Big Data Analytics: Lecture 8

Based on the slides of Dr. Tariq Mahmood

Fall 2025

1 Introduction to Hadoop

As datasets grow exponentially, they often exceed the storage capacity of a single physical machine. This necessitates partitioning the data across a network of separate machines.

- **Distributed Filesystems:** These are specialized filesystems designed to manage data storage across a network of interconnected machines, presenting them to the user as a single, cohesive filesystem.
- **Increased Complexity:** Because they operate over a network, distributed filesystems must handle all the inherent complexities of network programming, such as latency, bandwidth limitations, and partial failures. This makes them significantly more complex than traditional local disk filesystems.
- **Fault Tolerance:** A primary design goal is to be resilient to node failures. The system must be able to withstand the loss of individual machines without losing any data, ensuring high availability and data durability.

2 HDFS: The Hadoop Distributed Filesystem

HDFS is the primary storage component of the Hadoop ecosystem. It is a distributed filesystem specifically designed for big data workloads.

2.1 Core Design Principles

- **Storing Very Large Files:** HDFS is built to store massive files, typically in the range of hundreds of megabytes (MB), gigabytes (GB), terabytes (TB), or even petabytes (PB).
- **Streaming Data Access:** The system is optimized for a **write-once, read-many-times** access pattern. This means that data is typically written to HDFS once and then read and analyzed multiple times. In this model, the time it takes to read the entire dataset (high throughput) is more critical than the time it takes to read the first record (low latency).
- **Running on Commodity Hardware:** HDFS is designed to run on clusters built from inexpensive, off-the-shelf hardware. It anticipates that failures of such hardware components will be common and builds fault tolerance directly into the software layer, rather than relying on expensive, highly reliable hardware. This allows for massive, cost-effective scaling.

2.2 Use Case Suitability

2.2.1 What HDFS is NOT good for:

- **Low-Latency Data Access:** Applications that require very fast, random access to data in the millisecond range (e.g., transactional databases, real-time user-facing applications) are not a good fit for HDFS. HDFS achieves high throughput at the expense of higher latency.
- **Lots of Small Files:** HDFS is not efficient at storing a large number of small files. This is because the **NameNode**, the master server in HDFS, holds all filesystem metadata (filenames, permissions, block locations) in memory (RAM). Each file, directory, and block requires metadata.

  - **Rule of Thumb (ROT):** The metadata for a single block in HDFS occupies approximately 150 bytes of memory on the NameNode.
  - **Example:** If you store one million files, and each file is small enough to fit in one block, the NameNode would need at least: $1,000,000 \text{ files} \times 150\frac{\text{bytes}}{\text{file}} = 150,000,000$ bytes. To convert this to megabytes: $\frac{\text{150,000,000 bytes}}{\text{1,048,576 bytes/MB}} \approx 143.05$ MB. While this seems small, storing billions of files would require a NameNode with a very large amount of RAM, making it a bottleneck.

3 The Concept of Blocks in HDFS

3.1 Blocks in Traditional vs. Distributed Filesystems

- A physical disk reads and writes data in units called **disk blocks**, which are typically very small (e.g., 512 bytes).
- A traditional filesystem on a single disk also uses blocks, which are multiples of the disk block size, usually a few kilobytes.

3.2 HDFS Blocks

HDFS uses the concept of a block as its fundamental unit of storage, but its blocks are much larger.

- **Large Block Size:** The default block size in modern HDFS is **128 MB** (and can be configured to be even larger).
- **File Chunking:** Files in HDFS are broken into block-sized chunks, and these chunks are stored as independent units across the cluster.
- **Efficient Storage for Small Files:** Unlike a traditional filesystem, a file in HDFS that is smaller than the block size does **not** occupy the full block's worth of disk space.
  - **Example:** A 1 MB file stored in HDFS with a 128 MB block size will only consume 1 MB of physical disk space, not 128 MB.

3.3 Why Is a Block in HDFS So Large?

The primary reason for the large block size is to **minimize the cost of disk seeks**.

