Eightfold Backend Challenge Approach and Suggestions

Assumptions:

1. Since there is a single server node, the number of processes running on the server will also be limited.
2. If processes are limited, the number of log files will also be limited, since it is given that each process has logs in a single file and each file holds logs of a single process.
3. Thread count will be finite around the range O(L) where L is at the most 1000.
4. Each log file is sorted based on timestamp. I found a set of logs which were not sorted based on the timestamp in file 1.log.Text

   Description automatically generated
5. Cannot read whole file into memory.
6. Certain threads have the start delimiter, but not the end delimiter for a particular process. So, for the Average and Std Deviation, I will ignore that particular thread interval.

Proposed Solution:

**Sanitizing the Log Files**

Maintain separate files for each user-request/thread.

Approach:

* Read 1st log from each file simultaneously and offer it to Priority Queue based on the Timestamp.
* While Priority Queue is not empty, poll the log.
* If a file exists for that thread ID of the log, append the log to the file or create a new File and append the log.
* Offer the next log to the Priority Queue from the Log file which was polled.

Advantages:

* Easier to read the logs for a particular user-request.
* Faster access to the logs for a particular user-request.
* In case one of the log file gets corrupted/deleted, the separate files for other user-requests will not be affected.

Limitations:

* Managing more files for logs.
* In case logs are to be deleted after a period of time, it will be difficult to purge all files together.

**Basic API**

Given the range of time (t1, t2), provide the log information of the threads active in this range.

Approach:

* Maintain a HashMap<threadId, HashMap<processId,List<Interval>>> where Interval contains the Start Time of the Thread and End Time of the Thread.
* Iterate over the HashMap for each thread, which contains a HashMap for each process which has a sorted list of Intervals.
* If the Start Time is before t1 and End Time is after T1, or Start time is between t1 and t2, the thread is active.
* Return the threadID, processID and the reference of the file which is the <ThreadID>.log.
* The complexity will be O(L\*N), where N is the maximum number of times a particular thread was spawned in all processed. L indicates the number of threads which is about 1000.

Advantages:

* HashMap lookup is faster.

**Bonus API**

Part 1:

Highest count of concurrent threads running in any second over all the log files inclusive over all the files.

Approach:

* Maintain a HashSet for threadIDs to check the threads active at a particular timestamp.
* As the log is polled from the PriorityQueue for sanitization, I will store the timestamp and add the threadID into my HashSet.
* Update the count and timestamp for maximum concurrent threads to store, in case the HashSet size is greater for another timestamp.

Advantages:

* Along with Sanitization, this has been preprocessed.
* Quick response time for the API.

Part 2:

Find the Average and Standard Deviation of all the threads lifetime for all the logs.

Approach:

* From the HashMap made for Basic API, iterate over all the entries.
* For each entry, we have all the start and end times for a particular thread.
* Can find the time a thread is alive and find the Average and Standard deviation in O(L\*n) where n is the maximum number of times a particular thread is spawned and L is the number of threads (size of the HashMap) which is about 1000.

Advantages:

* Along with the Basic API, I can preprocess to get the Average and the Standard Deviation.
* Quick response time for the API

**Code**

1. The src/main/java/yashdoshi/eightfoldchallenge contains all my java classes.
2. The SanitizeFiles.java is first executed to read the Log files and write them to a sanitized Folder.
3. The BasicAPI.java is to be then executed to run the Basic API query for a given standard input timestamp range. I have used the following timestamp range for running.

2020-08-09 18:59:21,000

2020-08-09 18:59:28,000

1. In the Resources/ folder, you will be able to see my Sanitized logs folder and the API’s response in log/text files.

**Future Scope**

1. Write Unit tests for each component of my code.
2. Handle more edge cases for the logs.
3. Ability to delete the sanitized logs, and rebuild them after a period of time to maintain the latest logs.