standby (for /data)
  NN3: Active + Standby (for /archive)
  ```

Namenode federation - we can have more than one name node. Metadata is split across these namenodes…

Namenode federation is to give scalability

Data node failure
Replication factor - 3

Name node fails Secondary name node

secondary name node - fault tolerance

namenode fedration - scalability

rack

10 datanodes in bangalore (1 rack)
10 datanodes in san francisco (1 rack)
10 datanodes in australia (1 rack)

3 copies are not stored in a single rack