Hadoop Assignment

1. List the use and responsibilities of following tools of Hadoop Eco-System.

a. HDFS (Hadoop Distributed File System)

→ Use: HDFS is the primary storage system of Hadoop, designed to store large datasets across multiple machines in a distributed fashion. it handles large volumes of data and Ensures fault tolerance by replicating data across multiple nodes.

Responsibilities:

- Stores data across multiple nodes
- Ensures data replication (typically 3 copies) for fault tolerance
- Allows parallel processing of data stored across multiple nodes.
- Handles large file sizes Efficiently, supporting high throughput access.

b. YARN (Yet Another Resource Negotiator)

use:- YARN is the resource management layer of Hadoop that manages and schedules resources for the processing jobs. it decouples the resource management & job scheduling from the MapReduce Engine.

Responsibilities:-

- Allocates and manages system resources (CPU, Memory) across various applications
- Schedules and monitors jobs, Ensuring tasks are assigned Efficiently.
- Provides fault tolerance by reassigning failed jobs.
- Supports various processing models like MapReduce, Spark, Tez etc.

C. ~~Pig~~ Sqoop

use:- Sqoop is used to transfer bulk ~~on~~ data b/w Hadoop and relational databases (like MySQL, Oracle). It allows importing data from RDBMS into HDFS & Exporting processed data from HDFS back to RDBMS.

Responsibilities:

- Transfers data b/w relational databases & Hadoop
- Provides incremental import capabilities
- Supports data Export from Hadoop (HDFS / Hive) to relational databases.
- Ensures data integrity and Efficiency during transfers.

d. FLUME

use:- Flume is used for efficiently collecting, aggregating, and moving large volumes of log data from various sources (like web servers) to a centralized storage like HDFS.

Responsibilities:

- Collect streaming log data from multiple sources.
- Aggregates and transfers logs to HDFS or other destinations.
- Ensures reliable and fault-tolerant data flow.
- Supports various sources and sinks, & allows custom configurations for scaling.

e. Zookeeper :-

use:- Zookeeper is a co-ordination service that manages distributed applications. it provides services like configuration management, synchronization and naming for large, distributed systems.

Responsibilities

- Manages synchronization b/w distributed components
- Provides consistent configuration management across nodes.
- Offers leader election services for distributed applications
- Ensures reliable co-ordination in distributed systems by

g. Hive

use:- Hive is a data warehouse infrastructure built on top of Hadoop, it provides a SQL-like interface (HiveQL) to query and manage large datasets in HDFS

Responsibilites

- Allows SQL-like querying on large datasets in HDFS
- Translates HiveQL queries into MapReduce or Tez jobs
- Manages metadata in databases like MySQL (via the Hive metastore)
- optimizes query execution through partitioning, bucketing etc.

2. Explain the responsibilities of Name node and Data nodes in HDFS. Differentiate blw normal file system with HDFS.

→ Responsibilites of NameNode and DataNodes in HDFS

1. NameNode (Master node)

The NameNode is the central authority in HDFS that manages the metadata and structure of the file system.

Responsibilites :-

- Metadata Management :- It stores and manages metadata information such as file names, file locations, blocks and directories. It does not store the actual data

but maintains a mapping of which DataNode holds with block.

ii) Namespace Management:- It manages the file system namespace, keeping track of all the directories & files in the HDFS.

iii) File operations:- It handles operations like file creation, deletion, and renaming.

iv) Block Management:- It knows the location of all the blocks that make up a file and manages the replication of blocks to ensure fault tolerance, if a DataNode fails, the NameNode re-replicates the missing blocks to other nodes.

v) Client Interaction:- when a client wants to read/write a file, the NameNode provides the necessary information about where to find or store the data blocks.

DataNodes

These are the worker nodes in HDFS, responsible for storing & retrieving data blocks.

Responsibilities:

- Metadata Management:- It stores and manages metadata information such as file names, file location, blocks, and

directories. it does not store the actual data but maintains a mapping of which DataNode holds which block.

