1. Define Big Data. Explain any two key characteristics of Big Data.

Big Data – Definition
Big Data refers to extremely large, fast-growing, and diverse datasets that cannot be stored, managed, or processed efficiently using traditional relational database systems. Such data requires advanced technologies like distributed storage, parallel processing frameworks, and scalable analytics tools to extract useful information .

Modern digital life generates continuous streams of information through social media usage, online transactions, mobile applications, sensors, and connected devices. When the size, speed, and variety of this data exceed the capacity of conventional databases and analytical tools, it is recognised as Big Data

Key Characteristics of Big Data: **Big Data is commonly described using several characteristics known as the "V's" of Big Data. Any two of these characteristics are explained below**

1) Volume

Volume refers to the massive amount of data generated every second from digital platforms such as social media, sensors, mobile applications, and online transactions. The data size ranges from terabytes to petabytes. Traditional databases are not capable of handling data at this scale, which is why distributed systems like Hadoop are used.

2) Velocity

Velocity refers to the speed at which data is generated, collected, and processed. Examples include online banking transactions, fraud detection systems, stock market data, and live navigation services. Big Data systems must process data in real time or near real time to be useful.

Conclusion:
Big Data is characterized by huge data volumes and high-speed data generation, which demand scalable storage and parallel processing technologies to extract value from data

Unit-3-Notes

.

---

2. Differentiate between structured, semi-structured, and unstructured data with suitable examples.

Type of Data	Explanation	Examples
Structured	Fully organized data with fixed schema, stored in rows and columns	RDBMS tables, employee records
Semi-Structured	No rigid schema but contains tags or keys	JSON, XML, emails
Unstructured	No predefined format or structure	Images, videos, chats

Explanation:
Real-world data does not always arrive in a uniform format. Some data is neatly organized, some has partial structure, and some has no structure at all. Based on this, data is classified into structured, semi-structured, and unstructured data

1. Structured Data

Explanation:

- Structured data is fully organized and follows a fixed, predefined schema.
- It is stored in rows and columns, and every field has a known datatype.
- This makes it easy to store in relational databases and query using SQL.

Examples:
Employee table, banking transactions, airline bookings.

EmpID	Name	Age	Department	Salary
101	Rohan Kumar	28	Engineering	52,000
102	Meera Shah	31	HR	45,000
103	Arjun Rao	26	Finance	48,500

---

2. Semi-Structured Data

Explanation:

- Semi-structured data does not follow a rigid table structure.
- However, it contains tags, keys, or hierarchy that provides organization.
- It is flexible and widely used in web applications and APIs.

Examples:
JSON, XML, email metadata, log files.

```
{
  "employees": [
    {
      "EmpID": 101,
      "Name": "Rohan Kumar",
      "Age": 28,
      "Department": "Engineering",
      "Salary": 52000
    },
    {
      "EmpID": 102,
      "Name": "Meera Shah",
      "Department": "HR"
    },
    {
      "EmpID": 103,
      "Name": "Arjun Rao",
      "Age": 26,
      "Salary": 48500
    }
  ]
}
```

---

3. Unstructured Data

Explanation:

- Unstructured data has no predefined format or schema.
- It cannot be neatly placed into tables.
- It includes rich media, free text, and any content that requires advanced processing like NLP or image analysis.

Examples:
Images, videos, PDFs, WhatsApp chats, recorded audio, social media posts.

Rohan Kumar is 28 years old and works in the Engineering team
with a salary of around 52,000 per month.
Meera Shah works in the HR department.
Arjun Rao, who is 26, works in Finance.

MVSR_SALD_Internal_Exam_Solution

.

---

3. What are the two main phases of a MapReduce job? Briefly explain each.

The two main phases of a MapReduce job are:

1) Map Phase

In the Map phase, input data stored in HDFS is divided into input splits. Each split is processed by a mapper. The mapper reads the input data and converts it into intermediate key–value pairs. All map tasks execute in parallel, which enables faster processing of large datasets.

2) Reduce Phase

In the Reduce phase, the intermediate key–value pairs produced by the mappers are grouped based on keys. The reducer processes each key and its associated values to generate the final output. The reducer output is then written back to HDFS.

Conclusion:
The Map phase performs parallel data processing, while the Reduce phase aggregates results to generate final output

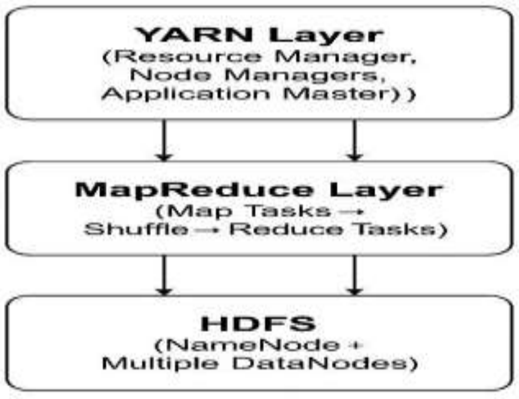
Unit-4-Notes

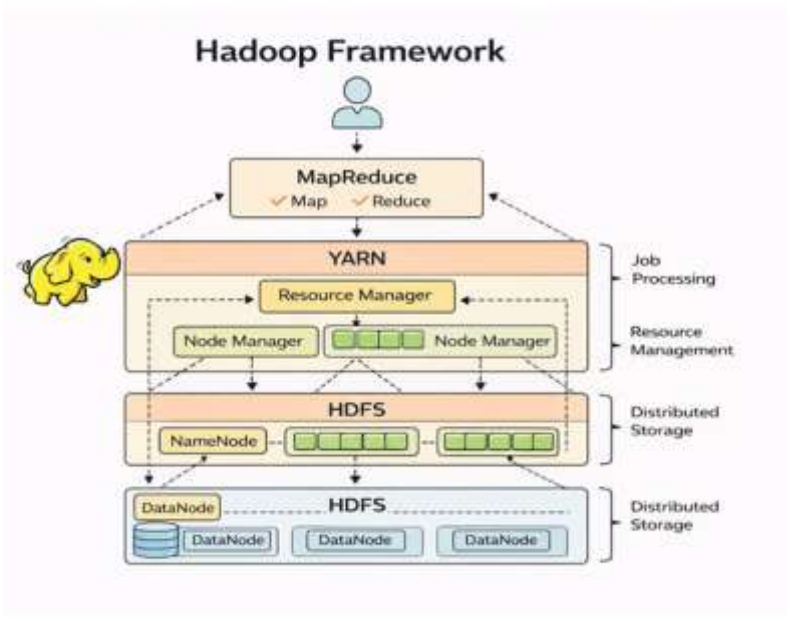
.

