AI 601 - Assignment 2: Batch Analytics Pipeline Project

Muhammad Huraira Anwer
Department of Computer Science
Lahore University of Management Sciences
25100314@lums.edu.pk

Muhammad Ahmad
Department of Electrical Engineering
Lahore University of Management Sciences
25100076@lums.edu.pk

GitHub Repository: https://github.com/muhmmad-ahmad-1/ai601-g11

1 Overview

The batch analytics pipeline is designed to ingest, store (in daily batches), transform, and analyze large-scale user activity logs from MediaCo's streaming platform. The ingestion process starts with a shell script that moves daily log files in *CSV* format from a local directory into **HDFS**, organizing them into a structured format (by date). The static metadata is also stored in a similar hierarchical structure.

Once ingested, the data is registered in **Hive** using external tables. The user activity logs are partitioned by date (year, month, day) to optimize query performance. Then, a star schema is implemented, consisting of a fact table `fact_logs` storing user interactions and dimension tables (`dim_content`, `dim_device`, `dim_region` and `dim_action`) containing metadata. The transformation step leverages Hive DDL to move data from the external tables (already partitioned by date) into an optimized Parquet format with star schema for efficient querying.

The final stage involves executing analytical queries to extract business insights, such as monthly active users by region. By leveraging **HDFS** for scalable storage and **Hive** for SQL-based querying, this pipeline ensures efficient batch processing of large datasets enabling MediaCo to derive actionable insights while maintaining a structured and automated workflow.

2 Data Generation and Ingestion

We generated user activity logs and content metadata tables following a structured approach to support robust data analysis and insights. This was done by prompting ChatGPT to provide a Python Script (present as `generate_csv.py` in our repository) for generating these logs and metadata. We chose a time window of 28 days, from 2024-03-01 to 2024-03-28 for generating these tables. The user activity logs table captured key behavioral interactions (such as play, pause, and forward) across various devices and regions, timestamped for temporal analysis. Each record was tied to a session ID, allowing for session-based user tracking and engagement analysis. Similarly, the inclusion of user IDs and content IDs enabled mapping user actions to specific content, facilitating trend analysis.

The content metadata table complemented this by storing essential attributes of media items, including title, category, length, and artist, which provide contextual information for user engagement patterns. By integrating both tables, we were able to run queries to analyze content popularity across different regions, session behaviors, and device preferences, enabling insights for optimizing content delivery, recommendation systems, and user experience enhancements. However, efficient querying meant a further conversion to star schema, which will be discussed later.

The first five rows of the user activity logs dataset and the content metadata are provided in Table. 1 and Table.2 respectively.

The user activity logs CSV was then partitioned by date, where each partition was stored in a separate directory such as *'2024-03-01'*. The content metadata CSV was not partitioned due to its static nature.

These files were then uploaded to HDFS through the `ingest_logs.sh` script leveraging the **hdfs dfs** `put` and `mkdir` commands. The logs were partitioned by data into directories such as `raw/logs/year/month/day` while the metadata was stored in the `raw/metadata` directory. We wrote an extra shell script (`ingest_all_logs.sh`) to automate the process for the 28 days.

User ID	Content ID	Action	Timestamp	Device	Region	Session ID
182	1000	Play	2024-03-01 17:37:21	Tablet	EU	w44j17
169	1005	Play	2024-03-01 06:01:35	Mobile	US	b118t7
121	1001	Play	2024-03-01 11:35:53	Tablet	EU	0qbgbi
139	1002	Forward	2024-03-01 09:51:08	Desktop	US	hlyc6i
157	1005	Pause	2024-03-01 05:18:36	Desktop	US	r7p821

Table 1: User Activity Log

Content ID	Title	Category	Length	Artist
1000	Classical Moods	Podcast	647	MC Flow
1001	Jazz Classics	Podcast	273	Anchor FM
1002	History Deep Dive	Acoustic	571	Professor X
1003	Daily News	Electronic	593	Guru Mike
1004	Indie Dreams	Pop	1102	Guru Mike

