Final Rport

Ziyang Xu

December 3, 2024

1 Introduction

In the era of big data, organizations are increasingly leveraging data-driven strategies to gain insights, improve decision-making processes, and enhance operational efficiency. At the core of this paradigm lies the data warehouse, a centralized repository designed to store, manage, and analyze large volumes of structured and semi-structured data. This project focuses on building a distributed data warehouse pipeline using Hadoop and complementary tools such as Flume, Kafka, and Hive, to simulate a real-world big data processing and analytics environment.

The primary objective of the project is to design and implement a robust pipeline capable of handling data generation, real-time ingestion, processing, and visualization. The scope of this work includes deploying a distributed Hadoop ecosystem across multiple virtual machines, configuring tools for seamless data transfer, and conducting advanced analytics to extract meaningful insights. By leveraging Hive for querying and MySQL for storing processed results, the project also incorporates Superset to create visually appealing dashboards for presenting analytical outcomes. This end-to-end pipeline serves as a foundational framework for exploring how big data technologies can be integrated to solve complex data analysis challenges in distributed systems.

2 Literature Review

The concept of data warehousing has evolved significantly since its inception in the late 1980s, when it was primarily focused on providing a centralized repository for decision support systems. Early foundational work by Bill Inmon and Ralph Kimball laid the groundwork for data warehouse design, emphasizing the importance of organizing data into either normalized relational structures or dimensional models to support analytical queries. These principles continue to influence modern data warehousing approaches, particularly with the integration of big data frameworks.

Recent advancements in big data technologies have extended the capabilities of traditional data warehouses. The Hadoop ecosystem, with its distributed storage (HDFS) and resource management (YARN), has become a popular platform for handling large-scale data processing. Research highlights the synergy between Hadoop and Hive, where Hive provides an SQL-like interface to interact with data stored in HDFS, making it accessible to users familiar with relational database systems. Similarly, tools like Kafka and Flume have emerged as key components for building real-time data pipelines. Kafka's high-throughput messaging system ensures reliable data streaming, while Flume specializes in collecting and transferring unstructured data into Hadoop-compatible formats.

The integration of real-time and batch processing has been a subject of considerable research, with frameworks like Lambda Architecture proposing hybrid approaches. However, significant gaps remain in practical implementation, particularly in optimizing data transfer and query execution in distributed systems. Furthermore, while tools like Apache Superset are widely acknowledged for their data visualization capabilities, there is limited documentation on their application in end-to-end pipelines, especially in scenarios involving real-time data ingestion and processing.

This project addresses some of these gaps by implementing a real-world data pipeline that combines data generation, ingestion, processing, analysis, and visualization. By documenting each step, this work contributes to the understanding of how distributed systems and big data tools can be effectively integrated to create scalable and efficient data analytics solutions.

3 Methology

This section outlines the methods and techniques employed in the project, focusing on data collection, preprocessing, and analysis. Before delving into these aspects, we provide a detailed explanation of the structure and significance of the mock data used in the pipeline.

