Total No. of Questions: 6 *Total No. of Printed Pages:3*

Enrollment No.....

Faculty of Management Studies
End Sem Examination Dec 2024
MS5EB04 Big Data Analytics & Hadoop

Programme: MBA Branch/Specialisation: Business Analytics

Duration: 3 Hrs. **Maximum Marks: 60**

Note: All questions are compulsory. Internal choices, if any, are indicated. Answers of Q.1 (MCQs) should be written in full instead of only a, b, c or d. Assume suitable data if necessary. Notations and symbols have their usual meaning.

			Marks	BL	PO	CO	PSO
Q.1	i.	Which of the following is NOT one of the 4Vs of big data? (a) Volume (b) Variety (c) Velocity (d) Veracity	**1**	1	1	1	
	ii.	What does the YARN (Yet Another Resource Negotiator) component in hadoop primarily handle? (a) Data storage (b) Task scheduling (c) Resource management and job scheduling (d) Data processing	**1**	2		2	
	iii.	Which of the following is true regarding HDFS? (a) HDFS is a distributed file system that stores data on a single machine (b) Data in HDFS is split into blocks and replicated across multiple machines (c) HDFS does not provide redundancy for data storage (d) HDFS is a database system for structured data	**1**	2		2	
	iv.	In MapReduce, which component is responsible for sorting the output before passing it to the reducer? (a) Mapper (b) Combiner (c) Partitioner (d) Shuffler	**1**	2	2	2	
	v.	In PySpark, which transformation operation would you use to combine two RDDs into a single RDD? (a) filter() (b) map() (c) reduce() (d) union()	**1**	3	2	3	

P.T.O.

vi.	Which of the following is the primary difference between RDDs and DataFrames in spark? (a) RDDs are immutable, while DataFrames are mutable (b) DataFrames provide optimizations such as catalyst optimizer, while RDDs do not (c) RDDs are more efficient than DataFrames for structured data (d) DataFrames are used for batch processing, whereas RDDs are used for stream processing	**1**	1	1	2
vii.	Which of the following is NOT a critical component of data architecture for AI/ML initiatives? (a) Data storage (b) Data recovery (c) Data visualization (d) Data labeling	**1**	4	2	4
viii.	Which of the following is a major challenge in AI/ML initiatives? (a) Lack of data labeling tools (b) Ensuring data privacy and protection (c) Absence of data storage systems (d) Limited computational power	**1**	2	5	5
ix.	Which of the following is a key pillar of a successful AI/ML initiative? (a) Data quality (b) Data collection (c) Data visualization (d) Data destruction	**1**	1	5	5
x.	Which of the following is a major challenge in AI/ML initiatives? (a) Lack of data labeling tools (b) Ensuring data privacy and protection (c) Absence of data storage systems (d) Limited computational power	**1**	2	5	5
Q.2 i.	What is big data, and what are the key characteristics that define it?	**2**	1	1	1
ii.	Explain about the Structured APIs.	**3**	2	3	3
iii.	List and briefly explain the 4Vs of big data.	**5**	1	1	1
OR iv.	Design a simple data processing pipeline using. Your pipeline should perform transformations on RDDs, aggregate the data, and then use Spark SQL to query the result. Demonstrate how this can be applied to a real-world dataset like user interactions or sales data.	**5**	3	3	3

Q.3 i.	Write features and Limitations of HDFS.	**2**	2	1	1
ii.	Discuss the key components of Hadoop's ecosystem. How does the HDFS work, and what are its limitations? Provide an example of data storage and retrieval in HDFS.	**8**	3	2	2
OR iii.	How does map reduce using python? Justify your answer.	**8**	4	2	2
Q.4 i.	Differentiate between Spark and Mapreduce.	**3**	4	3	3
ii.	Compare apache spark with hadoop MapReduce in terms of performance, ease of use, and ecosystem features.	**7**	5	3	3
OR iii.	Explain word count example in spark.	**7**	3	3	3
Q.5 i.	Explain how data would flow through the collect, store, process, and consume phases.	**4**	3	3	3
ii.	Explain the principles of data architecture for AI/ML systems. Discuss how these principles align with organizational goals such as scalability, flexibility, and performance.	**6**	3	3	4
OR iii.	Describe the different types of data storage systems (relational, NoSQL, distributed file systems) and explain how each can be used in various phases of AI/ML data architecture.	**6**	4	4	4
Q.6	Attempt any two:				
i.	What are the main differences between data recovery and data protection?	**5**	4	4	4
ii.	Explain the significance of data labeling and data context in the design of data architecture for AI/ML initiatives.	**5**	2	4	4
iii.	Explain the phases involved in the data architecture process for an AI/ML initiative. Discuss the tools and technologies used in each phase.	**5**	3	4	4