Table 2: Content Metadata

Then we used Hive SQL to create **external tables** pointing to these log partitions and metadata file. This was done using the following Hive DDL commands, creating an external table partitioned by date for the logs named as *logs* and a similar, un-partitioned external table named *metadata*:

- **Logs:**

```
CREATE EXTERNAL TABLE logs (
user_id INT,
content_id INT,
action STRING,
event_timestamp TIMESTAMP,
device STRING,
region STRING,
session_id STRING
)
PARTITIONED BY (log_date STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hadoop/raw/logs'
TBLPROPERTIES ('skip.header.line.count'='1');
```

- **Metadata**:

```
CREATE EXTERNAL TABLE metadata (
content_id INT,
title STRING,
category STRING,
length INT,
artist STRING
```

```
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hadoop/raw/metadata'
TBLPROPERTIES ('skip.header.line.count'='1');
```

The partitions were added sequentially for each date. A sample command for a single date is as follows:

```
ALTER TABLE logs ADD PARTITION (log_date='2024-03-01') LOCATION

user/hadoop/raw/logs/2024/03/01;
```

We used an additional shell script (`automate_hive.sh`) to automate the steps of creating external tables, their partitions, and then further the conversion into star schema, and the loading (transformation) of data in the star schema. The main SQL commands for the conversion to star schema are discussed in the next section.

3 Schema Design

As previously mentioned, the external tables were converted to a star schema with a `fact_logs` table containing user interaction activity and dimension tables containing metadata including `dim_content`, `dim_device`, `dim_region` and `dim_action`. The original external tables were in a text-based format where scanning a specific partition means reading full text files, which lack compression and require full row scans. In contrast, the star schema tables were stored in Parquet columnar format, which ensures only relevant columns are scanned, significantly reducing I/O. In addition, we converted the repeated entries of region, device, action as strings in the original tables (which made filtering and aggregation expensive through string comparisons) to integer based foreign keys (region_id, device_id, action_id), making filters, joins, and aggregations much faster. The star schema performs integer joins, much faster than text comparisons. The actual device, region, action strings were stored in their respective dimension tables instead for access only when needed. This also saves storage space, as relatively massive strings are stored only once with redundancy of only less costly integer ids in the fact table. The star schema, therefore, reduces log size, allowing faster scans and lower storage costs. This allows much faster querying and more optimum storage.

The star schema is also more scalable and better for dynamic entries because dimensions can be updated separately without modifying the fact table. Introduction of new entries only causes additional new integer ids to appear in the fact table. In this way, information is kept nicely compartmentalized. We provide a diagram of our star schema in Figure. 1.

The following queries were used for creating the star schema and filling it. We omit redundant queries for the different dimension tables (such as `dim_device`, `dim_region` and `dim_action`) which only have very minor differences. Note that the dimension tables were filled before the fact table:

Creating Tables

- Fact Table (fact_logs):

```
CREATE TABLE fact_logs (
user_id INT,
content_id INT,
event_timestamp TIMESTAMP,
action_id INT,
device_id INT,
region_id INT,
session_id STRING
)
PARTITIONED BY (log_date STRING)
STORED AS PARQUET;
```

