Big Data(BCS061/BCDS-601/KOE-097

Unit –2 Hadoop & Map Reduce

II	**Hadoop:** History of Hadoop, Apache Hadoop, the Hadoop Distributed File System, components of Hadoop, data format, analyzing data with Hadoop, scaling out, Hadoop streaming, Hadoop pipes, Hadoop Echo System. **Map Reduce:** Map Reduce framework and basics, how Map Reduce works, developing a Map Reduce application, unit tests with MR unit, test data and local tests, anatomy of a Map Reduce job run, failures, job scheduling, shuffle and sort, task execution, Map Reducetypes, input formats, output formats, Map Reduce features, Real-world Map Reduce

What is Hadoop?

Hadoop is an **open-source framework** designed to store and process **large datasets** across multiple computers in a **distributed manner**. It uses cheap hardware to handle big data efficiently. The two main components of Hadoop are:

- **HDFS (Hadoop Distributed File System)**: For storing data.
- **MapReduce**: For processing data.

✓ **History of Hadoop**

- **Inspired by Google**: In 2003-2004, Google published papers on the **Google File System (GFS)** and **MapReduce**.
- **Development**:
- **Doug Cutting** and **Mike Cafarella** created Hadoop as part of the **Nutch project** for web search engines.
- They built Hadoop to handle **large-scale data processing** using clusters.
- **Named After a Toy**: Hadoop was named after a toy elephant belonging to Doug Cutting's son.
- **Adopted by Yahoo**: In 2006, Yahoo started using Hadoop and contributed to its development.
- It became an **Apache project** in 2008, making it widely accessible.

❖ Apache Hadoop :

Apache Hadoop is an **open-source platform** used to store and analyze large datasets using distributed systems. It is scalable, meaning you can add more computers to handle more data. It has three main components:

- **HDFS**: A file system that splits data into blocks and stores them across computers.
- **YARN**: Manages resources in the cluster.
- **MapReduce**: Processes data in parallel across the cluster.

❖ What is Hadoop Distributed File System (HDFS)?

HDFS is the storage system used in **Hadoop**. It is designed to store very large files by breaking them into smaller pieces (blocks) and distributing them across multiple computers in a cluster. HDFS is fault-tolerant, meaning it keeps your data safe even if some computers fail.

How Does HDFS Work?

1. Splitting the File:

Large files are broken into **blocks** (default size is 128 MB or 256 MB).

For example, if you have a 1GB file and the block size is 128MB, it will be split into 8 blocks.

2. Storing Blocks on Nodes:

The blocks are stored across the computers (called **DataNodes**) in the cluster.

Each block is **replicated** (copied) to multiple nodes (default: 3 copies) for fault tolerance.

3. Components of HDFS:

NameNode: Acts like a "manager" or "directory" that keeps track of where each block is stored

It doesn't store data itself, just the metadata (information about blocks).

DataNodes: The "workers" that store the actual blocks of data.

4. Writing Data:

When you upload a file, HDFS splits it into blocks.

These blocks are sent to DataNodes and stored with replication.

The NameNode updates its metadata to record where the blocks are stored.

5. Reading Data:

When you request a file, the NameNode provides the location of the blocks.

The system retrieves these blocks from the DataNodes and combines them to give you the complete file.

6. Fault Tolerance:

If a DataNode fails, the other copies (replicas) of the block are used.

The system automatically creates a new copy of the lost block on another DataNode.

✓ **Key Features of HDFS**

- **Scalable**: Easily add more computers to store more data.
- **Fault-Tolerant**: Data is safe even if some computers fail, thanks to replication.
- **Distributed Storage**: Files are split and spread across multiple computers for efficiency.
- **High Throughput**: Optimized for large data reads and writes, not small files.

In simple terms, HDFS works like a library where:

The **NameNode** is the librarian who knows where every book (block) is stored.

The **DataNodes** are the bookshelves storing the actual books (data blocks).

If a book is damaged (a DataNode fails), the library has backup copies to replace it.

Components of Hadoop :

Hadoop has four main components that work together to store and process big data. Let's think of Hadoop as a "data factory" and understand each part like sections of the factory.

1. HDFS (Storage)

- Stores big files by breaking them into small parts and spreading them across many computers.
- **NameNode**: Tracks where files are stored.
- **DataNodes**: Stores the actual data.