******

Marking Scheme
MS5EB04 Big Data Analytics & Hadoop (T)

Q.1 i) **1 mark awarded to student** **1**

ii) **Answer:** c) Resource management and job scheduling **1**

iii) **Answer:** b) Data in HDFS is split into blocks and replicated across multiple machines **1**

iv) **Answer:** d) Shuffler **1**

v) **Answer:** b) Data Frames provide optimizations such as Catalyst optimizer, while RDDs do not **1**

vi) **Answer:** d) union () **1**

vii) **Answer:** b) Data visualization c) Data Visualization **1**

viii) **Answer:** b) Ensuring data privacy and protection **1**

ix) **Answer:** a) Data Quality **1**

x) **Answer**: b) Ensuring data privacy and protection **1**

Q.2 i. What is Big Data, and what are the key characteristics that define it? **2**

Big Data refers to large and complex datasets that are difficult to process and analyze using traditional data management tools and techniques. These datasets often involve high volumes of structured, semi-structured, and unstructured data from various sources. **1**

Key Characteristics of Big Data:

1. **Volume**: Refers to the large amount of data generated every second. This could be terabytes, petabytes, or even exabytes of data.
2. **Variety**: Big Data comes in many different formats, including structured (like databases), semi-structured (like XML), and unstructured data (like text, images, videos, etc.). **1**
3. **Velocity**: Refers to the speed at which data is generated and needs to be processed. In the modern era, data is often generated in real-time, requiring fast processing.
4. **Veracity**: Refers to the quality and reliability of the data. With large datasets, the accuracy of data can vary, and data may contain errors or inconsistencies.

ii. Explain about the Structured APIs. **3**

Answer: **1**

Structured APIs are interfaces that allow users to interact with structured data in a systematic, organized manner. These APIs provide a way for applications to retrieve, manipulate, and manage structured data, typically represented in rows and columns (like relational databases or tables). They support structured data formats such as SQL tables, Data Frames, and Datasets.

Key Features of Structured APIs:

1. **Data Abstraction**: Structured APIs abstract the underlying complexities of data processing by providing high-level operations (like SQL queries, aggregations, etc.) to interact with structured data. **2**
2. **Optimization**: These APIs often include built-in optimizations, such as query optimization, indexing, and caching. For example, Apache Spark's **Catalyst Optimizer** in Structured APIs optimizes SQL queries for performance.
3. **Integration with DataFrames and Datasets**: Structured APIs are closely integrated with DataFrames and Datasets, enabling efficient data handling and manipulation through high-level functions such as filtering, grouping, and joining data.

Example:
In **Apache Spark**, Structured APIs are used through the

DataFrame and **Dataset** abstractions to allow users to execute SQL-like queries directly on data.

iii. List and briefly explain the 4Vs of Big Data **5**

The **4Vs of Big Data** are essential characteristics that define the complexity and challenges of handling large datasets. They are:

1. **Volume**:
   - **Explanation**: Refers to the **vast amount of data** generated and collected from various sources such as social media, sensors, devices, transactions, and more. Big Data typically involves large datasets that cannot be processed using traditional data management tools.
   - **Example**: Social media platforms generating terabytes of data every day from user interactions, posts, and images.
2. **Variety**:
   - **Explanation**: Refers to the **diverse types and formats of data**. Big Data can be structured (like databases and tables), semi-structured (like JSON or XML), or unstructured (like text, images, audio, and videos). Managing and integrating this variety of data types is a significant challenge.
   - **Example**: Combining data from databases (structured), sensor logs (semi-structured), and social media posts (unstructured).
3. **Velocity**:
   - **Explanation**: Refers to the **speed at which data is generated and needs to be processed**. Real-time or near-real-time processing is required in many Big Data scenarios, where decisions must be made quickly based on incoming data streams.
   - **Example**: Streaming data from financial transactions or social media feeds that require immediate processing to detect fraud or trends.
4. **Veracity**:
   - **Explanation**: Refers to the **trustworthiness and quality of the data**. With massive amounts of data,

ensuring that the data is accurate, consistent, and reliable is crucial for making sound decisions. Incomplete or incorrect data can lead to misleading results.
   - **Example**: Inaccurate or missing sensor readings in IoT systems that may affect predictive maintenance models.

Marks Distribution:

- **Volume** (1 mark)
- **Variety** (1 mark)
- **Velocity** (1 mark)
- **Veracity** (1 mark)
- **Example for each V** (1 mark)

OR iv. **Design a simple data processing pipeline using Apache Spark. Your pipeline should perform transformations on RDDs, aggregate the data, and then use Spark SQL to query the result. Demonstrate how this can be applied to a real-world dataset like user interactions or sales data. Question: Design a simple data processing pipeline using Apache Spark. Your pipeline should perform transformations on RDDs, aggregate the data, and then use Spark SQL to query the result. Demonstrate how this can be applied to a real-world dataset like user interactions or sales data. (5 Marks)** **5**

Answer:

To design a simple data processing pipeline using Apache Spark, we will follow these steps:

1. Data Collection (RDD Creation)

First, we create an RDD from a sample dataset (e.g., user interaction or sales data). Let's assume we're working with a sales dataset where each entry contains information about a product purchase (e.g., product_id, user_id, timestamp, amount).

Sample dataset:

product_id, user_id, timestamp, amount

1, 101, 2024-01-01 10:00, 50

2, 102, 2024-01-01 10:30, 150

1, 103, 2024-01-01 11:00, 75

3, 104, 2024-01-01 12:00, 200

2, 105, 2024-01-01 12:30, 100

Spark code to create an RDD:

```
from pyspark import SparkContext


sc = SparkContext("local", "Sales Data Processing")

sales_data = sc.textFile("sales_data.csv")
```

2. Data Transformation on RDDs

We can perform transformations on the RDD, such as filtering, mapping, or aggregating data. For example, we can filter the sales data for purchases greater than a certain amount and then aggregate the total sales by product.

Transformation example:

```
# Remove header and split data

header = sales_data.first()

filtered_sales_data = sales_data.filter(lambda line: line != header)
\

                .map(lambda line: line.split(','))
```



```
# Filter for sales above $100

high_value_sales = filtered_sales_data.filter(lambda x: float(x[3])
> 100)


# Map to (product_id, amount) and aggregate by product_id

product_sales = high_value_sales.map(lambda x: (x[0],
float(x[3]))) \

                .reduceByKey(lambda a, b: a + b)
```

3. Convert to DataFrame and Use Spark SQL

After performing RDD transformations, we can convert the RDD to a DataFrame, which allows us to leverage Spark SQL for querying the data. For example, let's query the total sales by product.

Conversion to DataFrame:

```
from pyspark.sql import SparkSession


spark = SparkSession.builder.appName("Sales Data
SQL").getOrCreate()


# Convert the RDD to DataFrame

sales_df = spark.createDataFrame(product_sales, ["product_id",
"total_sales"])


# Register the DataFrame as a SQL temporary table

sales_df.createOrReplaceTempView("sales")
```

4. Querying the Data with Spark SQL

Once the DataFrame is registered as a temporary table, we can run SQL queries to analyze the data.

Spark SQL Query Example:

```
# Query the total sales by product

query = "SELECT product_id, SUM(total_sales) as total_sales
FROM sales GROUP BY product_id ORDER BY total_sales
DESC"

result = spark.sql(query)



# Show the result

result.show()
```

Real-World Example Application:

For a **sales dataset**, the pipeline could be used to:

- **Filter** sales transactions above a certain threshold (e.g., $100).
- **Aggregate** the sales by product ID to find which products are generating the most revenue.
- **Use Spark SQL** to run complex queries and identify trends, such as top-selling products.