---

4. Explain the architecture of Hadoop with a neat diagram.

Hadoop architecture is designed to store and process very large datasets in a **distributed, scalable, and fault-tolerant** manner. The Hadoop framework consists of **three major components** that work together: **HDFS, YARN, and MapReduce**.


YARN Layer
(Resource Manager, Node Managers, Application Master))
MapReduce Layer
(Map Tasks → Shuffle → Reduce Tasks)
HDFS
(NameNode + Multiple DataNodes)



HDFS (Hadoop Distributed File System) is the **storage layer of Hadoop**. It stores huge datasets by splitting them into blocks and distributing these blocks across multiple DataNodes. The NameNode manages metadata about file locations and block mapping, while DataNodes store the actual data. HDFS ensures fault tolerance through replication so that data remains available even if some nodes fail.

YARN (Yet Another Resource Negotiator) is the **resource management and job scheduling layer**. It decides how cluster resources such as memory and CPU are allocated to different applications. YARN consists of the Resource Manager (which makes global decisions), Node Managers (running on each node), and the Application Master (created per job). YARN ensures that tasks are launched on appropriate nodes, monitors execution, and handles failures.

MapReduce is the **processing layer** used to perform computation on the data stored in HDFS. It consists of two phases: Map and Reduce. Map tasks read data blocks from HDFS, process them in parallel, and generate intermediate key-value pairs. Reduce tasks aggregate these intermediate results to produce final output, which is again written back to HDFS.

These components interact as follows:
A user submits a job to Hadoop; YARN allocates resources and launches an Application Master. Map tasks are executed on the DataNodes where the data resides (achieving data locality). Map outputs are shuffled and sent to reducers, which then write final results to HDFS. Throughout this process, YARN coordinates resource allocation and task execution, HDFS provides reliable storage, and MapReduce performs distributed computation.

Together, this integrated framework enables Hadoop to handle large-scale data processing efficiently and reliably.

Below is **Question 5 answered strictly from your Unit-4 notes and the MVSR internal solutions**, with **neat bolding**, **exam-ready language**, and **clear page references**.

---

5. Describe the roles of NameNode and DataNode in HDFS.

HDFS (Hadoop Distributed File System) follows a **master–slave architecture** consisting of a **NameNode** and multiple **DataNodes**.

Role of NameNode

The **NameNode** is the **master node** of HDFS. It **does not store actual data**. Instead, it manages and maintains **metadata** related to the files stored in HDFS.

The NameNode keeps information such as:

- **File names**
- **Number of blocks**
- **Block locations**
- **Mapping of blocks to DataNodes**
- **Replication information**

The NameNode acts like a **directory or index** of the file system and controls all file system operations such as **file creation, deletion, and access control**.

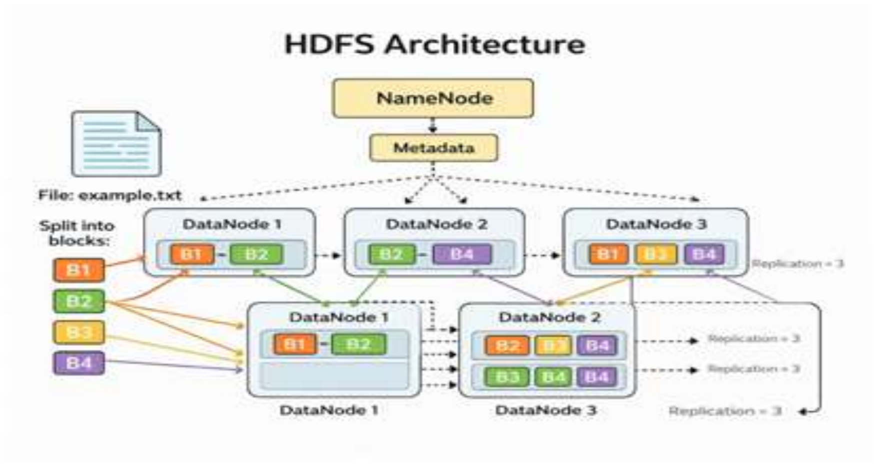
Role of DataNode

The **DataNodes** are the **slave nodes** in HDFS. They are responsible for **storing the actual data blocks** on their local disks.

The main functions of DataNodes include:

- **Storing data blocks**
- **Serving read and write requests** from clients
- **Sending heartbeat signals** to the NameNode to indicate they are active
- **Reporting block information** to the NameNode periodically

DataNodes also perform **block creation, deletion, and replication** as instructed by the NameNode.



Conclusion

Thus, in HDFS, the **NameNode manages metadata and controls the file system**, while the **DataNodes store the actual data blocks**. This separation enables **scalable, reliable, and fault-tolerant storage** of large datasets.

.

6. What is YARN? Explain its purpose in the Hadoop framework.

YARN (Yet Another Resource Negotiator) is the **resource management and job scheduling layer** of the Hadoop framework. It is responsible for managing **cluster resources** and coordinating the execution of distributed applications such as **MapReduce**.

