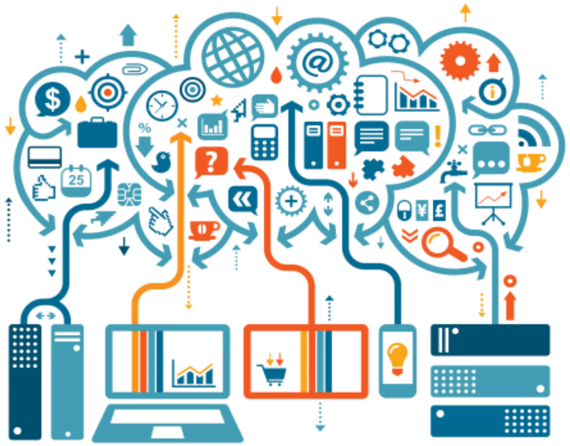
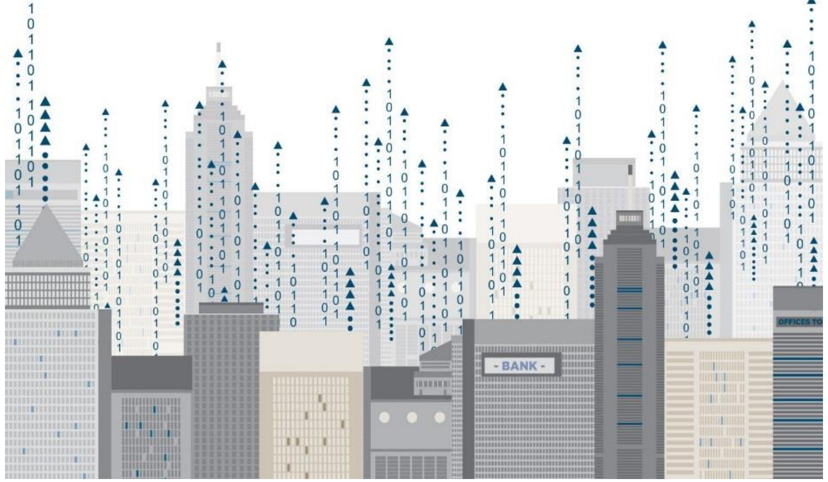




Big Data Analytics

Fall 2025

Lecture 8

Hadoop

- When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines.
- Filesystems that manage the storage across a network of machines are called distributed filesystems.
- Since they are network based, all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk filesystems.
- Making the filesystem tolerate to node failure without suffering data loss.

HDFS

- HDFS: Hadoop Distributed Filesystem.
- HDFS: filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware
- Very large files: hundreds of MB, GB, TB or PB in size
- Streaming data access
  - Write-once, read-many-times pattern.
  - A dataset is copied from source and analyzed
  - Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

HDFS

- Commodity hardware
  - Doesn’t require expensive, highly reliable hardware.
  - Designed to run on clusters of commodity hardware for which the chance of node failure across the cluster is high, at least for large clusters.
  - HDFS is designed to carry on working without a noticeable interruption