Marks Distribution:

1. **Data Collection & RDD Creation** (1 mark)
    - Explanation of how to load data into RDD and handle a sample dataset.
2. **RDD Transformations** (1 mark)
    - Example of transformations on RDD, such as filtering, mapping, and reducing data.
3. **DataFrame Conversion & Spark SQL** (1 mark)

    - Explanation of converting an RDD to a DataFrame and registering it for SQL queries.
4. **Spark SQL Querying** (1 mark)
    - Example of a SQL query to aggregate and analyze the data.
5. **Real-World Example Application** (1 mark)
    - Demonstration of how the pipeline could be applied to real-world sales data, showing its practical use in data analysis.

This design provides a simple yet effective pipeline for processing and analyzing sales data using Spark, showcasing key concepts such as RDD transformations, aggregation, and SQL querying in a real-world context.

Q.3 i. Write features and Limitations of HDFS. **2**

Question: Write the features and limitations of HDFS. (2 Marks)

Answer:

Features of HDFS:

1. **High Fault Tolerance**:
    - HDFS ensures data reliability by replicating each data block multiple times (default is 3 replicas). If one replica is lost due to a node failure, the system can still retrieve the data from other replicas.
2. **Scalability**:
    - HDFS is highly scalable, allowing data to be stored across multiple nodes in a cluster. As the data volume grows, more nodes can be added to the cluster without affecting the system's performance.
3. **Large Data Storage**:

- HDFS is optimized for storing large files (terabytes to petabytes) in a distributed manner. It efficiently handles very large datasets, which is essential for big data applications.

4. **High Throughput**:
   - HDFS is designed to provide high throughput for data access, making it ideal for applications that require streaming data processing or read-heavy workloads.

Limitations of HDFS:

1. **Small File Handling**:
   - HDFS is not efficient for storing and managing a large number of small files because each file in HDFS is stored as a separate block, which incurs high overhead in terms of metadata and system resources.
2. **Write-Once, Read-Many**:
   - HDFS is optimized for write-once, read-many applications. It does not support random write operations well. Once a file is written, it cannot be modified or updated; it can only be appended to.
3. **Latency**:
   - HDFS is not designed for low-latency data access, so it is not suitable for real-time processing or interactive querying where quick response times are required.
4. **Single NameNode**:
   - HDFS relies on a single **NameNode** to manage the metadata of the entire file system. This creates a single point of failure. If the NameNode goes down, the entire HDFS system can become unavailable unless failover mechanisms are implemented.

Marks Distribution:

- **Features (2 points)**: 1 mark
- **Limitations (2 points)**: 1 mark

ii. Discuss the key components of Hadoop's ecosystem. How does the HDFS work, and what are its limitations? Provide an example of data storage and retrieval in HDFS. **8**

Question: Discuss the key components of Hadoop's ecosystem. How does the HDFS work, and what are its limitations? Provide an example of data storage and retrieval in HDFS. (8 Marks)

Answer:

Key Components of Hadoop's Ecosystem (4 Marks):

1. **HDFS (Hadoop Distributed File System)**:
   - **Function**: HDFS is the storage layer of Hadoop, designed to store large datasets across a distributed cluster of machines. It divides large files into blocks and distributes them across multiple nodes for fault tolerance and parallel processing.
   - **Key components**:
     - **NameNode**: Manages the metadata (structure) of HDFS. It keeps track of where file blocks are stored in the cluster.
     - **DataNode**: Stores the actual data blocks on the local disks of nodes. Each DataNode is responsible for reading/writing data on request.
2. **MapReduce**:
   - **Function**: The processing layer of Hadoop. MapReduce enables parallel data processing by dividing tasks into smaller chunks. The **Map** phase

processes input data, while the **Reduce** phase aggregates the results.