2. YARN (Manager)

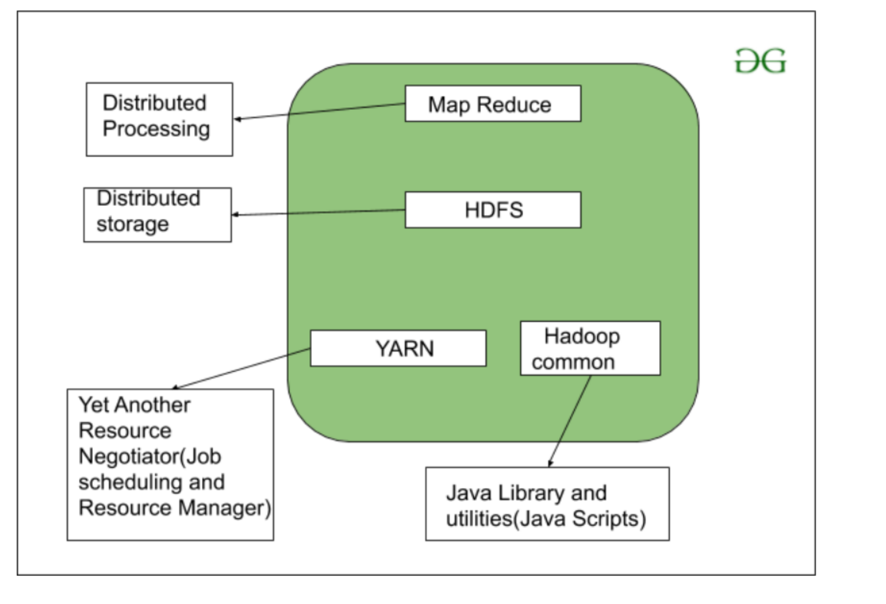
Manages resources like memory and CPU across the cluster. Decides which computer does what job.

3. MapReduce (Processing)

Processes big data in two steps:

Map: Breaks work into smaller tasks.

Reduce: Combines results for the final output.



EduShine CLASSES By- ARMAN ALI

4. Hadoop Common (Toolbox)

Provides tools and libraries to help all the components work together.

Simple Example

Think of it like a pizza shop:

- **HDFS**: Stores the ingredients.
- **YARN**: Decides who prepares, bakes, and delivers the pizza.
- **MapReduce**: Prepares and combines ingredients to make the pizza.
- **Hadoop Common**: Tools like ovens and knives to get the job done.

Data Formats Used in Hadoop

Hadoop can work with different types of data formats. Here are the most common ones, explained simply:

1. Text Format

Stores data as plain text files (e.g., CSV, JSON).

Example: A CSV file with rows like name,age,city.

Usage: Easy to read and debug but not efficient for large files.

2. Sequence File Format

Stores data in key-value pairs (binary format).

Example: key1 -> value1, key2 -> value2.

Usage: Faster to process than text files.

3. Avro

A row-based data format with a schema (structure).

Example: A structured table with columns like ID, Name, Salary.

Usage: Used for serialization and deserialization of data.

4. Parquet

A column-based data format.

Example: Data is stored column-wise (good for analytics).

Usage: Efficient for reading specific columns in big datasets.

5. ORC (Optimized Row Columnar)

Stores data in rows and columns, optimized for Hadoop.

Usage: High performance for Hive queries.

❖ Tools for Analyzing Data in Hadoop

Hadoop has many tools in its ecosystem for analyzing big data. Here are some common ones:

1. Hive

What it does: Converts SQL-like queries into MapReduce jobs.

Why use it: Makes it easy for people who know SQL to analyze big data.

EduShine CLASSES
By- ARMAN ALI

2. Pig

What it does: Uses a scripting language (Pig Latin) to process data.

Why use it: Great for complex data transformations.

3. HBase

What it does: A NoSQL database for fast access to large datasets.

Why use it: Handles real-time queries on massive data.

4. Spark

What it does: Processes data much faster than MapReduce.

Why use it: Ideal for machine learning and real-time data processing.

5. Sqoop

What it does: Transfers data between Hadoop and databases.