Purpose of YARN in Hadoop

The main purpose of **YARN** is to **allocate and manage cluster resources** such as **CPU and memory** efficiently among multiple applications running on a Hadoop cluster.

YARN performs the following functions:

- It **decides how resources are allocated** to different applications.
- It **schedules tasks** and determines **where tasks should run** in the cluster.
- It **launches and manages application execution** using an **Application Master**.
- It **monitors task execution** and **handles failures** by restarting tasks when required.

- It enables **multiple applications to run simultaneously** on the same Hadoop cluster, not just MapReduce.

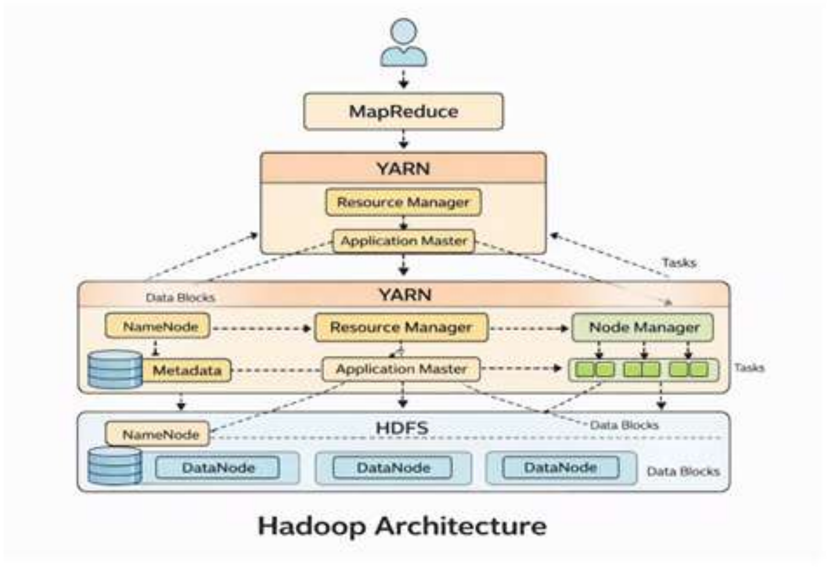
YARN Components (Mentioned for Purpose)

YARN consists of the following components:

- **Resource Manager** – makes global resource allocation decisions.
- **Node Managers** – run on each node and manage resources locally.
- **Application Master** – manages the lifecycle of a specific application.

Conclusion

Thus, YARN separates **resource management from data processing**, making Hadoop more **scalable, efficient, and flexible**. It plays a crucial role in ensuring optimal utilization of cluster resources and reliable execution of Big Data applications.



.

---

7. Explain the role of ResourceManager, NodeManager, and ApplicationMaster in executing a MapReduce job.

YARN (Yet Another Resource Negotiator) is responsible for executing MapReduce jobs in Hadoop. It coordinates job execution using three main components: **ResourceManager**, **ApplicationMaster**, and **NodeManager**.

1) Role of ResourceManager (RM)

The **ResourceManager** is the central authority responsible for managing and allocating resources across the entire Hadoop cluster.

Its roles include:

- **Receiving the MapReduce job** submitted by the client
- **Allocating the first container** to start the ApplicationMaster
- **Allocating containers** for Map and Reduce tasks as requested
- **Ensuring fair and efficient resource sharing** among multiple running applications

The ResourceManager **does not execute tasks**; it only manages resources.

---

2) Role of ApplicationMaster (AM)

The Application Master is created specifically for each MapReduce job and controls the complete lifecycle of that job

Its roles include:

- **Requesting containers** from the ResourceManager for Map and Reduce tasks
- **Deciding where and when tasks should run**
- **Monitoring task execution status**
- **Handling failures** by re-executing failed tasks
- **Reporting final job status** back to the client

The ApplicationMaster **understands the logic of the MapReduce job**.

---

3) Role of NodeManager (NM)

The **NodeManager** runs on **each node in the cluster** and is responsible for executing tasks.

Its roles include:

- Launches containers for Map and Reduce tasks as instructed
- Monitors resource usage (CPU, memory, disk) inside each container
- Sends regular heartbeat messages to the Resource Manager
- **Reporting task status** to the ApplicationMaster

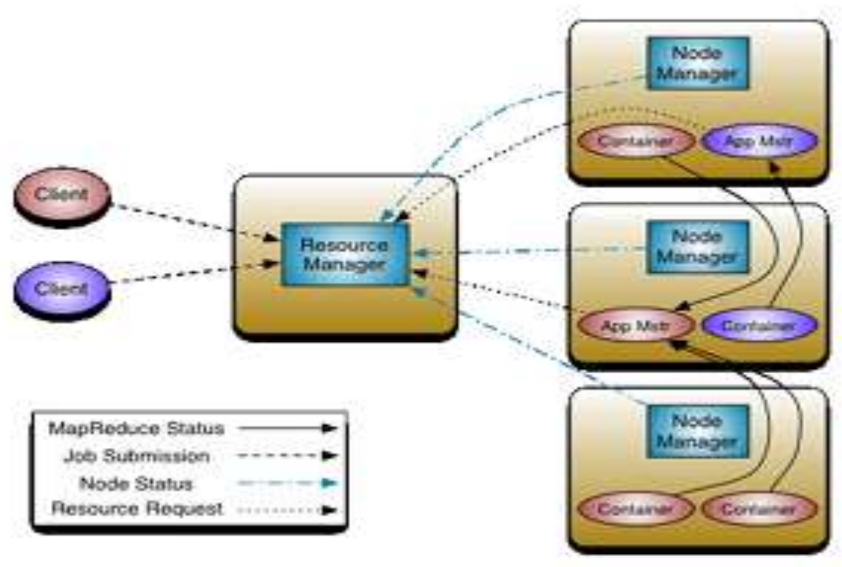
Each worker node has **exactly one NodeManager**.

