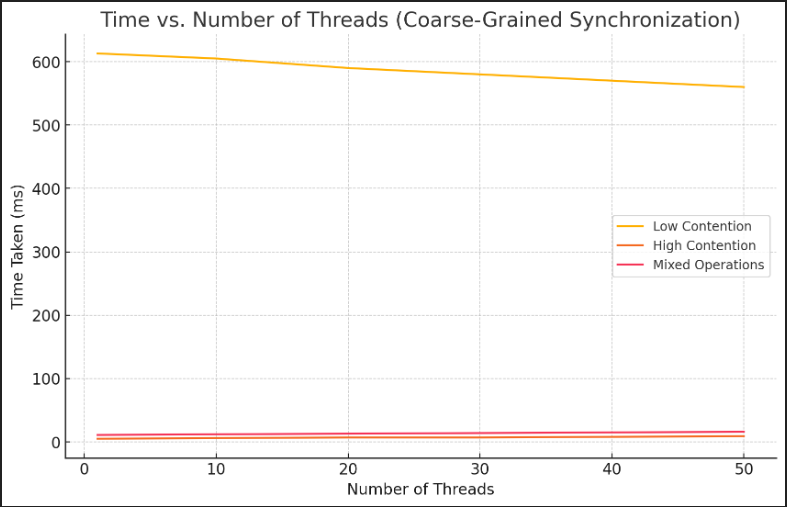
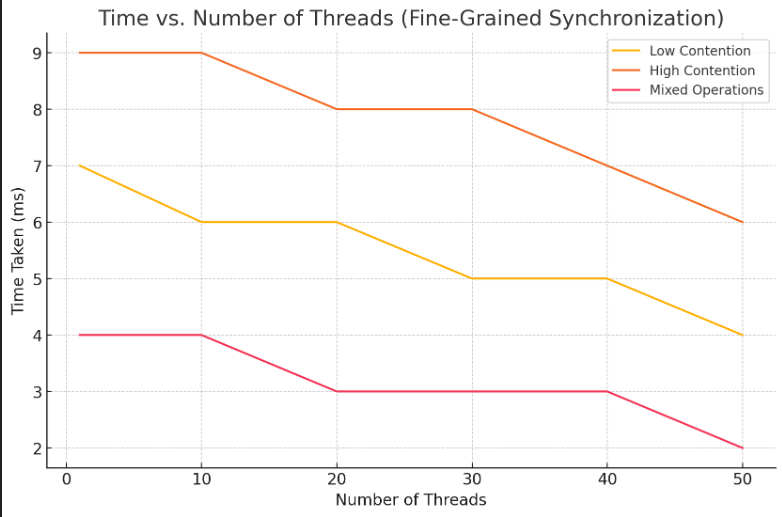
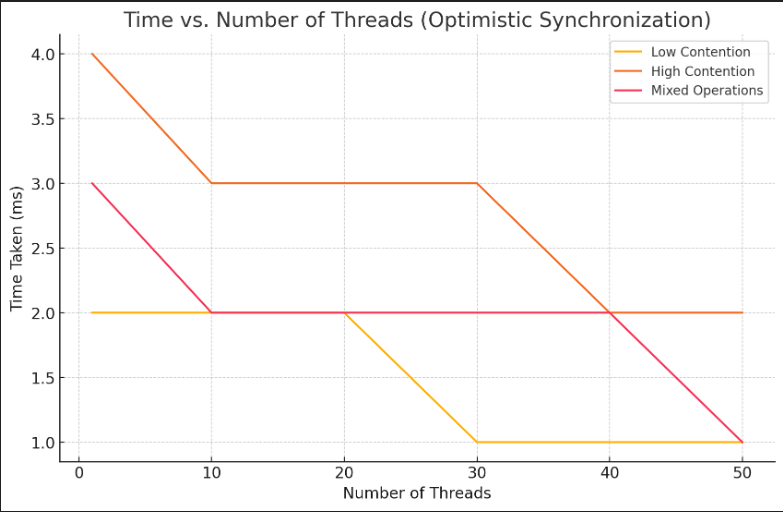
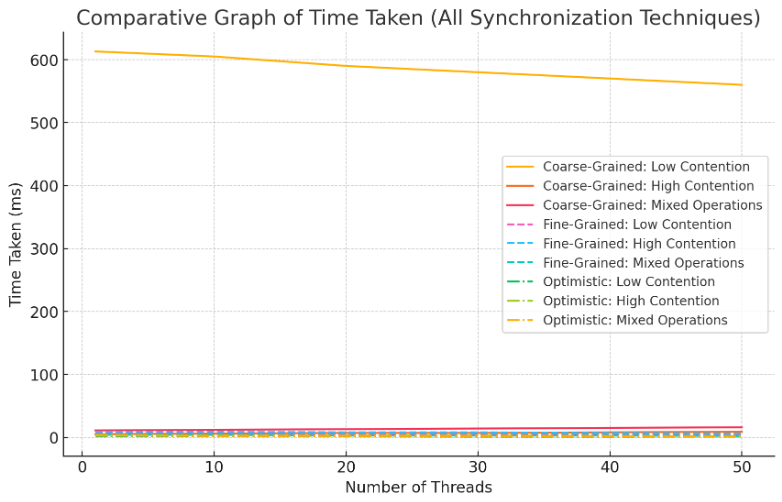
COS226 : Practical 6

