

Distributed Log Analyzer

1. Abstract

This project implements a distributed system designed to **collect, process, and analyze log data** in real-time using **Apache Kafka** for streaming and **Hadoop MapReduce** for batch processing.

The system demonstrates how modern distributed architectures can handle large-scale log data efficiently by decoupling data ingestion (Kafka) from storage and analysis (Hadoop).

It automatically stores log data from multiple producers into **HDFS**, where MapReduce jobs perform analytics, such as counting occurrences, detecting anomalies, or summarizing events.

The project provides a clear demonstration of how **Big Data pipelines** operate in real-world analytics environments.

2. Introduction

With the exponential growth of data generated by servers, applications, and IoT devices, organizations require scalable solutions to collect and analyze logs efficiently.

Traditional single-node systems fail to handle the high volume and velocity of data.

This project leverages:

- **Apache Kafka** as a distributed streaming platform to handle real-time ingestion of logs.
- **Hadoop HDFS** for distributed storage.
- **Hadoop Streaming with MapReduce** for large-scale batch analytics.

By integrating these tools, the system showcases a complete **end-to-end data flow** from real-time input to distributed analysis.

3. Objectives

- To simulate a distributed log data pipeline using **Kafka** and **Hadoop**.
- To store the collected logs in **HDFS** for persistence and scalability.
- To process and analyze these logs using **MapReduce** through a custom Python-based mapper and reducer.
- To demonstrate a reliable and automated workflow for continuous log ingestion and analysis.
- To ensure consistent and reproducible execution for testing and presentations.

4. System Architecture

Data Flow:

Producers (Servers) → Kafka (Streaming Layer) → HDFS (Storage) → MapReduce (Analysis)

Components:

1. **Kafka Producer:** Simulates multiple servers generating log messages.
 2. **Kafka Broker & Zookeeper:** Manage and coordinate log data streams.
 3. **Kafka Consumer:** Consumes log messages and uploads them automatically to HDFS.
 4. **HDFS (Hadoop Distributed File System):** Stores the incoming log data in a distributed manner.
 5. **Hadoop MapReduce (Streaming):** Executes Python mapper and reducer scripts to process and analyze logs.
-

5. Technologies Used

Component	Technology	Purpose
Data Streaming	Apache Kafka	Ingests and manages log data streams
Coordination Service	Apache Zookeeper	Coordinates Kafka brokers
Data Storage	Hadoop HDFS	Distributed and fault-tolerant data storage
Data Processing	Hadoop Streaming (MapReduce)	Processes large-scale log data using Python
Programming Language	Python	For Producer, Consumer, Mapper, and Reducer scripts
Operating System	Ubuntu (Linux VM)	Execution environment for all components

6. Workflow

Step 1: Start Zookeeper

```
hdoop@youssef-virtual-machine:~$ cd kafka
hdoop@youssef-virtual-machine:~/kafka$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

Starts the Zookeeper service, which manages configuration and synchronization between Kafka brokers.

Step 2: Start Kafka Broker

```
hdoop@youssef-virtual-machine:~$ cd kafka
hdoop@youssef-virtual-machine:~/kafka$ bin/kafka-server-start.sh config/server.properties
```

Starts the Kafka broker, which acts as the main message hub that receives log messages from producers and stores them in topics for consumers to read.

Step 3: Start Three Producer Nodes

```
hadoop@youssef-virtual-machine:~$ cd dis_logs
hadoop@youssef-virtual-machine:~/dis_logs$ python3 producer.py node1
```

```
hadoop@youssef-virtual-machine:~$ cd dis_logs
hadoop@youssef-virtual-machine:~/dis_logs$ python3 producer.py node2
```

```
hadoop@youssef-virtual-machine:~$ cd dis_logs
hadoop@youssef-virtual-machine:~/dis_logs$ python3 producer.py node3
```

Begins generating simulated log messages from three servers and sends them to the Kafka topic.

Step 4: Start Consumer

```
hadoop@youssef-virtual-machine:~/dis_logs$ python3 consumer.py
```

The consumer subscribes to the Kafka topic, continuously reads incoming messages, and uploads them automatically into the HDFS input directory (/user/hadoop/dis_logs/input).

Step 5: Run the Hadoop MapReduce Job

```
hadoop@youssef-virtual-machine:~/dis_logs$ hdfs dfs -rm -r -f /user/hadoop/dis_logs/output
$HADOOP_HOME/bin/hadoop jar "$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar" \
-files /home/hadoop/dis_logs/mapper.py,/home/hadoop/dis_logs/reducer.py \
-mapper "python3 mapper.py" \
-reducer "python3 reducer.py" \
-input /user/hadoop/dis_logs/input \
-output /user/hadoop/dis_logs/output
```

Runs the MapReduce job using Hadoop Streaming.

The **mapper** and **reducer** scripts written in Python process the input logs stored in HDFS.

The job outputs results (such as log counts or summaries) into the HDFS output directory.

Step 7: View the Results

```
hadoop@youssef-virtual-machine:~/dis_logs$ hdfs dfs -cat /user/hadoop/dis_logs/output/part-00000
Total Requests  201
Error Requests  104
Error Rate      51.74%

Requests per Node:
node1  106
node2   84
node3   11

Top 5 IPs by Requests:
172.16.0.3      25
10.0.0.5        24
192.168.3.4     24
10.0.2.8        22
172.16.2.14     22

HTTP Status Counts:

Top 3 Nodes by Errors:
node1  106
node2   84
node3   11

Average Requests per Node      67.00

Percent of Total Requests per Node:
node1  52.74%
node2  41.79%
node3   5.47%
```

7. Implementation Details

7.1 Kafka Producers

- Each producer represents a server node generating real-time log data.
- Logs are sent to a Kafka topic (e.g., `dis_logs`) for centralized streaming.

7.2 Kafka Consumer

- Continuously listens to the Kafka topic and writes messages into the HDFS input directory.
- Ensures that the Hadoop system always has updated log data for processing.

7.3 Hadoop MapReduce

- **Mapper:** Processes each log line (e.g., extracting timestamp, severity level, or component).
- **Reducer:** Aggregates or summarizes the results (e.g., counts logs per level or node).
- Executes in parallel across nodes for high performance.

7.4 Output

- Output is saved in `/user/hadoop/dis_logs/output/part-00000`.
 - Contains insights such as:
 - Total log entries per node.
 - Count of ERROR messages per node.
 - Top 5 IPs by Requests.
-

8. Results

After execution, the system successfully:

- Streamed logs from multiple simulated servers into Kafka.
 - Stored logs automatically into HDFS through the consumer.
 - Processed logs using Hadoop MapReduce to extract analytical summaries.
 - Demonstrated distributed and automated data flow for large-scale log analytics.
-

9. Conclusion

The **Distributed Log Analyzer** effectively showcases a modern big data pipeline integrating **real-time streaming** with **batch processing**.

It highlights how distributed technologies like **Kafka** and **Hadoop** can be combined to handle large volumes of data reliably and efficiently.

The system's modular design ensures:

- Scalability through distributed producers and parallel MapReduce processing.
- Automation via Kafka–HDFS integration.
- Consistency in repeated tests and demonstrations.

10. Future Improvements

- Integrate **Apache Spark** for faster, real-time analytics.
 - Build a **web interface** to visualize processed log data.
 - Add **machine learning models** to detect anomalies automatically.
 - Use **workflow schedulers** like Airflow to automate job execution.
 - Introduce **monitoring dashboards** for Kafka and Hadoop cluster performance.
-