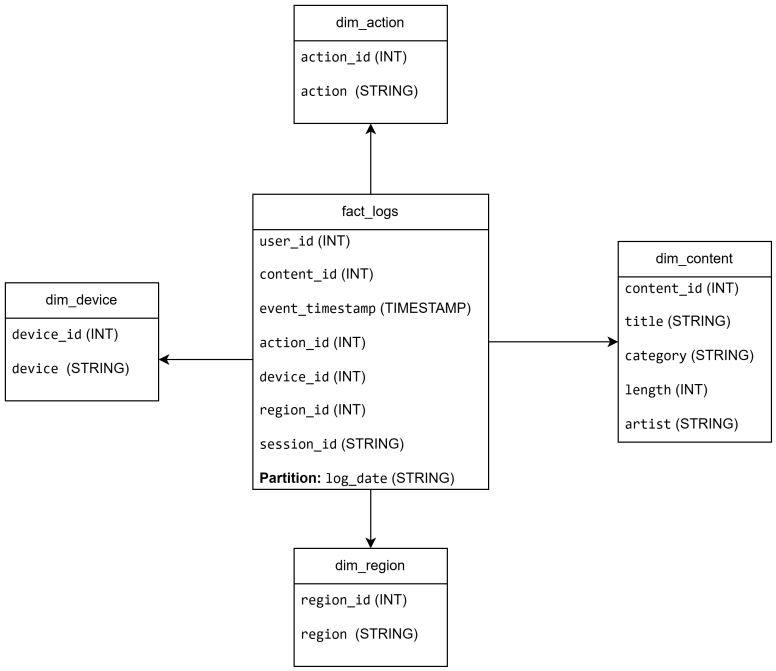

Figure 1: Star Schema

- Dimension Table (dim_content):

```
CREATE TABLE dim_content (
content_id INT,
title STRING,
category STRING,
length INT,
artist STRING
)
STORED AS PARQUET;
```

- Other dimension tables (e.g. dim_device):

```
CREATE TABLE dim_device (
device_id INT,
device STRING
)
STORED AS PARQUET;
```

Filling Tables

- Fact Table (fact_logs):

```
INSERT INTO fact_logs
PARTITION (log_date)
SELECT
    l.user_id,
    l.content_id,
    l.event_timestamp,
    a.action_id,
    d.device_id,
    r.region_id,
```

```
    l.session_id,
    l.log_date
FROM logs l
JOIN dim_action a ON l.action = a.action
JOIN dim_device d ON l.device = d.device
JOIN dim_region r ON l.region = r.region;
```

- Dimension Table (dim_content):

```
INSERT INTO dim_content
SELECT content_id, title, category, length, artist FROM metadata;
```

- Other dimension tables (e.g. dim_device):

```
INSERT INTO dim_device
SELECT
    t.device_id,
    t.device
FROM (
    SELECT
        ROW_NUMBER() OVER (ORDER BY device) AS device_id,
        device
    FROM (SELECT DISTINCT device FROM logs) AS unique_devices
) t;
```

We report the time taken for each step (i.e., ingestion into HDFS, and each step in making the external tables, and the star schema) and the total time taken in Table 3.

Process	**Time (s)**
Ingestion - HDFS	182
Creation - logs	2
Creation - metadata	1
Creation - fact_logs	<1
Creation - dim_tables	<1
Filling - fact_logs	53.392
Filling - dim_content	24.822
Filling - dim_device	68.152
Filling - dim_region	65.273
Filling - dim_action	68.459
Total - Hive	294
Total - Hive + HDFS	476

Table 3: Time Taken for Ingestion and Creation of Tables

4 Analytical Queries

The first query was to find Top Category by Play Counts. The query is as follows:

```
SELECT c.category, COUNT(*) AS play_count
FROM fact_logs f
JOIN dim_action a ON f.action_id = a.action_id
JOIN dim_content c ON f.content_id = c.content_id
WHERE a.action = 'play'
AND f.log_date BETWEEN '2024-03-01' AND '2024-03-28'
GROUP BY c.category
ORDER BY play_count DESC
LIMIT 10;
```