- A disk seek is the physical movement of the disk's read/write head to the correct location. This is a time-consuming mechanical operation.
- By making the block size very large, the time spent transferring the data from the disk can be made significantly longer than the time spent seeking to the start of the block.
- This ensures that for large files, the system operates at the disk's maximum transfer rate, providing high throughput.
- **Calculation Example:**
  - Assume disk seek time (T_{seek}) is 10 ms (0.01 s).
  - Assume the disk transfer rate is 100 MB/s.
  - To make the seek time only 1% of the total data transfer time (T_{transfer}), we have:

$$T_{\text{seek}} = 0.01 \times T_{\text{transfer}} \tag{1}$$

$$T_{\text{transfer}} = \frac{T_{\text{seek}}}{0.01} = \frac{0.01 \text{ s}}{0.01} = 1 \text{ s} \tag{2}$$

$$\text{Block Size} = \text{Transfer Rate} \times T_{\text{transfer}} = 100\frac{\text{MB}}{\text{s}} \times 1 \text{ s} = 100 \text{ MB} \tag{3}$$

  This calculation shows that a block size around 100 MB is needed to make seek time negligible. The HDFS default of 128 MB aligns with this principle.

3.4 Benefits of the Block Abstraction

1. **Files Larger Than Any Single Disk:** A file can be larger than any individual disk in the network because its blocks are distributed across the entire cluster.
2. **Simplified Storage Subsystem:** The system manages a uniform unit of storage (the block) rather than variable-sized files. This simplifies storage management, replication, and fault tolerance.
3. **Foundation for Fault Tolerance:** Blocks are a natural fit for replication. To provide fault tolerance, HDFS creates multiple copies (replicas) of each block and stores them on different machines. If one machine fails, the data can be retrieved from another.

The 'hdfs fsck / -files -blocks' command can be used to inspect the block composition of files within HDFS, demonstrating this abstraction in practice.

4 Worked Examples & Problems

4.1 Problem 1: Calculating Blocks

Scenario: You have a file of size 800 MB, and the HDFS block size is set to 128 MB. How many blocks will this file be split into?

- **Calculation:** $\frac{800\text{ MB}}{128\text{ MB/block}} = 6.25$ blocks.
- **Solution:** Since a file is made of whole blocks, HDFS rounds this up. The file will be split into **7 blocks**.
  - 6 full blocks of 128 MB each.
  - 1 final block containing the remainder: $800 - (6 \times 128) = 800 - 768 = 32$ MB. (Note: The slide says 16 MB, which is incorrect for this problem).

4.2 Problem 2: Calculating Disk Space with Replication

Scenario: You have a file of size 1 GB stored in HDFS, and the replication factor is set to 3. How much disk space will be used?

- **Explanation:** The replication factor dictates how many copies of the data are stored.
- **Solution:** Total disk space = File Size $\times$ Replication Factor = 1 GB $\times$ 3 = **3 GB**.

4.3 Problem 3: Multiple Files

Scenario: You have 5 files, each of size 500 MB, and the HDFS block size is 256 MB. How many blocks are needed?

- **Calculation per file:** $\frac{500\text{ MB}}{256\text{ MB/block}} = 1.953...$ blocks. This rounds up to 2 blocks per file.
- **Solution:** Total blocks = 2 blocks/file $\times$ 5 files = **10 blocks**.

4.4 Problem 4: Small File Disk Consumption

Scenario: You have a file of size 10 MB, and the HDFS block size is 128 MB. The replication factor is set to 2. How much disk space will this file consume?

- **Explanation:** A file smaller than the block size only consumes its actual size on disk, not the full block size. The replication factor then multiplies this actual size.
- **Solution:** Total disk space = File Size $\times$ Replication Factor = 10 MB $\times$ 2 = **20 MB**.
- **Note:** This is a common point of confusion. While the NameNode's metadata tracks this as occupying one block location, the physical disk usage on the DataNodes is only 10 MB per replica.

