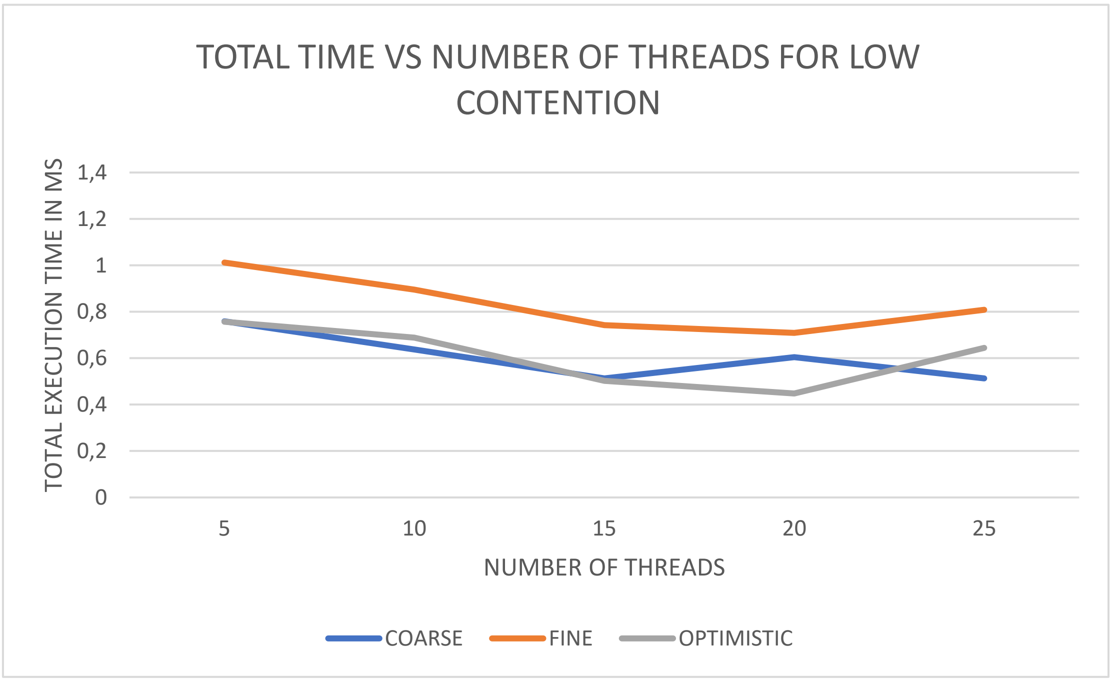
226 PRACTICAL 6

BAHIYA HOOSEN

U22598546

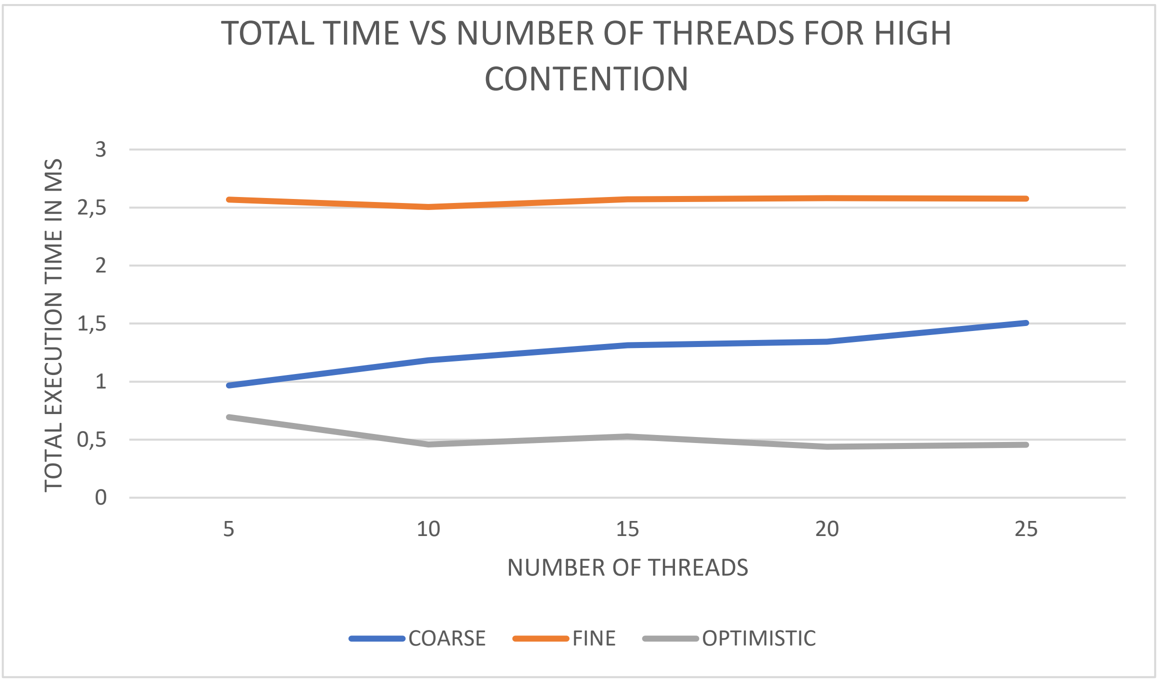
INTRODUCTION

Once upon a time in the land of concurrency, threads frolicked joyfully while battling the notorious monsters of contention. The brave programmers wielded fine-grained locks for precision but often tripped over deadlocks, while their coarse-grained friends got stuck waiting at the castle gate. Then came the optimistic souls, charging in without locks, only to flee from pesky retries! In the end, each approach had its quirks, and the villagers learned that laughter was the best synchronization strategy. And so, they lived happily ever after… until the next update!