3.1 Data Overview

The mock data is generated in JSON format, simulating real-world interactions with a mobile application. Each JSON object represents a single event, encompassing a variety of information about user interactions, app behavior, and system logs. Below is an example of the generated data:

```
{
    "common": {
        "ar": "15",                 -- Province ID
        "ba": "iPhone",             -- Phone brand
        "ch": "Appstore",           -- Channel
        "is_new": "1",              -- First day usage: On the first day of usage, this f
        "md": "iPhone 8",           -- Phone model
        "mid": "YXfhjAYH6As2z9Iq",  -- Device ID
        "os": "iOS 13.2.9",         -- Operating system
        "sid": "3981c171-558a-437c-be10-da6d2553c517", -- Session ID
        "uid": "485",               -- User ID
        "vc": "v2.1.134"            -- App version
    },
    "actions": [{
        "action_id": "favor_add",   -- Action ID
        "item": "3",                -- Target ID
        "item_type": "sku_id",      -- Target type
        "ts": 1585744376605         -- Action timestamp
    }],
    "displays": [{
        "displayType": "query",     -- Display type
        "item": "3",                -- Display object ID
        "item_type": "sku_id",      -- Display object type
        "order": 1,                 -- Order of appearance
        "pos_id": 2,                -- Display position
        "pos_seq": 1                -- Display sequence number (position number for multi
    },
    {
        "displayType": "promotion",
        "item": "6",
        "item_type": "sku_id",
        "order": 2,
        "pos_id": 1,
        "pos_seq": 1
    },
    {
        "displayType": "promotion",
        "item": "9",
        "item_type": "sku_id",
        "order": 3,
        "pos_id": 3,
        "pos_seq": 1
    },
    {
        "displayType": "recommend",
        "item": "6",
        "item_type": "sku_id",
        "order": 4,
        "pos_id": 2,
        "pos_seq": 1
    },
    {
        "displayType": "query",
        "item": "6",
        "item_type": "sku_id",
        "order": 5,
        "pos_id": 1,
        "pos_seq": 1
    }],
    "page": {
        "during_time": 7648,          -- Duration in milliseconds
        "item": "3",                  -- Target ID
        "item_type": "sku_id",        -- Target type
        "last_page_id": "login",      -- Previous page ID
        "page_id": "good_detail",     -- Current page ID
        "from_pos_id": 999,           -- Source position ID
        "from_pos_seq": 999,          -- Source position sequence number
        "refer_id": "2",              -- External marketing channel ID
        "sourceType": "promotion"     -- Source type
    },
    "err": {
        "error_code": "1234",         -- Error code
        "msg": "***********"          -- Error message
    },
    "ts": 1585744374423               -- Entry timestamp
}
```


Figure 1: Data Example

3.2 Data Collection

The data used in this project was synthetically generated in JSON format, simulating real-world user interactions with a mobile application. The mock data contained multiple nested structures, including user attributes, actions, displays, page details, and error logs. Key fields include device information, session IDs, timestamps, and user interactions with app features.

Data generation was performed on two virtual machines (XU002 and XU003) using a custom Java program. The program generated realistic mock data streams, which were then captured and ingested into the pipeline. Apache Flume was used as the primary tool for data ingestion. Flume collected data from the Java program and passed it to Apache Kafka, a distributed messaging system. Kafka ensured reliable and high-throughput data transfer between Flume and downstream systems. The data was eventually stored in Hadoop Distributed File System (HDFS) for subsequent processing and analysis.

The data collection process faced a critical challenge known as the zero drift problem. This issue arises when log data is generated close to midnight (e.g., 23:59:59), and during its journey through the pipeline, the automatically generated timestamp in the Kafka source header may reflect the following day due to processing delays. Consequently, the data could be mistakenly stored in the wrong partition in Hive, which organizes data by date.

To address this, the following solution was implemented:

1. Log Data Generation: Data was generated on XU002 and XU003 using a Java program that produced realistic JSON log entries. Each log entry included a timestamp (ts field) from the application's context, reflecting the exact moment the event occurred.

2. Flume Kafka Channel with Interceptor: The data was first collected by Apache Flume and forwarded to Kafka. Upon retrieval from Kafka by the second Flume channel, the Kafka source header added a new timestamp, which could potentially differ from the original log timestamp.

3. Timestamp Correction: To prevent partitioning errors in Hive, an interceptor was added to the second Flume channel. The interceptor extracted the original ts timestamp from the log data body and replaced the Kafka source's auto-generated timestamp in the header. This ensured that the correct timestamp was retained throughout the pipeline, preserving the integrity of the partitioning scheme in Hive.

4. Data Storage in HDFS: The corrected data was stored in HDFS, partitioned by the original date extracted from the log timestamp. This approach guaranteed that log events were placed in the appropriate daily partitions, regardless of delays in the pipeline.

By incorporating this mechanism, the zero drift problem was effectively mitigated, ensuring accurate data partitioning in Hive and maintaining the reliability of downstream analytical processes.

3.3 Data Preprocessing

The preprocessing phase of the project involved structuring, organizing, and preparing the JSON-style log data for analysis in Hive. This section explains how the table was created in Hive using nested structures (STRUCT, ARRAY) and maps, and how the daily data was loaded into Hive partitions using a custom bash script.

