BUSINESS INTELLIGENCE AND DATA ANALYTICS
– WE PREDICT THE FUTURE –

University of Petra

307401
Big Data and Data Warehouses

Introduction to Hadoop

Types of Processing in Computing

Processing Type	Description	Example
Sequential Processing	Tasks are executed one after another in a linear sequence.	Reading and executing lines of code in a single-threaded program.
Multi-Tasking	Multiple tasks share the same CPU by switching between them rapidly.	Running multiple applications (browser, music player, and document editor) on a computer.
Parallel Processing	Multiple processors or cores execute different parts of a task simultaneously.	Image processing using multiple CPU cores.
Distributed Processing	Tasks are divided across multiple computers working together over a network.	Cloud computing and blockchain networks.

Speedup Concept

Formula:

$$Speedup(S) = \frac{T_1}{T_p}$$

Where:

- T_1 = Execution time with a single processor.
- T_p = Execution time with p processors.

Key Points:

- **Definition:** Speedup measures the performance gain achieved by using multiple processors compared to a single one.
- **Linear Speedup:** Ideal scenario where $S = p$ (e.g., 4 processors give 4× speedup).
- **Superlinear Speedup:** Occasionally, $S > p$ due to better memory utilization or caching effects.
- **Sublinear Speedup:** Speedup is less than the number of processors due to communication and synchronization overhead.
- **Amdahl's Law:** The speedup is limited by the sequential portion of the program:

$$S = \frac{1}{(1-P) + \frac{P}{p}}$$

- **Gustafson's Law:** Speedup increases with problem size, making large-scale distributed computing more efficient:

$$S = p - (1-P) \times (p-1)$$

Introduction to Hadoop

- Hadoop is an open-source framework overseen by Apache Software Foundation which is written in Java for storing and processing of huge datasets with the cluster of commodity hardware.
- There are mainly two problems with the big data that Hadoop addresses:
  1. The need to store huge amount of data.
  2. The need to process that stored data.
- The traditional approach like RDBMS is not sufficient due to the heterogeneity of the data.
- Hadoop serve as the solution to the problem of big data i.e. storing and processing big data.

History of Hadoop

Doug Cutting

Mike Cafarella

- Early Beginnings (2002-2003):
  - Nutch Project: Open-source search engine project by Doug Cutting and Mike Cafarella.
  - Challenges in managing and processing large volumes of web data.
- Google's Influence (2003-2004):
  - Google Papers: Introduction of Google's GFS (Google File System) and MapReduce.
  - Inspiration for Hadoop's core concepts.
- Birth of Hadoop (2005-2006):
  - Creation of Hadoop as a separate project within Apache Nutch.
  - Named after Doug Cutting's son's toy elephant.
- Yahoo! began investing in and contributing to Hadoop's development.
- Apache Hadoop (2008):
  - Hadoop became an Apache top-level project.
  - Core components: HDFS (Hadoop Distributed File System) and MapReduce.
  - Rapid growth in community contributions and adoption.
- Ecosystem Expansion (2009-2011):
  - New projects: Hive, Pig, HBase, Zookeeper.
  - Formation of companies like Cloudera for commercial support.
- Industry Adoption (2012-2014):
  - Mainstream use by major companies: Facebook, LinkedIn, eBay.
  - Introduction of Hadoop 2.0 with YARN (Yet Another Resource Negotiator).
  - Hadoop Summit: Annual conference for advancements and use cases.
- Modern Era (2015-Present):
  - Widespread enterprise adoption for various Big Data applications.
  - Integration with cloud platforms: AWS EMR, Google Cloud Dataproc.
  - Ongoing development and enhancements to the Hadoop ecosystem.
- Impact of Hadoop:
  - Pioneered the Big Data revolution.
  - Enabled scalable, distributed data processing.
  - Fostered a strong open-source community and ecosystem.

Hadoop Echo System

- Hadoop ecosystem is a suite of tools and frameworks

that enables Hadoop to handle large amounts of data.

