VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Department of Mathematics

School of Advanced Sciences

Vellore Institute of Technology, Vellore.

PMDS507L- Big Data Analytics

Module – 1 (Understanding Big Data)

Introduction to Big Data:

Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. Big data is data that is big in volume, velocity, and variety.

According to Gartner, the definition of Big Data –

"Big data" is high-volume, velocity, and variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making."

However, there are certain basic tenets of Big Data that will make it even simpler to answer what Big Data is:

- It refers to a massive amount of data that keeps on growing exponentially with time.
- It is so voluminous that it cannot be processed or analyzed using conventional data processing techniques.
- It includes data mining, data storage, data analysis, data sharing, and data visualization.
- The term is an all-comprehensive one including data, data frameworks, along with the tools and techniques used to process and analyze the data.

The History of Big Data :

The evolution of Big Data spans several decades, shaped by advancements in data storage, processing, and analysis technologies:

1960s–1970s: Foundation Era

The origins of Big Data can be traced back to the establishment of the first data centers and the development of relational database systems, notably advanced by Edgar F. Codd. This period marked a significant milestone in the history of data management, introducing structured approaches to storing, organizing, and retrieving information efficiently.

1980s–1990s: Growth of Data Systems

The introduction of **data warehouses** and **business intelligence** tools allowed organizations to collect and analyze increasing volumes of structured data.

Early 2000s: Emergence of Big Data

As the internet and digital services expanded, the volume, velocity, and variety of data grew exponentially. In 2005, **Roger Mougalas** from O'Reilly Media coined the term **"Big Data"** to describe datasets that were too large to process using traditional tools. Around this time, **Google** introduced the **MapReduce** programming model and **Google File System (GFS)** to handle large-scale data processing, which inspired the creation of **Apache Hadoop** a major breakthrough in distributed computing.

2010s: Big Data Technologies Maturity

Open-source platforms like **Hadoop**, **Spark**, and **NoSQL** databases (e.g., MongoDB, Cassandra) became widely adopted. Cloud computing platforms such as **AWS**, **Azure**, and **Google Cloud** enabled scalable storage and real-time data analytics. The rise of **machine learning** and **AI** further emphasized the importance of Big Data in predictive analytics and automation.

2020s–Present: Data-Driven Economy

Today, Big Data is a critical asset for industries, governments, and researchers. Technologies like **edge computing**, **IoT**, and **real-time stream processing** have expanded the data ecosystem. With growing focus on **data ethics**, **privacy**, and **regulatory compliance**, organizations are challenged to manage not only the size but also the responsibility of handling massive datasets.

Importance of Big Data :

Big Data has become a critical asset in the digital age, driving innovation, efficiency, and strategic decision-making across a wide range of industries. Its importance lies in the ability to extract meaningful insights from massive and complex datasets that were previously difficult or impossible to analyze using traditional methods.

1. **Informed Decision-Making:**

Big Data enables organizations to make evidence-based decisions by uncovering patterns, trends, and correlations within large datasets. This leads to more accurate forecasting, risk assessment, and policy development.

2. **Enhanced Operational Efficiency:**

By analyzing real-time and historical data, businesses can identify inefficiencies, optimize processes, and reduce operational costs. In manufacturing, logistics, and supply chain management, Big Data contributes to better resource utilization and productivity.

3. **Personalized Customer Experience**

In sectors like retail, finance, and healthcare, Big Data allows for detailed analysis of consumer behavior, preferences, and feedback. This enables organizations to deliver customized services, improve customer satisfaction, and build stronger relationships.

4. **Real-Time Monitoring and Predictive Analytics**

Big Data technologies support real-time data processing and predictive modeling. This is particularly valuable in areas such as fraud detection, financial forecasting, and equipment maintenance, where early intervention can prevent major losses.

5. **Smart Systems and Automation**

Big Data powers intelligent systems in smart cities, autonomous vehicles, and IoT devices. These systems rely on continuous data streams to make decisions, optimize energy usage, manage traffic, and improve quality of life.

