Performance Analysis Report

By Dinuka Yohan Sinhalage
Student ID: 24011006

Introduction

In modern applications, logging is crucial because it enables developers to track and address issues as they appear. However, it becomes crucial to comprehend how various logging systems manage this stress when applications are operating at high loads and producing massive volumes of log data. In this analysis, I evaluated the performance of various logging systems, including FileAppender, ConsoleAppender, and MemoAppender (which uses both ArrayList and LinkedList), in terms of memory usage, CPU activity, and overall speed.

To track memory, CPU, garbage collection (GC) activity, and thread behavior, the experiment generated a lot of log messages using VisualVM. I evaluated the performance of each component under prolonged stress by adjusting the appenders' maxSize and experimenting with two layouts: PatternLayout and VelocityLayout.

With maximum sizes changed from small to large, thousands of log entries were written to the various appenders in total. The effectiveness of each appender's memory management and log overflow handling was assessed by measuring performance both before and after the appenders reached their maximum size.

Key Results and Observations

The tests provided valuable insights into how each appender performed under load.

1. MemAppender with ArrayList and LinkedList:

ArrayList: Logging times were faster with this implementation, especially when adding new logs. However, because the array was frequently resized as new elements were added, it used more memory. Because of this, memory usage increased more quickly as log entries increased, particularly after the maxSize was reached.

LinkedList: on the other hand, managed memory more effectively but performed slower. The nature of linked lists meant that append times were a little longer, but memory usage stayed constant as more logs were produced. Because of this, it is a better choice for settings where memory efficiency is crucial.

2. ConsoleAppender:

When handling log outputs to the console, this appender was reliable. It used the CPU moderately, particularly when there were big log event bursts. ConsoleAppender's dependence on system I/O, however, limits its applicability in high-frequency logging scenarios where I/O may become a bottleneck.

3. FileAppender:

Writing to disk overhead caused FileAppender to operate more slowly. The time required to process and store logs was the trade-off, even though memory usage stayed constant. Although it is not advised for use cases requiring real-time or high-speed logging, this appender is ideal for use cases requiring persistent log storage.

4. PatternLayout vs VelocityLayout

When it came to formatting logs, PatternLayout outperformed VelocityLayout, which makes it better suited for situations where speed is crucial. Higher processing time is a drawback of VelocityLayout, despite its flexibility and ability to customize log formats. When a lot of log entries need to be processed quickly, this can add overhead.

Visual Analysis

1. Memory Usage:

The memory usage increased steadily during the tests as more log entries were created. MemAppender with ArrayList consumed more heap space as it resized itself when logs exceeded its capacity. LinkedList, on the other hand, showed more stable memory consumption. The memory spikes were especially noticeable after reaching the maximum size of the appenders.

