BUSINESS INTELLIGENCE AND DATA ANALYTICS
– WE PREDICT THE FUTURE –

University of Petra

307401
Big Data and Data Warehouses

Introduction to Hadoop

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.

APACHE PIG

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 – First-Generation Framework

- A disk-based distributed processing system designed for large-scale, fault-tolerant batch computation.
- Follows a linear execution model:
  - Read data from HDFS
  - Apply map() function
  - Write intermediate data to disk
  - Shuffle and sort
  - Apply reduce() function
  - Write final output to HDFS
- Every phase involves reading from and writing to disk, which increases latency.
- Suitable for simple, one-pass data processing but inefficient for iterative tasks like machine learning or graph algorithms.
- Key Limitation: Disk I/O at every stage leads to poor performance for multi-stage workflows.

MapReduce

Apache Spark – Third-Generation Framework

- An in-memory distributed processing engine that improves on MapReduce's model.
- Uses Resilient Distributed Datasets (RDDs) to manage distributed data in memory.
- Supports chaining multiple transformations without writing intermediate results to disk.
- Writes to disk only when necessary (e.g., final output or fault recovery).
- Key Advantage: Significantly faster than MapReduce for iterative and interactive workloads.

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 built for **high-throughput, real-time data processing**. Created at **LinkedIn**, it is widely used as a **scalable message broker** for modern data pipelines.

Architecture Basics

- **Producer:** Sends data to Kafka topics
- **Consumer:** Reads data from topics
- **Broker:** Kafka server storing and serving data
- **Cluster:** Multiple brokers working together
- **Topic & Partitions:** Logical data streams split into partitions for scalability

Apache Flink

- **Apache Flink** is an open-source system for **real-time stream processing** and **batch processing**, offering **low latency**, **exactly-once reliability**, and **fault-tolerant stateful processing**.
- Its architecture includes **sources** (Kafka, DBs), **transformations** (filter, join, aggregate), and **sinks** (Kafka, DBs, lakes), with strong support for **event-time processing** using **watermarks**.

Apache Oozie

Apache Oozie is a workflow scheduler for Hadoop that automates and manages multi-step data processing pipelines. It organizes tasks like MapReduce, Hive, Pig, and Sqoop into a **DAG** to ensure proper execution order.

Key Points

- Schedules and coordinates multi-step Hadoop workflows
- Handles task dependencies, retries, and failure recovery
- Supports time-based and data-driven triggers
- Integrates with most Hadoop ecosystem tools

Use Cases

- Automating ETL and batch pipelines
- Daily data ingestion jobs
- Multi-tool workflows (Sqoop → Hive → MapReduce)

Example
A retail company uses Oozie to refresh its data warehouse nightly—importing data with Sqoop, transforming with Hive, aggregating with MapReduce, and updating reporting tables—fully automated with alerts on failures.

Apache Zookeeper

Apache ZooKeeper is a centralized coordination service for distributed systems. It maintains shared state and helps services work together reliably.

Key Functions

- **Centralized configuration management**
- **Naming service** for locating distributed components
- **Synchronization** of processes to avoid conflicts
- **Group management** for dynamic membership
- **Watchers** for real-time notifications on state changes

Apache Ambari

Apache Ambari is a web-based platform for **provisioning, managing, and monitoring Hadoop clusters**. It simplifies administration and provides an easy interface for handling complex Hadoop environments.

Key Features

- **Cluster Provisioning:** Guided installation and configuration of Hadoop clusters; add/remove nodes and services easily.
- **Service Management:** Start/stop/restart Hadoop services like HDFS, YARN, Hive, HBase.
- **Monitoring & Alerts:** Dashboards for real-time health, performance, and resource usage; customizable alerts.
- **Configuration Management:** Centralizes and simplifies Hadoop service configuration.
- **User Management:** Controls user access and permissions securely.
- **Metrics Integration:** Works with Ambari Metrics & Alerts for comprehensive monitoring.

Apache Ambari

Common Data File Formats

Format	Text / Binary	Row vs Column	Schema Handling	Compression	Splitable	Typical Use Cases
CSV	Plain-text	Row	None (external schema)	No (gzip/zip external)	No (gzipped) / Yes (plain)	Spreadsheets, exports
JSON / NDJSON	Plain-text	Row (hierarchical)	Self-describing (names and values)	No (gzip/bz2 common)	No (compressed) / Yes (NDJSON)	APIs, logs, documents
Avro	Binary	Row	Required (schema in header)	Yes (Snappy/Deflate)	Yes	Kafka, service exchange
Parquet	Binary	Column	Required (embedded)	Yes (Snappy/Gzip/Zstd)	Yes	Analytics, data lakes
ORC	Binary	Column	Required (embedded)	Yes (Zlib/Snappy/Zstd)	Yes	Hive, warehousing
XML	Plain-text	Tree-structured	Self-describing (tags)	No (gzip external)	No (compressed) / Yes (plain)	Legacy configs/docs