CS131HW3 Report: Java shared memory performance races

Lyla Liu
UCLA

Abstract

This report evaluates the effects of synchronization strategies on Java application performance and reliability. It explores the implications of using synchronized and unsynchronized methods in shared memory contexts. The tests were conducted using modified versions of a Java application, UnsafeMemory, on SEASnet GNU/Linux servers, under Java version 22.0.1. The results provide insights into the trade-offs between execution speed and data consistency, offering a basis for deciding on synchronization strategies in multithreaded Java applications. The test cases are provided in the Appendix.

1 Introduction

In high-performance computing environments, synchronization overhead can significantly impact application speed. This study was initiated to investigate the potential benefits and drawbacks of removing the synchronized keyword in Java applications that use shared memory. The primary focus was to assess the impact on performance and reliability, with a secondary evaluation of Java's virtual threads feature.

2 Test Data

2.1 Thread Count

In the UnsynchronizedState, the thread count significantly influences performance and error rates. With a single thread, the performance is optimal due to the absence of synchronization overhead, recording a swift total real time of 1.35551 seconds for 100 million swaps. However, as threads increase, performance deteriorates sharply, and errors such as sum mismatches become prevalent, evidenced by a total real time of 3.65362 seconds with eight threads. This increase in threads leads to data races, underlining the necessity for synchronization in multithreaded environments.

The SynchronizedState demonstrates a clear degradation in performance as the number of threads increases. This is due to the increased synchronization overhead and lock contention among multiple threads. For example, average real swap times escalate from 33.6616 nanoseconds with one thread to 1016.25 nanoseconds with thirty threads, illustrating the significant slowdown caused by the synchronized keyword in high-concurrency settings.

2.2 State Array Size

For UnsynchronizedState, as the array size increases, from small (size 5) to medium (size 50) and large (size 100), the complexity of managing concurrent accesses without synchronization leads to increased performance degradation and error rates. For example, larger arrays show a marked increase in total real time and a proportional rise in data inconsistencies like sum mismatches, highlighting the growing challenge of handling more elements that can be accessed concurrently. This demonstrates that larger data structures significantly exacerbate race conditions, stressing the necessity for robust synchronization mechanisms to maintain data integrity and system stability.

Increasing the state array size in the SynchronizedState directly impacts performance. Larger arrays require more complex synchronization, which significantly extends access and operation times. This relationship is evidenced by the jump in total real time from almost negligible (0.00121585 seconds for an array of one) to several seconds (4.16544 seconds for an array of fifty), highlighting the increased demand on synchronization mechanisms as array size grows.

2.3 Virtual vs. Platform Threads

Comparative performance analysis between virtual and platform threads in the Unsynchronized State reveals negligible differences. This outcome suggests that the advantages of virtual threads, which are designed to be lightweight and managed efficiently by the JVM, are negated by the absence of effective synchronization. Both thread types show similar performance metrics, such as total real time and average swap time, indicating that the efficiency gains expected from virtual threads do not materialize in unsynchronized environments.

For SynchronizedState, data indicates that virtual threads experience higher average swap times, suggesting they incur additional overhead. For example, in tests with 100 million swaps, platform threads averaged 260 nanoseconds per swap, while virtual threads were slower at 340 nanoseconds. This supports the view that platform threads, benefiting from direct OS-level management, are more efficient in scenarios requiring robust synchronization, effectively leveraging underlying system resources with less performance penalty compared to the more abstracted virtual threads.

2.4 NullState

In the NullState, which performs no operations on the state array, performance remains consistent regardless of thread count or state array size, highlighting that the observed overhead is primarily due to thread management. This consistency is evident as changes in the number of threads or the size of the state array do not significantly impact performance metrics, demonstrating the minimal baseline overhead involved. Additionally, there are negligible differences in performance between virtual and platform threads in the Null State, indicating that without data manipulation, the type of thread management strategy has little effect on overall performance.

3 Problem Encountered

During the performance testing of the 'UnsafeMemory' application, variability in outcomes increased with higher thread counts, primarily due to greater resource contention and synchronization challenges. As threads competed for CPU and memory, issues like context switching and cache misses also contributed to instability. Strategies such as optimizing thread count, employing finer-grained locking or lock-free structures, and adjusting system settings like processor affinity proved effective in stabilizing performance. Additionally, an implementation error occurred when incorporating the 'UnsynchronizedState' class due to incorrect command line argument referencing ('args[1]' instead of 'args[2]'), leading to an 'ArrayIndexOutOfBoundsException'. This was corrected by adjusting the instantiation line, which resolved the error and ensured correct processing of user inputs. Together, these solutions enhanced both the stability and efficiency of the application in high-concurrency environments.