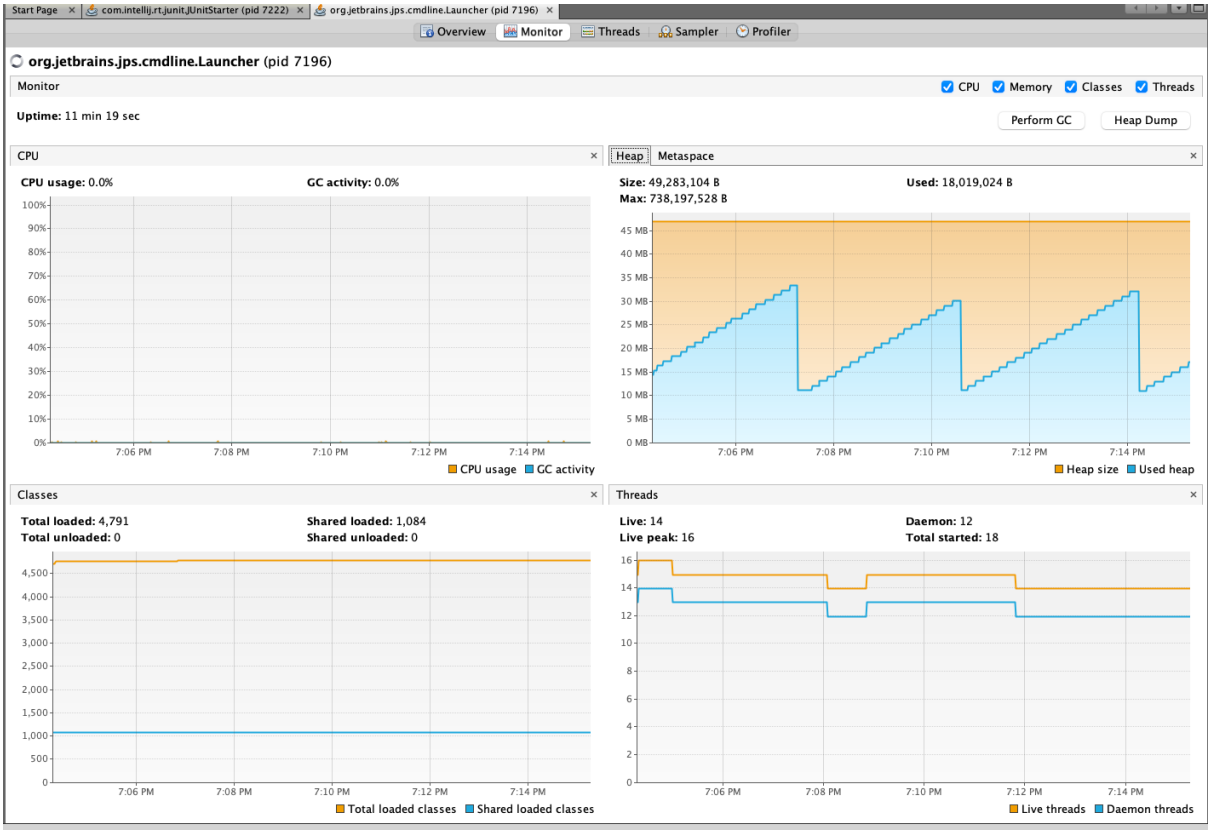

Figure 1: Initial Memory and Thread Usage during Log Generation

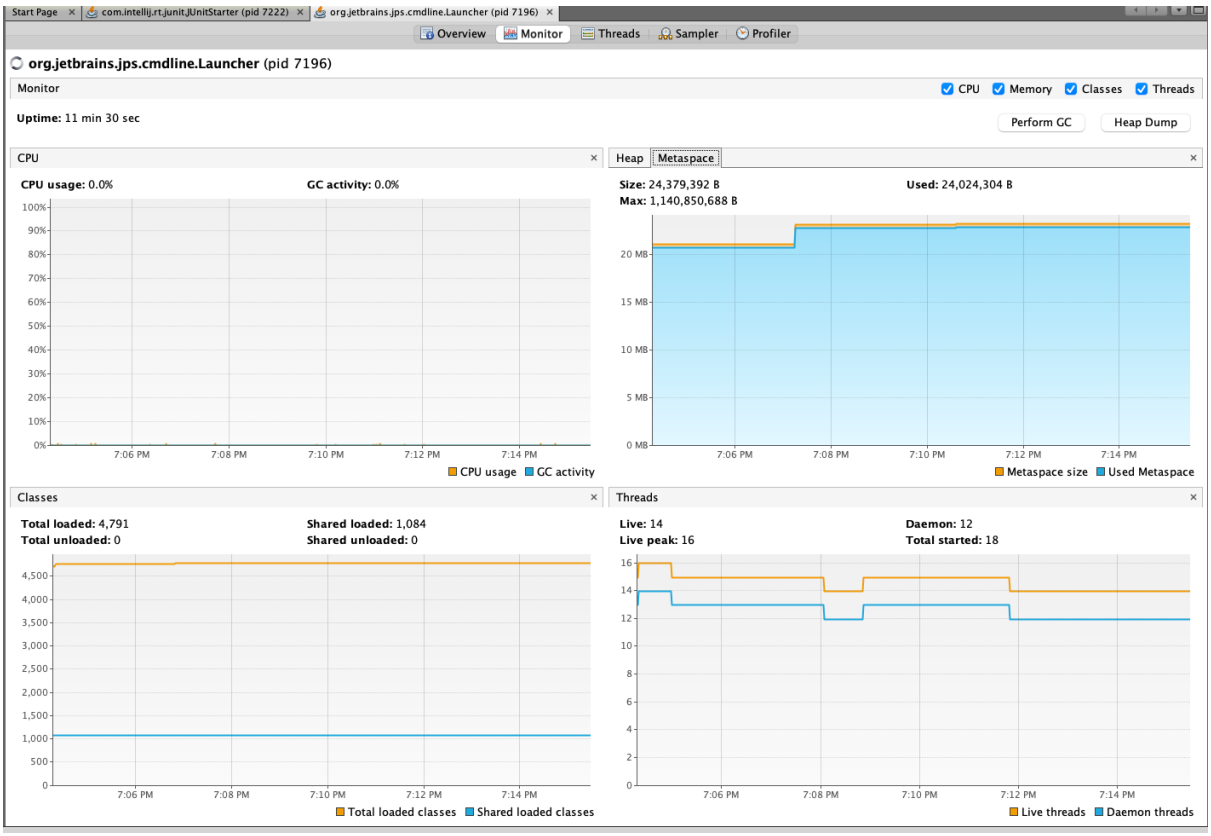

Figure 2: Memory Spike After MaxSize is Reached

2. CPU Usage:

CPU usage remained relatively low for all appenders, with ConsoleAppender and FileAppender experiencing moderate spikes during log bursts due to their reliance on I/O operations. MemAppender, on the other hand, demonstrated efficient CPU utilization even under stress.

3. GC Activity:

Garbage collection activity remained minimal throughout the tests, indicating that the JVM handled memory well even under heavy logging loads. This was particularly noticeable in the LinkedList-based MemAppender, where memory usage remained consistent, reducing the need for frequent GC.

Conclusion

The trade-offs between different appenders and layouts under high logging workloads are highlighted in this analysis. When combined with a LinkedList, which balances memory management and performance, MemAppender offers a versatile and effective way to handle massive volumes of logs. Although array lists can use more memory, they are better in situations requiring quick append operations.

In terms of formatting logs, PatternLayout proved to be the quicker choice; VelocityLayout, on the other hand, provides more flexibility at the expense of longer

processing times. It is advised to use PatternLayout in conjunction with MemAppender (LinkedList) for real-time logging requirements. Although FileAppender is helpful for persistent logs, it works best in situations with a moderate logging frequency and no disk I/O constraints.

All things considered, the tested logging solutions performed satisfactorily under stress. In high-performance logging environments, the findings of this study can help direct the selection of appenders and layouts.