Basically, Hadoop has the following three components:

1. HDFS (Hadoop Distributed File System): A distributed file system that provides high-throughput access to application data.
2. MapReduce: A framework for parallel processing of large data sets.
3. YARN, Resource Manager.

Other ecosystem components: Hive, Pig, HBase, Storm, Spark, Kafka, Flume, Sqoop, Oozie, Zookeeper…etc. These components **work together** to provide a comprehensive solution for managing and processing large data sets in a distributed computing environment.

HDFS – Hadoop Distributed File System

What is HDFS?

- **Hadoop Distributed File System (HDFS)** is the storage layer in Hadoop.
- It enables **distributed storage** and **fault tolerance** by replicating data across multiple nodes.

- **Key Components of HDFS**

1. NameNode

- Acts as the **master node**.
- Manages **metadata** (file locations, permissions, and structure).
- Does not store actual data.

2. DataNodes

- Store actual data in **blocks**.
- Handle read/write requests from clients.
- Perform **data replication** to ensure fault tolerance.

How HDFS Works?

1. **Client requests data** from HDFS.
2. **NameNode retrieves metadata** and provides block locations.
3. **Client reads/writes data blocks** directly from/to the DataNodes.
4. **Data replication** occurs across multiple DataNodes for redundancy.

YARN, Yet Another Resource Negotiator

What is YARN?

- YARN (Yet Another Resource Negotiator) is the resource management layer in Hadoop.
- It efficiently allocates resources and schedules tasks for processing big data jobs.

Key Components of YARN

1. Resource Manager (RM) - Central component managing resource allocation.

- It has two main parts:
  - **Application Manager** – Manages job submission and execution.
  - **Scheduler** – Allocates resources to tasks based on policies.

2. Node Managers (NM)

- Runs on each DataNode to execute tasks.
- Manages **Containers**, which are allocated resources for executing tasks.

3. Application Master (AM)

- Runs within a container and coordinates job execution.
- Communicates with the Resource Manager and Node Managers.

How YARN Works?

1. **Clients submit jobs** to the Resource Manager.
2. **Scheduler assigns resources** based on job size and priority.
3. **Application Master is launched** in a container to oversee execution.
4. **Node Managers execute tasks** in their respective containers.
5. **Results are sent back to clients** after completion.

MapReduce

- Map Reduce is a programming model for processing large data
- sets with a parallel, distributed algorithm on a cluster.
- Key Components:
  - Mapper: Takes a set of data and converts it into another set of data, where individual elements are broken down into key/value pairs.
  - Reducer: Takes the output from a mapper as an input and combines those data tuples into a smaller set of tuples.
- Workflow:
  - Input Splits: The input file is split into multiple pieces.
  - Map Phase: Processes the input splits and produces intermediate key-value pairs.
  - Shuffle and Sort Phase: Organizes the intermediate data by key.
  - Reduce Phase: Aggregates the sorted key-value pairs and generates the final output.

Real-World Use Case: Log Analysis for Ad Click Fraud Detection

Company Example: Facebook
Objective: **Detect fraudulent ad clicks** by analyzing massive volumes of server logs.
Data Source: Web server logs from Facebook's ad delivery infrastructure.

Data Structure (Sample Log Entry):


```
{  "timestamp": "2025-03-20T14:33:21Z",
  "user_id": "u98374",
  "ad_id": "ad4562",
  "ip_address": "192.168.1.5",
  "click": true,
  "device_type": "mobile",
  "location": "New York, USA"}
```


Processing with MapReduce:
Map Phase:
- Extract relevant fields: user_id, ip_address, ad_id, timestamp.
- Emit key-value pairs like (ip_address, 1) to count clicks per IP.

Reduce Phase:
- Aggregate counts of clicks per IP address.
- Identify IPs with suspiciously high click counts in a short time frame.