4.5 Problem 5: File Size Conversion

Scenario: You have a file of size 5 GB, and the HDFS block size is 512 MB. How many blocks will this file occupy?

- **File Size Conversion:** First, convert GB to MB. 5 GB $\times 1024 \frac{\text{MB}}{\text{GB}} = 5120$ MB.
- **Calculation:** $\frac{5120 \text{ MB}}{512 \text{ MB/block}} = 10$ blocks.
- **Solution:** The file will occupy exactly **10 blocks**.

4.6 Problem 6: Multiple Block Sizes

Scenario: You have a file of size 2.5 GB. How many blocks are required with a 128 MB block size versus a 256 MB block size?

- **File Size Conversion:** 2.5 GB $\times 1024 \frac{\text{MB}}{\text{GB}} = 2560$ MB.
- **Solution with 128 MB blocks:** $\frac{2560 \text{ MB}}{128 \text{ MB/block}} =$ **20 blocks**.
- **Solution with 256 MB blocks:** $\frac{2560 \text{ MB}}{256 \text{ MB/block}} =$ **10 blocks**.

4.7 Problem 7: Combined Calculation

Scenario: You have 10 files, each of size 300 MB. The HDFS block size is 128 MB, and the replication factor is 3. How much total disk space will be consumed?

- **Total Data Size (Original):** 10 files $\times 300 \frac{\text{MB}}{\text{file}} = 3000$ MB.
- **Solution:** Total disk space = Total Data Size $\times$ Replication Factor = 3000 MB $\times$ 3 = 9000 MB (or approximately 8.79 GB).

5 Hadoop Component Responsibilities

Hadoop's functionality is divided among several key components. Below are detailed tables outlining the responsibilities of each.

5.1 NameNode Tasks

The NameNode is the master server that manages the filesystem namespace and regulates access to files by clients.

NameNode Task	Task Details	Explanation
1. Namespace Management	Maintains the entire HDFS directory tree — file names, directories, permissions, timestamps, and ownership.	Think of it as the "file system table of contents" (like the output of 'ls -lR') for the whole cluster.
2. Block Management	Keeps a mapping of each file to its constituent blocks and each block to the DataNodes that store its replicas.	If a file is 1 GB and split into 8 blocks, the NameNode knows which DataNodes hold each of those 8 blocks.
3. Replication Control	Ensures every block maintains its configured replication factor (default = 3). If a DataNode fails, it schedules replication from other nodes to restore balance.	Acts as a "health monitor" for data, maintaining the required number of copies of each piece.

NameNode Task	Task Details	Explanation
4. Metadata Persistence (EditLog + FsImage)	Periodically writes metadata to disk. **FsImage:** a snapshot of the namespace. **EditLog:** a record of recent changes.	Merges them periodically into a new FsImage (checkpointing). Ensures HDFS can recover its state after a restart or crash.
5. Client Coordination & Access Control	When a client requests to read/write a file, the NameNode: authenticates the user, returns block locations for reading, and chooses target DataNodes for new block writes.	Acts as a "traffic controller" that coordinates clients and DataNodes without transferring the actual data itself.

5.2 DataNode Tasks

DataNodes are the slave nodes that store the actual data blocks.

DataNode Task	Task Details	Explanation
1. Block Storage and Retrieval	Stores the actual HDFS blocks (e.g., 128 MB each) on local disks and reads/writes them upon NameNode or client request.	Acts like the "warehouse shelves" that hold the real data chunks.
2. Block Reporting	Periodically sends a Block Report to the NameNode containing a list of all blocks it currently holds.	Like an inventory list telling the central manager (NameNode) what's in stock.
3. Heartbeat Transmission	Sends regular heartbeats (every 3 seconds by default) to the NameNode to indicate that it's alive and functioning.	A "pulse check" confirming this DataNode is healthy and reachable.
4. Data Replication (and Deletion)	When instructed by the NameNode, the DataNode replicates blocks to other nodes or deletes excess and obsolete copies to maintain the desired replication factor.	Think of a warehouse worker copying or removing boxes as per central orders.
5. Data Transfer between DataNodes (Pipeline)	During writes, DataNodes form a replication pipeline (Node 1 → Node 2 → Node 3) to stream data efficiently. They also handle rebalancing transfers.	Like a production line where data flows through multiple nodes to form 3 copies.