Types of Big Data :

A) **Supervised:**
Structured data refers to processed, stored, and retrieved data in a fixed format. It relates to highly organized information that can be readily and seamlessly stored and accessed from a database by simple search engine algorithms. For instance, the employee table in a company database will be structured as the employee details, their job positions, their salaries, etc., will be present in an organized manner.

B) **Unsupervised data:**
Unsupervised data refers to a dataset that lacks predefined labels or target outputs. This makes it very difficult and time-consuming to process and analyze unstructured data. Email is an example of unstructured data.

C) **Semi-supervised data:**
Semi-supervised data that combines elements of both supervised and unsupervised learning. It utilizes a dataset that contains both a small amount of labeled data and a large amount of unlabeled data.

Characteristics of Big Data :

The four Vs of Big Data are Volume, Velocity, Variety, and Veracity, describing the characteristics of data that make it "big".

Volume: Volume is one of the characteristics of big data. We already know that Big Data indicates huge volumes of data that is being generated on a daily basis from various sources like social media platforms, business processes, machines, networks, human interactions, etc. Such a large amount of data is stored in data warehouses.

Velocity: Velocity essentially refers to the speed at which data is being created in real-time. In a broader perspective, it comprises the rate of change, linking of incoming data sets at varying speeds, and activity bursts.

Variety: Variety of Big Data refers to structured, unstructured, and semistructured data that is gathered from multiple sources. While in the past, data could only be collected from spreadsheets and databases, today data comes in an array of forms such as emails, PDFs, photos, videos, audios, SM posts, and so much more.

Veracity: Veracity in big data refers to the accuracy, reliability, and trustworthiness of the data. For example, in an autonomous vehicle's system, sensors provide data, but the software must accurately interpret it to determine if something is a collision threat, demonstrating how data must be accurate for reliable action.

Applications of Big Data:

Big data applications are software and systems that leverage large and complex datasets to extract insights, support decision-making, and address various challenges across different industries.

Healthcare :

- Predictive diagnostics
- Medical imaging analysis
- Electronic Health Records (EHR)

Retail and E-Commerce:

- Customer behavior analysis
- Dynamic pricing
- Inventory optimization

Banking and Finance:

- Fraud detection
- Risk management
- Customer insights

Transportation and Logistics:

- Route optimization
- Fuel efficiency
- Demand forecasting
- Improve delivery performance

Government & Public Services:

- Smart city planning

- Public safety monitoring
- Census & policy development

Agriculture:

- Precision farming
- Yield prediction
- Resource optimization

Distributed File System (DFS) :

In the context of Big Data, a Distributed File System (DFS) is a scalable data storage and management system that distributes and stores large files across multiple servers or nodes within a network. It enables users and applications to access and process data seamlessly, as though it resides on a single local device, while ensuring fault tolerance, high availability, and efficient data retrieval.

Key Features of DFS:

Scalability: Easily expands by adding more nodes.

Fault Tolerance: Redundant copies ensure reliability.

Parallel Access: Supports multiple users simultaneously.

Example:

HDFS (Hadoop DFS) – used in Big Data.

GFS (Google File System) – used by Google.

Amazon S3 – cloud-based distributed object storage.

Components of DFS:

Name Node: Manages metadata.

Data Nodes: Store actual file blocks.

Client Nodes: Access and interact with the DFS.

Algorithms using MapReduce:

Here are several commonly used **algorithms that can be implemented using the MapReduce framework**, especially for processing large-scale data in distributed environments:

1. Word Count - Counts the frequency of each word in a large text dataset.
2. Sorting - Sorts data based on keys
3. PageRank - Ranks web pages based on their importance using link structure.
4. K-Means Clustering - Groups data points into K clusters.
5. Join Operations - Combines data from multiple datasets based on a common key.
6. Matrix Multiplication - Performs large matrix operations in parallel (used in machine learning, graph algorithms).