3. **YARN (Yet Another Resource Negotiator)**:
    - **Function**: YARN manages resources and job scheduling. It allocates resources to different applications and maintains job execution across the cluster.
    - **Components**:
        - **ResourceManager**: Manages resources and schedules jobs.
        - **NodeManager**: Manages resources on individual nodes and monitors resource usage.
4. **Hive**:
    - **Function**: A data warehouse system built on top of Hadoop for querying and managing large datasets using SQL-like queries. It abstracts the complexity of Hadoop's MapReduce by providing a query language called **HiveQL**.
5. **Pig**:
    - **Function**: A platform for analyzing large datasets that provides a higher-level scripting language called **Pig Latin**. It simplifies the complex MapReduce programming model.
6. **HBase**:
    - **Function**: A NoSQL database that runs on top of HDFS. It is designed for real-time read/write access to large datasets, unlike HDFS, which is optimized for batch processing.
7. **Zookeeper**:
    - **Function**: A coordination service for distributed applications. It is used to manage and synchronize distributed systems (e.g., coordinating HDFS replication and handling job scheduling).
8. **Sqoop**:
    - **Function**: A tool for transferring bulk data between Hadoop and relational databases. It allows for importing data from databases into HDFS and

exporting data from HDFS back to relational databases.

How HDFS Works (2 Marks):

1. **Data Splitting and Block Distribution**:
    - When a file is uploaded to HDFS, it is split into large blocks (typically 128 MB or 256 MB each). These blocks are then distributed across different DataNodes in the cluster. This ensures data is spread out for parallel processing and high availability.
2. **Replication**:
    - For fault tolerance, each block is replicated (usually 3 copies) across different DataNodes. This ensures that even if one node or block fails, there are multiple copies available for access.
3. **Metadata Management**:
    - The **NameNode** stores metadata (such as file structure and block locations) in memory. It does not store the actual data but keeps track of where the blocks are located. The **DataNodes** store the actual data blocks.
4. **Data Access**:
    - When a user or application requests data, the **Client** communicates with the NameNode to find out where the blocks are located. The client then communicates directly with the DataNodes to retrieve the data.

Limitations of HDFS (1 Mark):

1. **Small File Handling**:
    - HDFS is not efficient for storing many small files because each file is treated as a separate block.

Managing metadata for a large number of small files increases overhead.

2. **Write-Once, Read-Many Model**:
    - HDFS is optimized for write-once, read-many scenarios. It does not allow for random writes or modifications to existing files. You can only append data to existing files.
3. **Single Point of Failure (NameNode)**:
    - The **NameNode** is a single point of failure in HDFS. If it goes down, the entire system is impacted. Though high-availability configurations can be set up, it remains a vulnerability.
4. **Latency**:
    - HDFS is designed for high-throughput, batch processing, and is not suitable for low-latency access or real-time processing.

Example of Data Storage and Retrieval in HDFS (1 Mark):

Data Storage: To store data in HDFS, you first need to upload it using the hadoop fs command:

hadoop fs -put localfile.txt /user/hadoop/input/

- This command stores the local file localfile.txt in the HDFS directory /user/hadoop/input/.

Data Retrieval: To retrieve data from HDFS, you can use the -get command to copy it back to the local file system:

hadoop fs -get /user/hadoop/input/localfile.txt /localpath/

- This command retrieves the localfile.txt from HDFS and stores it at /localpath/ on the local file system.

Reading from HDFS using a Java Application: In Java, you can

use the **FileSystem** API to read and write files in HDFS:

```
Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(new
URI("hdfs://namenode_host:9000"), conf);


// Reading file from HDFS

Path filePath = new Path("/user/hadoop/input/localfile.txt");

FSDataInputStream in = fs.open(filePath);

BufferedReader br = new BufferedReader(new
InputStreamReader(in));


String line;

while ((line = br.readLine()) != null) {
   System.out.println(line);
}

br.close();
```

Marks Distribution:

1. **Key Components of Hadoop's Ecosystem** (4 Marks):
    - Explanation of HDFS, MapReduce, YARN, Hive, Pig, HBase, Zookeeper, Sqoop.
2. **How HDFS Works** (2 Marks):
    - Description of data splitting, block distribution, replication, metadata management, and access.
3. **Limitations of HDFS** (1 Mark):
    - Explanation of small file handling, write-once model, single NameNode, and latency issues.

