Submitted by: Minahil Umar

Phase 5: Big Data & Distributed Computing

Big Data refers to extremely large and complex datasets that traditional data processing software can't handle efficiently. These datasets are not only huge in **volume** but also grow rapidly and come in various **types and formats**.

🔑 Key Characteristics of Big Data (The 5 V's):

1. **Volume** – The amount of data (e.g., terabytes, petabytes).
   - Example: Facebook generates over 4 petabytes of data every day.
2. **Velocity** – The speed at which data is generated and processed.
   - Example: Real-time data from sensors or stock market feeds.
3. **Variety** – Different types of data (structured, semi-structured, unstructured).
   - Example: Text, images, videos, emails, audio files, etc.
4. **Veracity** – The reliability and accuracy of the data.
   - Example: Social media posts can be misleading or noisy.
5. **Value** – The usefulness of the data once processed.
   - Example: Predicting customer behavior to boost sales.

How Big Data and Hadoop Are Linked

Big Data Need	Hadoop Solution
Need to store huge amounts of data	**HDFS** (Hadoop Distributed File System) stores big data across multiple machines
Need to process large data fast	**MapReduce** processes data in parallel

Need fault tolerance	HDFS replicates data across nodes
Need scalability	Hadoop can easily add more nodes
Need cost-effective solution	Runs on commodity (cheap) hardware

---

What is Hadoop?

Apache Hadoop is an **open-source framework** used to store and process **large-scale data (Big Data)** across multiple computers using simple programming models.

🧠 Why Do We Need Hadoop?

Traditional systems **fail** when:

- Data becomes **too large** to fit on a single machine
- Hardware **fails**
- You need **parallel processing**

✅ Hadoop solves all of this by:

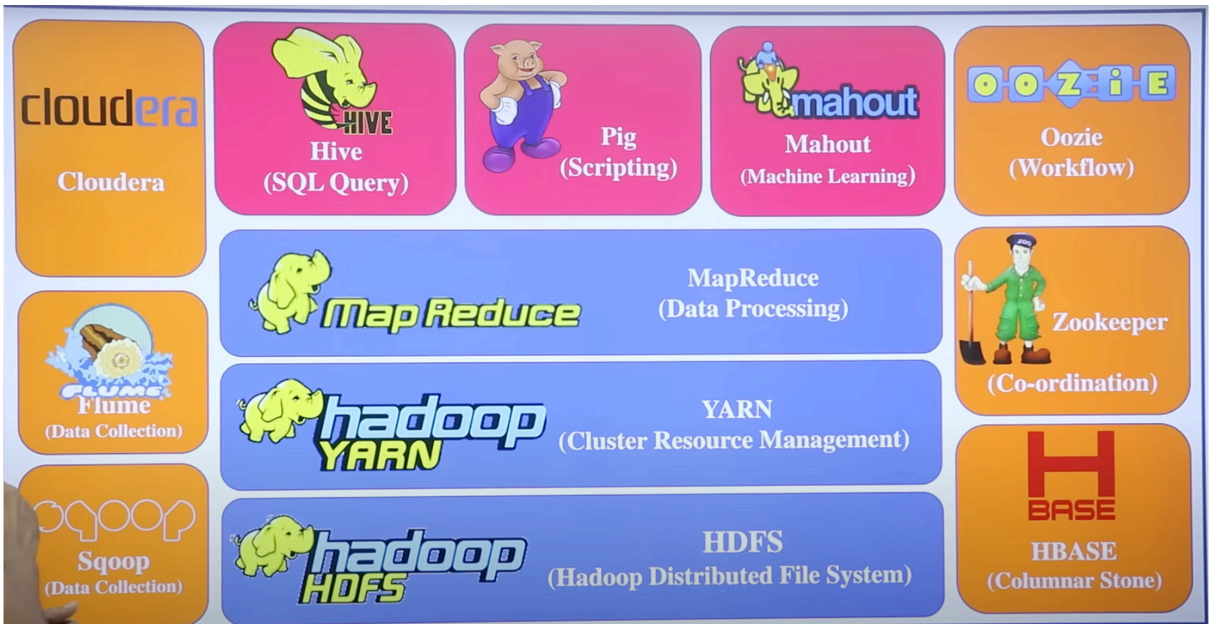
- **Distributing storage** using **HDFS**
- **Distributing computation** using **MapReduce**
- Running on **low-cost hardware** (commodity machines)

🔹 Key Features of Hadoop

Feature	Description
Distributed Storage	Splits and stores data across multiple nodes
Fault Tolerant	Replicates data to handle node failure
Scalable	Easily add more machines (horizontal scaling)

Open Source	Free and maintained by Apache
Cost-Effective	Runs on commodity hardware

🔸 Hadoop Architecture (Layer-wise)



🧱 1. Hadoop Distributed File System (HDFS)

Manages **storage** across the cluster.