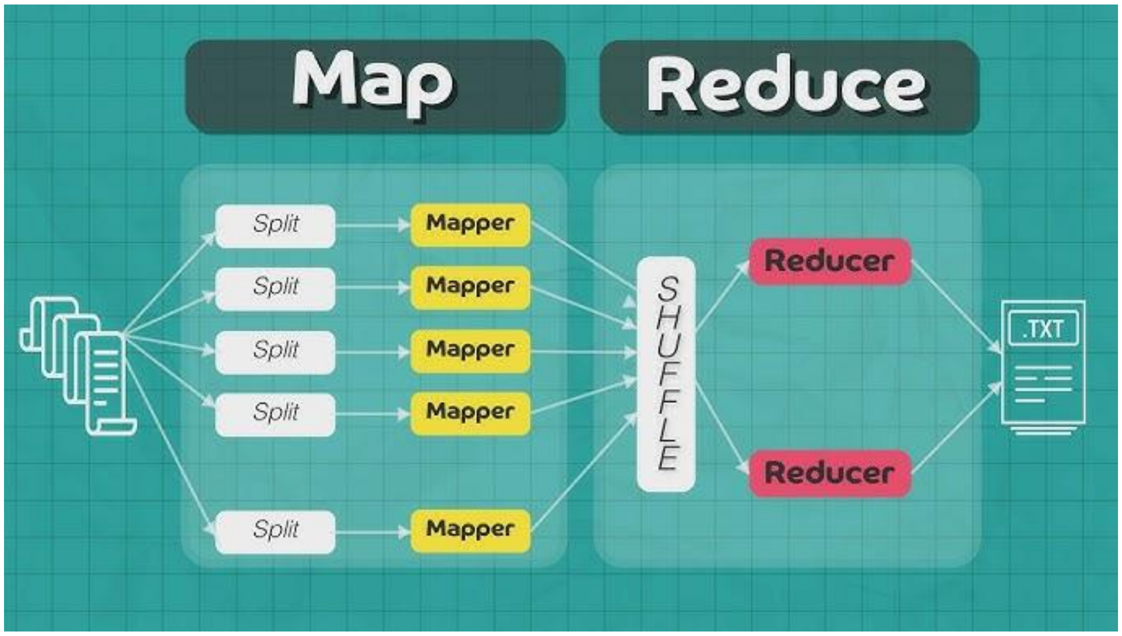
Question Bank:

1. What is Big Data analytics? What are the challenges in Big Data analytics? Why is Big Data considered important in the modern digital age?
2. Discuss the evolution of Big Data across different decades. How did technological advancements contribute to this evolution?
3. List and explain the key characteristics of Big Data.
4. Discuss the role of structured, unstructured, and semi-structured data in Big Data analytics. How do their formats influence data processing methods and tools?
5. Explain the four V's of Big Data. How do they differentiate Big Data from traditional data?
6. What is a Distributed File System (DFS) in Big Data? Describe its architecture and explain its key features.
7. Applications of Big Data Analytics.
8. Why is matrix-vector multiplication important in data analysis?
9. Explain the components and features of DFS.
10. Difference between Traditional Vs Big Data Analytics.

Module – 2 (Map Reduce Applications)

What is MapReduce?

- A distributed programming model for large-scale data processing.
- developed by Google.
- It is designed to process large-scale data sets across clusters of computers efficiently.



MapReduce - Use Cases:

1. Search Indexing

- **What**: Building indexes for search engines (like Google).
- **Why MapReduce**: Efficiently processes huge volumes of web data (crawled pages), tokenizes words, and creates inverted indexes.
- **Map**: Emits (word, documentID).
- **Reduce**: Aggregates all document IDs per word.

2. Log Analysis

- **What**: Analyzing server or application logs to detect trends or issues.
- **Why MapReduce**: Handles terabytes of log files across distributed servers.
- **Use Cases**: Error counting, traffic trends, clickstream analysis.