4. **Example of Data Storage and Retrieval in HDFS** (1 Mark):
    - Practical example of storing and retrieving data from HDFS using commands or code.

This answer provides a comprehensive understanding of Hadoop's ecosystem, HDFS operations, its limitations, and a practical example of data storage and retrieval.

OR iii. How does Map reduce using python . Justify your answer. **8**

Marks Distribution:

1. **Introduction to MapReduce** (2 Marks):
   Explanation of the **Map** and **Reduce** phases and the general working of MapReduce.
2. **MapReduce Implementation in Python using Hadoop Streaming** (3 Marks):
   Detailed steps to implement the **Mapper** and **Reducer** in Python and how to run the job using Hadoop Streaming.
3. **Justification for Using Python in MapReduce** (2 Marks):
   Explanation of the benefits of using Python for MapReduce jobs, including ease of use, integration with Hadoop, and availability of libraries.
4. **Example of Data Storage and Retrieval** (1 Mark):
   A practical example of how data is processed, stored, and retrieved from the MapReduce job.

Q.4 i. Differentiate between Spark and Mapreduce. **3**

Marks Distribution:

1. **Execution Model** (1 Mark): Explanation of how Spark uses in-memory processing versus MapReduce's disk-based execution.
2. **Performance** (1 Mark): Comparison of performance based on disk vs in-memory processing.
3. **Fault Tolerance** & **Data Processing** (1 Mark): Overview of fault tolerance and types of data processing supported by Spark and MapReduce

ii. Compare Apache Spark with Hadoop MapReduce in terms of performance, ease of use, and ecosystem features. **7**

Marks Distribution:

1. **Performance** (2 Marks):
    - Explanation of Spark's **in-memory processing** and how it makes Spark faster than MapReduce (disk-based processing).
    - Comparison of **real-time processing capabilities** in Spark vs batch processing in MapReduce.
2. **Ease of Use** (2 Marks):
    - Comparison of **programming complexity** between Spark and MapReduce.
    - Mention of **higher-level APIs** and the **unified programming model** in Spark, which improves ease of use compared to the more complex MapReduce model.
3. **Ecosystem Features** (3 Marks):
    - Comparison of the **Hadoop ecosystem** (MapReduce) with the **rich ecosystem** of Spark (including SQL, machine learning, streaming, and graph processing).
    - Highlight of **Spark's unified framework** for various workloads, making it a more comprehensive solution compared to MapReduce's batch-processing limitation.

OR iii. Explain word count Example in spark. **7**

Marks Distribution:

1. **Overview of the Word Count Example (1 Mark)**:
   - Brief description of the word count problem and its purpose.
2. **Detailed Steps for Word Count (3 Marks)**:
   - Explanation of how data is processed using Spark's RDDs and transformations (map, reduce, etc.).
   - Code examples for reading input, processing data, and aggregating results.
3. **Explanation of Key Concepts (2 Marks)**:
   - Understanding of **RDDs**, **transformations**, **actions**, and **in-memory processing** in Spark.
4. **Output (1 Mark)**:
   - Expected output with the word counts.

Q.5 i. Explain how data would flow through the Collect, Store, Process, and Consume phases. **4**

Marks Distribution:

1. **Collect Phase (1 Mark)**: Explanation of how data is gathered from various sources, including batch and real-time ingestion methods.
2. **Store Phase (1 Mark)**: Description of how data is stored in databases, data lakes, or cloud storage.
3. **Process Phase (1 Mark)**: Explanation of how data is processed (transformed, cleaned, aggregated) using tools like Spark or MapReduce.
4. **Consume Phase (1 Mark)**: Clarification of how processed data is consumed or delivered to users or other systems, such as through visualizations, reports, or APIs.

ii. Explain the principles of Data Architecture for AI/ML systems. Discuss how these principles align with organizational goals such **6**

as scalability, flexibility, and performance.

