ACHIEVE.AI

PROJECT DOCUMENTATION

Name: Alankrit Arya

College : Dwarkadas J Sanghvi College of Engineering

Problem Statement:

We usually use our log database to query our logs. But now and then, we may have to query older logs for customer support or debugging. In most of these cases, we know the time range for which we need to analyze the logs.

We need a tool that could extract the log lines from a given time range and print it to the console in a time-effective manner.

The command line (CLI) for the desired program is as below

`LogExtractor.exe -f "From Time" -t "To Time" -i "Log file directory location"` The time format will be in "ISO 8601".

The extraction process should complete in a few seconds, minimizing the engineer's wait time.

Approach:

I choose Golang to solve the problem statement that was given. This language consists of Go Routines which makes parallel processing easier in a simpler version of code. Since the problem defines that a huge amount of data will be given as an input, I thought that parallel processing will be perfect for optimizing the task given at hand.

I used the approach which linearly iterated over all the files to process them. This is done by considering one file at one point in time since the data stream cannot be stored in a single file and was stored in multiple files that was mentioned in the problem statement.

For Web App I established a persistent connection between the user and the server using websockets. WebSockets provide a bidirectional, full-duplex communications channel that operates over HTTP through a single TCP/IP socket connection. At its core, the WebSocket protocol facilitates message passing between a client and server.

Algorithm:

1. Input start and end time
2. Loop through the collection of files
3. For each file check the last log with the start and end time to check if the logs in the files are in the time frame that we are searching.
4. If the log lies in the time frame, process the file
   4.1. Divide the whole file into multiple chunks
   4.2. Run the code to process these chunks parallelly so that all these chunks get processed at the same time
   4.3. For each parallel execution, the log's timestamp is checked to verify if it lies between the start and end time
   4.4. If the conditions are met then the entry from the log file is displayed on the interface
5. If not, then move to the next file

Drawbacks:

- If the logical order of the logs is important then it is not achieved because parallel processing is done and hence any log can be displayed at any random time.
- Chunk size is ambiguous, if the chunk is too large then it burdens the processor and slows the parallel process while if the chunk size is too small then the parallel overhead increase

Solutions for drawbacks (Future Optimization):

- The solution for logical order is using synchronous algorithms on top of an asynchronous processor network
- An appropriate chunk size can be defined so that the chunk is not too large or too small.

Comparison Table:

File size	Total Allocation (Space Complexity)	Time Taken (Time Complexity)
9.72 MB	53 MiB	35.9061 ms
19.4 MB	153 MiB	73.7498 ms
29.1 MB	157 MiB	114.2355 ms
38.9 MB	230 MiB	153.5614 ms
43.7 MB	160 MiB	128.4251 ms

References:

[1] https://medium.com/swlh/processing-16gb-file-in-seconds-go-lang-3982c235dfa2

[2]https://socket.io/docs/v4/

[3]https://www.windows-commandline.com/how-to-create-large-dummy-file/

[4]https://www.botify.com/learn/basics/log-file-analysis

[5] https://www.educative.io/answers/what-is-a-goroutine