- Namespace Management :- It manages the file system name space, keeping track of all the directories & files in the HDFS.
- File Operations :- It handles operations like file creation, deletion, and renaming
- Block Management :- It knows the location of all the blocks that made up a file and manages the replication of blocks to other nodes.
- client
- Data Storage :- Each DataNode stores the actual blocks of data in the local file system. A file in HDFS is split into multiple blocks, & these blocks are distributed across DataNodes.
- Block Management :- DataNodes periodically send block reports to the NameNode, informing it of the blocks they are storing.
- Read/write Operations :- when clients request to read or write data, DataNodes serve or store the data as directed by the NameNode.

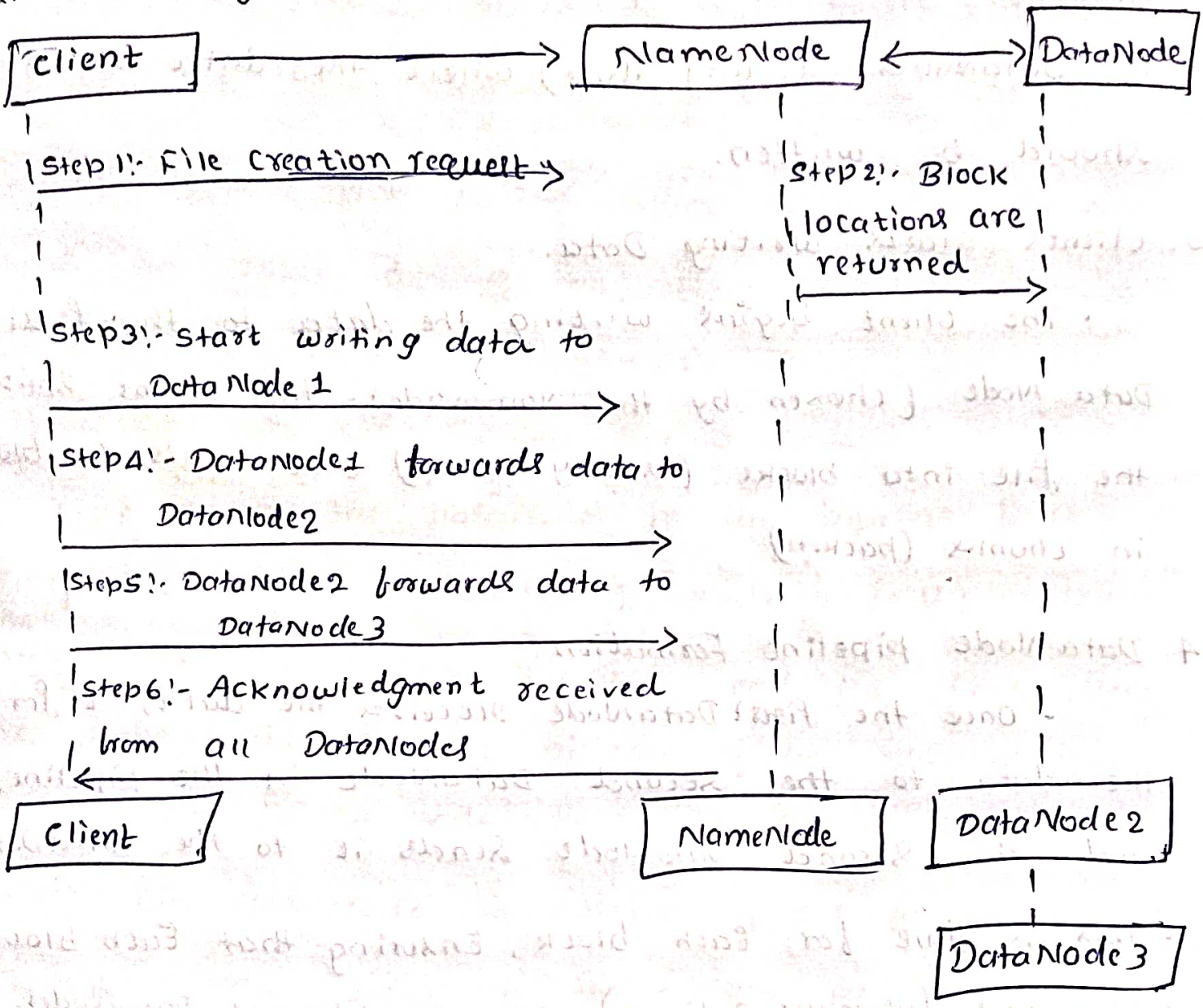
- Heartbeat Mechanism:- DataNodes send regular heartbeat signals to the NameNode to inform it that they are alive and functioning. if a DataNode fails to send a heartbeat, the NameNode assumes the node has failed and reassigns its tasks.
- Replication:- DataNodes replicate blocks to other DataNodes to meet the replication factor specified by the system (usually 3 copies per block)

③ Differentiation btw HDFS and a Normal File System.