Why use it: Moves data in and out of Hadoop easily.

6. Flume

What it does: Collects and moves logs into Hadoop.

Why use it: For streaming data like server logs.

7. Oozie

What it does: Schedules and manages Hadoop jobs.

Why use it: Automates workflows.

❖ Difference Between Scale-Up and Scale-Out

Feature	Scale-Up	Scale-Out
What It Means	Adding more power to a single machine (e.g., better CPU, more RAM).	Adding more machines to the system.
Cost	Expensive, as high-performance hardware is costly.	Cheaper, as it uses regular, low-cost computers.
Performance Limit	Limited by the capacity of one machine.	Flexible, as more machines can always be added.
Failure Risk	If the machine fails, everything stops.	If one machine fails, others continue to work.

✓ How Hadoop Uses Scale-Out

Hadoop is designed to **scale out**, meaning it improves performance by adding more computers (nodes) to the cluster. Here's how:

1. Distributed Storage with HDFS:

Files are split into blocks and stored across multiple machines.
Adding more machines means more storage capacity.

2. Parallel Processing with MapReduce:

Tasks are divided and run on multiple machines at the same time.
Adding more machines means tasks can be processed faster.

3. Fault Tolerance:

If one machine fails, Hadoop uses replicas (copies) stored on other machines.
Adding more machines increases data safety.

Example of Scale-Out in Hadoop

Scenario: You need to process 10TB of data for analysis.
Without Scale-Out: One powerful machine might take 10 hours.
With Scale-Out: By using 10 regular machines in a cluster, each machine processes 1TB, reducing the time to **1 hour**.

EduShine CLASSES By- ARMAN ALI

✓ What is Hadoop Streaming?

Hadoop Streaming allows you to use **any programming language** (like Python, Ruby, or Perl) to write MapReduce programs instead of using Java. It acts as a bridge to process data in Hadoop.

✓ What is Hadoop Pipes?

Hadoop Pipes allows you to write MapReduce programs in **C++**. It is useful when high performance is needed, as C++ programs run faster than Java in some cases.

Feature	**Hadoop Streaming**	**Hadoop Pipes**
Language	Any language with stdin/stdout support	C++ only
Performance	Slightly slower due to flexibility	Faster, as C++ is compiled code
Ease of Use	Easier for non-Java programmers	Requires C++ programming knowledge

EduShine CLASSES
By- ARMAN ALI

❖ Hadoop Ecosystem :

The Hadoop Ecosystem is like a **toolbox** with different tools to store, process, and analyze big data. Here's an overview of the main tools:

✓ Core Tools

1. HDFS (Storage):

Stores big data by splitting it into smaller pieces and distributing it across many computers.

2. MapReduce (Processing):

Processes the data in parallel across multiple computers.

3. YARN (Manager):

Manages resources and assigns tasks to computers in the cluster.

✓ Data Storage & Management

1. HBase:

A database for storing and accessing large amounts of data quickly.

2. Hive:

Lets you query data using SQL-like commands.

Big Data(BCS061/BCDS-601

EduShine CLASSES
By- ARMAN ALI

3. Pig:

Allows you to write scripts for processing big data.

✓ **Data Movement**

1. Sqoop:

Transfers data between Hadoop and traditional databases.

2.Flume:

Moves streaming data (like logs) into Hadoop.

✓ **Data Analysis & Processing**

1. Spark:

A faster tool for big data processing and machine learning.

2. Mahout:

Provides machine learning algorithms for tasks like recommendations and clustering.

✓ **Workflow Management**

1. Oozie:

Schedules and manages Hadoop jobs.

