Overview

This project details a scalable, end-to-end solution for real-time log analysis using Hadoop and Apache Spark. The main objective is to create a pipeline that can ingest, process, aggregate, and visualize various system, web, and application logs in near real-time. This provides instant operational insights and allows for anomaly detection, which aids in better monitoring, troubleshooting, and decision-making.

Problem Statement

Digital applications and systems continuously generate vast streams of log data. Traditional batch processing methods are inadequate for providing the immediate visibility required to respond quickly to operational issues. This project addresses this by:

- Ingesting live log data streams from multiple sources
- Processing and aggregating them instantly with Spark Streaming.
- Storing logs for both historical and real-time analytics.
- Displaying insights, anomalies, and trends through dashboards and visualizations.

Architecture & Workflow

The system follows a clear data flow from collection to visualization.

Data Flow Overview

- **Data Collection**: Logs are gathered from web, application, and system servers using tools like Flume or Kafka.
- **Streaming Ingestion**: Data is published to Kafka topics or Flume channels for reliable and scalable routing.
- **Real-Time Processing**: Spark Structured Streaming consumes, parses, filters, and aggregates logs within small time windows (e.g., 5–10 seconds).
- **Storage**: Raw and processed logs are stored in HDFS (Hadoop Distributed File System), partitioned by date and hour for efficiency. Aggregated summaries are written to Hive tables.
- **Visualization**: Metrics are visualized on dashboards (using Grafana, Kibana, or Power BI) to show trends, error spikes, and usage patterns.

System Data Pipeline

Stage	Tool(s)	Purpose
Collection	Kafka/Flume	Ingest log streams from sources
Processing	Spark Streaming	Transform, aggregate, and analyze
Storage	HDFS, Hive	Durable storage for logs/analytics
Visualization	Kibana/Grafana	Dashboards for real-time monitoring

---

Tools and Technologies

The project utilizes the following technologies:

- **Apache Kafka/Flume**: For high-throughput, fault-tolerant log ingestion and delivery.
- **Apache Spark Streaming (PySpark/Scala)**: For real-time processing and aggregation.
- **HDFS**: For reliable, distributed storage of logs.
- **Hive/Spark SQL**: For fast, ad-hoc querying and data organization.
- **Visualization Tools**: Kibana, Grafana, or Power BI for dashboards.
- **Python**: As the main language for the Spark program logic.

Implementation Steps

1. **Data Collection and Ingestion**: Logs are continuously pushed from servers and forwarded by Kafka/Flume agents to streaming topics.

2. **Spark Streaming Processing**: Spark reads data in batches, parses entries into structured fields (timestamp, IP, URL, status), applies filters (e.g., for status ≥ 500), and performs windowed aggregations to find top URLs, frequent IPs, and error rates.

3. **Data Storage**: All raw and aggregated logs are written to HDFS. Aggregates are saved as partitioned Hive tables for efficient historical analysis.

4. **Analysis and Detection**: The system performs frequency counts (top IPs, URLs, error distribution), uses time-windowed aggregations to detect spikes for alerting, and can optionally use Spark MLlib for anomaly detection.

5. **Visualization**: Data is fed from Hive or Elasticsearch into Kibana or Grafana dashboards to present traffic trends, top URLs, error rates, and alerts.

Code & Sample Outputs

Example: Real-Time Aggregation in PySpark

Python

```
from pyspark.sql.functions import window, col
# Streaming logs DataFrame: logs_df
# Windowed count of errors per 10-min window, sliding every 2 minutes
error_counts = logs_df.filter(col("status") >= 500) \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(window(col("timestamp"), "10 minutes", "2 minutes"), "url") \
    .count()
```

Example: Top URLs Visualization

Python

```
import matplotlib.pyplot as plt
top_urls = df.groupBy("url").count().orderBy("count", ascending=False).limit(10).toPandas()
plt.figure(figsize=(10,6))
plt.barh(top_urls['url'], top_urls['count'], color="#64cc85")
plt.xlabel("Requests")
plt.ylabel("URL")
plt.title("Top 10 Requested URLs")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

Example Output Table: Top Status Codes

status	count
200	62
404	14
500	7
401	9
301	2
...	...

Visualization of Results

- **Hourly Requests Bar Chart**: Shows periods of high and low activity.
- **Pie Chart of Top 10 Clients/IPs**: Reveals which users drive the most load.
- **Line Chart of Error Rate by Hour**: Makes error surges and system issues obvious.
- **Bar Chart of Requests by Status**: Visualizes reliability and load patterns.

Conclusion

The real-time log analysis platform enables enterprises to monitor system health in real time, rapidly detect and resolve anomalies, and make proactive data-driven decisions. It also allows for the cost-efficient storage of logs for deep historical analysis and visualizes trends for strategic planning. By using Hadoop and Spark, the system is scalable and provides low-latency analytics suitable for any log-rich environment, paving the way for future enhancements like machine learning and automated remediation.

Sayyad Chandan
24M11MC158
Aditya Univeristy