Outcome:
- Detects click patterns that suggest automated scripts or bot activity.
- Enables the system to block fraudulent IPs and protect ad revenue.

Apache Hive

What is Hive?

- Hive is a data warehouse system built on top of Hadoop.
- It allows querying and managing large datasets using a SQL-like language called HiveQL.
- Created by Facebook to simplify complex MapReduce programming for analysts.

Why Hive?

- Writing raw MapReduce code in Java is complex and time-consuming.
- Hive allows users to write declarative queries in HiveQL.
- Hive compiles these queries into efficient MapReduce tasks.

Hive Architecture

- Metastore: Stores metadata about Hive tables (schemas, locations, etc.).
- Driver: Manages the lifecycle of a HiveQL query and its execution plan.
- Compiler: Translates HiveQL into MapReduce jobs to run on Hadoop.

Real-World Use Case: Marketing Campaign Analysis

Company Example: NETFLIX

Objective: **Analyze customer behavior** to evaluate marketing campaign effectiveness.

Data Source: User activity logs stored in HDFS, including views, clicks, watch duration, and subscription status.

Data Structure (Hive Table Schema):

```
CREATE TABLE user_activity (
  user_id STRING,
  timestamp TIMESTAMP,
  action STRING,        -- e.g., "play", "pause", "subscribe",
"click_ad"
  content_id STRING,
  device_type STRING,
  campaign_id STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Processing with Hive (HiveQL):
Perform aggregations:

- Total views per campaign
- Conversion rates from ad click to subscription
- Device-specific campaign performance

Sample Query:
SELECT campaign_id, COUNT(*) AS views, SUM(CASE WHEN action = 'subscribe' THEN 1 ELSE 0 END) AS subscriptions FROM user_activity WHERE action IN ('click_ad', 'subscribe’) GROUP BY campaign_id;

Outcome:

- Identifies high-performing campaigns and channels.
- Supports data-driven decisions for targeting and ad spend allocation.
- Let me know if you'd like this turned into a presentation slide or want a Hive use case from retail, telecom, or IoT.

Apache PIG – High Level SQL Like Language on top of Hadoop

- Run on Hadoop and translates Pig Latin scripts into MapReduce jobs.
- Created by YAHOO! to solve MapReduce challenges
- Pig Latin is a high-level data processing language that simplifies writing complex MapReduce code directly, this allows users to work with big data using simpler syntax compared to Map Reduce.
- Used for data transformations and analysis tasks over large datasets in Hadoop.
- **Architecture**:
  - **Pig Latin Scripts**: Define data transformation tasks.
  - **Pig Engine**: Executes these scripts on Hadoop.

https://www.analyticsvidhya.com/blog/2022/06/getting-started-with-apache-pig/

Hive vs Pig

Real-World Use Case: Web Log Processing for User Behavior Analysis

Company Example: *LinkedIn (used Pig before transitioning to Spark)*

Objective: Analyze web server logs to understand user navigation paths and session behavior.

Data Source:

- Daily web server log files stored in HDFS.
- Data Structure (Sample Log Fields):
- timestamp, user_id, session_id, page_url, referrer_url, device_type

Example Data:

2025-03-22T12:01:45Z, u12345, s7890, /jobs, /home, mobile

Processing with Apache Pig:

Load and Transform Logs:

```
logs = LOAD '/logs/2025/03/22'
    USING PigStorage(',')
    AS (timestamp:chararray, user_id:chararray,
session_id:chararray,
        page_url:chararray, referrer_url:chararray,
device_type:chararray);
```

Group and Analyze Sessions:

```
grouped = GROUP logs BY session_id;
session_counts = FOREACH grouped GENERATE group AS
session_id,
            COUNT(logs) AS page_views;
```

Filter High-Engagement Sessions:

```
engaged_sessions = FILTER session_counts BY page_views > 5;
```

Outcome:

- Identifies highly engaged users and frequent navigation paths.
- Insights used for UX improvements and targeted content placement.

Apache HBase - Distributed Database on top of Hadoop

What is HBase?

- HBase is a distributed, column-oriented NoSQL database built on top of Hadoop HDFS.
- Designed for sparse, semi-structured, and unstructured data.
- Provides real-time read/write access to large datasets.
- Inspired by Google's Bigtable.

Apache HBase

Apache HBase Architecture

Apache HBase Architecture

- **HMaster**: Manages the cluster.
- **RegionServer**: Handles read/write requests for regions of the database.
- **ZooKeeper**: Manages distributed coordination.

https://www.scaler.com/topics/hadoop/hbase-architecture/

Insert Data in HBase

- In HBase, we create tables and define column families.

create 'employees', 'personal', 'job'	**Create a table named employees with two families (personal and job)**

- Then we insert values

put 'employees', 'emp1', 'personal:name', 'Alice'	emp1 has only name in personal
put 'employees', 'emp2', **'personal:phone'**, '123456789'	emp2 has only phone in personal
put 'employees', 'emp3', **'job:salary'**, '60000'	emp3 has only salary in job, no personal column family

- Notice that must comply with column families when we insert data into the table.
- But we can use different column names within the column family e.g.

☑ **Schema-less?** Yes, **for columns** (qualifiers) – flexible and dynamic.

Schema-fixed? Yes, **for column families** – must be defined when the table is created.

Key Concepts

- Row Key: Unique identifier for each row (like a primary key).
- Column Family: Logical group of columns stored together on disk.
- Column Qualifier: Actual field name inside a column family.
- Timestamp: Each cell value is versioned by time.
- Cell: Combination of Row Key + Column Family + Qualifier + Timestamp → Value.

Each entry in HBase is stored as a **triplet (technically a quintuple):**

pgsql Copy Edit

```
(Row Key, Column Family, Column Qualifier, Timestamp) => Value
```

This means **each field you insert becomes a cell**, and there's no fixed schema for columns like in a relational table.

Example: Employee Data Internally in HBase

Let's say you added this:

bash Copy Edit

