LOVELY PROFESSIONAL UNIVERSITY

RESULT MANAGEMENT SYSTEM USING HADOOP, SPARK, AND HIVE

Big Data Processing for Student Analytics

Submitted by

Arjun Kallatt (12216676)

Department of Computer Science

1. Introduction

Result Processing System relies on Hadoop, Spark, and Hive to process large student data, compute marks, and pull out useful statistical data efficiently. Legacy result processing systems are non-scalable and non-performant, but the system avoids these problems by using Hadoop's HDFS for distributed storage, Spark for in-memory efficient computation, and Hive for SQL-like querying of large data. The system begins with data ingestion into HDFS, followed by MapReduce processing wherein the Mapper reads student records and subjects and the Reducer computes subject-wise averages. The processed data is then processed in Spark DataFrames, enabling complex transformations like filtering, aggregation, and partitioning for best performance. In addition, Hive tables store structured results for SQL-based analysis, e.g., top performers and pass/fail trends. Visualization libraries like Matplotlib and Seaborn are employed to generate insights through bar charts, histograms, and box plots. The system enhances scalability, accuracy, and performance significantly, making it a valuable alternative to legacy result processing systems. Future potential enhancements are real-time streaming through Kafka, adaptive query execution in Spark, and an interactive dashboard for dynamic visualization and querying.

2. Technologies Used

- **Hadoop 2.7.7**: Provides distributed storage and processing of large datasets.
- **Apache Spark**: Enables in-memory data processing for faster computations.
- **Hive**: Allows SQL-like query execution on structured data stored in Hadoop.
- **HDFS (Hadoop Distributed File System)**: Manages large-scale data storage across distributed nodes.
- **Python (PySpark, Pandas, Matplotlib, Seaborn)**: Used for scripting, data manipulation, and visualization.

3. Project Workflow

1. **Environment Setup:**
    - Installed OpenJDK 8, Hadoop, and PySpark in preparation for constructing the necessary execution environment.
    - Establish key Hadoop files (hdfs-site.xml and core-site.xml) to enable distributed file storage and processing.
    - Establishing structured and organized NameNode and DataNode for smooth system operation.
2. **Hadoop HDFS Configuration:**

- Created directories in HDFS to store different categories of data:
  - /input: Stores raw student data files.
  - /output: Stores processed results.
  - /user/root: Stores user-specific files and configurations.
- Uploaded input.txt containing student records into HDFS for batch processing.

3. **MapReduce Processing:**
   - Implemented **Mapper (mapper.py)**:
     - Reads input student data.
     - Extracts student names, subjects, and marks.
     - Emits key-value pairs (subject, marks) for processing.
   - Developed **Reducer (reducer.py)**:
     - Receives grouped key-value pairs from Mapper.
     - Computes **subject-wise average marks**.
     - Outputs the final computed results for storage in HDFS.
   - Executed Hadoop Streaming jobs to run the MapReduce process and generate subject-wise aggregated data.
4. **Data Ingestion and Processing in Spark:**
   - Used Python to generate 10,000 synthetic student records for large-scale testing.
   - Loaded data into a Spark DataFrame for structured, distributed processing.
   - Applied key transformations:
     - Filtering: Removed incorrect or missing values.
     - Aggregations: Computed subject-wise performance metrics.
   - Stored the cleaned and processed data back into HDFS (/user/root/all_data) for further analysis.
5. **Statistical Analysis and Visualization:**
   - Calculated essential statistical metrics such as mean, max, and min marks for each subject.
   - Stored computed statistics in HDFS (/user/root/stats_data).
   - Used Matplotlib and Seaborn for data visualization:
     - Bar charts representing average marks per subject.
     - Histograms illustrating student performance distribution.

        - Box plots identifying outliers in student marks.

6. **Hive Integration and Query Execution:**
    - Created Hive tables to store structured student results for querying.
    - Executed SQL queries in Hive for in-depth analysis:
        - Identified top-performing students across subjects.
        - Computed pass/fail rates based on predefined thresholds.
        - Analyzed subject performance trends over different time periods.

4. Explanation of Code Components

- **Mapper (mapper.py)**: It reads student information from input files, parses each line to retrieve student information, and emits key-value pairs (subject, marks) for later aggregation.
- **Reducer (reducer.py)**: It takes sorted key-value pairs from the Mapper, calculates the average marks for each subject, and prints out the final processed data to HDFS.
- **Spark DataFrame Transformations**: It encompasses null values filtering, aggregating scores subject-wise, and performance optimisation through data partitioning.
- **Hive Queries**: Structured queries for retrieving student information, allowing for efficient query of outcomes like top-performing students, pass and failure rates, and performance in different subjects.
- **Visualization Scripts**: Utilizes Matplotlib and Seaborn to create visualizations of resultant data in bar graphs and histogram form.

5. Key Findings and Observations

- Processed and graded student information efficiently in a Hadoop-Spark workflow.
- MapReduce jobs computed efficiently subject-wise averages for enormous amounts of data.
- Apache Spark DataFrame API improved performance and efficiency in processing dramatically.
- Visual analytics provided richer insights regarding student performance trends.
- Ease and flexibility in accessing structured insights arose via Hive SQL queries.

6. Challenges and Solutions

- **Encoding Issues:** Encountered UTF-8 errors during data processing; resolved by setting fallback encoding to Latin-1 in mapper.py.
- **Data Format Errors:** Implemented exception handling in Reducer to skip malformed input records and ensure continuity in processing.
- **Performance Bottlenecks:** Optimized Spark transformations using:
    - **Partitioning:** Distributed data more efficiently across nodes.
    - **Caching:** Reduced redundant computations by storing intermediate results.

7. Conclusion

This project amply shows the potential of Big Data technologies to transform student result management. With the use of Hadoop, Spark, and Hive, the system is highly scalable, accurate, and efficient in processing large-scale student data. MapReduce, Spark DataFrames, and Hive SQL queries integrated ensure systematic analysis and easy retrieval of student performance measures. Distributed computing, in-memory processing, and statistical analysis are essential in transforming raw student data into actionable information for informed school decision-making. Visual analytics add further to the understanding of data, enabling easy identification of performance trends, subject-wise analysis, and pass/fail patterns. This project is a scalable and high-performance alternative to conventional result-processing systems, with future prospects for real-time result streaming, advanced predictive analytics, and interactive web-based dashboards to further improve student data management.

8. Future Enhancements

- Implement real-time streaming of student results using Apache Kafka for instant updates.
- Integrate Apache Sqoop into this.
- Develop an interactive web-based dashboard using Flask/Django for real-time visualization and querying.