Component	Role
NameNode	Master that stores metadata (file info, block locations)
DataNode	Slaves that store actual file blocks

⚙️ 2. MapReduce

Handles **data processing** by splitting it into:

- **Map**: Filters and sorts data

- **Reduce**: Aggregates the result

🎛️ 3. YARN (Yet Another Resource Negotiator)

Manages **cluster resources** and **job scheduling**.

Component	Role
ResourceManager	Global manager for resource allocation
NodeManager	Runs on each node and manages jobs and containers

---

🔹 Hadoop Ecosystem

Hadoop is **not just HDFS + MapReduce** — it's a complete ecosystem.

Tool	Purpose
HDFS	Distributed file storage
MapReduce	Distributed processing
YARN	Cluster resource manager
Hive	SQL on Hadoop
Pig	High-level data flow scripting
HBase	NoSQL database on HDFS
Sqoop	Data transfer between Hadoop and RDBMS
Flume	Stream logs into HDFS
Oozie	Job scheduler
Zookeeper	Coordination between services
Spark	In-memory data processing (faster alternative to MapReduce)

(limitation of hadoop is that in it data is stored in disk which makes data processing very slow and it also processes data in batches we have to wait to complete one process before starting another one, so there is a need to process data faster and in real-time thats when

Apache Spark is introduced) (in 2009 , researchers of university of california built it as a research project)

"Instead of reading and writing intermediate data to disk like Hadoop, Apache Spark does most of the work in memory (RAM)."

What is Apache Spark?

Apache Spark is an **open-source, distributed computing system** used to process **large-scale data** quickly.

It helps in:

- **Storing**, **processing**, and **analyzing** big data
- Running tasks in **parallel** on multiple machines (called a cluster)

🔹 Why Use Spark? (Compared to Hadoop MapReduce)

Feature	Hadoop MapReduce	Apache Spark
Speed	Slow (writes to disk)	Fast (in-memory processing)
Ease of Use	Complex (Java code)	Easy (Python, SQL, Scala)
Libraries	Limited	Rich (MLlib, SparkSQL, GraphX)
Real-time	❌ Not suitable	✅ Supports streaming

🔸 Spark's Core Components

✅ 1. RDD (Resilient Distributed Dataset)-->lower level abstraction

RDD (Resilient Distributed Dataset) automatically **splits your data across different machines (or nodes)** in a Spark cluster so it can be **processed in parallel.**

✅ 2. DataFrame

- A **distributed table** with rows and columns (like Excel or SQL)

- Built on top of RDDs
- Easier to use and **more optimized**
- You can run SQL-like queries on it

📌 Example:

python
CopyEdit
```
df = spark.read.csv("data.csv", header=True, inferSchema=True)
```

✅ 3. SparkSQL

- Allows you to run **SQL queries** on DataFrames
- Supports both **SQL syntax** and **DataFrame API**

📌 Example:

python
CopyEdit
```
df.createOrReplaceTempView("students")
spark.sql("SELECT * FROM students WHERE Age > 20").show()
```

✅ 4. SparkSession

- The **entry point** for Spark functionality (from Spark 2.0+)
- You use it to create DataFrames, run SQL, read files, etc.

📌 Example:

python
CopyEdit
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("MyApp").getOrCreate()
```

🔹 Spark Libraries

Library	Purpose
SparkSQL	Query structured data with SQL
Spark Streaming	Process real-time data streams
MLlib	Machine learning (clustering, classification)
GraphX	Graph processing (like Facebook's friend graph)

🔹 How Spark Works (Simplified)

1. You Write Code →sparkSession(entry point)

   ↓

2. Spark Builds DAG(directed Acyclic graph→execution plan) & Splits into Tasks

   ↓

3. Tasks Sent to Executors (Workers or worker nodes)

   ↓

4. Executors Process in RAM (Parallel)(**Loads a chunk of data into RAM** ,Applies transformations like `filter()`, `map()`, `groupBy()` etc.,**Stores intermediate results in memory** (not on disk like Hadoop)

   ↓

5. Final Result Returned or Saved

🔹 Spark Supports Multiple Languages

- Python (via **PySpark**) ✅
- Java
- Scala
- R

We use **PySpark** because it's easy and works well with Python's ecosystem.

🔸 What is the CAP Theorem?

The **CAP Theorem** is a fundamental concept in **distributed databases and systems**.
It says that **a distributed system can only guarantee 2 out of the 3 properties at any time**:

Property	Meaning
C – Consistency	Every read gets the **most recent** write
A – Availability	Every request gets a **response**, even if it's not the latest
P – Partition Tolerance	The system keeps working even if **some network links fail** (nodes can't talk to each other)

🔺 Visual Representation