3. Recommendation Systems

- **What**: Suggesting items to users based on past behavior.
- **Why MapReduce**: Processes massive user-item interaction data to calculate similarity scores or preferences.
- **Example**: Amazon product suggestions or Netflix movie recommendations.

4. Data Warehousing and ETL

- **What**: Extracting, Transforming, and Loading data from various sources.
- **Why MapReduce**: Cleanses and aggregates structured/semi-structured data for analytics.
- **Example**: Pre-processing data before loading into Hive or a data warehouse.

5. Bioinformatics / Genomic Data Processing

- **What**: Analyzing DNA sequences, genome assembly.
- **Why MapReduce**: Scales with large genome datasets.
- **Example**: Searching for gene patterns, sequence alignment.

6. Web Crawling & Web Graph Analysis

- **What**: Analyzing the link structure of the web (e.g., PageRank).
- **Why MapReduce**: Efficiently computes on massive graph data.
- **Example**: Ranking web pages based on number/quality of backlinks.

7. Fraud Detection in Banking

- **What**: Identifying suspicious or fraudulent transactions.
- **Why MapReduce**: Aggregates and filters billions of transaction logs for anomalies.

8. Sentiment Analysis / Social Media Mining

- **What**: Analyzing tweets, posts, and reviews for public opinion.
- **Why MapReduce**: Can process large-scale, real-time feeds like Twitter or Facebook.
- **Use Case**: Election analysis, brand monitoring.

MapReduce vs Traditional Parallel Processing:

Feature	**MapReduce**	**Traditional Parallel Processing**
Programming Model	Declarative (Map, Reduce functions)	Procedural/Imperative (threads, MPI, OpenMP, etc.)
Abstraction Level	High-level abstraction	Low-level control over threads/processes
Data Distribution	Automatically handled by the framework (e.g., Hadoop)	Manually managed by the programmer
Fault Tolerance	Built-in (automatic re-execution of failed tasks)	Programmer must handle failures explicitly
Scalability	Highly scalable (handles petabytes of data)	Limited scalability without custom implementation
Ease of Use	Easier to write distributed programs using Map and Reduce	Complex due to manual parallelization and synchronization
Execution Environment	Runs on distributed systems (e.g., Hadoop cluster)	Typically runs on multicore systems or tightly-coupled clusters
Data Locality Optimization	Yes, Map tasks run where data is located	No inherent data locality; must be programmed
I/O Handling	Relies on distributed file systems (e.g., HDFS)	Uses traditional file systems or in-memory operations
Development Time	Shorter (due to abstraction and frameworks)	Longer (due to complexity and error handling)
Communication Between Tasks	Not required (Map and Reduce are independent)	Often needed (e.g., message passing, shared memory)
Load Balancing	Handled by framework automatically	Programmer must manage task distribution and load balancing

MapReduce Workflow:

1. Input Splitting
2. Mapping
3. Shuffling & Sorting
4. Reducing
5. Output Generation

Input Splitting:

- Input files stored in HDFS.
- Files are split into blocks (e.g., 128MB).
- Each block is processed by a separate Mapper.

Map Phase:

- Mapper processes each input split.
- Emits key-value pairs.
- Example: "hello world" → [(”hello”, 1), (”world”, 1)]

Shuffling and Sorting :

- Keys are grouped and sorted.
- All values of a key are sent to a single Reducer.
- Optimized for performance and bandwidth.

Reduce Phase:

- Reducer receives key and list of values.
- Aggregates values.
- Emits final key-value pair.