HDFS (Hadoop Distributed File System)	Normal File System (e.g. NTFS, EXT4)
* Data is distributed across multiple nodes in the cluster	Files are stored on a single physical machine or device
* Provides fault tolerance through data replication (usually 3 copies) Fault tolerance	* No built-in fault tolerance. if disk fails data is lost unless backups are made.
* It can scale horizontally by adding more nodes to the cluster	* Limited scalability; adding more storage usually involves a single machine
* Data size can be optimized for handling large datasets	* works efficiently with small to medium sized datasets.
* Supports parallel processing through framework like MapReduce	* No built-in mechanism for parallel data processing.

④ with Diagram show how write operation is performed in HDFS. List the steps to recover from failure during HDFS write operation.

→



Steps of the HDFS write operation

1. Client Initiates File Creation.

- The Client contacts the NameNode to initiate a file creation request. the NameNode checks if the file already exists and whether the client has the necessary permissions.

2. NameNode Allocates Blocks and DataNodes

• The NameNode doesn't store the data but allocates new data blocks and provides the client with a list of DataNodes (usually three) where the data blocks should be written.

3. Client starts writing Data.

• The client begins writing the data to the first Data Node (chosen by the NameNode). the client splits the file into blocks (usually 128 MB) and sends the blocks in chunks (packets)

4. Data Node pipeline Formation:

• Once the first DataNode receives the data, it forwards the data to the second DataNode in the pipeline, and the second DataNode sends it to the third DataNode

• This is done for each block, Ensuring that each block is replicated (usually 3 times) across different DataNodes.

5. Acknowledgments:

• Each DataNode in the pipeline sends an acknowledg-ment to the previous DataNode after successfully receiving the data block. the third DataNode sends an acknowledgment to the second DataNode, which passes it to the first DataNode, and then the client gets the acknowledgment.

6. Completion of Write Operation

- After all blocks are written and acknowledged by the DataNodes, the client informs the NameNode that the write operation has been completed.

Steps to Recover from Failure During HDFS Write Operation

Failures can occur during the HDFS write operation, such as DataNode failure or network issues.

1. Handling DataNode Failure During Write

- If one of the DataNodes in the pipeline fails during the write process (e.g. DataNode 2 fails)

Step 1:- The remaining DataNodes and the client detect the failure due to the absence of an acknowledgment from the failed node.

Step 2:- The NameNode is notified of the failure, & it selects a new DataNode to replace the failed one.

Step 3:- The client or DataNode restarts the write operation from the point of failure, sending the data to the new DataNode.

2. Replication Mechanism for Fault Tolerance

- If the write to one DataNode fails, the replication factor ensures that the data is still stored on the other DataNodes. the system will ensure the required

replication factor is met by copying the data to additional nodes if needed.

2. Replication Mechanism for Fault tolerance

- If the write to one DataNode fails, the replication factor ensures that the data is still stored on the other DataNodes.

3. NameNode Handling Metadata

- The NameNode monitors the health of DataNodes through periodic heartbeats and block reports

4. Client Retries the write

- If the failure occurs at the client-side or during the writing process, the client retries the write operation from the point of failure.

5. Block Replication Adjustment.

- If any of the replicated blocks are lost due to DataNode failure, the NameNode replicates the blocks from the surviving DataNodes to maintain the required replication factor.

6. Temporary Unavailability of NameNode:

- In case of a NameNode failure, the data write operation will stop until the NameNode is back online. (using a backup NameNode or high availability setup).

(5) Explain how HDFS Ensures high availability of data and Services.