3.3.1 Table Creation in Hive

The JSON data was highly nested, with fields representing user information, session details, actions, and display information. To efficiently query this data, a Hive table named `ods_log_inc` was created using STRUCT and ARRAY types to preserve the nested format. Below is the explanation of the table structure:

1. STRUCT Data Type:
   The STRUCT type was used to encapsulate related fields into a single logical grouping. For instance:

   - The common field groups attributes such as ar (area), ba (brand), and mid (member ID), representing device and session metadata.

- The page field encapsulates information about user navigation, including `page_id` and `last_page_id`.
- Error details (err) and startup details (start) were also stored as STRUCT types.

2. ARRAY Data Type:
   The actions and displays fields, which contain lists of user interactions and display events, were represented as arrays of STRUCT. This approach allows efficient querying of repeated events. For example:
   - The actions array stores actions such as adding items to favorites, with each action containing an `action_id`, item, `item_type`, and timestamp.
   - The displays array records information about displayed items, including the type, position, and item details.
3. Partitioning:
   The table was partitioned by the dt field, which represents the date. Partitioning enables efficient queries by filtering data based on specific dates, which is critical for large datasets.
4. Storage and Format:
   The table was defined as an external table to ensure the data remains in its original HDFS location and is not deleted if the table is dropped. The data was stored in JSON format using Hive's JsonSerDe, with Gzip compression applied to reduce storage requirements.

3.3.2 Daily Data Loading with Bash Script

To automate the process of loading daily log data into Hive, a bash script was written. This script dynamically loads data into the appropriate partition based on the date, ensuring that each day's data is stored in the correct location.

Key features of the script:

- Dynamic Date Handling: If a date is provided as an argument, the script uses it; otherwise, it defaults to the previous day's date (date -d "-1 day").
- Dynamic SQL Generation: The script constructs a LOAD DATA statement to load data from HDFS into the Hive table's partition corresponding to the date.
- Partitioned Loading: Data is directly loaded into the dt partition, ensuring efficient storage and query performance.

3.4 Analysis Techniques

The analysis phase of this project involved querying and aggregating data stored in Hive to extract meaningful insights. This section describes the techniques used to perform user and behavioral analytics, as well as the challenges faced during execution.

3.4.1 Analysis in HIVE

The primary analysis tasks focused on understanding user behavior and app performance. A notable example was the creation of a table to analyze user distribution by province. This table provided insights into the number of unique users in each area, enabling further analysis of regional trends and app usage patterns.

Example: Creating a Province-Based User Analytics Table The `user_province_info` table(shown in appendix) aggregates data about user activity by province and stores it in a compact and efficient format. The code in appendix B demonstrates how the table was created and populated.Explanation The `user_province_info` table aggregates unique user counts (`user_count` table) based on their area code (`ar_code`).It joins the `ods_log_inc` table with a `province_mapping` table to enrich the data with province names and ISO codes.The data is stored in the Parquet format, compressed using the Snappy codec, for efficient storage and faster query performance. And I also got `channel_count` table to analyze the distribution of users across different app channels, such as the App Store, Google Play Store, and other sources. And `user_province_info` table provides insights into the geographical distribution of users.

3.4.2 Challenges and Adjustments

During the implementation, a significant challenge was the insufficient memory allocation for the master node (XU002). Since Hive relies heavily on system resources for executing complex queries, tasks like aggregating large datasets or performing joins could cause Hive to crash. This limitation required a shift in strategy:

- Focusing on Simple Analyses: Instead of running complex multi-level aggregations, simpler queries such as counting users by channel or province were prioritized.
- Resource Monitoring: Memory usage was closely monitored, and execution plans were optimized to reduce resource consumption.
- Future Improvement: To enable more complex analytics, it is recommended to allocate additional memory to the master node or offload HiveServer2 to another machine in the cluster.