4 Conclusion

The UnsynchronizedState class was implemented by removing the synchronized keyword, allowing for concurrent modifications without explicit synchronization. This approach was expected to increase throughput by reducing the overhead caused by lock contention. The SynchronizedState class uses Java's synchronized keyword to ensure that updates to the shared memory structure are thread-safe. This traditional method of synchronization guarantees that the operations are atomic, preventing race conditions and ensuring data consistency. The findings confirm that removing the synchronized keyword can significantly improve the performance of Java applications by reducing synchronization overhead. However, this comes at a considerable cost to the reliability of the application. For applications where data consistency is critical, the use of traditional synchronization methods remains essential. Conversely, in environments where speed is prioritized over absolute accuracy, such as certain types of real-time data processing, using an unsynchronized approach may be acceptable.

5 Appendix: Tests

5.1 UnsynchronizedState

time timeout 3600 java UnsafeMemory Platform 5 Unsynchronized 1 100000000 Total real time 1.35551 s Average real swap time 13.5551 ns real 0m1.467s user 0m1.445s sys 0m0.036s

time timeout 3600 java UnsafeMemory Platform 5 Unsynchronized 8 100000000 Total real time 3.65362 s Average real swap time 292.290 ns output sum mismatch (-9356 != 0) real 0m3.769s user 0m14.394s sys 0m0.054s

time timeout 3600 java UnsafeMemory Platform 1 Unsynchronized 1 100000000 Total real time 0.000271110 s Average real swap time 0.00271110 ns real 0m0.123s user 0m0.102s sys 0m0.038s

time timeout 3600 java UnsafeMemory Platform 5 Unsynchronized 1 1000000 Total real time 0.0280298 s Average real swap time 28.0298 ns real 0m0.150s user 0m0.147s sys 0m0.038s

time timeout 3600 java UnsafeMemory Virtual 5 Unsynchronized 1 100000000 Total real time 1.33337 s Average real swap time 13.3337 ns real 0m1.453s user 0m1.421s sys 0m0.042s

5.2 SynchronizedState

time timeout 3600 java UnsafeMemory Platform 5 Synchronized 1 100000000 Total real time 3.36616 s Average real swap time 33.6616 ns real 0m3.476s user 0m3.443s sys 0m0.040s

time timeout 3600 java UnsafeMemory Platform 5 Synchronized 10 100000000 Total real time 3.53923 s Average real swap time 353.923 ns real 0m3.648s user 0m4.197s sys 0m0.108s

time timeout 3600 java UnsafeMemory Platform 5 Synchronized 30 100000000 Total real time 3.38750 s Average real swap time 1016.25 ns real 0m3.499s user 0m3.992s sys 0m0.123s

time timeout 3600 java UnsafeMemory Platform 1 Synchronized 8 100000000 Total real time 0.00121585 s Average real swap time 0.0972676 ns real 0m0.125s user 0m0.104s sys 0m0.040s

time timeout 3600 java UnsafeMemory Platform 50 Synchronized 8 100000000 Total real time 4.16544 s Average real swap time 333.236 ns real 0m4.280s user 0m5.859s sys 0m0.216s

time timeout 3600 java UnsafeMemory Platform 100 Synchronized 8 100000000 Total real time 3.36596 s Average real swap time 269.277 ns real 0m3.476s user 0m3.910s sys 0m0.114s

time timeout 3600 java UnsafeMemory Platform 5 Synchronized 8 1000 Total real time 0.00338503 s Average real swap time 27080.2 ns real 0m0.116s user 0m0.102s sys 0m0.041s

time timeout 3600 java UnsafeMemory Platform 5 Synchronized 8 1000000 Total real time 0.0921891 s Average real swap time 737.512 ns real 0m0.205s user 0m0.266s sys 0m0.048s

time timeout 3600 java UnsafeMemory Platform 5 Synchronized 8 1000000000 Total real time 32.8796 s Average real swap time 263.037 ns real 0m32.994s user 0m36.221s sys 0m0.640s

time timeout 3600 java UnsafeMemory Virtual 5 Synchronized 8 100000000 Total real time 10.6617 s Average real swap time 852.933 ns real 0m10.787s user 0m24.059s sys 0m2.920s

5.3 NullState

time timeout 3600 java UnsafeMemory Platform 50 Null 8 100000000 Total real time 0.524498 s Average real swap time 41.9599 ns real 0m0.639s user 0m1.949s sys 0m0.029s

time timeout 3600 java UnsafeMemory Platform 5 Null 8 100000000 Total real time 0.424476 s Average real swap time 33.9580 ns real 0m0.540s user 0m1.733s sys 0m0.047s

time timeout 3600 java UnsafeMemory Platform 5 Null 40 100000000 Total real time 0.509644 s Average real swap time 203.858 ns real 0m0.620s user 0m1.914s sys 0m0.046s

time timeout 3600 java UnsafeMemory Platform 5 Null 8 1000000 Total real time 0.113403 s Average real swap time 907.220 ns real 0m0.236s user 0m0.524s sys 0m0.046s