Big Data Pipeline: Hive and Hadoop Implementation

Github Repository: https://github.com/Amin-AQ/Batch-Analytics-HDFS

1. Introduction

This document outlines the implementation of a data pipeline using **Hadoop and Hive** to process user activity logs and content metadata efficiently. The pipeline follows a **star schema** design with a central fact table and supporting dimension tables for analytical queries.

2. Data Ingestion and Storage

Raw Data Storage

- A folder named **`raw_data`** contains the input files before ingestion.
- The data is ingested into **HDFS** under directories `/raw/logs/` and `/raw/metadata/`.
- The ingestion process is automated using a shell script **`ingest_logs.sh`**.

3. Hive Schema Definitions (DDL)

Raw Tables (External Tables)

```
CREATE EXTERNAL TABLE IF NOT EXISTS raw_user_logs (
   user_id INT,
   content_id INT,
   action STRING,
   event_timestamp STRING,
   device STRING,
   region STRING,
   session_id STRING)
PARTITIONED BY (year INT, month INT, day INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/raw/logs/';

CREATE EXTERNAL TABLE IF NOT EXISTS raw_content_metadata (
   content_id INT,
   title STRING,
    category STRING,
    length INT,
    artist STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/raw/metadata/';
```

Star Schema Tables (Parquet Format)

```
CREATE TABLE IF NOT EXISTS dim_users (
    user_id INT,
    region STRING,
    device STRING)
STORED AS PARQUET;

CREATE TABLE IF NOT EXISTS dim_content (
    content_id INT,
    title STRING,
    category STRING,
    length INT,
    artist STRING)
STORED AS PARQUET;

CREATE TABLE IF NOT EXISTS dim_sessions (
    session_id STRING,
    user_id INT)
STORED AS PARQUET;

CREATE TABLE IF NOT EXISTS fact_user_actions (
    user_id INT,
    content_id INT,
    session_id STRING,
    action STRING,
    event_timestamp STRING)
PARTITIONED BY (year INT, month INT, day INT)
STORED AS PARQUET;
```

4. Data Transformation Commands

```
-- Load Data into `dim_users`
INSERT OVERWRITE TABLE dim_users
SELECT DISTINCT user_id, region, device FROM raw_user_logs;

-- Load Data into `dim_content`
```

```
INSERT OVERWRITE TABLE dim_content
SELECT DISTINCT * FROM raw_content_metadata;
-- Load Data into `dim_sessions`
INSERT OVERWRITE TABLE dim_sessions
SELECT DISTINCT session_id, user_id FROM raw_user_logs;

-- Load Data into `fact_user_actions`
SET hive.exec.dynamic.partition.mode=nonstrict;
SET hive.exec.dynamic.partition=true;

INSERT OVERWRITE TABLE fact_user_actions PARTITION (year, month, day)
SELECT user_id, content_id, session_id, action, event_timestamp, year, month, day
FROM raw_user_logs;
```

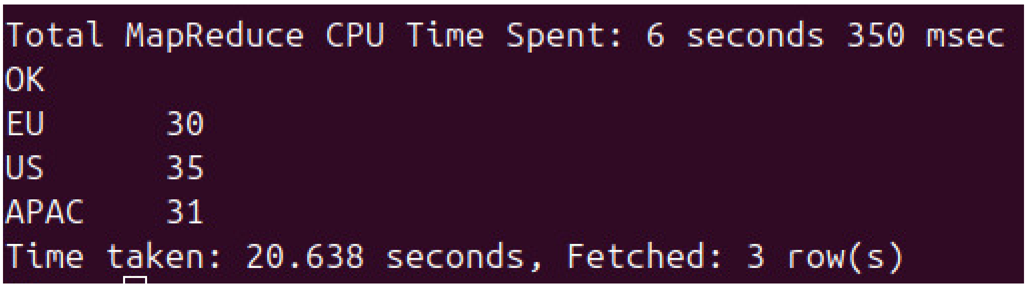
5. Sample Queries and Execution Results

Query 1: Monthly Active Users by Region

```
SELECT dim_users.region, COUNT(DISTINCT fact_user_actions.user_id) AS active_users
FROM fact_user_actions
JOIN dim_users ON fact_user_actions.user_id = dim_users.user_id
WHERE fact_user_actions.year = 2023 AND fact_user_actions.month = 9
GROUP BY dim_users.region;
```

Execution Time: ~20.638 seconds

Result:



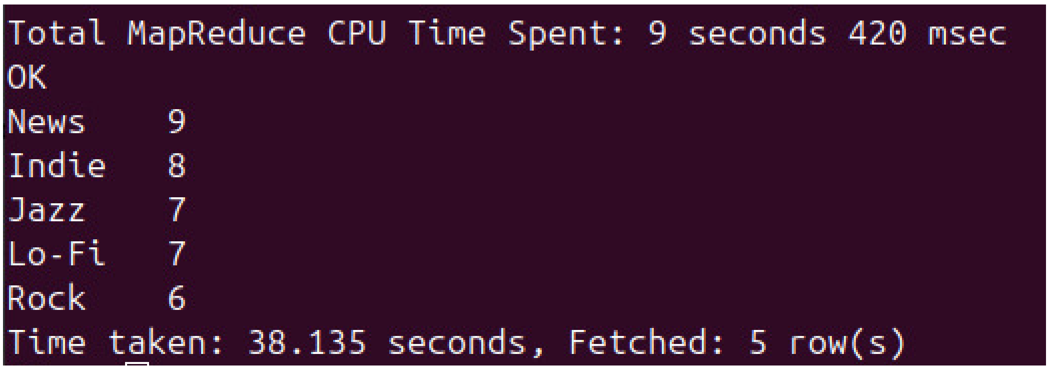
```
Total MapReduce CPU Time Spent: 6 seconds 350 msec
OK
EU      30
US      35
APAC    31
Time taken: 20.638 seconds, Fetched: 3 row(s)
```

Query 2: Top Categories by Play Count

```
SELECT dim_content.category, COUNT(*) AS play_count
FROM fact_user_actions
JOIN dim_content ON fact_user_actions.content_id = dim_content.content_id
WHERE fact_user_actions.action = 'play'
GROUP BY dim_content.category
ORDER BY play_count DESC
LIMIT 5;
```

Execution Time: ~38.135 seconds

Result:



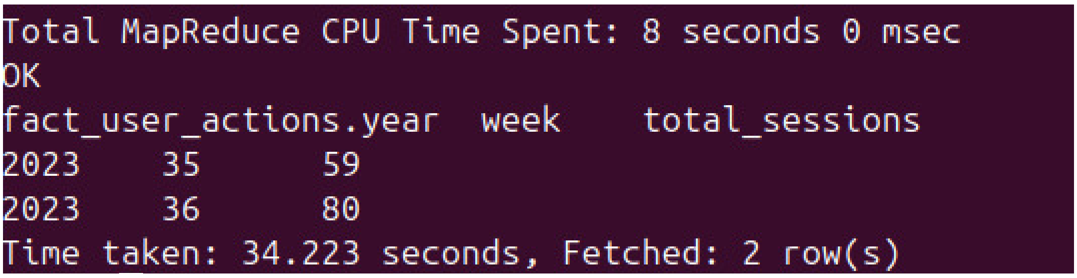
```
Total MapReduce CPU Time Spent: 9 seconds 420 msec
OK
News     9
Indie    8
Jazz     7
Lo-Fi    7
Rock     6
Time taken: 38.135 seconds, Fetched: 5 row(s)
```

Query 3: Average Session Count Per Week

```
SELECT fact_user_actions.year, WEEKOFYEAR(fact_user_actions.event_timestamp) AS week,
    COUNT(DISTINCT fact_user_actions.session_id) AS total_sessions
FROM fact_user_actions
GROUP BY fact_user_actions.year, WEEKOFYEAR(fact_user_actions.event_timestamp)
ORDER BY fact_user_actions.year, week;
```

Execution Time: ~34.223 seconds

Result:



```
Total MapReduce CPU Time Spent: 8 seconds 0 msec
OK
fact_user_actions.year  week    total_sessions
2023    35      59
2023    36      80
Time taken: 34.223 seconds, Fetched: 2 row(s)
```

6. Design Considerations & Performance Optimization

1. Partitioning Strategy

- `fact_user_actions` table is **partitioned by (year, month, day)** to optimize query performance.
- Dynamic partitioning is enabled for efficient data ingestion.

2. Data Storage Format

- **Raw Data:** Stored in `TEXTFILE` format for easy ingestion.
- **Transformed Data:** Stored in `PARQUET` format to benefit from compression and faster query execution.

3. Parallel Processing & Execution Optimization

Hive Execution Settings:

```
SET hive.exec.dynamic.partition.mode=nonstrict;
SET hive.exec.dynamic.partition=true;
```

MapReduce Optimization: Number of reducers dynamically determined for optimal resource utilization. Queries leverage **partition pruning** to scan only necessary data.

4. Execution Time Analysis

Stage	Execution Time
Data Ingestion (HDFS)	~10 sec
Raw Table Creation	~20 sec
Data Transformation (ETL)	~15 - 35 sec per table
Query Execution (Hive)	~20 - 40 sec per query

7. Conclusion

This pipeline efficiently processes and transforms large-scale user activity logs using **Hadoop and Hive**. The adoption of **partitioning, Parquet storage, and dynamic partitioning** significantly improves performance, making the system scalable for real-time analytics. Future enhancements may include **bucketing** and **Apache Spark integration** for further optimization.