4 Discussion

The primary goal of this project was to design and implement a data pipeline for processing, analyzing, and visualizing JSON-style log data in a distributed environment. All data are mocked, so it does not accurately represent real-world user behaviors or operational patterns.Despite the limitations of mock data, the visualization phase of the project serves to demonstrate the pipeline's capabilities for processing and presenting insights. Below are examples of visualizations created using Apache Superset based on the processed data:

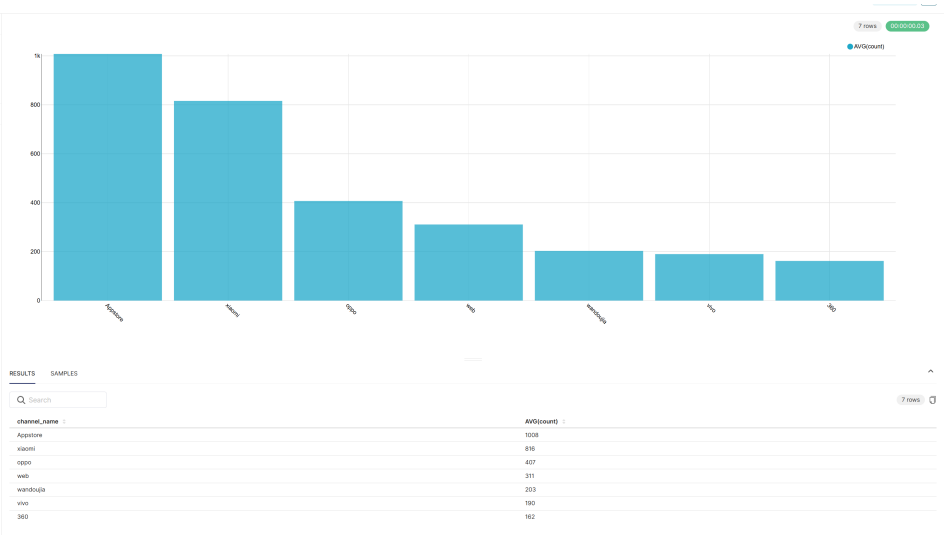

Figure 2: User Channel Count

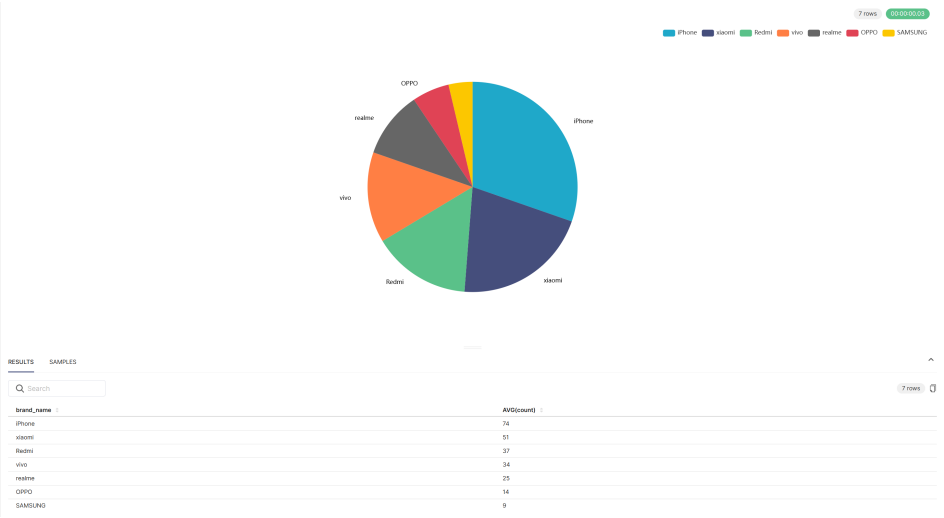

Figure 3: User Device

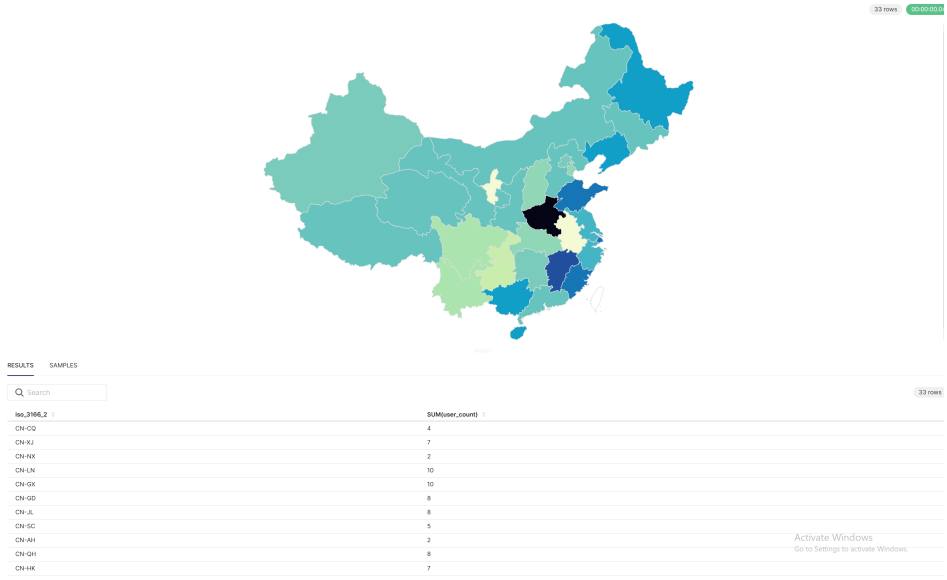

Figure 4: User position

5 Conclusion

5.1 about Project

Throughout this project, I gained invaluable insights into data engineering and the complexities of building a comprehensive data pipeline. One of the most challenging yet rewarding aspects was setting up the environment. I spent a significant amount of time configuring the infrastructure, which involved setting up multiple virtual machines and integrating tools such as Hadoop, Hive, Flume, Kafka, and MySQL into a cohesive system. This process was far from straightforward, as each component had its own intricacies and required meticulous configuration to ensure seamless communication. Aggregating all parts together was an arduous task but a vital step toward building a functional and robust pipeline.

A significant technical takeaway from this project was understanding how tools like Sqoop work, particularly its requirement for data to be in plain-text format, such as CSV, to successfully transfer data from Hive to MySQL. This insight played a critical role in ensuring smooth data export and integration. I also learned to use DataGrip, a powerful database management tool, which greatly

enhanced my efficiency in managing and querying Hive tables. Additionally, I encountered and resolved complex problems such as zero-drifting, which required careful handling to maintain data consistency and accuracy throughout the pipeline.