Coordination Flow (Step-by-Step)

1. Client submits MapReduce job to **ResourceManager**
2. ResourceManager launches **ApplicationMaster**
3. ApplicationMaster requests containers
4. ResourceManager allocates containers
5. NodeManagers launch Map and Reduce tasks
6. ApplicationMaster monitors execution
7. Job completes and status is reported

Conclusion

Thus, the **ResourceManager allocates resources**, the **ApplicationMaster manages job execution**, and the **NodeManagers execute tasks**. Together, they ensure **distributed, scalable, and fault-tolerant execution** of MapReduce jobs..



---

8. Explain the working of a MapReduce program using the WordCount example.

MapReduce is a programming model for processing large data sets with a **parallel, distributed algorithm** on a cluster.

Hadoop MapReduce is a software framework for easily writing applications which process **vast amounts of data (multi-terabyte datasets)** in-parallel on large clusters (thousands of nodes) of commodity hardware in a **reliable and fault-tolerant manner**.

The **WordCount example** is the most widely used program to understand how MapReduce works. It counts how many times each word appears in a text file and helps explain how

Hadoop divides work, processes it, and produces final results in a structured sequence of stages.

Stage One: Input Stage

The process begins with a text file stored in the **Hadoop Distributed File System (HDFS)**. This file may contain several lines of text such as:

```
Deer Bear River
Car Car River
Deer Car Bear
```

HDFS stores the file by **breaking it into blocks** and **distributing these blocks across different machines** in the cluster.

Stage Two: Input Splitting

Before processing begins, Hadoop divides the input file into **logical units known as input splits**. Each split contains a portion of the file. For example, one split may contain the first line, another split may contain the second line, and so on.

Each split is processed **independently and in parallel**.

Input splitting ensures that the work is divided into **manageable portions** that can be handled by different mappers at the same time.

Stage Three: Mapping Stage

Each mapper receives **one input split**. The mapper reads the text in its split, breaks the text into individual words, and records that **each word has occurred once**.

For example, if the split contains the line *Deer Bear River*, the mapper identifies three words and notes that each word has occurred once.

Mappers do not combine or total the counts. They simply recognise words and mark a **single occurrence** for each.

All mappers across the cluster work independently, which makes the processing **highly parallel**.

Stage Four: Shuffle and Sort Stage

Once all mappers complete their work, Hadoop automatically begins the **shuffle and sort phase**. This is the most critical part of MapReduce.

During shuffle, Hadoop collects all the mapper outputs from different machines and reorganises them so that **every occurrence of the same word is brought together in one place**.

During sort, Hadoop arranges the words in a **sorted manner** to maintain order.

This step ensures that all data belonging to a particular word is grouped accurately, even if it came from different parts of the input and different machines.

Stage Five: Reducing Stage

Each reducer receives **one word along with the complete list of occurrences** collected from all mappers during the shuffle and sort stage.

The reducer simply adds these occurrences to determine **how many times the word appears** in the entire dataset.

For example, the reducer may determine that:

- Bear appears two times
- Car appears three times
- Deer appears two times
- River appears two times

These results represent the **final word counts**.

Stage Six: Output Stage

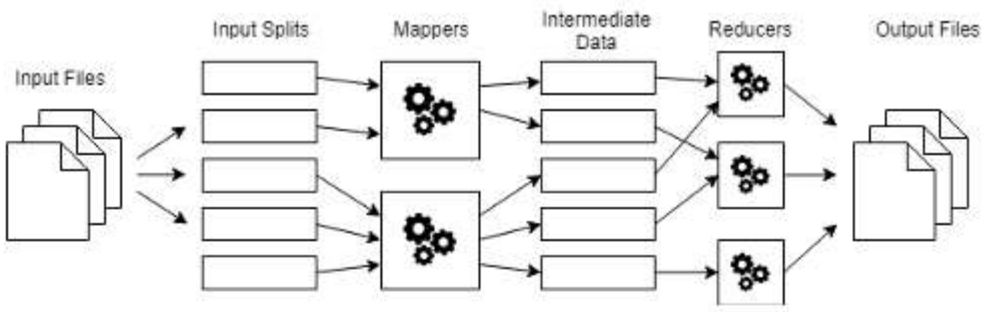
After completing the counting, each reducer writes the **final word and its count** into the Hadoop Distributed File System.

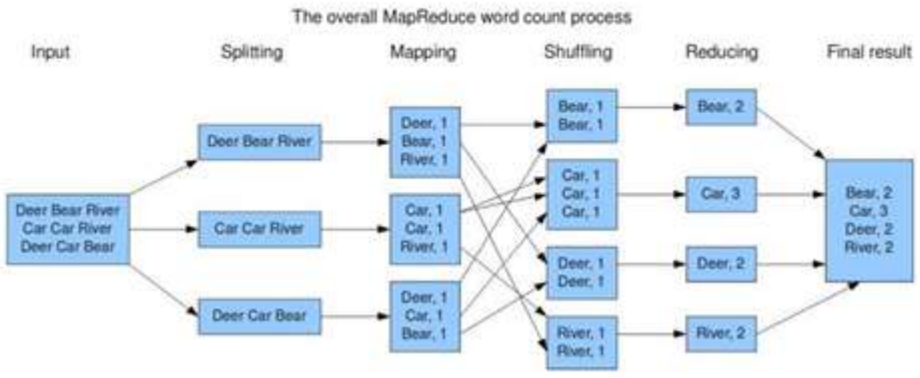
The output consists of text files that contain the final results of the WordCount program. For example:

```
Bear two
Car three
Deer two
River two
```

Summary

The WordCount program demonstrates the complete **MapReduce flow**: the input file is split into smaller units, processed in parallel by mappers, reorganised through shuffle and sort, and finally aggregated by reducers to produce the final counts. This simple example reveals the core strengths of Hadoop, which include **massive parallelism, scalability, fault tolerance, and efficient processing of very large datasets**.