→ HDFS (Hadoop Distributed File System) Ensures high availability of data and Services through a combination of features designed to handle hardware failures, Provide redundancy, & Ensure that the System remains operational Even in the Event of failures.

1. Data Replication

HDFS Ensures data availability and fault tolerance by replicating Each block of data across multiple nodes in the ~~Cookies~~ cluster.

- Default Replication Factor:- By default, Each block of data is replicated three times, meaning that if one DataNode fails, there are still two copies of the data on other nodes.
- Rack Awarness:- HDFS can be configured to Ensure that replicas are distributed across different racks in the data center. this Ensures that Even if an Entire rack fails
- Automatic Re-replication:- If a DataNode fails or becomes unreachable, the NameNode detects the failure & automatically replicates that lost data blocks to other healthy DataNodes to maintain the desired replication factor.

2. High Availability (HA) of the NameNode

The NameNode is a critical component of HDFS, as it stores metadata about the file system.

(6) Explain the terms heart-beat, Edit-log and check-points

-> Heartbeat :- In HDFS, a heartbeat is a signal sent by each DataNode to the Name: Node at regular intervals to indicate that it is alive & functioning properly.

key points:

Purpose: The heartbeat mechanism helps the NameNode monitor the health of DataNodes and detect failures. If a DataNode fails to send a heartbeat within a specified time, NameNode considers it dead or unreachable.

Failure Detection:- If a DataNode is marked as dead, the NameNode stops directing read/write requests to that DataNode and begins the process of re-replicating the blocks stored on that node to other healthy DataNodes to maintain fault tolerance and Data availability.

Block report:- In addition to heartbeats, DataNodes also send block reports periodically to the NameNode, providing information about the blocks they store.

Edit log :- The Edit log is a file maintained by the NameNode in HDFS that records all changes made to the file system's metadata, such as file creation, deletion, or block allocation.

Key Points:

Purpose: The Edit log captures every modification to the HDFS namespace (file or directory changes) in a sequential manner.

Persistence:- When a client modifies the file system, the NameNode immediately records this change in the Edit log to ensure durability in case of a system crash.

Replay: If the NameNode crashes or is restarted, the Edit log can be replayed to recover the state of the file system.

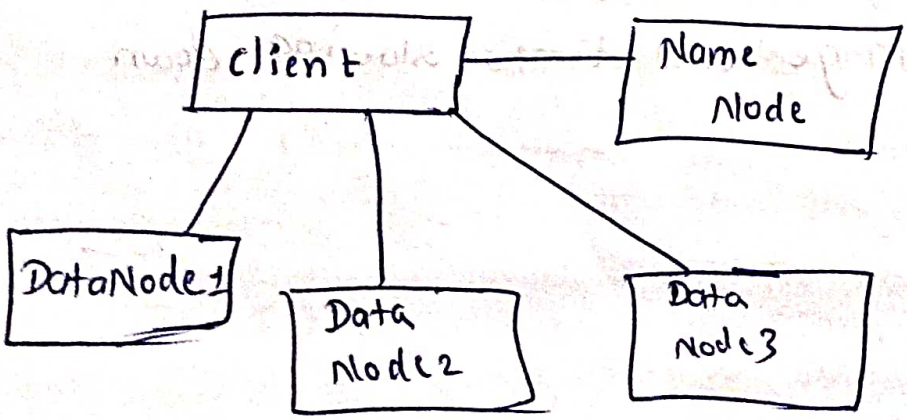
Size Management:- As the Edit log grows with every change, it can become large over time, slowing down recovery processes.

3. checkpoints :- A checkpoint is a snapshot of the HDFS metadata that combines the edit log & the FsImage file.

Key Points:

- FsImage : The FsImage is the file that stores the entire file system namespace and block information as it was at the time of the last checkpoint.
- Checkpointing Process:
  - The Secondary NameNode or Standby NameNode periodically merges the edit log with the current FsImage file to create a new FsImage, and a fresh edit log is started.
- Purpose:- To limit the size of the edit log and prevent it from becoming too large
- Frequency:- checkpoints happen periodically based on a configured time interval or the size of the edit log.

