UNIT IV

Frameworks and Applications: Frameworks: Applications on Big Data Using Pig and Hive, Data processing operators in Pig, Hive services, HiveQL, Querying Data in Hive, fundamentals of HBase and ZooKeeper.

→ Apache Pig and Apache Hive are two popular frameworks for processing large-scale data in the Hadoop ecosystem.

→ They provide abstraction layers over Hadoop's MapReduce, making it easier to process and analyze big data without having to write complex Java-based MapReduce programs.

1. Apache Pig

Overview:

- Pig is a high-level scripting platform that runs on Hadoop.
- It is designed for data transformation, ETL (Extract, Transform, Load) operations, and iterative data processing.
- It uses **Pig Latin**, a procedural language that abstracts MapReduce.

Key Features:

- Handles **structured, semi-structured, and unstructured data**.
- Automatically optimizes execution plans.
- Supports **UDFs (User-Defined Functions)** in Java, Python, and other languages.
- Works efficiently with large datasets.

Use Cases:

- Data preprocessing and cleaning before storage in a data warehouse.
- ETL operations for analytics pipelines.
- Batch processing for log analysis and sentiment analysis.

2. Apache Hive

Overview:

- Hive is a **data warehouse system** built on top of Hadoop.

- It allows users to query large datasets using **HiveQL (SQL-like syntax)** instead of writing MapReduce programs.
- Designed for batch processing and analytics.

Key Features:

- Provides **schema-on-read** (data does not need a predefined schema).
- Optimized for **aggregations and reporting**.
- Supports **ACID transactions** and partitioning for better performance.
- Integrates with **BI tools** like Tableau, Spark, and Tez.

Use Cases:

- Business Intelligence and reporting.
- Ad hoc querying of massive datasets.
- Data warehousing solutions for structured data.

Comparison: Pig vs. Hive

Feature	Apache Pig	Apache Hive
Language	Pig Latin (Procedural)	HiveQL (Declarative SQL-like)
Best For	Data transformation (ETL, preprocessing)	Querying and analysis
Learning Curve	Easier for programmers	Easier for SQL users
Performance	Good for raw data processing	Optimized for querying large datasets
Integration	Works with Java, Python, UDFs	Works with BI tools, JDBC/ODBC

Data processing operators in Pig

Apache Pig provides a rich set of data processing operators in Pig Latin to manipulate and analyze large datasets efficiently. These operators can be categorized into different groups based on their functionality:

1. Load and Store Operators

These operators are used to read and write data in Pig.

LOAD – Loads data from HDFS or other storage into a Pig relation.

data = LOAD 'hdfs://path/to/file' USING PigStorage(',') AS (id:int, name:chararray, age:int);

`STORE` – Stores the output data into HDFS or other storage.

```
pig

STORE data INTO 'hdfs://path/to/output' USING PigStorage(',');
```

2. Relational Operators

Used to **manipulate and transform** data in Pig.

a. Filtering and Transformation

- `FILTER` – Selects rows based on a condition.

```pig
adults = FILTER data BY age > 18;
```

- `FOREACH ... GENERATE` – Transforms data by selecting and computing new fields.

```pig
names = FOREACH data GENERATE name, age * 2 AS double_age;
```

- `FLATTEN` – Converts nested data structures into a simpler format.

```pig
flat_data = FOREACH nested_data GENERATE FLATTEN(some_column);
```

b. Grouping and Aggregation

- `GROUP` – Groups data by a specified field.

```pig
grouped_data = GROUP data BY age;
```

- `COUNT`, `SUM`, `AVG`, `MAX`, `MIN` – Aggregate functions used with `GROUP`.

```pig
age_count = FOREACH grouped_data GENERATE group AS age, COUNT(data);
```

c. Sorting and Ordering

- `ORDER BY` – Sorts data based on a field.

```pig
sorted_data = ORDER data BY age DESC;
```

- `LIMIT` – Limits the number of rows.

```pig
limited_data = LIMIT data 10;
```

3. Join and Set Operators

Used to **combine** data from multiple datasets.

- `JOIN` – Performs an inner join between two datasets.

```pig
joined_data = JOIN students BY id, scores BY student_id;
```

- `CROSS` – Returns the Cartesian product of two relations.

```pig
cross_data = CROSS students, subjects;
```

- `UNION` – Combines two datasets with the same schema.

```pig
combined_data = UNION data1, data2;
```

- `DISTINCT` – Removes duplicate records.

```pig
unique_data = DISTINCT data;
```

4. Debugging Operators

Useful for **debugging and inspecting data**.