---

9. Describe the input–output data flow in a MapReduce job.

Step-by-Step Working Procedure of a MapReduce Job

Step 1: Job Submission

- The client submits a **MapReduce job**
- Job details include:
    - **Input path**
    - **Output path**

    - **Mapper and Reducer classes**
- The job is submitted to **YARN's Resource Manager**

Step 2: Application Master Launch

- **Resource Manager allocates a container**
- **Application Master (AM)** is launched for this job
- AM manages the **execution of the job**
- **One Application Master is created per job**

Step 3: Input Splitting

- Input data stored in **HDFS** is divided into **input splits**
- Each split corresponds to **one map task**
- Splits enable **parallel execution**
- Splitting is **logical**, based on **HDFS blocks**

Step 4: Map Task Execution

- **Application Master requests containers** for map tasks
- **Node Managers launch map tasks** inside containers
- Each mapper processes its **assigned input split**
- Map tasks run **close to data (data locality)**

Step 5: Shuffle and Sort Phase

- Mapper outputs are **transferred across the network**
- Data is **grouped by key**
- **Sorted data** is sent to reducers
- This is an **automatic system-managed phase**

Step 6: Reduce Task Execution

- **Application Master requests containers** for reducers
- Reducers process **grouped data**
- **Final output is generated**
- Reducers run **in parallel where possible**

Step 7: Output Writing

- Reducer output is written back to **HDFS**
- Output is stored as **result files**
- Once output is written, the job is considered **complete**

Step 8: Job Completion

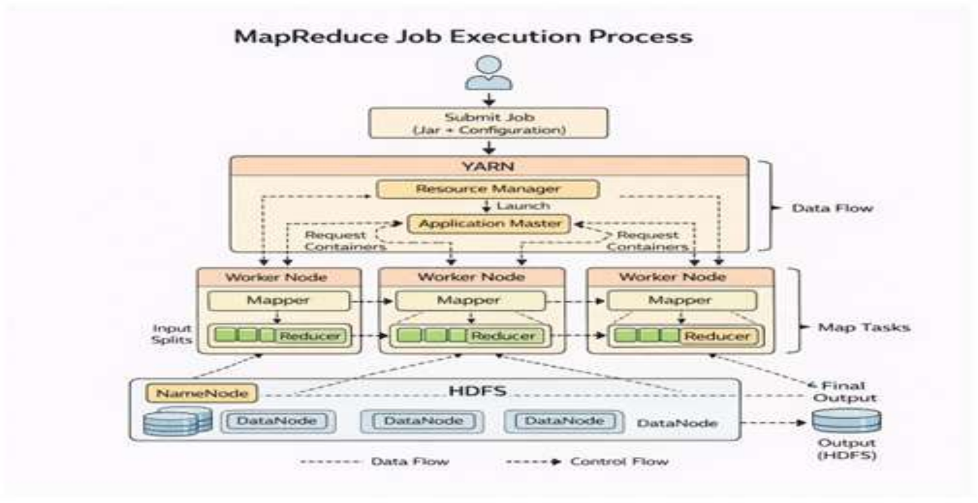
- **Application Master reports job status** to client
- **Resources are released**
- **Application Master shuts down**

Role of Hadoop Components in Execution

Component	Role
HDFS	Stores input and output data
YARN	Allocates resources and manages execution
MapReduce	Executes map and reduce tasks

Key Characteristics of MapReduce Execution

- **Parallel processing**
- **Data locality**
- **Fault tolerance (task retries)**



-

10. Explain the Mapper and Reducer logic in MapReduce.

In MapReduce, the **Mapper** and **Reducer** are the two main functional components that perform data processing in a distributed manner.

Mapper Logic

The **Mapper** is responsible for **processing input data** and generating **intermediate key–value pairs**.

Mapper logic works as follows:

- The mapper receives **input splits** from HDFS.
- Each input split is processed independently.
- The mapper reads the input record (for example, a line of text).
- It transforms the input into **intermediate key–value pairs**.
- The mapper **does not perform aggregation** or counting.

Example (WordCount):
For each word in the input text, the mapper outputs *(word, 1)*.

Reducer Logic

The **Reducer** is responsible for **aggregating intermediate results** produced by the mappers.

Reducer logic works as follows:

- The reducer receives **grouped key–value pairs** after shuffle and sort.
- All values related to the same key are processed together.
- The reducer performs **aggregation** (such as summing values).
- It produces the **final output**.

Example (WordCount):
The reducer adds all the values for a word to compute the **total count**.

Conclusion

Thus, the **Mapper transforms input data into intermediate key–value pairs**, and the **Reducer aggregates these intermediate results** to generate the final output. Together, they enable **parallel and scalable processing** of large datas

.

---

11. Explain the Shuffle and Sort phase in MapReduce.

The **Shuffle and Sort phase** is a **system-managed intermediate phase** in MapReduce that takes place **between the Map and Reduce phases**. It plays a crucial role in ensuring correct data aggregation.

Shuffle Phase

During the **shuffle phase**, Hadoop **collects the intermediate output** produced by all mappers across different nodes.

- Mapper outputs are **transferred across the network**
- All intermediate **key–value pairs** are moved from mappers to reducers
- Data belonging to the **same key** is sent to the **same reducer**

The shuffle phase ensures that **all occurrences of a key**, even if generated by different mappers, are brought together.

Sort Phase

During the **sort phase**, Hadoop **sorts the intermediate keys** before sending them to the reducers.

- Keys are arranged in a **sorted order**
- Values are **grouped based on identical keys**
- Sorting helps reducers process data efficiently

Sorting guarantees that each reducer receives **all values of a key together**.