⑦ with neat diagram, Explain the architecture of HDFS



1. NameNode (Master Node):

Role:- The NameNode is the master node in the HDFS architecture, it manages the metadata of the file system & coordinates access to files stored on the DataNodes.

responsibilities:

- Stores metadata (information like file names, directories)
- Keeps track of the mapping blw file names & block locations on the DataNodes.

weakness:- The NameNode is critical, & if it fails, the file system becomes temporarily unavailable.

2. DataNodes (Slave Nodes)

Role:- DataNodes are responsible for storing actual data blocks. they are the worker nodes in the architecture, performing the read & write requests from clients.

Responsibilities:-

- Stores the blocks of data that are part of a file.
- Sends periodic heartbeats & block reports to the NameNode to confirm that they are functioning & to report which blocks they are storing.

Replica Management:- Each block of a file is replicated across multiple DataNodes to ensure fault tolerance.

3. Block Storage :-

- HDFS Blocks Files in HDFS are split into blocks (usually 128 MB or 256 MB in size), which are stored on different DataNodes. Each block is replicated across multiple DataNodes to Ensure data reliability & fault tolerance.

4. Client :-

- The Client interacts with HDFS by sending file system requests such as read, write or Delete

responsibilities:

- Sends read and write requests to the NameNode
- Communicates directly with the DataNodes for data retrieval or storage

5. Secondary NameNode:

The Secondary NameNode does not serve as a backup to the NameNode but helps manage the file system's metadata by performing checkpointing.

6.

HDFS Operations Overview

1. Write Operation :

- The Client requests the NameNode to write a file
- The NameNode responds with the DataNode locations where the blocks can be written.

2. Read Operation:-

- The Client requests the NameNode for the block locations of a specific file.
- The Client directly reads the data from the nearest DataNode

8. List the major components of YARN. Explain the responsibilities of YARN

Major Components of YARN (Yet Another Resource Negotiator)

Yarn is a core component of Hadoop and provides resources management and schedoling for application running in the Hadoop EcoSystem.

major components of YARN are

i) Resource Manager (RM):- Acts as the global resource manager and governs the use of cluster resources across multiple applications.

ii) NodeManager (NM):- A per-node agent responsible for managing containers and monitoring their resource usage on individual node.

iii) ApplicationMaster (AM):- A per-application component responsible for negotiating resources from the resource manager & coordinating the application's execution

5. Scheduler:-

A part of the Resource Manager that allocates resources to various running applications based on scheduling polices like FIFO, Capacity

Responsibilities of YARN

1. Resource Management:-

- Centralized Resource Management: YARN acts as a central resource manager that allocates cluster resources (CPU, memory, disk) dynamically across all running applications.
- Fine-Grained Resource Allocation: It ensures optimal resource usage by assigning containers based on resource needs.

2. Job Scheduling

YARN allows prioritizing certain jobs over others based on organizational needs, using scheduling policies such as FIFO, Capacity, or Fair Scheduling.

3. Fault Tolerance :- Failure Detection & Recovery: if a NodeManager, ApplicationMaster, or container fails, YARN detects the failure and automatically re-launches the failed components on other available nodes.

4. Application Management :-

- Application Lifecycle Management :- YARN is responsible for managing the lifecycle of an application, starting from resource allocation, task Execution, to completion.

5. Data Security & Isolation :-

- Security & Access control : YARN provides mechanisms for Ensuring Security & access control to resources through authentication & authorization frameworks

6. Resource Negotiation and coordination :

- ApplicationMaster Coordination : Each application has its own ApplicationMaster, which negotiates with the resource Manager for resources & co-ordinates with the NodeManagers to run tasks in allocated containers

7. Monitoring and Reporting :

- Resource Usage Monitoring : The NodeManagers monitors the resource usage (memory, CPU, etc) of Each container running on a node and reports back to the Resource Manager.

10. Explain different schedulers used in YARN

1. FIFO Scheduler (First In, First Out)

• The FIFO scheduler processes jobs in the order they are submitted, with no regard for resource availability, priority, or user quotas.

Pros:- • Simple & Easy to implement

• Ensures jobs are completed in the order they arrive.