5.3 YARN (ResourceManager) Tasks

YARN (Yet Another Resource Negotiator) is Hadoop's cluster resource management layer. The ResourceManager is the master scheduler.

YARN Task	Task Details	Explanation
1. Resource Management	Allocates CPU, memory, and containers across the cluster for multiple jobs. The ResourceManager (RM) acts as a central scheduler.	Like an "airport control tower" deciding which plane (job) gets which runway (resource).
2. Job Scheduling & Queuing	Determines which application runs first and how cluster resources are divided among users or queues (e.g., FIFO, Capacity, Fair Scheduling).	Similar to a "task manager" prioritizing processes on your OS.

YARN Task	Task Details	Explanation
3. Application Lifecycle Management	Launches, monitors, and terminates ApplicationMasters and their containers. Keeps track of app states (RUNNING, FINISHED, FAILED).	Like a project manager supervising each running job from start to finish.
4. Fault Tolerance & Recovery	Detects failed ApplicationMasters or containers and re-launches them automatically on healthy nodes. Maintains high cluster uptime.	Like restarting a crashed service without user intervention.
5. Resource Monitoring & Reporting	Collects usage metrics (CPU, RAM, container logs) from NodeManagers, updates cluster status, and provides data to the Web UI and job history server.	Like a system monitor showing cluster-wide performance stats.

5.4 NodeManager (in YARN) Tasks

NodeManagers are the slave nodes in the YARN framework, running on each machine to manage containers.

NodeManager Task	Task Details	Explanation
1. Container Management	Launches, monitors, and terminates containers (isolated environments where actual tasks run). Each container runs part of an application.	Like Docker running and managing containers on each node.
2. Resource Monitoring	Tracks local resource usage (CPU, memory, disk, network) of all running containers and ensures no container exceeds its allocated limits.	Like a system resource monitor that enforces quotas per process.
3. Node Health Reporting	Periodically sends heartbeat signals to the ResourceManager, confirming that the node is alive, and reporting container statuses.	Like sending "I'm alive and healthy" signals to the central controller.
4. Log Aggregation & Cleanup	Collects logs from all containers, aggregates them, and sends them to the YARN log server. Also cleans up old containers and temporary files.	Like a janitor who collects and archives logs, then cleans up old workspace files.
5. Communication with ApplicationMaster	Provides the ApplicationMaster with updates about container status, failures, and resource availability, enabling job progress tracking and retries.	Like a site supervisor updating the project manager about worker progress.

5.5 ApplicationMaster Tasks

Each application running on YARN gets its own ApplicationMaster to manage its execution.

ApplicationMaster Task	Task Details	Explanation
1. Resource Negotiation	Communicates with the ResourceManager (RM) to request containers with specific resource requirements (CPU, memory, locality).	Like a project manager asking headquarters for workers and equipment.

ApplicationMaster Task	Task Details	Explanation
2. Container Launch & Coordination	Once containers are allocated, the AM contacts the relevant NodeManagers to launch the tasks inside those containers.	Like assigning workers (tasks) to different construction sites (nodes).
3. Task Monitoring & Progress Tracking	Tracks the status of each task running inside containers — success, failure, progress %, and time. Handles retries for failed tasks.	Like a manager checking every team member's daily progress.
4. Fault Tolerance & Task Recovery	Detects failed containers or nodes and requests replacement containers from the RM to re-run failed tasks.	Like reassigning a task to another worker when one falls sick.
5. Reporting & Communication with Client	Sends periodic status updates and final results back to the client program that submitted the job.	Like a manager reporting job status and results to the client.

5.6 Citations

- Apache Hadoop 3.3.4 Documentation. (2022). *HDFS Architecture*. The Apache Software Foundation. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html