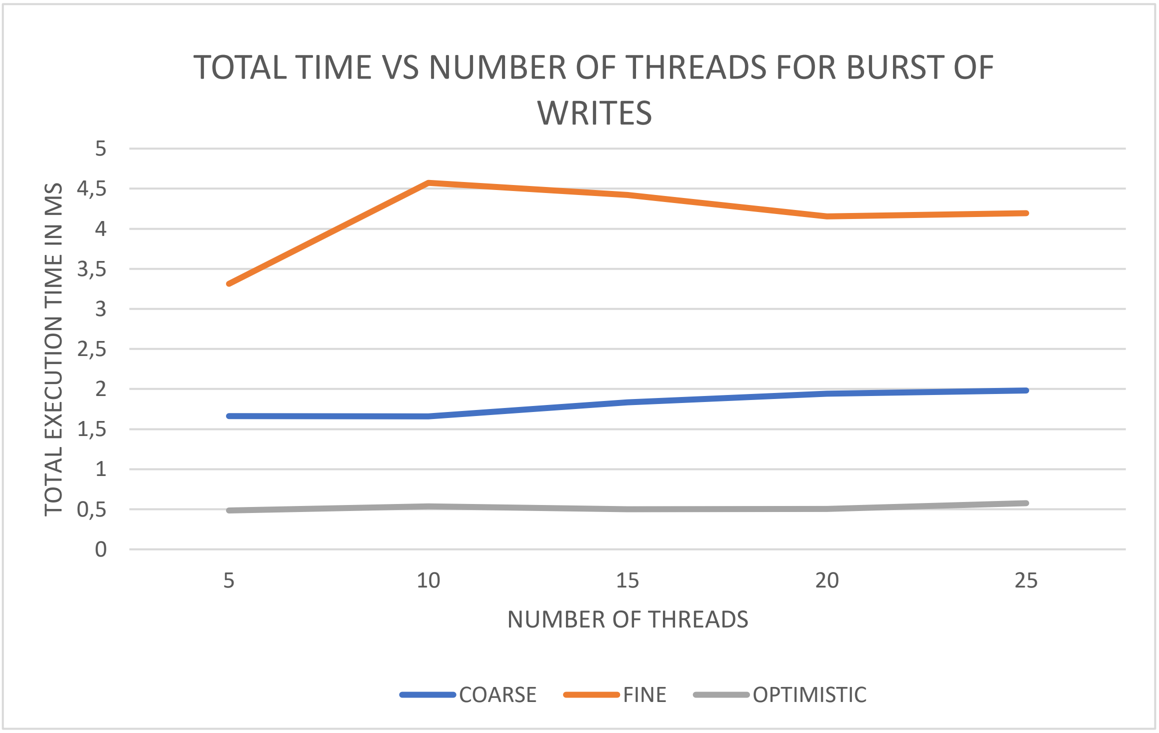
TEST 1 : LOW CONTENTION

- COARSE : consistently lower execution time compared to fine-grained synchronization. Beneficial in low contention scenarios due to its simplicity and reduced overhead. However, its performance does not scale as effectively when the number of threads increases significantly.
- FINE: highest across all thread counts. The high execution time in fine-grained synchronization can be attributed to the overhead of managing multiple locks, which can lead to increased contention and waiting time, particularly in scenarios where thread counts are lower.
- OPTIMISTIC : handle contention by allowing threads to proceed without locking until a conflict is detected. This reduces waiting times under low contention scenarios and can outperform both coarse-grained and fine-grained approaches.



TEST 2 : HIGH CONTENTION

- COARSE : The increase in execution time indicates that while fewer locks simplify the lock management process, they still lead to contention as more threads attempt to access the same critical sections. Performs better than fine-grained synchronization because it involves fewer context switches and lower overhead.
- FINE : In high contention scenarios, the fine-grained approach leads to excessive locking and unlocking, causing threads to spend a lot of time waiting for access to the critical sections leading to the worst synchronization for high contention.
- OPTIMISTIC : Lowest execution times across all thread counts, with the best performance. Allows threads to execute their operations without immediate locks, reducing waiting time and improving throughput in high contention scenarios.



TEST 3 : BURST OF WRITE OPERATIONS

- COARSE : while it may limit concurrency, incurs less overhead than fine because it uses a single lock for the entire data structure. it can create contention if many threads are attempting to access the resource simultaneously, leading to higher wait times. Despite this, the simplicity of the implementation results in relatively stable performance with fewer variations in execution time.
- FINE : This increased time indicates the overhead associated with managing multiple locks, which can lead to contention when threads compete for access.