Cons:-

• No fairness or resource sharing blw jobs

• Not suitable for multi-tenant environments

use case:-

• useful in environments with few users or in cases where job execution order is important & resource competition is minimal

2. Capacity Scheduler.

• The capacity scheduler is designed for multi-tenant clusters, where multiple users or organizations share the same cluster.

• Resources are divided into multiple queues, department or user, with guarantees on the minimum amount of resources that each queue will get

10. Explain different schedulers used in YARN

1. FIFO Scheduler (First In, First Out)

- The FIFO scheduler processes jobs in the order they are submitted, with no regard for resource availability, priority, or user quota.

Pros :- • Simple & Easy to implement

• Ensures jobs are completed in the order they arrive.

Cons :-

• No ~~fairness~~ fairness or resource sharing blw jobs

• Not suitable for multi-tenant Environments

use case :-

• useful in Environments with few users or in cases where job execution order is important & resource competition is minimal

2. Capacity Scheduler.

• The capacity scheduler is designed for multi-tenant clusters, where multiple users or organizations share the same cluster.

• Resources are divided into multiple ~~Users~~ queues, department or user, with guarantees on the minimum amount of resources that each queue will get

Pros:- Ensures resource guarantees for multiple tenants or groups while Enabling flexible resource usage

- Supports hierarchical queues, allowing for better organization & control over resource allocation.

Cons:- More complex to configure & manage compared to FIFO

Use case:

- Suitable for Enterprise clusters where multiple teams or departments share resources and have varying workloads with different priorities

3. Fair Scheduler

- The Fair Scheduler allocates resources such that all jobs receive a fair share of the cluster's resources.
- when multiple jobs are submitted, they share the available Equally, & if one job finishes,

Pros:- Ensures fair resource distribution among all jobs, Preventing any job from monopolizing resources.

Cons:-

can lead to resource fragmentation if jobs require very specific resource amounts.

15. With Ex:, Explain Hadoop MapReduce Process

Hadoop MapReduce is a programming model used for processing & generating large datasets. the process involves breaking down a task into smaller units of work, which are distributed across a cluster of computers.

Ex: word count.

1. Job Submission.

The user writes a MapReduce job in Java, Python or another supported language. the job is submitted to the Hadoop cluster,

2. Job Initialization

: Job tracker: Receives the job & splits it into smaller tasks. the job is divided into Map tasks & Reduce tasks.

3. Map Task Execution.

Input: "Hello Hadoop"

Output (key-value pairs): ("Hello", 1) ("Hadoop", 1)

```
Public class WordCounterMapper Extends Mapper < LongWritable
Text, Text, IntWritable > {
    String line = value.toString();
```

```
StringTokenizer token = new StringTokenizer (line);
while (token.hasMoreTokens()) {
    value.set (token.nextToken());
    context.write (value, new IntWritable(1));
```