Marks Distribution:

1. **Data Quality** (1 Mark): Importance of accurate, complete, and consistent data.
2. **Data Accessibility and Integration** (1 Mark): Ensuring data from diverse sources is accessible and integrated.
3. **Real-Time Data Processing** (1 Mark): Importance of processing data in real-time for AI/ML applications.
4. **Data Security and Privacy** (1 Mark): Protecting sensitive data while ensuring compliance and security.
5. **Scalability and Elasticity** (1 Mark): Enabling the architecture to scale and adapt to growing data needs.
6. **Automation and Orchestration** (1 Mark): Streamlining processes to improve efficiency and scalability.

OR iii. Describe the different types of data storage systems (relational, NoSQL, distributed file systems) and explain how each can be used in various phases of AI/ML data architecture. **6**

Question: Describe the different types of data storage systems (relational, NoSQL, distributed file systems) and explain how each can be used in various phases of AI/ML data architecture. (6 Marks)

Answer:

Data storage is a crucial component in AI/ML data architecture, as it determines how data is stored, accessed, and processed at each stage of the machine learning pipeline. Different types of data storage systems are designed to meet the specific requirements of data volume, speed, complexity, and scalability, which are crucial in AI/ML initiatives. The three main types of data storage systems are **Relational Databases**, **NoSQL Databases**, and **Distributed File Systems**. Let's examine these in detail and discuss how each is used in the different phases of AI/ML data architecture: **Collect, Store, Process, and Consume**.

---

1. Relational Databases (1.5 Marks)

Description:
Relational databases (RDBMS) use a structured format with tables, rows, and columns to store data. They are built on SQL (Structured Query Language) and support ACID (Atomicity, Consistency, Isolation, Durability) properties for reliable transactions. Examples include **MySQL**, **PostgreSQL**, **Oracle**, and **SQL Server**.

Use in AI/ML Phases:

- **Collect Phase**: Relational databases may be used to **store transactional or structured data** collected from internal systems, such as user profiles, transactions, and logs, which are important for building features for machine learning models.
- **Store Phase**: Data from the collect phase is stored in relational tables with predefined schema. This is suitable for structured data that can be organized into tables (e.g., customer data, sales data).
- **Process Phase**: SQL queries can be used to **filter, aggregate, and transform** data for training AI/ML models. For example, aggregating user activity data to create features for recommendation systems.
- **Consume Phase**: Relational databases can provide data to visualization tools or reporting systems, and users can query the data for insights. Data is consumed by analysts, decision-makers, or external systems for reporting and decision-making.

---

2. NoSQL Databases (1.5 Marks)

Description:
NoSQL databases are designed for **unstructured** or **semi-**

structured data and are highly scalable. They are flexible in terms of schema design and can handle high volumes of diverse data types (text, images, JSON, etc.). Examples include **MongoDB**, **Cassandra**, **Couchbase**, and **Redis**.

Use in AI/ML Phases:

- **Collect Phase**: NoSQL databases are commonly used to **collect unstructured or semi-structured data** like social media posts, sensor data, logs, or user interactions. This is particularly important for AI/ML projects that involve large volumes of varied data, such as image or text data.
- **Store Phase**: NoSQL storage systems such as **document stores** (e.g., MongoDB) or **wide-column stores** (e.g., Cassandra) allow for flexible schema designs and efficient storage of large, unstructured datasets. Data such as customer reviews, clickstreams, or sensor readings is stored in a distributed, scalable manner.
- **Process Phase**: In the process phase, NoSQL databases are used to quickly access and process large datasets. For instance, data preprocessing tasks for training machine learning models may involve transforming raw JSON data into structured features, which can be efficiently done using NoSQL systems.
- **Consume Phase**: NoSQL systems are often queried in real-time for applications such as recommendation engines, personalized content delivery, or fraud detection, where data needs to be accessed and consumed quickly for decision-making.

---

3. Distributed File Systems (2 Marks)

Description:
Distributed File Systems, such as **HDFS (Hadoop Distributed File System)** or **Amazon S3**, are designed to store large volumes of data across multiple nodes in a cluster. They provide **high availability**, **fault tolerance**, and **scalability**. These systems are

essential for big data processing and AI/ML applications that require high throughput for large datasets.