Importance of Shuffle and Sort

- Ensures **correct grouping of data**
- Enables **accurate aggregation**
- Handles data coming from **multiple mappers**
- Is performed **automatically by Hadoop**

Conclusion

Thus, the **shuffle phase transfers and groups mapper outputs**, and the **sort phase orders the data by keys** before it reaches the reducers. Together, they ensure accurate and efficient processing in MapReduce.

12. Types of NoSQL databases with examples.

NoSQL Introduction

NoSQL databases are used when data is **large**, **fast-changing**, or **not suitable for tables**.
They are mainly of **four types**, each storing data in a different way:

1. **Key–Value Databases**
2. **Document Databases**
3. **Column-Family Databases**
4. **Graph Databases**

---

1. Key–Value Databases

These databases store data as a **simple pair of key and value**.
The **key** is like an ID, and the **value** is the data linked to that ID.

Example

To store user preference:

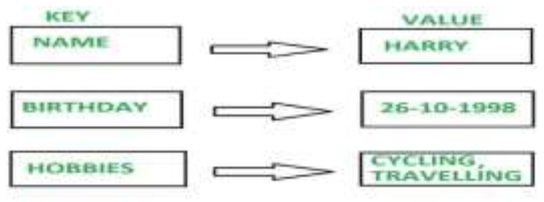
- **Key:** User101
- **Value:** Dark Mode On

Another example:

- **Key:** Product55
- **Value:** Out of Stock

Where used

- Session data
- Shopping carts
- Quick lookups



2. Document Databases

Data is stored in the form of **documents**, usually similar to **JSON**.
Each document can have **different fields**, making it flexible.

Example

Document for one student:

- Name: Rohan
- Course: CSE
- Skills: Java, Python

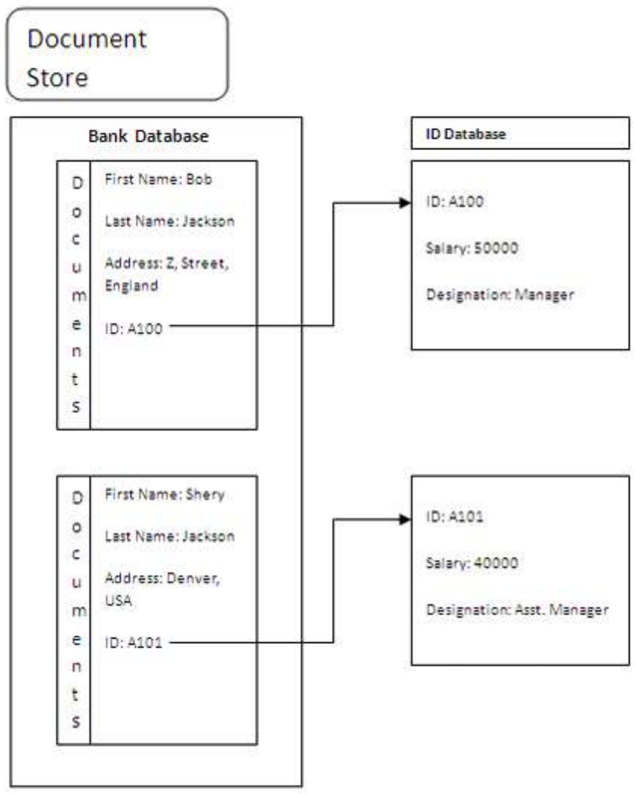
Document for another student:

- Name: Meera
- Skills: Python

Both documents are valid even though they have **different fields**.

Where used

- E-commerce product details
- User profiles
- Mobile app data



---

3. Column-Family Databases

Data is stored in **rows**, but grouped into **column families**.
Each row can have **different sets of columns**.

Example

Row for Student101:

- Personal: Name Rohan, Age 20
- Marks: Maths 85, Science 90

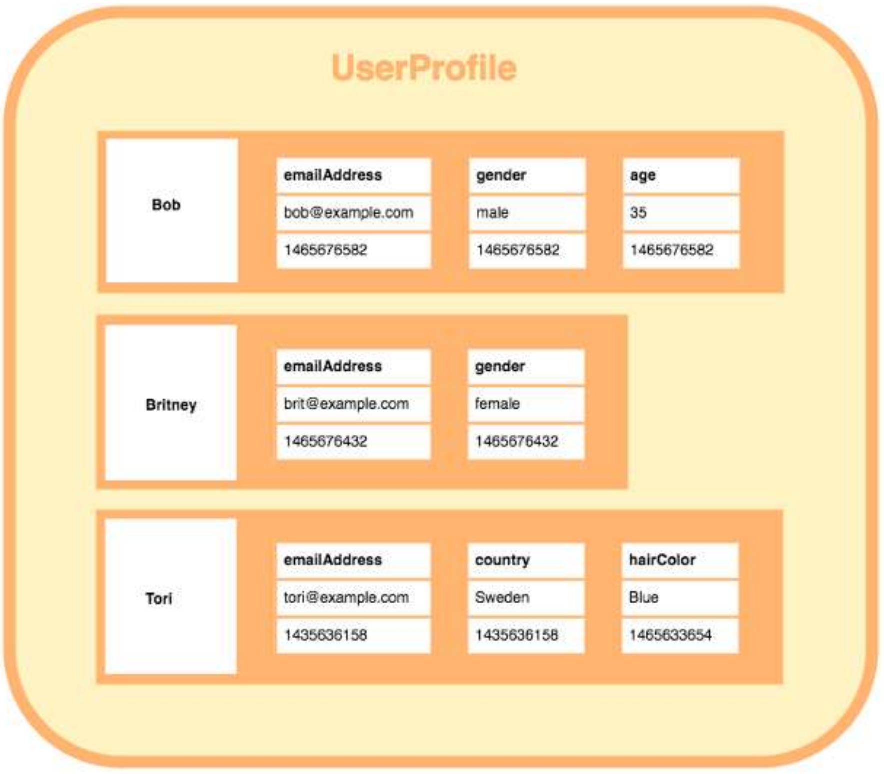
Row for Student102:

- Personal: Name Meera
- Marks: Science 95

Each row has **flexible columns** under the same families.