```
put 'employees', 'emp1', 'info:name', 'Alice Smith'
put 'employees', 'emp1', 'info:salary', '72000'
```

Internally, it looks more like:

Row Key	Column Family	Column	Timestamp	Value
emp1	info	name	1679553600000	Alice Smith
emp1	info	salary	1679553600000	72000

> HBase auto-generates timestamps unless you set them manually.

Real-World Use Case: Messaging Platform – Real-Time Message Storage & Retrieval

Company Example: *WhatsApp*

Objective: Store and retrieve billions of real-time chat messages with low latency and high throughput.

Data Source: User chat messages from mobile devices, synced to a backend service.

Data Structure (HBase Table Design):
Table: messages
Row Key: user_id + timestamp
Column Families:
- **meta: sender_id, receiver_id, message_type**
- **content: message_type, attachments**
- **status: delivered, read, timestamp**

Sample Record (in HBase):
Row Key: user123|2025-03-22T14:33:21Z
Column Family: meta
- sender_id: user123
- receiver_id: user456
Column Family: content
- message_text: individual
Column Family: status
- delivered: true
- read: false

Processing with HBase:
- Messages are written and read with low latency.
- Efficient for real-time features: chat history retrieval, unread message counts.
- Scales horizontally to billions of rows (messages) per day.

Outcome:
- Ensures seamless chat experience across devices.
- Supports real-time sync, delivery status tracking, and storage durability.

HBase vs Relational DB

Feature	HBase	Relational DB
Schema	Flexible (schema-less columns)	Fixed schema
Row Structure	Rows can have different columns	Rows have same set of columns
Access Pattern	Real-time random read/write	Typically, transactional
Query Language	API (Java, REST, Thrift)	SQL
Storage Model	Column-oriented, sparse	Row-oriented
Best Use Case	Time-series, logs, IoT, real-time	Structured, normalized data

Apache Spark

Definition:

- Apache Spark is an open-source distributed data processing engine optimized for large-scale data processing.
- In-Memory Processing: Spark keeps data in memory between transformations, speeding up processing times significantly compared to disk-based MapReduce.
- Spark can handle both batch and real-time processing.
- Spark offers APIs for various languages (Java, Scala, Python, R).

Core Components:

1. **Spark Core:** The foundation for Spark's API, handling basic data processing and distributing tasks across the cluster.
2. **Spark SQL:** Enables SQL-like queries on data stored in distributed datasets (DataFrames and Datasets).
3. **Spark Streaming:** Allows processing of real-time data streams, enabling near real-time analytics.
4. **MLlib:** Spark's machine learning library, offering tools for scalable machine learning.
5. **GraphX:** A graph processing library for large-scale graph computations.

MapReduce vs. Spark

- MapReduce is a first-generation distributed data processing system. It processes data that is parallelizable and performs computation on a distributed, horizontally scalable infrastructure.
- As a distributed system, MapReduce applies a particular linear data flow structure, which consists of MapReduce programs that read input from a disk, map a function across the data, reduce the resulting data of the map, and store it on the disk again.

MapReduce

- Spark is a third-generation data processing framework that enhances MapReduce's performance by processing data in memory instead of writing them to the disk in each step.
- Spark distributes the in-memory data in a logically partitioned way on many machines and calls these Resilient Distributed Datasets (RDDs), which are then used as an abstraction layer to manage the logically distributed data.

Apache Spark

https://www.redpanda.com/blog/data-stream-processing-spark-flink-ksqldb

Apache Kafka – Messaging System

- Apache Kafka is a distributed event streaming platform designed to handle high-throughput, real-time data feeds.
- Originally developed by LinkedIn, Kafka has become the backbone for data pipelines and real-time analytics in many large-scale systems.
- It serves as a message broker that allows various applications to produce and consume streams of data reliably, efficiently, with low latency and at scale.
- **Architecture**:
  - **Producer:** An application that sends data to Kafka topics.
  - **Consumer:** An application that reads data from Kafka topics.
  - **Broker:** A Kafka server that stores data and serves it to consumers.
  - Kafka clusters consist of multiple brokers.
  - **Topic:** A category or feed name to which records are published. Each topic can have multiple partitions, helping Kafka to parallelize data across the cluster.

Kafka in E-Commerce Order Processing

- **Scenario:**
- An online store uses **Kafka** to manage orders, inventory, payments, and shipping in real time.
- **Process Flow:**

1. **A customer places an order** on the storefront (website/app).
2. The order details are sent to the **Kafka orders topic**.
3. The **Order Processing Service** consumes the order, checks inventory, and verifies payment.
4. If items are in stock, the order is confirmed, and a message is sent to the **Inventory & Payment topics**.
5. Once the payment is confirmed, the **Shipping Service** picks up the order details from Kafka and generates a shipment.
6. The **Customer Notification Service** listens for updates and sends notifications.

Producer (Sends Data)	Kafka Topic	Description of Producer Output	Consumer (Receives & Processes Data)	Usage of Received Data
Online Storefront	orders	Sends new customer orders with item details and total amount	Order Processing Service	Validates order, checks stock, and verifies payment
Order Processing Service	inventory	Sends stock update requests after order validation	Inventory Management Service	Updates stock levels and confirms availability
Warehouse System	inventory	Sends real-time stock updates when items are received or dispatched	Order Processing Service	Adjusts order fulfillment based on inventory
Payment Gateway	payments	Sends payment confirmation or failure status	Order Processing Service	Marks order as paid or cancels if payment fails
Order Processing Service	shipping	Sends confirmed orders ready for shipping	Shipping Service	Creates a shipment and assigns a tracking number
Shipping Provider	shipping	Sends delivery updates such as shipment dispatched or delivered	Customer Notification Service	Notifies the customer about shipping progress
Order Processing & Shipping	notifications	Sends status updates like "Order Shipped" or "Payment Failed"	Customer Notification Service	Triggers email/SMS notifications to customers

Kafka Message Example

Order Message Sent to Kafka (orders topic):