✓ **Visualization**

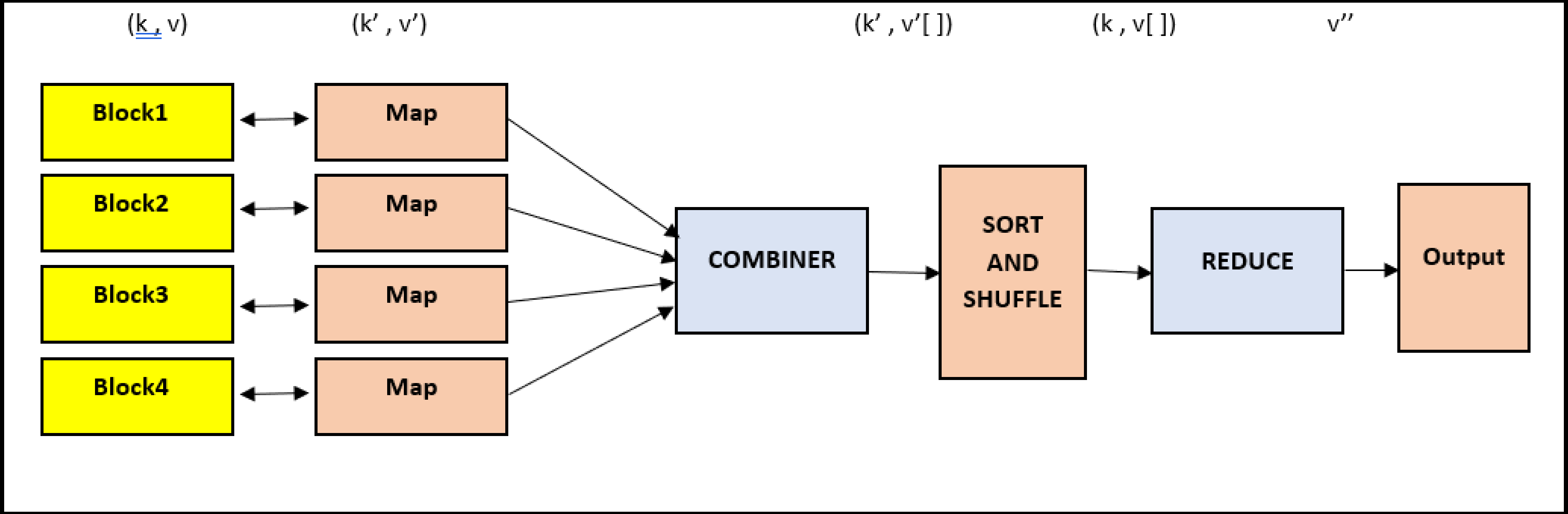
1.Zeppelin or **Hue**:

Helps you view and analyze data with dashboards or notebooks.

What is MapReduce?

MapReduce is a programming model in Hadoop that processes large amounts of data by dividing it into small tasks that run on many computers at the same time. It has two main steps:

- **Map**: Breaks data into smaller parts and processes it.
- **Reduce**: Combines the results from the Map step to produce the final output.



Big Data(BCS061/BCDS-601

EduShine CLASSES
By- ARMAN ALI

✓ Phases (Stages) of MapReduce

1. Map Phase (Splitting and Mapping)

- **What Happens:**
  - Input data is split into small chunks (blocks).
  - Each chunk is processed by a **Mapper** to produce key-value pairs.
  - **Example:** If analyzing a document, Mapper counts how many times each word appears and outputs results like:
    - word1 -> 1
    - word2 -> 1

2. Shuffle and Sort Phase (Grouping)

- **What Happens:**
  - The key-value pairs from the Map phase are grouped by their key (like sorting all values for the same word together).
  - **Example:**
    - All results for word1 are grouped: word1 -> [1, 1, 1]

3. Reduce Phase (Combining)

- **What Happens:**
  - The grouped data is processed by the **Reducer**, which combines the values for each key.
  - **Example:**
    - For word1 -> [1, 1, 1], the Reducer adds the values:
      - word1 -> 3

How MapReduce Works :

1. **Input**: Splits big data into smaller parts (blocks).
2. **Map**: Processes each part to create **key-value pairs** (e.g., ("word", 1)).
3. **Shuffle & Sort**: Groups data by key (e.g., ("word", [1, 1, 1])).
4. **Reduce**: Combines values for each key to produce the final result (e.g., ("word", 3)).
5. **Output**: Saves the combined result.

Example: Counting words in a file:

Input: "Hadoop is cool. Hadoop is fast."

Output:

Hadoop -> 2

is -> 2

cool -> 1

fast -> 1

EduShine CLASSES
By- ARMAN ALI

Phases of Developing a MapReduce Application :

Here are the main steps involved in creating a MapReduce application:

1. Understand the Problem

Identify what you want to achieve with your data.

Example: Counting how many times each word appears in a document.