Example : Word Count

1. **Mapper Phase:**
   Each line is split into words, and for every word, the mapper emits a key-value pair.
   Input:
   Line 1: "cat dog cat fish"
   Line 2: "dog cat bird"

Mapper Output:

(cat, 1)

(dog, 1)

(cat, 1)

(fish, 1)

(dog, 1)

(cat, 1)

(bird, 1)

2. **Shuffle & Sort Phase (Handled by Hadoop):**
   Groups all values by key (i.e., word).
   Grouped Output:
   cat → [1, 1, 1]
   dog → [1, 1]
   fish → [1]
   bird → [1]
3. **Reducer Phase:**
   Sums up the values for each word.
   Reducer Output:
   (cat, 3)
   (dog, 2)
   (fish, 1)
   (bird, 1)
4. **Final Output in HDFS:**
   cat 3
   dog 2
   bird 1
   fish 1

Matrix – Multiplication:

A = | 1 2 3 |

| 4 5 6 |

B = | 7 8 |

| 9 10 |

|11 12 |

C = | 58 64 |

| 139 154 |

Step 1: Represent Matrices as Input Records (Triplets):

Each matrix element is represented as:

(matrix_name, row, column, value)

Matrix A:

(A, 0, 0, 1)

(A, 0, 1, 2)

(A, 0, 2, 3)

(A, 1, 0, 4)

(A, 1, 1, 5)

(A, 1, 2, 6)

Matrix B:

(B, 0, 0, 7)

(B, 0, 1, 8)

(B, 1, 0, 9)

(B, 1, 1, 10)

(B, 2, 0, 11)

(B, 2, 1, 12)

Step 2: Mapper Phase – Emits Key-Value Pairs

Each matrix element is **emitted multiple times** based on the target output position.

For A[i][k], emit:

Key: (i, j) → j from 0 to #columns in B

Value: (A, k, A[i][k])

For B[k][j], emit:

Key: (i, j) → i from 0 to #rows in A

Value: (B, k, B[k][j])

Example Mapper Output:

A[0][*] = 1, 2, 3 (row 0 of A):

Emit for j = 0 and j = 1:

(0,0) → (A,0,1)

(0,0) → (A,1,2)

(0,0) → (A,2,3)

(0,1) → (A,0,1)

(0,1) → (A,1,2)

(0,1) → (A,2,3)

B[*][0] = 7, 9, 11 (column 0 of B):

Emit for i = 0 and i = 1:

(0,0) → (B,0,7)

(0,0) → (B,1,9)

(0,0) → (B,2,11)

(1,0) → (B,0,7)

(1,0) → (B,1,9)

(1,0) → (B,2,11)

B[*][1] = 8, 10, 12 (column 1 of B):

(0,1) → (B,0,8)

(0,1) → (B,1,10)

(0,1) → (B,2,12)

(1,1) → (B,0,8)

(1,1) → (B,1,10)

(1,1) → (B,2,12)

Step 3: Shuffle & Sort Phase

Hadoop groups values by key:

Key = (i, j)

Value = list of (A, k, A[i][k]) and (B, k, B[k][j])

Example for key (0,0):

Values:

(A,0,1), (A,1,2), (A,2,3)

(B,0,7), (B,1,9), (B,2,11)

Step 4: Reducer Phase – Compute Dot Product

For each key (i,j), reducer:

1. Matches all A[i][k] with B[k][j]
2. Computes sum of A[i][k] × B[k][j]

Key (0,0):

A[0][0] = 1, B[0][0] = 7 → 1×7 = 7

A[0][1] = 2, B[1][0] = 9 → 2×9 = 18

A[0][2] = 3, B[2][0] = 11 → 3×11 = 33

Sum: 7 + 18 + 33 = 58

Similar Computations:

Key (0,1):

1×8 + 2×10 + 3×12 = 8 + 20 + 36 = 64

Key (1,0):

4×7 + 5×9 + 6×11 = 28 + 45 + 66 = 139

Key (1,1):

4×8 + 5×10 + 6×12 = 32 + 50 + 72 = 154

Step 5: Final Output:

Key	**Value**
(0,0)	58
(0,1)	64
(1,0)	139
(1,1)	154

Final matrix:

```
|  58   64  |

| 139  154  |
```

MRUnit:

- MRUnit is a Java library that simplifies the unit testing of Hadoop MapReduce jobs.
- It allows developers to test mappers and reducers independently or together, without needing a full Hadoop cluster.
- MRUnit provides a way to define input data and expected output, making it easier to verify the correctness of MapReduce logic.