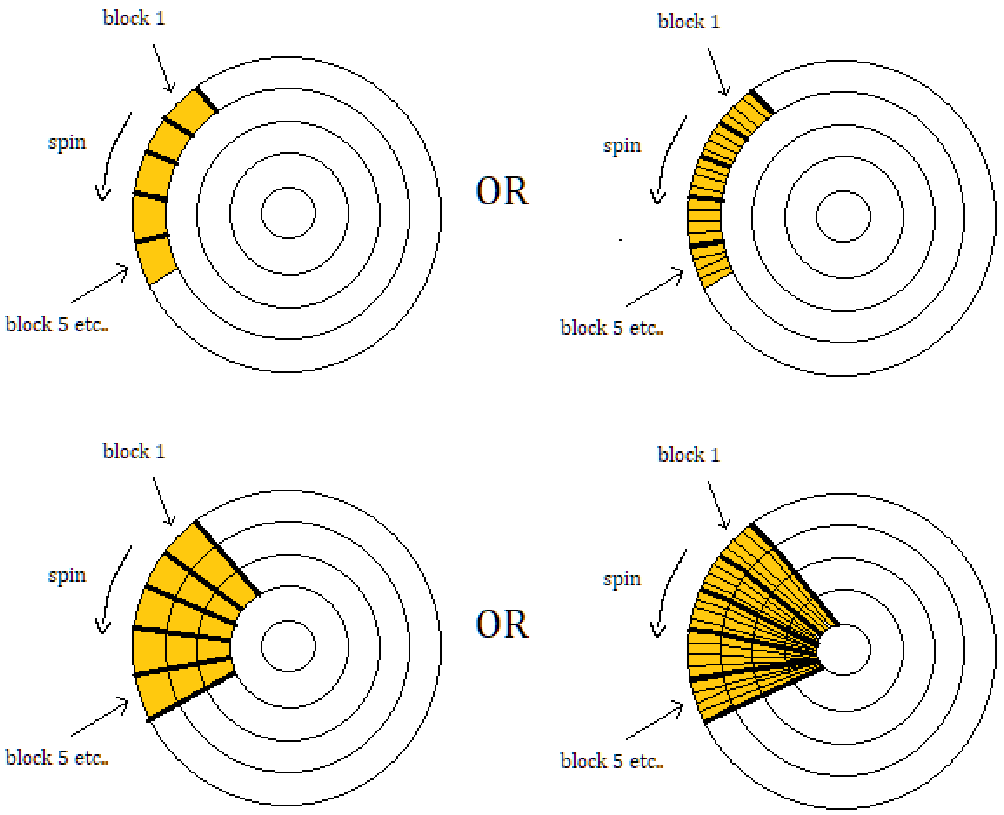
- Not a good-fit for Low-latency data access:
  - Applications that require low-latency access (milliseconds range), will not work with HDFS
  - HDFS: optimized for delivering high throughputs at the expense of latency

HDFS

- Not a good-fit for lots of small files:
  - Because the namenode holds HDFS metadata in memory - the limit to the number of files is governed by the amnt of memory on namenode.
  - ROT: Each HDFS block's metadata takes about 150 bytes on namenode
  - So, for example, if you had one million files, each taking one block, you would need at least 150 MB of memory.
  - 150 * 1 million = 150 million
  - 1 MB = 1048576 bytes
  - So mem = 143.05 MB

Blocks

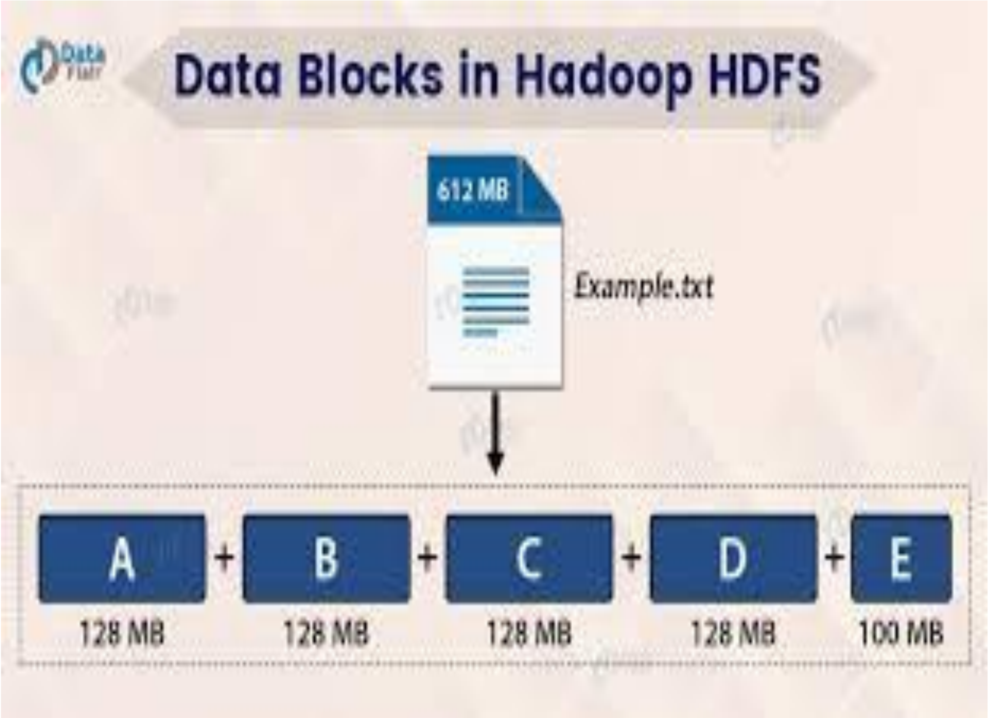
- A disk has a block size, which is the minimum amount of data that it can read or write.
- Filesystems also deal with data in blocks, which are an integral multiple of the disk block size.
- Filesystem blocks: typically a few kilobytes in size, whereas disk blocks are normally 512 bytes.



