Log to Columnar Converter

Project Summary:

Overview

The "Log to Columnar Converter" is a tool designed to efficiently convert large JSON-based log files into columnar format. The tool processes incoming log data in a streaming manner and writes the transformed data to separate column files. This project aims to provide a scalable and high-performance solution suitable for handling log files ranging from 4GB to 48GB.

Components

The project consists of several components:

- Producer Component (producer.py): This component reads an input log file, extracts log entries, and sends them to an Apache Kafka topic for real-time streaming.

- Consumer Component (consumer.py): The consumer subscribes to the Kafka topic, processes incoming log data, and writes the data to columnar files in the specified output directory.

- CLI Tool (log_to_columns.py): This Command Line Interface (CLI) tool leverages multiprocessing to parallelize the producer and consumer tasks.

- Streamlit User Interface (streamlit_app.py): A web-based user interface powered by Streamlit provides a user-friendly way to interact with the CLI tool and initiate log-to-columnar file conversion.

Usage

- Users can utilize the CLI tool or the Streamlit user interface to convert JSON-based log files to columnar format.
- The CLI tool accepts command-line arguments for specifying input, output, and initiates conversion. It also provides additional information like the time taken and the maximum RAM usage for conversion.
- The Streamlit interface allows users to upload log files, set output directory, and initiates the conversion process with a simple button click.
- Table demonstrating the performance using time and space complexity:

Filename	**Time taken**	**Max RAM usage**
dummy.log	4.14 seconds	15.35 MB

128MB.log	122.38 seconds	15.42 MB
256MB.log	203.63 seconds	15.61 MB
512MB.log	465.89 seconds	15.84 MB

```
(.kafka) aditya@LAPTOP-AEIAQV3T:/mnt/c/Users/Aditya/OneDrive/Desktop/Personal Projects/achiev
e.ai$ python log_to_columns.py "dummy.log" "dirr10"
Conversion took 4.14 seconds
Max. RAM Usage: 15.35 MB
(.kafka) aditya@LAPTOP-AEIAQV3T:/mnt/c/Users/Aditya/OneDrive/Desktop/Personal Projects/achiev
e.ai$ python log_to_columns.py "128MB.log" "dir11"
Conversion took 122.38 seconds
Max. RAM Usage: 15.42 MB
(.kafka) aditya@LAPTOP-AEIAQV3T:/mnt/c/Users/Aditya/OneDrive/Desktop/Personal Projects/achiev
e.ai$ python log_to_columns.py "256MB.log" "dir12"
Conversion took 203.63 seconds
Max. RAM Usage: 15.61 MB
(.kafka) aditya@LAPTOP-AEIAQV3T:/mnt/c/Users/Aditya/OneDrive/Desktop/Personal Projects/achiev
e.ai$ python log_to_columns.py "512MB.log" "dir13"\
>
Conversion took 465.89 seconds
Max. RAM Usage: 15.54 MB
```

Please keep in mind that calculating the time and space complexities could itself have added some overhead.

Benefits

- Efficiently processes large log files in real-time using Apache Kafka.
- Utilizes multi-processing for parallelism, enhancing performance.
- User-friendly Streamlit interface for easy interaction and control.
- Automatically clears the application state after a successful conversion for a fresh start.

Future Enhancements

- Extend support for streaming logs in real-time from various sources.
- Optimize performance for even larger log files. Currently only tested with file size up to 512 MB.
- Implement advanced error handling and robust log parsing.
- Add security features to protect data in transit.

Bottlenecks and Optimization Suggestions:

1. Limited Error Handling:
   - Bottleneck: The current implementation lacks advanced error handling for scenarios such as malformed JSON data, connection issues with Apache Kafka, or file I/O errors.
   - Optimization: Implement comprehensive error handling and logging to handle unexpected situations.
2. Security:
   - Bottleneck: The tool currently lacks security measures, which can be a concern when dealing with sensitive log data.
   - Optimization: Implement encryption for data in transit to secure log files.

Handling Edge Cases:

1. Fault Tolerance: Improve fault tolerance mechanisms to handle unexpected shutdowns and restarts.
2. Parallelism: Achieve further parallelism within the consumer component to process different portions of the log data in parallel.

The "Log to Columnar Converter" project aims to provide a scalable and high-performance solution for converting large JSON-based log files into a columnar format while supporting real-time processing and user-friendly interaction. This tool can be a valuable asset for organizations dealing with extensive log data.