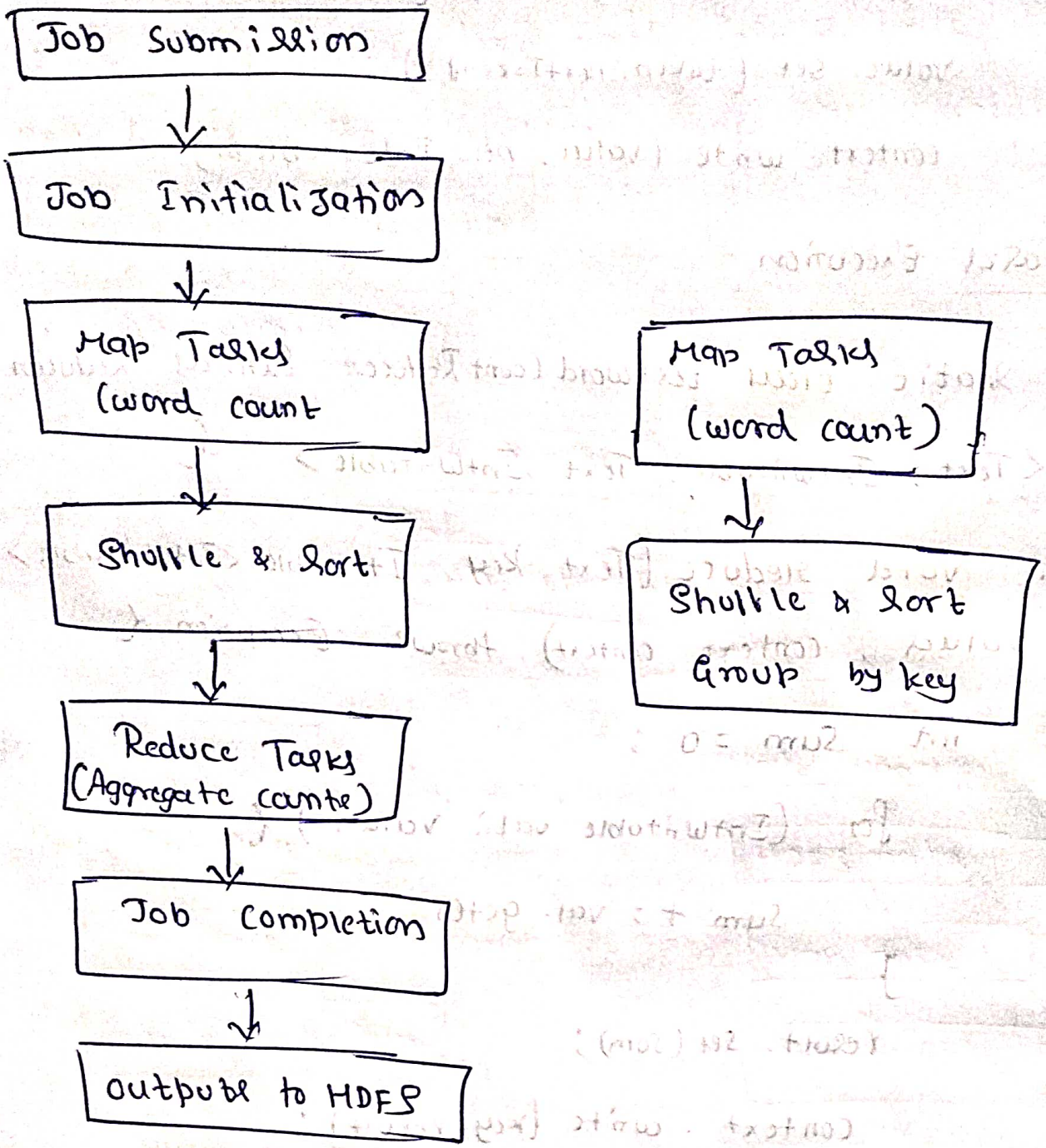
Reduce Task Execution

```
Public Static class testWordCountReducer Extends Reducer
<Text, IntWritable, Text, IntWritable> {
Public void reduce (Text key, Iterable <IntWritable>
values, context context) throws Exception {
    int sum = 0;
    for (IntWritable val: values) {
        sum += val.get();
    }
    result.set (sum);
    context.write (key, result);
}
}
```

cleanup

Temporary data, logs and intermediate files are cleaned up, and resources are released.

Diagram of MapReduce Process



with neat diagram, show how map reduce jobs are Executed in Hadoop Environment.

MapReduce Job Execution workflow

1. Job Submission

- The Client submits a MapReduce job to the JobTracker or ResourceManager. the job consists of the

Map & Reduce tasks, input data & configuration.

2. Job Initialization.

- Job Tracker | Resource Manager receives the job submission and splits the job into smaller Map & Reduce tasks

3. Task Scheduling.

- The JobTracker | ResourceManager schedules the tasks to be executed on various nodes in the Hadoop cluster. the task scheduling depends on the availability

4. Map Tasks Execution

- Map tasks read input data splits from the Hadoop Distributed file system, process the data & produce intermediate key-value pairs

5. Shuffle and Sort

- After Map tasks complete, the intermediate results are transferred to the Reduce tasks.
- Data is transferred to the reducers over the network,

6. Reduce Tasks Execution

- Reduce tasks take the intermediate data, process it to produce the final output, & write the results to HDFS

7. Job completion

- once all Map & Reduce tasks are completed, the job tracker verifies the job's status. it ensures that all

tasks have completed successfully.

Cleanup.

- Temporary data, logs and intermediate files are cleaned up. the system frees up resources & prepares for new jobs