The results returned were as follows

Category	Play Count
Podcast	52
Acoustic	35
Electronic	35
Indie	31
Jazz	22
Pop	18

Table 4: Play count by category

The second query was to find Average Weekly Session Length. The query is as follows:

```
WITH session_durations AS (
    SELECT
        session_id,
        DATE_SUB(TO_DATE(event_timestamp), Pmod(DAYOFWEEK(TO_DATE(event_timestamp)) - 2, 7))
        AS week_start, -- Get Monday
        MAX(UNIX_TIMESTAMP(event_timestamp)) - MIN(UNIX_TIMESTAMP(event_timestamp))
        AS session_length_seconds
        FROM fact_logs
        WHERE log_date BETWEEN '2024-03-01' AND '2024-03-28'
        GROUP BY session_id, DATE_SUB(TO_DATE(event_timestamp),
        Pmod(DAYOFWEEK(TO_DATE(event_timestamp)) - 2, 7))
    )
        SELECT
        week_start,
        AVG(session_length_seconds) AS avg_session_length_seconds
        FROM session_durations
        GROUP BY week_start
        ORDER BY week_start;
```

The results returned were as follows

Date	Avg Session Length
2024-02-26	3897.01
2024-03-04	3145.97
2024-03-11	2957.41
2024-03-18	1849.93
2024-03-25	2885.87

Table 5: Weekly Average Session Length

The third query was to find Weekly Active Users Region-wise. The query is as follows:

```
SELECT
    DATE_FORMAT(f.event_timestamp, 'yyyy-MM') AS month,
    r.region,
    COUNT(DISTINCT f.user_id) AS active_users
    FROM fact_logs f
    JOIN dim_region r ON f.region_id = r.region_id
    WHERE f.log_date >= '2024-03-01' AND f.log_date < '2024-03-14'
    GROUP BY DATE_FORMAT(f.event_timestamp, 'yyyy-MM'), r.region
    ORDER BY month, active_users DESC;
```

The results returned were as follows

The fourth query was to find Daily Peak Playback Hours Region-wise. The query is as follows:

Month	Region	Active Users
2024-03	US	94
2024-03	APAC	93
2024-03	EU	89

Table 6: Monthly Active Users by Region

```
SELECT
    TO_DATE(f.event_timestamp) AS play_date,
    r.region,
    HOUR(f.event_timestamp) AS peak_hour,
    COUNT(*) AS play_count
    FROM fact_logs f
    JOIN dim_action a ON f.action_id = a.action_id
    JOIN dim_region r ON f.region_id = r.region_id
    WHERE a.action = 'play'
    AND f.log_date BETWEEN '2024-03-01' AND '2024-03-28'
    GROUP BY TO_DATE(f.event_timestamp), r.region, HOUR(f.event_timestamp)
    ORDER BY play_count DESC;
```

The results returned were as follows

Date	Region	Peak Hour	Play Count
2024-03-16	US	12	3
2024-03-14	APAC	6	2
...	..	..	..
2024-03-01	APAC	16	1
2024-03-01	APAC	9	1

Table 7: Peak Playback Hours by Region

The query times were as follows:

Query	Runtime
1	53.58
2	47.54
3	50.95
4	55.49

Table 8: Query runtimes in seconds

5 Conclusion

The batch analytics pipeline effectively processes large-scale user activity logs, leveraging HDFS for scalable storage and Hive for SQL-based querying. The adoption of a star schema significantly optimized performance by reducing I/O, improving storage efficiency, and enabling faster query execution through integer-based foreign keys and Parquet storage.

The entire pipeline, including data ingestion, transformation, and table creation, took approximately 476 seconds. The transformation into the star schema, particularly the population of dimension and fact tables, accounted for a significant portion of this time.

Query execution times varied between 47 to 56 seconds, with the most computationally expensive queries involving aggregations over large partitions. A key limitation faced was the processing time for large partitions, especially when filtering text-based columns in the initial external tables. Performance could be improved by further partitioning the fact table by region to enhance query

filtering, bucketing on user_id to speed up joins, and using materialized views for frequently accessed aggregations.

Despite these limitations, the pipeline provides a structured and automated approach to batch analytics, allowing derivation of actionable insights using batched data. The nature of the company does, however, indicate real-time stream processing for more immediate insights to be desirable. Spark can be used for such stream processing.