Where used

- Analytics
- Log data
- Time-series data



---

4. Graph Databases

Data is stored as **nodes and relationships**.
This model is best when **connections between data matter**.

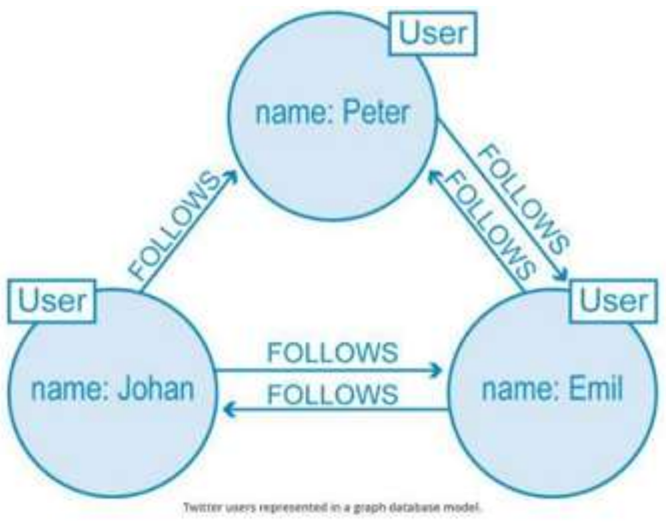
Example

- User A is a friend of User B
- Customer101 bought Product7
- Employee5 reports to Manager2

Here, data is represented as **entities connected by relationships**.

Where used

- Social networks
- Recommendation systems
- Fraud detection



Summary

- **Key–Value databases** store data as simple key–value pairs
- **Document databases** store flexible JSON-like documents
- **Column-Family databases** organize data using column groups for large-scale analytics
- **Graph databases** store connected data using nodes and relationships

Each type supports different requirements of **scalability**, **flexibility**, or **relationship management**.

---

13. What are ACID properties?

ACID Properties (Strong Consistency Model)

ACID is mainly used in **traditional relational databases** where **data accuracy is critical**.

ACID stands for:

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**

Atomicity

Meaning:
A transaction is treated as **all or nothing**.

Analogy:
ATM withdrawal
If cash is not dispensed, money is not deducted from the account.

Consistency

Meaning:
The database moves from **one valid state to another valid state** after a transaction.

Analogy:
Train ticket system
Seat count can never become negative after booking.

Isolation

Meaning:
Multiple transactions execute **independently without interfering**.

Analogy:
Two people filling different bank forms at the same time
One person’s form should not affect the other’s form.

Durability

Meaning:
Once a transaction is committed, it remains saved **permanently**, even after failure.

Analogy:
Once an exam result is published, it remains stored even if the server restarts.

Summary of ACID

- **Strong consistency**
- **Strict rules**
- **Reliable and accurate**
- **Slower and harder to scale**

Used in:
Banking, finance, OLTP systems

---

14. What are BASE properties?

BASE Properties (Flexible Consistency Model)

BASE is used in **Big Data and NoSQL systems**, where **availability and scalability** matter more than strict correctness.

BASE stands for:

- **Basically Available**
- **Soft State**
- **Eventual Consistency**

Basically Available

Meaning:
The system remains **available**, even during failures.

Analogy:
A shopping website still loads even if some prices update later.

Soft State

Meaning:
The data state may change over time, even without new input.

Analogy:
Cached weather data slowly updates across different apps.

Eventual Consistency

Meaning:
All data copies will **eventually become consistent**, not immediately.

Analogy:
WhatsApp message seen ticks appear after some time for all users.

Summary of BASE

- **High availability**
- **High scalability**
- **Faster performance**
- **Temporary inconsistency allowed**

Used in:
Social media, analytics, IoT, Big Data systems

.

---

15. Compare ACID and BASE properties.

Aspect	ACID Properties	BASE Properties
Consistency Model	Strong consistency	Eventual consistency
Focus	Correctness and reliability	Availability and scalability
Transaction Rules	Strict rules	Flexible rules
Performance	Slower performance	Faster performance
Scalability	Harder to scale	Highly scalable
Failure Handling	Rolls back transactions to maintain correctness	Continues operation even during failures
Data Accuracy	Always accurate and consistent	Temporary inconsistency allowed

Aspect	ACID Properties	BASE Properties
Consistency Model	Strong consistency	Eventual consistency
Focus	Correctness and reliability	Availability and scalability
Transaction Rules	Strict rules	Flexible rules
Performance	Slower performance	Faster performance
Scalability	Harder to scale	Highly scalable
Failure Handling	Rolls back transactions to maintain correctness	Continues operation even during failures
Data Accuracy	Always accurate and consistent	Temporary inconsistency allowed

Real-World Examples

ACID Example:
In **banking systems**, ACID properties are required. For example, during an ATM withdrawal, if cash is not dispensed, the amount should not be deducted from the account. This ensures **atomicity, consistency, isolation, and durability**.

BASE Example:
In **social media applications**, BASE properties are followed. For example, when a message is sent on WhatsApp, the “seen” status may not appear immediately for all users, but it will eventually update across all servers. This shows **eventual consistency**.

Conclusion

Thus, **ACID properties** are suitable for systems where **data accuracy and correctness are critical**, such as banking and finance, while **BASE properties** are suitable for **Big Data and NoSQL systems** where **high availability and scalability** are more important than immediate consistency.