```
{
  "order_id": "ord12345",
  "timestamp": "2025-03-25T10:10:30Z",
  "customer_id": "cust456",
  "items": [
    {"product_id": "prod678", "quantity": 2},
    {"product_id": "prod789", "quantity": 1}
  ],
  "total_amount": 129.99,
  "payment_status": "paid",
  "shipping_address": "123 Main St, Anytown, USA"
}
```

The Order Processing Service reads this message, checks stock levels, verifies payment, and updates Kafka with order status.

Kafka Data Structure (Sample Message):

Order Message (from storefront):

```
{
  "order_id": "ord12345",
  "timestamp": "2025-03-25T10:10:30Z",
  "customer_id": "cust456",
  "items": [
    {"product_id": "prod678", "quantity": 2},
    {"product_id": "prod789", "quantity": 1}
  ],
  "total_amount": 129.99,
  "payment_status": "paid",
  "shipping_address": "123 Main St, Anytown, USA"
}
```

Apache Oozie

- Apache Oozie is a workflow scheduler system designed to manage and coordinate Hadoop jobs in a sequence or directed acyclic graph (DAG) for end-to-end workflow management.
- Oozie is particularly useful for automating complex data workflows, managing dependencies, and orchestrating multiple jobs across the Hadoop ecosystem (such as MapReduce, Hive, Pig, and Sqoop jobs).
- It supports both time-based (frequency) and data-based triggers,
- making it versatile for different types of big data applications.
- **Main Functions:**
  - **Workflow Management**: Defines and manages job sequences.
  - **Coordination**: Manages job dependencies and schedules based on time and data availability.
  - **Retry Mechanism**: Automatically retries failed jobs.
- **Use Cases**: ETL workflows, batch processing, and managing complex job dependencies.

Apache Zookeeper

- Apache ZooKeeper is a centralized service designed for coordinating distributed applications.
- ZooKeeper plays a critical role in the Hadoop ecosystem and other distributed systems by helping to maintain the distributed state and providing a reliable way for services to work together.
- **Key Features of Apache ZooKeeper**

1. **Centralized Configuration Management**: ZooKeeper stores configuration data in a centralized manner, allowing distributed applications to access and modify configuration settings easily.
2. **Naming Service**: It provides a naming registry for distributed services, allowing them to find each other and manage resources dynamically.
3. **Synchronization**: ZooKeeper helps synchronize distributed processes, ensuring they work in coordination and without conflicts.
4. **Group Management**: Supports the creation and management of groups, enabling services to join and leave groups dynamically.
5. **Watchers**: ZooKeeper can set watchers on nodes to get notified of changes, allowing applications to react to changes in state.

Apache Ambari

Apache Ambari

- Apache Ambari is a web-based tool designed for provisioning, managing, and monitoring Apache Hadoop clusters.
- It provides a user-friendly interface that simplifies the management of complex Hadoop environments, making it easier for administrators and operators to deploy and manage various components of the Hadoop ecosystem.
- **Key Features of Apache Ambari**
  - **Cluster Provisioning**: Ambari simplifies the installation and configuration of Hadoop clusters, allowing users to easily add or remove nodes and services through a guided installation wizard.
  - **Service Management**: Users can start, stop, and restart various Hadoop services (such as HDFS, YARN, Hive, HBase, etc.) through the Ambari interface.
  - **Monitoring and Alerts**: Ambari provides dashboards for real-time monitoring of cluster health, service metrics, and resource usage. It also supports setting up alerts for various cluster events or metrics.
  - **Configuration Management**: It allows for easy management of configuration settings for Hadoop services, including the ability to apply configurations in a centralized manner.
  - **User Management**: Administrators can manage users and their permissions, ensuring secure access to cluster resources and configurations.