MRUnit Components:

Component	**Purpose**
MapDriver	Test only the **Mapper** logic
ReduceDriver	Test only the **Reducer** logic
MapReduceDriver	Test both Mapper and Reducer **end-to-end**

When to Use MRUnit:

- During development of new Mappers/Reducers.
- To verify bug fixes.
- For CI pipelines and TDD.
- For education and prototyping.

MRUnit Limitations:

- No simulation of Hadoop cluster or YARN.
- Doesn’t test scalability or performance.
- Limited support for Hadoop 3.x.
- Not ideal for testing distributed failures.
- Doesn’t test cluster behavior.
- No performance benchmarking.
- MRUnit is less maintained in Hadoop 3.x+.
- Alternatives: MiniCluster, Mockito, Apache Crunch.

Alternatives to MRUnit:

- Mini Cluster – lightweight local cluster for integration testing.
- Mockito + JUnit – mock Context objects.
- Apache Crunch – pipeline testing.
- Spark Testing Base – for Apache Spark jobs.

Best Practices for MRUnit Testing:

- Use small, well-scoped test cases.
- Normalize input for deterministic results.
- Test edge cases: empty lines, nulls, special characters.
- Combine MRUnit with integration tests.

Test Data and Local Testing:

- Create small datasets to simulate real-world input.
- Helps identify logic errors before running on the full cluster.
- Reduces turnaround time during development.
- Local mode (standalone) testing doesn’t require HDFS.
- Allows step-by-step debugging of Mapper and Reducer.
- Useful for testing custom Input/Output formats.
- Can be integrated with IDEs like Eclipse or IntelliJ.
- Essential for building confidence in job correctness.

Anatomy of a MapReduce Job:

- Starts with a user submitting a job to the Hadoop system.
- Job client checks input/output paths and JAR dependencies.
- Job is divided into map and reduce tasks.

- Input data is split into logical Input Splits.
- Mappers read data, process, and emit intermediate key-value pairs.
- Intermediate data is shuffled, sorted, and passed to reducers.
- Reducers aggregate or transform data and write the final output.
- Job status is updated and results stored in HDFS.

Job Submission and Configuration:

- Create a ‘Job‘ object using Hadoop API.
- Set configuration parameters (e.g., job name, input/output paths).
- Define Mapper, Reducer, and Combiner classes.
- Specify Input Format and Output Format.
- Configure the number of reduced tasks.
- Set job-level optimizations like compression and memory limits.
- Submit job using ‘job.waitForCompletion(true)‘.
- Monitor job progress through the Hadoop web interface.

Classic MapReduce Architecture:

- Based on Hadoop 1.x architecture.
- Job Tracker manages job scheduling and resource allocation.
- Task Tracker runs Map and Reduce tasks on data nodes.
- Heartbeats are used for status updates and fault detection.
- Handles task re-execution on failure.
- Lacked multi-tenancy and fine-grained resource control.
- Single Job Tracker was a performance and availability bottleneck.
- It was replaced by YARN in Hadoop 2.x for scalability.

Problems with Classic MapReduce:

- Centralized Job Tracker caused scalability issues.
- Poor handling of resource contention among jobs.
- No support for non-MapReduce applications (e.g., Spark).
- Resource allocation was coarse-grained and inefficient.
- Job Tracker is responsible for both scheduling and monitoring.
- It is difficult to manage workloads in multi-user environments.
- No containerization or resource isolation.
- Inefficient memory and CPU usage on large clusters.

