COS 226  
Practical 6  
Milan Kruger  
  
Fine-Grained, Coarse-Grained, Optimistic. Long ago, the three synchronization strategies lived together in contention. Then, everything changed when I got a thread exception. Only ChatGPT, master of C++, could stop them, but when I needed it most, it failed to compile.

A few lines of code later, I discovered a bug, and although my debugging skills are great, I still have a lot to learn before I can safely navigate multi-threaded programming. But I believe, with ChatGPT’s help, I can solve this.

Scenarios

I did 3 scenarios with a sample size of 1000  
Scenario 1:

High Contention

Coarse-grained performs consistently better than the rest as the number of threads increased, indicating less overhead in high-contention scenarios.

Fine-grained performed well initially, but as the thread amount increased so did the overhead for managing so many locks under contention.

Optimistic performed similarly to fine but did better at the higher number of threads.

Scenario 2:

Read Heavy

Coarse-grained performs poorly with increased threads, showing a significant rise in time as thread count grows. This is probably due to coarse blocking all reads except for one thread. Thus, limiting concurrent reads.

Fine-grained remains stable across all thread counts, this is due to having multiple locks being accessible thus better concurrent reads.

Optimistic performed the best, this suggests it efficiently handling read heavy scenarios by allowing more reads without locking.

Scenario 3:

Mixed Workload

Coarse grained performs the worst, with a significant increase as the number of threads increase. This is due to blocking all operations, including the reads when a thread accesses the set.

Fine grained, performs better than coarse showing a more gradual increase in time. It allows more parallel operations reducing contention but still having a lot of overhead.

Optimistic performed the best, This suggests that optimistic locking effectively managing concurrent reads and writes operations minimizing locking and thus reducing overhead.

Conclusion

Coarse grained

Good: Simple to implement and efficient for low contention or atomic operations.

Bad: High contention scenarios will generally be worse since the entire set is locked reducing concurrency.

Fine grained

Good: Better than coarse in high contention, as it allows more parallel operations.

Bad: Can still suffer from overhead due to having to manage so many locks.

Optimistic

Good: Performs best in High read and mixed workloads. This is due to allowing most operations without locking

Bad: Can be less efficient in write environments where you need to check each write.

References  
  
<https://www.baeldung.com/java-map-computeifabsent>  
<https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)>

<https://link.springer.com/chapter/10.1007/978-3-031-01738-4_3#:~:text=Coarse%2DGrain%20Multithreading%20(CGMT),different%20thread%20without%20software%20intervention>.

<https://link.springer.com/chapter/10.1007/978-3-031-01738-4_4#:~:text=A%20Fine%2DGrain%20Multithreading%20architecture,instruction%20in%20a%20particular%20cycle>.

<https://en.wikipedia.org/wiki/Optimistic_concurrency_control>

<https://systemdesignschool.io/blog/optimistic-locking>

Recipe for this report

Ingredients

1. 1 cup of code plagiarized
2. 2 tablespoons of dreams dying
3. A dash of redbull
4. 3 cups of google searches
5. 1 pinch of panic
6. 1 teaspoon of procrastination
7. 1 slice of deadline pressure

Instructions

1. Preheat your chatgpt
2. Get a bowl
3. Add the redbull
4. Chop the google searches
5. PANIC!!!!!
6. Add procrastination and let it rest
7. Remember deadline and remember your childhood dreams of getting a girlfriend and getting a successful career.
8. Remember COS221, cry about dreams again
9. Fold because of deadline
10. Sleep
11. Enjoy!