Tiyiso Hlungwani u21598127

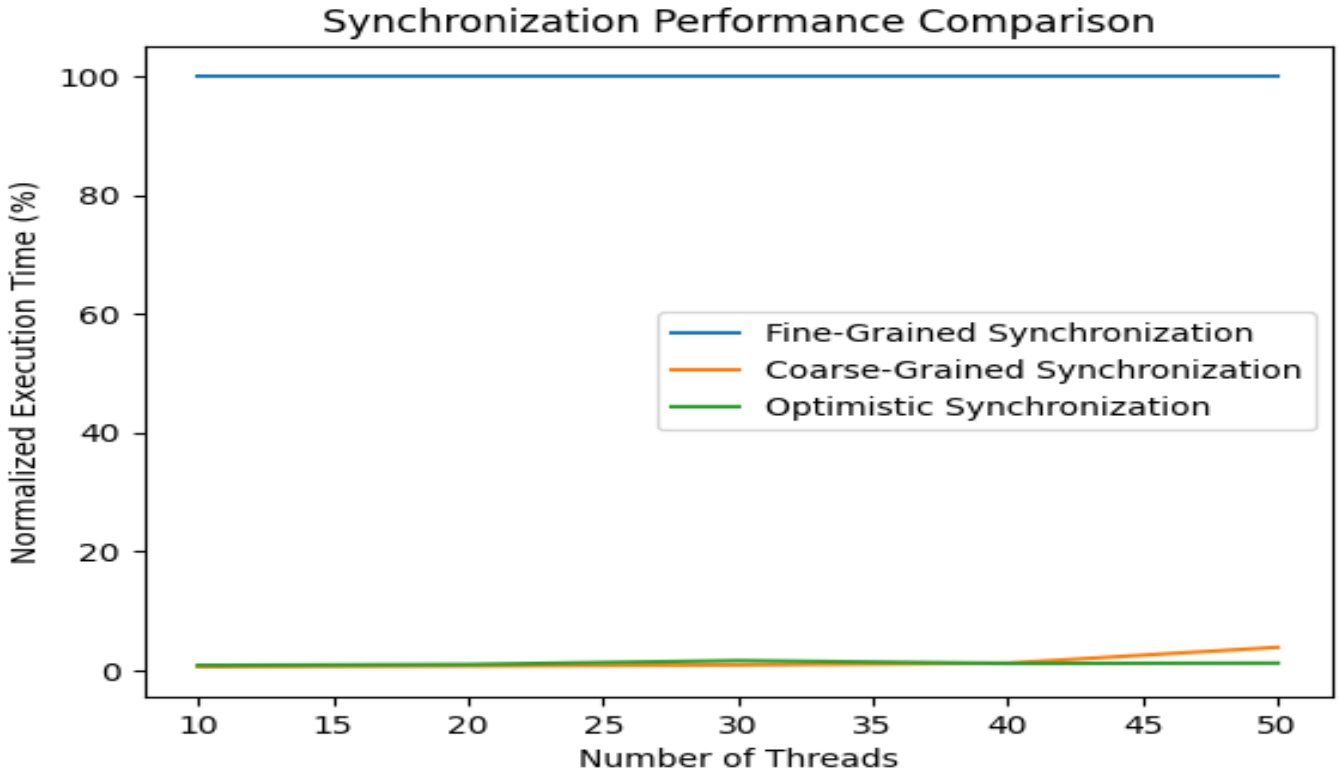
The Tale of Synchronization: A Journey Through Time and Threads

Once upon a time in the land of Computation, three brave synchronization methods set out on a quest to conquer the challenges of multi-threading: Fine-Grained Synchronization, Coarse-Grained Synchronization, and Optimistic Synchronization. Each had their unique strengths and weaknesses, and they were eager to prove their worth.

The Brave Warriors

1. **Fine-Grained Synchronization**: Known for its meticulous attention to detail, this method used multiple locks to ensure that each part of the data structure was protected. It was precise but sometimes slow due to the overhead of managing many locks.
2. **Coarse-Grained Synchronization**: This method preferred simplicity, using a single lock to guard the entire data structure. It was straightforward and fast for small numbers of threads but could become a bottleneck as the number of threads increased.
3. **Optimistic Synchronization**: The most hopeful of the three, this method assumed that conflicts were rare. It used validation to check if its optimistic assumptions held true, retrying operations if necessary. It was efficient in low-contention scenarios but could struggle under high contention.

The Quest: The three methods embarked on a series of trials, each facing different numbers of threads to test their mettle. Here are the results of their trials:



Scenarios: 10, 20, 30, 40, 50 Threads

The Findings

1. **Fine-Grained Synchronization**: As the number of threads increased, Fine-Grained Synchronization struggled, taking a significant amount of time (3563 ms) with 50 threads. The overhead of managing many locks became apparent.
2. **Coarse-Grained Synchronization**: This method performed well with fewer threads, but as the number of threads increased, the single lock became a bottleneck. The execution time increased from 20 ms with 10 threads to 137 ms with 50 threads.
3. **Optimistic Synchronization**: Optimistic Synchronization showed a balanced performance. It handled up to 50 threads with relatively stable execution times, peaking at 57 ms with 30 threads but managing to stay around 40-42 ms with higher thread counts.

The Trends

- **Fine-Grained Synchronization**: The trend showed that while fine-grained locking can be very precise, it incurs a high overhead with many threads due to the complexity of managing multiple locks.
- **Coarse-Grained Synchronization**: This method is efficient with fewer threads but becomes less effective as contention increases, leading to longer wait times for the single lock.
- **Optimistic Synchronization**: Optimistic Synchronization performs well in scenarios with low to moderate contention, maintaining stable execution times even with higher thread counts.

The Conclusions

- **Fine-Grained Synchronization**: Best used in scenarios where precision is critical, and the number of threads is relatively low.
- **Coarse-Grained Synchronization**: Suitable for simple applications with a small number of threads where ease of implementation is a priority.
- **Optimistic Synchronization**: Ideal for scenarios with low to moderate contention, where the overhead of retries is outweighed by the benefits of fewer locks.

The Recipe for This Report

Materials Needed:

- A cup of knowledge about synchronization methods
- A dash of fun and creativity
- A sprinkle of performance testing
- A pinch of graphing skills

Steps to Produce This Report:

1. Gather information about different synchronization methods.
2. Implement the methods in a programming language (Java in this case).
3. Conduct performance tests with varying numbers of threads.
4. Collect and analyze the results.
5. Create graphs to visualize the performance trends.
6. Write a story-like introduction to make the report engaging.
7. Explain the trends and provide reasons for the observed performance.
8. Conclude with recommendations on when to use each synchronization method.
9. Compile the report with all the findings and visualizations.

References

- Herlihy, M., & Shavit, N. (2012). *The Art of Multiprocessor Programming*. Morgan Kaufmann.