Use in AI/ML Phases:

- **Collect Phase**: Data from various sources (e.g., logs, sensors, internet of things (IoT) devices) can be collected and stored in **distributed file systems** for further processing. The system can handle diverse data formats like images, videos, and text.
- **Store Phase**: Distributed file systems are ideal for storing **massive datasets** generated in AI/ML projects. For example, in training deep learning models, large datasets (such as image datasets) are stored across multiple nodes, ensuring reliability and scalability.
- **Process Phase**: Distributed file systems are highly effective in the processing phase, especially when combined with tools like **Apache Spark** or **MapReduce**. These tools allow for **parallel processing** of data stored across the distributed file system, enabling faster processing of large-scale datasets for AI/ML model training.
- **Consume Phase**: After processing and training models, results (predictions, analysis) can be stored or retrieved from distributed file systems for consumption by end-users, business intelligence tools, or decision-making systems. For instance, the output from a machine learning model can be saved in **HDFS** or **Amazon S3** and then queried or used in applications.

Comparison Table:

Storage System	Data Type	Use in AI/ML Phases (Collect, Store, Process, Consume)
Relational Databases	Structured data (tables, rows)	Collect: Internal structured data; Store: Predefined schema; Process: SQL for transformation; Consume: Reporting and decision-making
NoSQL Databases	Unstructured/Semi-structured (e.g., JSON, logs, sensor data)	Collect: User interactions, logs, sensor data; Store: Flexible schema; Process: Data wrangling; Consume: Real-time applications (e.g., recommendations)
Distributed File Systems	Large-scale datasets (images, videos, large text files)	Collect: Big data from various sources; Store: Mass data across nodes; Process: Parallel processing (Hadoop, Spark); Consume: Results storage (e.g., S3, HDFS)

Conclusion:

Each type of data storage system plays a critical role in the AI/ML pipeline. **Relational databases** are ideal for structured data and transactional systems, **NoSQL databases** excel in managing unstructured or semi-structured data, and **distributed file systems** provide the scalability and performance needed for large-scale data storage and processing. By using these systems in tandem, organizations can build a robust, scalable, and efficient data architecture for AI/ML projects.

Marks Distribution:

1. **Relational Databases** (1.5 Marks): Description and use in AI/ML pipeline phases.
2. **NoSQL Databases** (1.5 Marks): Description and use in

AI/ML pipeline phases.

3. **Distributed File Systems** (2 Marks): Detailed explanation and use in AI/ML pipeline phases.
4. **Comparison Table** (1 Mark): Summary table comparing the three storage systems and their uses in AI/ML phases.

Q.6

i. What are the main differences between Data Recovery and Data Protection? **5**

Marks Distribution:

1. **Definition** (1 Mark): Clear distinction between Data Protection and Data Recovery.
2. **Objectives** (1 Mark): Focus on prevention (Data Protection) vs. restoration (Data Recovery).
3. **Key Methods and Technologies** (1.5 Marks): Detailed explanation of methods used in both.
4. **Timing of Action** (1 Mark): Emphasis on proactive vs. reactive nature of each.
5. **Example Scenarios** (1.5 Marks): Illustrating real-world applications of both concepts.

ii. Explain the significance of data labeling and data context in the design of data architecture for AI/ML initiatives. **5**

Marks Distribution:

1. **Data Labeling** (2 Marks): Explanation of data labeling and its role in supervised learning and model accuracy.
2. **Data Context** (3 Marks): Detailed explanation of data context, its importance in interpreting data, feature engineering, bias mitigation, and model performance

iii. Explain the phases involved in the data architecture process for an AI/ML initiative. Discuss the tools and technologies used in each **5**

phase.

Marks Distribution:

- **Collect Phase (1.5 Marks)**: Explanation of data ingestion and tools.
- **Store Phase (1 Mark)**: Explanation of data storage and tools.
- **Process Phase (1.5 Marks)**: Explanation of data processing and tools.
- **Consume Phase (1 Mark)**: Explanation of data consumption and tools.

4o mini

******