- `DUMP` – Prints the data to the console.

```pig
DUMP data;
```

- `DESCRIBE` – Displays schema of a relation.

```pig
DESCRIBE data;
```

- `EXPLAIN` – Shows the logical and physical execution plan.

```pig
EXPLAIN data;
```

- `ILLUSTRATE` – Shows sample data transformations.

```
pig

ILLUSTRATE data;
```

---

5. Special Operators

- `SPLIT` – Divides a dataset into multiple datasets based on conditions.

```
pig

SPLIT data INTO young IF age < 18, adults IF age >= 18;
```

- `RANK` – Assigns a rank to each row based on ordering.

```
pig

ranked_data = RANK data BY age;
```

Apache Hive: Services, HiveQL, and Querying Data

1. Hive Services

Apache Hive provides multiple services that work together to process queries and manage metadata. The key Hive services are:

a. Hive Metastore (HMS)

- Stores **metadata** about Hive tables, schemas, partitions, and data locations.
- Can use **MySQL, PostgreSQL, or Derby** as the underlying database.
- Metadata is essential for query optimization and execution.

b. Hive Driver

- Acts as the **interface between users and Hive**.

- Receives **queries**, compiles them, optimizes execution plans, and submits jobs to Hadoop.

c. Hive Compiler

- Converts **HiveQL queries into MapReduce jobs**.
- Optimizes execution by breaking down complex queries into efficient tasks.

d. Hive Execution Engine

- Executes the compiled MapReduce jobs or **Tez/Spark queries**.
- Sends jobs to Hadoop for distributed processing.

e. HiveServer2

- Enables **multi-client connections** via JDBC, ODBC, and Thrift.
- Supports external applications like **BI tools (Tableau, Power BI, etc.)**.

f. Web UI (Beeline & Hue)

- **Beeline**: A command-line client for running Hive queries.
- **Hue**: A web-based UI for executing queries and visualizing results.

2. HiveQL (Hive Query Language)

HiveQL is an SQL-like language used in Hive for querying and managing data. It includes:

a. Data Definition Language (DDL)

Used to create and manage databases, tables, and partitions.

Create Database

```sql
CREATE DATABASE mydb;
```

Create Table

```sql
CREATE TABLE employees (
    id INT,
    name STRING,
    age INT,
    department STRING
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  STORED AS TEXTFILE;
```

- Drop Table

```sql
DROP TABLE employees;
```

b. Data Manipulation Language (DML)

Used for inserting, updating, and deleting data.

- Insert Data

```sql
INSERT INTO TABLE employees VALUES (1, 'John', 30, 'HR');
```

- Load Data from HDFS

```sql
LOAD DATA INPATH '/hdfs/path/to/file.csv' INTO TABLE employees;
```

- Delete Data (Only for ACID tables)

```sql
DELETE FROM employees WHERE age < 25;
```

c. Query Language

Used to retrieve and analyze data.

- **Select Query**

```sql
SELECT name, age FROM employees WHERE department = 'IT';
```

- **Group By & Aggregation**

```sql
SELECT department, COUNT(*) AS total FROM employees GROUP BY department;
```

- **Join Tables**

```sql
SELECT e.id, e.name, d.department_name
FROM employees e
JOIN departments d ON e.department = d.id;
```

3. Querying Data in Hive

a. Partitioning (Improves Query Performance)

Partitions divide tables into smaller segments based on a column.

- **Create Partitioned Table**

```sql
CREATE TABLE employees_partitioned (
    id INT,
    name STRING,
    age INT
) PARTITIONED BY (department STRING);
```

b. Bucketing (Further Data Optimization)

- **Buckets group data into fixed-size files** for better query efficiency.
- Example:

```sql
CREATE TABLE employees_bucketed (
    id INT,
    name STRING
) CLUSTERED BY (id) INTO 4 BUCKETS;
```

c. Indexing

Speeds up queries by **creating indexes** on frequently queried columns.

- Create Index

```sql
CREATE INDEX emp_index ON TABLE employees (name) AS 'COMPACT';
```

d. Views

Used to create **virtual tables** for frequently used queries.

