CS 474
Davery Valdez

Project 4: Analysis Report

Overview:

This report evaluates the performance of three synchronization methods (Global Lock, Two-Lock, and Node-Level Lock) used for concurrent insertion into a sorted linked list. The performance was measured by varying the number of threads (1, 2, 4, 8, and 16) and analyzing the time taken for node insertions. The primary goal was to assess the scalability and efficiency of these methods under multithreaded conditions.

Analysis:

Global Lock:

Description:

•Uses a single lock for the entire linked list. All threads must acquire the same lock for any insertion operation, ensuring only one thread accesses the list at a time.

•Performance:

•The execution time increases as the number of threads increases due to contention for the single lock. For instance:

•At 1 thread: 0.000099 seconds

•At 16 threads: 0.000419 seconds

•This behavior demonstrates poor scalability, as more threads lead to higher contention and serialization.

•Use Case:

•Suitable for low-thread environments where contention is minimal and simplicity is preferred.

Two-Lock:

Description:

•Divides the linked list into two logical sections, each protected by a separate lock. Threads access one of the two locks based on the value being inserted.

Performance:

•Performs better than Global Lock in most cases, particularly as thread count increases:

•At 1 thread: 0.000067 seconds

•At 16 threads: 0.000295 seconds

•However, contention can still occur when multiple threads target the same section, particularly for evenly distributed values.

Scalability:

•Moderately scalable; performance begins to degrade at high thread counts when more threads contend for the same lock.

Use Case:

•Useful for moderate-thread environments with predictable data distribution.

Node-Level Lock:

Description:

•Each node in the linked list has its own lock. Threads lock only the nodes they interact with, allowing high concurrency.

Performance:

•Achieves the best scalability and efficiency across thread counts:

•At 1 thread: 0.000063 seconds

•At 16 threads: 0.000303 seconds

•Fine-grained locking minimizes contention and allows threads to operate on different parts of the list concurrently.

Scalability:

•Highly scalable due to reduced contention. Performance remains stable even at 16 threads.

Use Case:

•Ideal for high-thread environments where insertions are distributed across the list.

Graph Analysis

•The performance graph highlights the differences between the three methods:

•Global Lock: Shows the steepest increase in execution time as thread count grows.

•Two-Lock: Performance is better than Global Lock but still shows contention at high thread counts.

•Node-Level Lock: Maintains relatively stable performance and outperforms the other methods in most scenarios.

Nature of Node Insertions

•Uniform Node Distribution:

-For evenly distributed nodes (e.g., {1, 2, 4, 8, 16, 32}), Node-Level Lock is most effective as threads operate independently.

•Skewed Node Distribution:

-For skewed distributions (e.g., many nodes in the same range), Two-Lock and Node-Level Lock may still experience some contention, but Node-Level Lock handles this better due to finer granularity.

Conclusions

1. **Scalability:**
   •Node-Level Lock is the most scalable method, as it minimizes contention and allows high concurrency.
   •Global Lock suffers from contention and poor scalability with increasing threads.
   •Two-Lock offers moderate scalability, depending on the data distribution.
2. **Efficiency:**
   •Node-Level Lock consistently performs best across all thread counts.
   •Two-Lock provides better performance than Global Lock but has limitations at higher thread counts.
3. **Recommendations:**
   •Use Node-Level Lock for high-thread environments or when scalability is critical.
   •Use Two-Lock for moderate-thread environments where simplicity and reduced contention are acceptable.
   •Use Global Lock only in low-thread environments or for quick, simple implementations.