Introduction to YARN:

- YARN stands for Yet Another Resource Negotiator.
- It is a resource management layer for the Hadoop ecosystem.
- Introduced in Hadoop 2.0 to overcome scalability limitations.
- Separates resource management and job scheduling/monitoring.
- Enables multiple data processing engines (MapReduce, Spark).
- Acts as an OS for Hadoop clusters.
- Provides APIs for developing distributed applications.
- Improves cluster utilization and scalability.

YARN Architecture Components:

Component	**Role**
ResourceManager (RM)	Master daemon managing **cluster resources** and **job scheduling**
NodeManager (NM)	Per-node agent responsible for **containers** and **monitoring**
ApplicationMaster (AM)	Handles the **lifecycle of a single application/job**
Container	A bounded **resource unit (CPU + RAM)** used to run a task

1. ResourceManager (RM)

- Central authority
- Tracks resources across cluster nodes
- Schedules containers
- Has two main parts:
  - **Scheduler** (allocates resources)
  - **ApplicationManager** (manages job submissions)

2. NodeManager (NM)

- Runs on **each DataNode**
- Reports node health and resource usage to RM
- Launches and monitors containers as per AM instructions

3. ApplicationMaster (AM)

- Runs **per application** (job)
- Negotiates resources from RM
- Requests NMs to launch containers
- Manages execution and failure recovery of tasks

4. Container

- Basic unit of allocation
- Contains **one task** (map, reduce, or other)
- Includes a fixed amount of memory, CPU, and disk I/O

How YARN Works (Workflow)

1. **Client submits** a job to the **ResourceManager**
2. **RM allocates** a container to launch the **ApplicationMaster**
3. **AM requests containers** from RM for tasks
4. **Containers are launched** on various NodeManagers
5. **AM monitors** and manages the tasks
6. Once finished, AM **shuts down** and reports to RM

Key Features of YARN:

Feature	**Benefit**
Scalability	Can handle thousands of nodes
Multi-tenancy	Run different types of apps, not just MapReduce
Fault Tolerance	AM failure can restart
Better Resource Utilization	Resources are shared and optimized

Classic MapReduce vs YARN:

Feature	**Classic MapReduce**	**YARN**
Resource Manager	Fixed in Job Tracker	Centralized RM
Task Tracker	Executes tasks	NodeManager runs containers
Support for non-MapReduce	Only MapReduce	Any distributed application
Scalability	Limited	High
AM per job	No	Yes

Failures in YARN:

Failure Type	**Handled by**	**What Happens**
NodeManager Crash	ResourceManager	RM reschedules containers
ApplicationMaster Crash	ResourceManager	AM is restarted
Container Failure	ApplicationMaster	Retries task
ResourceManager Crash	Standby RM (HA)	Switches to backup RM (if HA enabled)

Question Bank:

1. Explain the MapReduce programming model. How does it facilitate large-scale data processing in distributed systems?
2. Discuss at least five real-world applications of MapReduce and explain why MapReduce is suitable for each use case.
3. Compare and contrast MapReduce with traditional parallel processing in terms of scalability, fault tolerance, and ease of development.
4. Illustrate the complete workflow of a MapReduce job with an example. Explain the roles of input splitting, mapping, shuffling, reducing, and output generation.

5. Describe the anatomy of a MapReduce job from submission to completion, including the components involved and their responsibilities.
6. Using the Word Count example, trace the flow of data through the Mapper, Shuffle/Sort, and Reducer phases. What is the significance of key grouping in this process?
9. What is MRUnit? Describe its components and explain how it helps in testing MapReduce jobs without deploying to a cluster.
10. List and explain the advantages and limitations of MRUnit in Hadoop development.
11. Explain the architecture of YARN. Describe the roles of ResourceManager, NodeManager, ApplicationMaster, and containers.
12. How does YARN overcome the limitations of classic MapReduce? Discuss improvements in scalability, fault tolerance, and multi-tenancy.
13. Describe the end-to-end workflow of job execution in YARN. How do AM and RM coordinate to manage task execution and resources?