- The thicker lines divides the sectors

Blocks

- HDFS: concept of a block, but much larger unit—128 MB by default.
- Files in HDFS are broken into block-sized chunks - stored as independent units.
- Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's worth of underlying storage.
- For example, a 1 MB file stored with a block size of 128 MB uses 1 MB of disk space, not 128 MB.



Why Is a Block in HDFS So Large?

- To minimize the cost of seeks: If the block is large, the time it takes to transfer the data from the disk can be longer than the time to seek to the start of the block.

- Transferring a large file made of multiple blocks operates at the disk transfer rate.

- If seek time is around 10 ms, and the transfer rate is 100 MB/s, to make the seek time 1% of the transfer time, we need to make the block size around 100 MB: default is actually 128 MB

Why Is a Block in HDFS So Large?

$$T_{seek} = 0.01\ s$$

$$T_{transfer} = \frac{\text{Block Size}}{\text{Transfer Rate}}$$

$$T_{seek} = 0.01 \times T_{transfer}$$

File Size	Configured Block Size	No. of Blocks	Block Sizes
<120 MB	128 MB	1	120 MB
=128 MB	128 MB	1	128 MB
=134 MB	128 MB	2	128 MB, 6 MB
=512 MB	128 MB	4	128 MB × 4

Benefits

- First: a file can be larger than any single disk in the network.
  - There's nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster.

- Second: making the unit of abstraction a block rather than a file simplifies the storage subsystem.

- Third: blocks fit well with replication for providing fault tolerance and availability.

Benefits

- Like its disk filesystem cousin, HDFS’s fsck command understands blocks.
- % hdfs fsck / -files -blocks
- This will list the blocks that make up each file in the filesystem

```
[[redacted] ~]$ hdfs fsck  /tmp/data
Connecting to namenode via [redacted]/fsck?ugi=m
FSCK started by manjunath (auth:SIMPLE) from /192.0.1.9 for path /tmp/data at
.Status: HEALTHY
 Total size:     15882 B
 Total dirs:     8
 Total files:    1
 Total symlinks:                0
 Total blocks (validated):      1 (avg. block size 15882 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
FSCK ended at Mon Jun 28 08:12:11 UTC 2021 in 0 milliseconds


The filesystem under path '/tmp/data' is HEALTHY
[[redacted] ~]$
```

Questions

- Scenario: You have a file of size 800 MB, and the HDFS block size is set to 128 MB.
- Question: How many blocks will this file be split into on HDFS?
- Number of blocks=800/128 = 6.25
- HDFS rounds this up, so the file will be split into **7 blocks**: 6 full blocks of 128 MB
- 1 block of 16 MB (remainder)

Questions

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

Questions

- Scenario: You have 5 files, each of size 500 MB, and the HDFS block size is set to 256 MB.
- Question: How many blocks are needed to store all the files?
- 500/256 ~ 2 blocks per file (in reality, 1 block + 0.953125 blocks)
- For 5 files ~ 10 blocks

Questions

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

- Since the block size is 128 MB, the file will consume 1 block even though it is smaller than the block size (in this case, < 1 block is not possible)
- With a replication factor of 2: 256MB