```
           Consistency (C)
            /         \
           /           \
Availability (A)      Partition Tolerance (P)
```

In a real-world failure scenario (like network partition), you can **only pick 2**:

- CP
- AP
- CA ❌ (Not practical in distributed systems — because network partition is unavoidable)

✅ Let's Explain Each Term in Simple Words

🧩 1. Consistency

> "If I write something, and you read it right after, you should see exactly what I wrote."

📌 Example: You transfer Rs.1000 in your bank app, and it immediately reflects on all devices.

🧩 2. Availability

> "The system always responds, even if it shows old or partial data."

📌 Example: Google shows you an old version of a page if the server is busy — but it **never fails to respond**.

🧩 3. Partition Tolerance

> "Even if the system is split into parts due to a network issue, it keeps running."

📌 Example: Two data centers can't talk due to a fiber cut, but **both still serve users** independently.

🎯 CAP Theorem in Practice

Combination	What It Means	Real Examples
CP (Consistency + Partition Tolerance)	Always consistent, but may be **slow or unavailable** if nodes are down	HBase, MongoDB (with consistency)
AP (Availability + Partition Tolerance)	Always responds, but may return **stale data**	CouchDB, DynamoDB
CA	Always fresh and always up — ❌ **Not possible** if there's a partition	Only possible in **single-machine systems**, not in distributed ones

---

🔶 Apache Cassandra

✅ Concept

- A **highly scalable, distributed, NoSQL database**
- Uses **column-family model** (like a table but more flexible than RDBMS)
- Designed for **high write throughput** across **multiple nodes**
- Decentralized (no master node)

✅ Use Case

- Ideal for **time-series data**, such as:
    - IoT device logs
    - Sensor data
    - Event tracking systems (e.g., clickstreams)

🔷 Apache HBase

✅ Concept

- A **NoSQL column-family database built on top of HDFS**
- Supports **real-time** read/write on Big Data
- Integrates tightly with the **Hadoop ecosystem**
- Designed for **sparse**, large datasets (like Facebook feeds)

✅ Use Case

- Best suited for **user feeds**, **chat systems**, **analytics log**

✅ What is Apache Cassandra?

Apache Cassandra is a **highly scalable NoSQL database** designed to handle **huge amounts of data** across many servers.
It uses a **column-family model** and is great for **fast writing and reading**, especially for time-series data like **sensor logs**, **IoT**, or **event tracking**.

👉 Example:

It's used by apps like **Instagram** or **Netflix** to store billions of logs and user actions.

✅ What is Apache HBase?

> **Apache HBase** is a **NoSQL database built on top of Hadoop (HDFS)**.
> It stores data in **tables with column families** and supports **real-time read/write** access to **Big Data**.
> It's great for use cases like **social media feeds**, **chat apps**, and **user posts**.

👉 Example:

> It's like a **big spreadsheet** that can store millions of rows for each user — like storing your Facebook timeline.

What is Data Streaming?

Data streaming means **data is continuously flowing** from one place to another **in real-time**, just like a **live video** or **live cricket match**.It is when data is sent, received, and processed **right away**, continuously — just like a **live feed**

📦 Batch vs Streaming

Batch Processing	**Data Streaming**
Data comes in chunks	Data comes continuously
Processed after collecting	Processed instantly as it arrives
Higher latency	Low latency
Use case: Reports, backups	Use case: Fraud detection, live dashboards

---

✅ What is Apache Kafka?

Apache Kafka is a **tool (platform)** used to handle **real-time data streams**.

It works like a **high-speed message delivery system** between apps.

🔁 Real-Life Example:

Imagine a **food delivery app** like Foodpanda:

1. **Rider app** sends location every second 📍
2. **User app** receives rider updates instantly
3. Kafka is like the **middleman** that quickly passes messages from rider → user

---

Kafka lets you send and receive data in real-time between systems — just like **a live messaging bus**.

💡 Key Concepts:

Term	Meaning
Producer	Sends messages (e.g. sensor, website)
Consumer	Receives messages (e.g. dashboard, database)
Topic	Like a channel or category of messages
Broker	Kafka server that stores and sends messages
Cluster	A group of Kafka servers (for big systems)

🔧 How Kafka Works:

1. A **producer** sends messages to **Kafka**.
2. Kafka stores the messages in a **topic**.
3. A **consumer** reads those messages in real-time.
4. The cycle continues — super fast and scalable.

What is Spark Streaming?

Spark Streaming is a feature of **Apache Spark** that lets you process **real-time data** — as it's being generated.

How it Works :

1. Data comes from a **streaming source** (e.g. Kafka, socket, sensor).
2. Spark receives data in **mini-batches** (every 1 or 2 seconds).
3. You apply operations like `filter()`, `map()`, `groupBy()`.
4. Spark **outputs the results** continuously (to console, file, database, etc.)

🔁 Real-Life Example:

- Imagine you're watching a **live cricket match score** on your phone.
- The score updates **every few seconds**.
- That continuous data flow is a **stream**.

Spark Streaming helps collect that stream and process it **right away** (e.g., calculate average runs every 5 balls).