- Create View

```sql
CREATE VIEW senior_employees AS
SELECT * FROM employees WHERE age > 40;
```

Fundamentals of HBase and ZooKeeper

1. HBase Fundamentals

Apache **HBase** is a **distributed, NoSQL database** that runs on top of Hadoop. It is designed for handling **large amounts of sparse data** in real time, providing **random read/write access** to structured data.

a. Key Features of HBase

- **Column-Oriented Storage** – Unlike traditional row-based databases, HBase stores data in **columns**.
- **Schema Flexibility** – It supports dynamic schema changes.
- **Horizontal Scalability** – Can scale by adding more nodes (RegionServers).
- **Strong Consistency** – Ensures data consistency across replicas.
- **Integration with Hadoop** – Works with Hadoop's **HDFS** for storage.

b. HBase Architecture

HBase consists of several key components:

1. HMaster

- Manages the **RegionServers**.
- Handles **metadata operations** and load balancing.

2. RegionServers

- Store actual data in **Regions** (distributed chunks of a table).
- Handle **read/write requests** from clients.

3. Regions

- A **Region** is a subset of table data stored in **HFiles** (HDFS).
- A table is **split into multiple Regions** to distribute the load.

4. HDFS (Hadoop Distributed File System)

- Stores the actual HBase data files.

5. ZooKeeper (Coordination Service)

- Manages **leader election, failure recovery, and distributed coordination**.
- Keeps track of the **live RegionServers** and **HMaster** status.

c. Data Model in HBase

HBase is a **NoSQL database** that uses a **key-value** structure.

Table Structure

- **Row Key** – The unique identifier for each row.
- **Column Families** – Logical groups of related columns.
- **Columns** – Contain actual data values.
- **Timestamp** – Each cell stores multiple versions of data (based on time).

Example HBase Table (Employee Data)

Row Key	Column Family: Personal	Column Family: Work
101	Name: John	Dept: HR
	Age: 30	Location: NY
102	Name: Alice	Dept: IT
	Age: 28	Location: SF

d. HBase Operations

- Create a Table

```shell
create 'employees', 'personal', 'work'
```

- Insert Data

```shell
put 'employees', '101', 'personal:name', 'John'
put 'employees', '101', 'personal:age', '30'
put 'employees', '101', 'work:dept', 'HR'
```

- Retrieve Data

```shell
get 'employees', '101'
```

- Scan a Table

```shell
scan 'employees'
```

- Delete a Row

```shell
deleteall 'employees', '101'
```

2. ZooKeeper Fundamentals

Apache **ZooKeeper** is a distributed **coordination service** used to manage **synchronization, configuration, and failure recovery** in distributed systems like HBase, Hadoop, and Kafka.

a. Key Features of ZooKeeper

- **Centralized Configuration Management** – Stores system settings for distributed applications.
- **Leader Election** – Helps select the leader node in a cluster.
- **Synchronization** – Ensures consistent data across distributed nodes.
- **Failure Detection and Recovery** – Keeps track of live and dead nodes.
- **High Availability** – Uses replication to avoid single points of failure.

b. ZooKeeper Architecture

ZooKeeper follows a **client-server model** with three key components:

1. ZooKeeper Ensemble (Cluster)

- Comprises multiple ZooKeeper nodes (**Leader + Followers**).
- Uses **replicated consensus (ZAB protocol)** to maintain consistency.

2. ZNodes (ZooKeeper Nodes)

- **Data units in ZooKeeper**, similar to files in a filesystem.
- Supports **ephemeral nodes** (temporary) and **persistent nodes**.

3. Clients (HBase, Hadoop, Kafka, etc.)

- Applications that connect to ZooKeeper to retrieve configuration data or participate in distributed coordination.

c. ZooKeeper Operations

- List Nodes (ZNodes)

```shell
ls /
```

- Create a Node

```shell
create /appconfig "version1"
```

- Get Data from a Node

```shell
get /appconfig
```

- **Delete a Node**

```shell
delete /appconfig
```

3. How HBase Uses ZooKeeper

HBase **depends on ZooKeeper** for:

- **Tracking Active HMaster** – Ensures only **one** HMaster is active at a time.
- **RegionServer Management** – Keeps a list of **live RegionServers**.
- **Failure Recovery** – Automatically detects and **reassigns Regions** in case of server failures.