Looking back, this project has been a transformative learning experience. I now have a deeper appreciation for the complexities involved in setting up distributed environments and managing large-scale data systems. While the implementation is functional, there is room for future enhancement, such as migrating the entire pipeline to the cloud for scalability and reliability, enabling real-time data processing, integrating advanced analytics and visualization tools, and further automating ETL processes. These experiences have not only expanded my technical expertise but also reinforced the importance of persistence and problem-solving in tackling complex challenges. The lessons learned and skills acquired during this journey will undoubtedly guide me in future projects and professional endeavors.

5.2 Future Work

While the current implementation is robust and functional, there are several enhancements I plan to undertake in the future:

- Migrating the Entire Pipeline to the Cloud: Deploying the pipeline on cloud platforms such as AWS or GCP for improved scalability, reliability, and ease of maintenance.
- Real-Time Data Processing: Implementing real-time data processing capabilities to handle streaming data more efficiently.
- Integrating advanced visualization tools for better insights and decision-making.

This project not only helped me build practical expertise in using tools like Hive, Sqoop, and DataGrip but also gave me a solid foundation in problem-solving and data pipeline architecture. Moving forward, I am confident in applying these skills to larger and more complex projects, with a vision of creating scalable and efficient systems.

References

1. Kimball, R., Ross, M. (2013). The data warehouse toolkit: The definitive guide to dimensional modeling (3rd ed.). Wiley.
2. Snowflake. (n.d.). Online course: Data warehousing workshop.
3. AmpCode. (2021, May 5). Streaming data to HDFS using Apache Flume — Big Data Hadoop Tutorial [Video]. YouTube.
4. Prof. Arif Shaikh. (2023, June 15). Hadoop 2.x Architecture [Video]. YouTube.
5. Data Engineering. (2020, June 1).Big Data Engineering Master Course in Tamil [Video]. YouTube.