- OPTIMISTIC : most effective strategy in this test scenario due to its ability to minimize blocking and manage high rates of concurrent operations efficiently. This non-blocking approach minimizes contention by allowing multiple threads to operate simultaneously, which is particularly advantageous in high-throughput scenarios.

CONCLUSIONS

- COARSE: Shows decent performance in low contention but experiences significant drops in throughput as contention increases.The waiting time and execution time are higher in high contention scenarios compared to fine-grained locking and optimistic locking.
- FINE : Performs reasonably well in low contention scenarios but starts to show some decline in high contention.Average waiting times increase as contention rises, indicating some overhead in managing multiple locks.
- OPTIMISTIC : Shows the best throughput across all tested scenarios, especially in low contention.Even under high contention, it maintains a competitive throughput compared to others.

REFERENCES

- GeeksforGeeks: Synchronization in Java - A comprehensive overview of synchronization mechanisms in Java- https://www.geeksforgeeks.org/synchronization-in-java/
- Java Concurrency Tutorial - An introduction to concurrency in Java, explaining various concepts including synchronization-https://www.youtube.com/watch?v=yKmkYE4QDrM
- "The Art of Multiprocessor Programming" by Maurice Herlihy and Nir Shavit

REPORT RECIPE

Materials Needed:

- A cup of knowledge (brewed from a lifetime of sleepless nights).
- A dash of fun (preferably in the form of dad jokes).
- A sprinkle of caffeine (to fuel those "Aha!" moments).
- A handful of snacks (for when the ideas start to dry up).
- A pinch of procrastination (to ensure maximum creativity under pressure).

Steps to Produce the Report:

1. Mix knowledge and fun in a bowl, add caffeine, and watch the magic happen.
2. Toss in snacks when creativity wanes—snack power is real!
3. Stir vigorously while avoiding procrastination, but remember: good ideas often come while scrolling cat videos.
4. Bake at 350°F until golden brown (or until you remember the deadline).
5. Serve hot, garnished with a witty conclusion!