  - **Integration with Ambari Metrics and Alerts**: Supports the collection and visualization of metrics from different Hadoop components, allowing for comprehensive monitoring and alerting.

Hadoop Components Summary

MainHadoop Components

Category / Component	Description
Storage Components	
HDFS	Distributed file system for scalable and fault-tolerant data storage.
HBase	NoSQL database for real-time read/write access to large datasets.
Phoenix	SQL layer on top of HBase for low-latency OLTP and analytics.
Batch Processing	
MapReduce	Traditional batch processing model for parallel computation on large data.
Tez	DAG-based execution engine; faster alternative to MapReduce.
Hive	SQL-like interface for querying large datasets, often runs on Tez or Spark.
Pig	Dataflow scripting platform using Pig Latin for batch data processing.
Impala	MPP (Massively Parallel Processing) SQL query engine for real-time, interactive querying on Hadoop data.
Real-Time / Stream Processing	
Spark Streaming	Micro-batch stream processing engine within Apache Spark.
Flink	True stream processing engine with high throughput and low latency.
Storm	Distributed real-time computation system for event-driven processing.
Data Ingestion & Integration	
Sqoop	Imports/exports structured data between Hadoop and relational databases.
Flume	Collects and transports large volumes of log or event data.

Main Hadoop Components

Category / Component	Description
Security & Governance	
Ranger	Centralized security for authorization, policies, and auditing.
Knox	Gateway that provides perimeter security and REST API access to Hadoop.
Atlas	Metadata management and data governance framework.
Machine Learning & Search	
Mahout	Scalable machine learning library for classification, clustering, etc.
Solr	Enterprise search platform; supports full-text search on Hadoop data.
Cluster Coordination	
YARN	Resource management layer for job scheduling and cluster resource utilization.
Zookeeper	Coordination service for configuration, synchronization, and leader election.
Serialization & File Formats	
Avro	Row-based data serialization framework with schema support.
Parquet	Columnar storage format optimized for performance and compression.
ORC	Columnar storage format optimized for Hive; enables efficient querying.
UI, Workflow & Dev Tools	
Oozie	Workflow scheduler to orchestrate complex Hadoop jobs.
Ambari	Web UI for provisioning, monitoring, and managing Hadoop clusters.
Hue	Web-based user interface for querying and interacting with Hadoop tools.
Zeppelin	Interactive data analytics notebook that supports Spark, SQL, and more.

Final Notes

Distributed Processing vs. Massive Parallel Processing (MPP)

Distributed Processing

- **General Concept**: Any system where multiple computers (or nodes) work together to process data.
- **Used in**: Hadoop (MapReduce), Spark, Flink, Storm, etc.
- **How it works**: The workload is split into chunks and processed in parallel across multiple machines, each with its own memory and CPU.
- **Flexible**: Can handle a wide range of tasks, not just databases—can be for file processing, stream processing, ML, etc.

MPP (Massively Parallel Processing)

- **Specific Type of Distributed Processing**: Tailored for **databases** and **query execution**.
- **Used in**: Impala, Amazon Redshift, Google BigQuery, Greenplum, etc.
- **How it works**: Tables are split into partitions and distributed across nodes. Each node processes its chunk **independently and simultaneously**, then results are combined.
- **Optimized for**: High-speed SQL queries on big data. Think of it like a turbocharged SQL engine.

Key Differences (Distributed Processing vs. MPP)

Feature	MPP	Distributed Processing
Primary Use	SQL query engines	General data processing
Optimization	Analytical queries (OLAP)	Broader workloads (ETL, ML, streaming)
Examples	Impala, Redshift	Spark, MapReduce, Flink
Independence	Nodes work mostly independently	Often requires coordination or intermediate shuffling

TL;DR:
All MPP systems are distributed, but not all distributed systems are MPP.
MPP is a specialized, high-performance **subset** of distributed processing focused on SQL analytics.