2. Set Up Input Data

Prepare the data you want to process (e.g., a text file).

Example: A file with sentences like "Hadoop is amazing. Hadoop is fast.".

3. Write the Mapper Function

- Break the input data into small pieces and process each piece.
- Mapper outputs **key-value pairs**.
- Example:
  - Input: "Hadoop is amazing"
  - Mapper output:
    - ("Hadoop", 1), ("is", 1), ("amazing", 1)

4. Write the Reducer Function

- Combine the key-value pairs from the Mapper.
- Reducer processes grouped data and produces the final output.
- Example:
  - Input: ("Hadoop", [1, 1])
  - Reducer output:
    - ("Hadoop", 2)

5. Configure the Job

Specify the **input file**, **output location**, **Mapper**, and **Reducer** in your code.
Set the number of Mappers and Reducers if needed.

6. Run the Application

Submit the job to a Hadoop cluster.
Hadoop handles splitting data, running Mappers and Reducers, and saving the output.

7. Verify Output

- Check the output file to ensure the results are correct.
- Example:
  - Output:
    - Hadoop -> 2
    - is -> 2
    - amazing -> 1

Summary

1. **Understand** the problem.
2. **Prepare** the input data.
3. Write the **Mapper** and **Reducer** functions.
4. **Configure** and run the job.
5. **Verify** the output.

EduShine CLASSES
By- ARMAN ALI

Unit Tests with MRUnit (Short Note)

When you write a MapReduce application, you need to ensure your **Mapper**, **Reducer**, and overall logic work correctly. **MRUnit** is a testing tool specifically designed to test MapReduce code without needing to run it on a full Hadoop cluster.

Why Use MRUnit?

- It’s faster than testing on a real Hadoop cluster.
- Helps find bugs in your Mapper and Reducer logic before deploying.

✓ Test Data and Local Testing in MapReduce (Easy Explanation)

When developing a MapReduce program, you need to make sure it works as expected before running it on large datasets. **Test data** and **local testing** help you do this efficiently.

What is Test Data?

Test data is a small, sample dataset that represents your actual data.
It’s used to check if your **Mapper**, **Reducer**, and the entire job produce the correct results.

Example:

Real data: 10 GB of website logs.

Test data: 10 lines of logs.

What is Local Testing?

Local testing runs your MapReduce program on your computer (not on a Hadoop cluster). It's faster and easier to debug compared to testing on a full cluster.

Anatomy of a MapReduce Job Run (Easy Explanation)

When you run a MapReduce job, Hadoop follows several steps to process the data and produce results. Here's how it works in simple terms:

1. Job Submission

- You submit your MapReduce program (job) to Hadoop.

It includes:

- Input location (where the data is).
- Output location (where to save results).
- Mapper and Reducer code.

EduShine CLASSES
By- ARMAN ALI

2. Input Splitting

- Hadoop splits the input data into smaller chunks (blocks).
- Example: A 100 MB file might be split into ten 10 MB chunks.
- Each chunk is processed by a separate **Mapper**.

3. Mapper Phase

- **Mappers** process each chunk of data in parallel.
- They generate **key-value pairs**.
- Example:
    - Input: "Hadoop is fast"
    - Mapper output: ("Hadoop", 1), ("is", 1), ("fast", 1)

4. Shuffle and Sort

- After the Mapper phase, Hadoop:
    - **Shuffles** (groups) the key-value pairs by their keys.
    - **Sorts** them to prepare for the Reducer phase.
- Example: Grouped data: ("Hadoop", [1, 1]), ("is", [1, 1]), ("fast", [1])

5. Reducer Phase

- **Reducers** process the grouped data.
- They combine the values for each key to produce the final result.
- Example:
  - Input: ("Hadoop", [1, 1])
  - Reducer output: ("Hadoop", 2)

6. Output Writing

The final results from Reducers are written to the output location (e.g., a file on HDFS).

Hadoop -> 2

is -> 2

fast -> 1

Failures in MapReduce (Short Points)

- ✓ **Task Failure**: Mapper/Reducer crashes; Hadoop restarts the task.
- ✓ **Node Failure**: Node goes offline; tasks reassigned to healthy nodes.
- ✓ **Job Tracker Failure**: Master node crashes; modern YARN handles recovery.
- ✓ **Disk Failure**: Hard drive fails; Hadoop uses replicas to recover data.
- ✓ **Network Failure**: Communication breaks; Hadoop retries or reassigns tasks.