---

16. List any two advantages of NoSQL databases over traditional RDBMS.

1) Horizontal Scalability

NoSQL databases can **scale horizontally** by adding more machines (nodes) to the system. This makes them suitable for **Big Data and distributed environments**, where data volume grows rapidly.

2) Flexible Schema (Schema-less Design)

NoSQL databases do not require a fixed table structure. **New fields can be added at any time** without altering existing data, making them ideal for **semi-structured and unstructured data**.

Conclusion

Thus, NoSQL databases provide **better scalability** and **schema flexibility** compared to traditional RDBMS, making them suitable for modern Big Data applications.

---

17. Explain input splitting in MapReduce.

Input splitting is the process of dividing the input data stored in **HDFS** into smaller logical units called **input splits** before MapReduce processing begins.

Each **input split** is assigned to **one mapper**, allowing multiple mappers to process data **in parallel**.

Explanation

- Input data is first stored in **HDFS** as large files.
- Hadoop divides these files into **logical input splits**.
- **Each input split corresponds to one map task**.
- Input splitting enables **parallel execution** of map tasks.
- Splits are usually based on **HDFS block size**, but they are **logical divisions**, not physical copies.

Importance of Input Splitting

- Enables **parallel processing**
- Improves **performance**
- Allows efficient handling of **large datasets**
- Supports **scalability** in MapReduce jobs

Conclusion

Thus, input splitting divides large input files into manageable units so that multiple mappers can process data simultaneously, resulting in faster and scalable data processing.

.

---

18. Why is data locality important in Hadoop?

Data locality is a key concept in Hadoop that means **processing data on or near the node where the data is stored**, instead of moving data across the network.

Importance of Data Locality

- In Hadoop, data is stored in **HDFS across multiple DataNodes**.
- When a MapReduce job is executed, **map tasks are scheduled on the same DataNodes** where the data blocks reside.
- This **reduces network traffic**, because large data blocks do not need to be transferred over the network.
- It **improves performance** and **reduces execution time**.
- Efficient use of data locality leads to **better resource utilization** in the cluster.

Why Hadoop Uses Data Locality

- Network bandwidth is **limited and expensive**
- Moving computation is **cheaper than moving data**
- Hadoop achieves **high throughput** by processing data locally

---

Conclusion

Thus, data locality is important in Hadoop because it **minimizes data movement**, **reduces network overhead**, and **improves performance**, making Hadoop suitable for processing very large datasets efficiently.

.

---

19. Give one real-world application of Hadoop and explain briefly.

Application: Fraud Detection

Hadoop is widely used in **fraud detection systems**, especially in banking, finance, and e-commerce industries.

Explanation

- Banks and online platforms generate **huge volumes of transaction data** every second.
- Hadoop stores this large transactional data in **HDFS** and processes it using **distributed computing**.
- By analysing **historical and real-time transaction patterns**, Hadoop helps identify **abnormal or suspicious activities**.
- MapReduce processes large datasets in parallel to compare normal behaviour with unusual behaviour.
- This allows organisations to **detect fraud early** and reduce financial losses.

Conclusion

Thus, Hadoop plays an important role in fraud detection by enabling **large-scale storage, parallel processing, and analysis of massive transaction data**, making it suitable for handling Big Data applications..

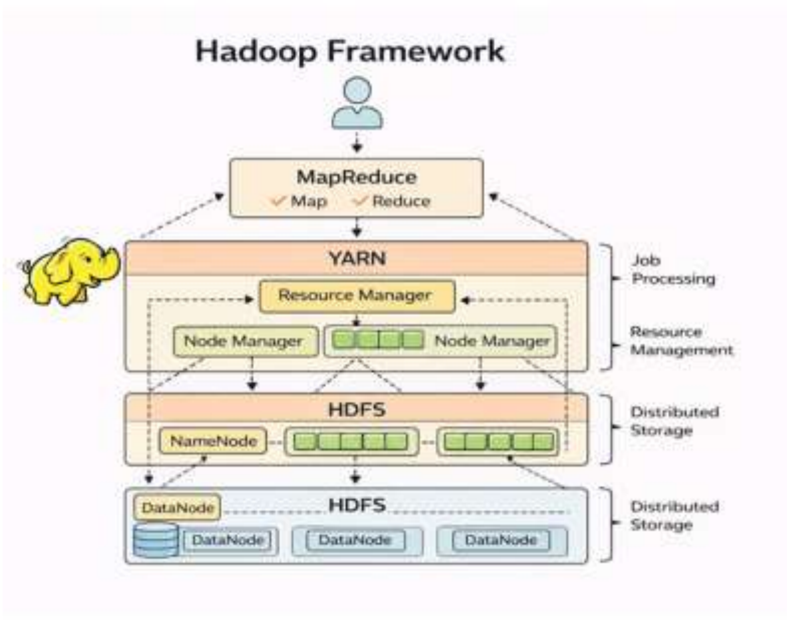
20. Write short notes on:

a) HDFS (Hadoop Distributed File System)

HDFS is the **storage layer of Hadoop** used to store very large files in a **distributed manner**. It follows a **master–slave architecture** consisting of a **NameNode** and multiple **DataNodes**.

- Files are **split into fixed-size blocks**
- Blocks are **distributed across DataNodes**
- **Replication** provides fault tolerance
- The **NameNode stores metadata**
- **DataNodes store actual data blocks**

HDFS is designed to provide **high scalability, fault tolerance, and high throughput**, making it suitable for Big Data storage.



b) MapReduce

MapReduce is the **data processing framework** of Hadoop used to process large datasets in a **parallel and distributed** manner.

It consists of **two main phases**:

- **Map Phase** – processes input data and produces intermediate key–value pairs
- **Reduce Phase** – aggregates intermediate data to generate final output

MapReduce works closely with **HDFS** for data storage and **YARN** for resource management. The final output is written back to **HDFS**.

MapReduce provides **parallel processing, data locality, fault tolerance, and scalability**.