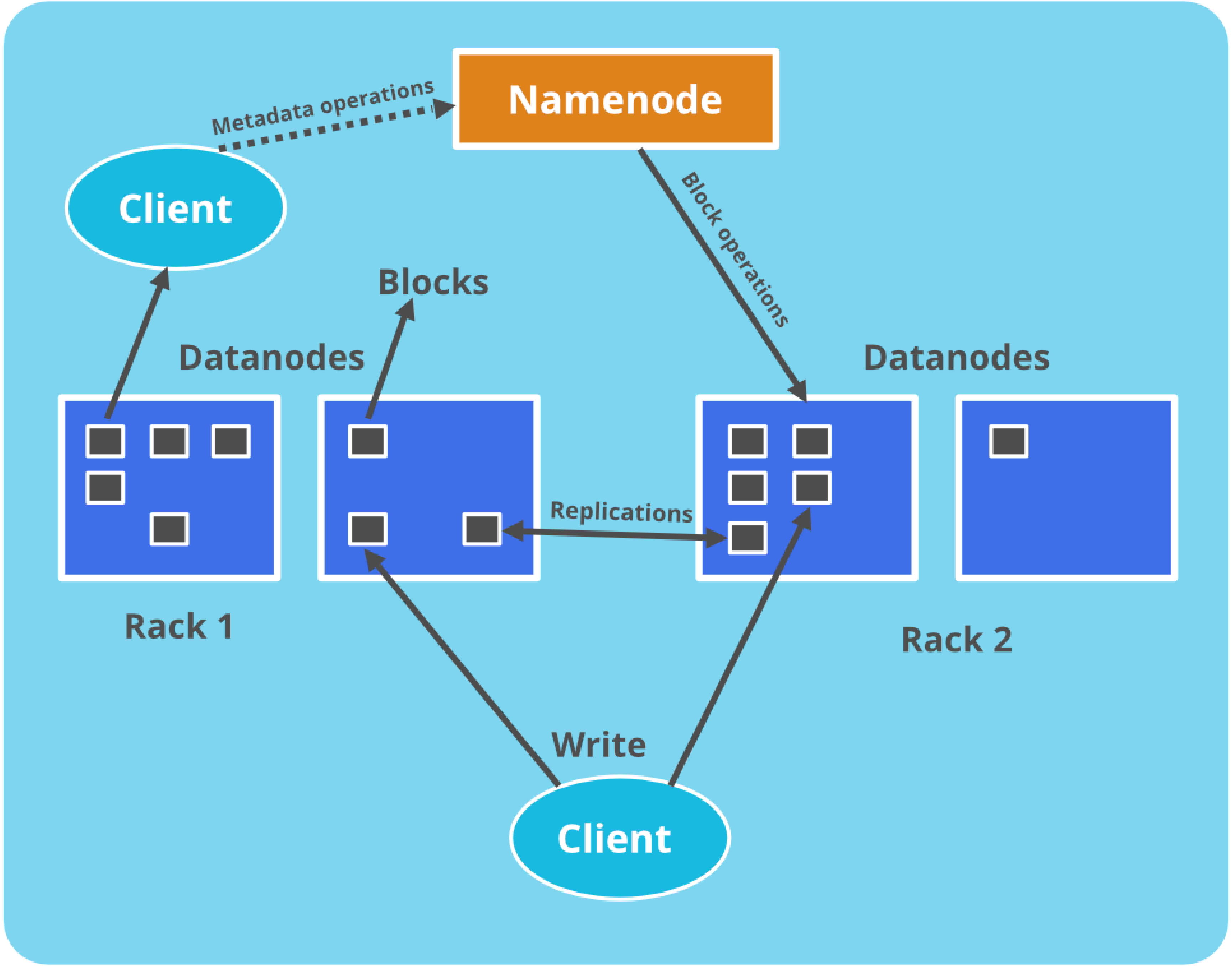
Questions

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

- 5000 MB / 512 MB = 9.77 ~ 10 blocks
- 9 full blocks of 512 MB
- 1 block of 232 MB

Questions

- Scenario: You have a file of size 2.5 GB, and HDFS supports two block sizes: 128 MB and 256 MB.
- Question: How many blocks are required to store the file with each block size?
- Scenario: You have 10 files, each of size 300 MB, the HDFS block size is set to 128 MB, and the replication factor is set to 3.
- Question: How much total disk space will be consumed to store these files in HDFS?


Namenode
Metadata operations
Client
Block operations
Blocks
Datanodes
Datanodes
Replications
Rack 1
Rack 2
Write
Client

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” (like ls output for the whole cluster).
2. Block Management	Keeps a mapping of each file to its blocks and each block to its DataNodes.	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 has the configured replication factor (default = 3). If a DataNode fails, it schedules replication from other nodes to restore balance.	Like a “health monitor” maintaining 3 copies of each piece.
4. Metadata Persistence (EditLog + FsImage)	Periodically writes metadata to disk: **FsImage:** snapshot of the namespace **EditLog:** record of recent changes It	Merges them periodically into a new FsImage (checkpointing). | Ensures HDFS can recover state after 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 Chooses target DataNodes for new block writes	Acts as a “traffic controller” that coordinates clients and DataNodes without transferring the actual data.

DataNode Tasks	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 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.

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 (FIFO, Capacity, Fair Scheduling).	Similar to a “task manager” prioritizing processes on your OS.
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.

NodeManager (in YARN) Tasks	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 (for later viewing via the Web UI). 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.

ApplicationMaster Tasks	Task Details	Explanation
1. Resource Negotiation with ResourceManager	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.
2. Container Launch & Coordination via NodeManagers	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 (the user who submitted the job). Updates are also written to the YARN Job	Like a manager reporting job status and results to the client.