Types of Schedulers in MapReduce (Easy Explanation)

Schedulers in MapReduce decide how tasks are assigned to available resources (nodes). Here are the main types of schedulers:

1. FIFO Scheduler

- Processes jobs in the order they arrive (First In, First Out).
- For simple setups with one or very few users.
- Doesn't prioritize or share resources well.

2. Fair Scheduler

- Divides resources equally among all running jobs.
- If 3 jobs are running, each gets 1/3 of the cluster’s resources.
- Ensures no job is starved of resources.

Use Case:

- When multiple users or jobs need fair access.

3. Capacity Scheduler

- Divides the cluster into **queues**, each with a fixed amount of resources.
- Jobs within a queue share resources based on priority.
- Allows organizations to reserve resources for specific teams or projects.

Use Case:

When different teams or departments share the same cluster.

❖ Shuffle and Sort in Hadoop MapReduce :

Shuffle and Sort happens between the **Mapper** and **Reducer** phases. It organizes the data so Reducers can process it efficiently.

1. Shuffle

What It Does:

Collects the key-value pairs output by Mappers.

Groups all values with the same key together.

Example:

Mapper Output:

("apple", 1), ("banana", 1), ("apple", 1)

Shuffle groups :

("apple", [1, 1]), ("banana", [1])

EduShine CLASSES
By- ARMAN ALI

2. Sort

What It Does:

Sorts the grouped keys in order (alphabetical or numerical).

Example:

After sorting:

("apple", [1, 1]), ("banana", [1])

Why Is It Important?

- **Ensures the data is ready for the Reducer to process efficiently.**
- **Happens automatically in Hadoop; you don’t need to write code for it.**

EduShine CLASSES
By- ARMAN ALI

i. Speculative Execution

If a task is running too slowly, Hadoop runs the same task on another computer at the same time.
The result from the faster one is used.

Why It's Done:

To avoid delays caused by slow machines.

Example:

Imagine one worker is too slow at a task. Another worker starts doing the same task, and the quicker one's work is taken.

ii. Task JVM Reuse

Hadoop reuses the same "working space" (JVM) for multiple tasks instead of creating a new one every time.

Why It's Done:

To save time and computer memory.

Example:

Instead of cleaning and preparing a desk for every new task, you use the same desk for 5 tasks in a row.

iii. Skipping Bad Records

- If some data is bad or corrupt and causes errors, Hadoop skips it and processes the rest of the data.

EduShine CLASSES
By- ARMAN ALI

❖ Input and Output Formats in MapReduce :

Input Formats (How data is read):

- **TextInputFormat**: Reads data line by line (default).
- **KeyValueTextInputFormat**: Reads key-value pairs (split by tab).
- **SequenceFileInputFormat**: Reads data in binary format.
- **NLineInputFormat**: Reads a fixed number of lines as one block.

❖ Output Formats (How results are written):

- **TextOutputFormat**: Writes results as plain text (default).
- **KeyValueTextOutputFormat**: Writes key-value pairs as text.
- **SequenceFileOutputFormat**: Saves results in binary format.

EduShine CLASSES
By- ARMAN ALI

Advanced Features of MapReduce :

1. Combiner:

Combines data during the **Mapper** step to reduce the amount sent to Reducers.

Benefit: Saves time and resources.

2. Counters:

Keeps count of things like errors or processed records.

Benefit: Helps track job progress.

3. Speculative Execution:

Runs slow tasks on another machine at the same time.

Benefit: Speeds up job completion.

4. Partitioner:

Decides which Reducer gets which data.

Benefit: Makes sure Reducers get balanced work.

5. Custom Input/Output Formats:

Lets you create special formats to read and write your data.

Benefit: Works with any kind of data.

6. Skips Bad Records:

Ignores bad data that causes errors.

Benefit: Keeps the job running smoothly.

7. Chaining Jobs:

Connects multiple jobs to run one after the other.

Benefit: Handles complex tasks step by step.

EduShine CLASSES
By- ARMAN ALI

Thank You....