Marco Geral – u23584565

Once upon a time, in the world of computer science, there lived three noble synchronization techniques: Coarse-Grained, Fine-Grained, and Optimistic. Each of them was tasked with guarding a mighty treasure—the Shared List—against the unruly chaos of multithreading. As more threads came to test their strength, each synchronization technique demonstrated its unique abilities in the face of both peace and contention. This tale is about their journeys through three great trials: Low Contention, High Contention, and Mixed Operations, and what we learned from their victories and defeats.

**Graphs:**

1. **Time vs. Number of Threads (Coarse-Grained Synchronization)**
2. **Time vs. Number of Threads (Fine-Grained Synchronization)**
3. **Time vs. Number of Threads (Optimistic Synchronization)**
4. **Comparative Graph of Time Taken (All Synchronization Techniques)**

**Explanation of Trends:**

* **Scenario 1 (Low Contention, Many Threads):**  
  In this scenario, **Coarse-Grained Synchronization** took the longest time (613 ms), while **Fine-Grained Synchronization** (7 ms) and **Optimistic Synchronization** (2 ms) were much faster. The high time for coarse-grained locking is because every thread, even when working on separate parts of the list, has to wait for the single lock. Fine-grained locking performs better since threads can lock different parts of the list, and optimistic synchronization avoids locking altogether during traversal, making it the fastest in this case.
* **Scenario 2 (High Contention, Many Threads):**  
  With more threads trying to modify the same part of the list, **Fine-Grained Synchronization** (9 ms) showed reasonable performance, while **Coarse-Grained Synchronization** took less time (5 ms) due to less lock acquisition overhead, as threads were frequently waiting for the single lock. Surprisingly, **Optimistic Synchronization** (4 ms) also performed well, as retries were minimal due to the small range of contention.
* **Scenario 3 (Mixed Operations):**  
  In a mixed workload, **Optimistic Synchronization** (3 ms) proved to be the most efficient, followed by **Fine-Grained Synchronization** (4 ms), and **Coarse-Grained Synchronization** (11 ms). This demonstrates that fine-grained locking is effective when different operations (add, remove, contains) are being performed concurrently, while optimistic locking handles a variety of tasks with minimal retries.

**Reasons for the Trends:**

* **Coarse-Grained Synchronization:**  
  This method uses a single lock for the entire list, making it simple but limiting concurrency. It performs well in high contention scenarios with fewer operations, as threads are often blocked waiting for the lock, reducing lock acquisition overhead. However, its performance drastically drops in low-contention scenarios because even disjoint threads must wait for the lock.
* **Fine-Grained Synchronization:**  
  Fine-grained locking uses locks at the node level, allowing greater concurrency. This method excels in scenarios with low contention because multiple threads can operate on different parts of the list simultaneously. However, in high contention, the overhead of acquiring and releasing locks for individual nodes slightly impacts performance.
* **Optimistic Synchronization:**  
  Optimistic synchronization is ideal in low and mixed contention scenarios. By traversing the list without locking and only locking at the end of an operation, this method minimizes lock contention. In high-contention scenarios, retries can become an issue, but in our case, contention was moderate, leading to good performance even in these situations.

**Conclusions:**

* **Coarse-Grained Synchronization** is best used in scenarios where contention is high, and the simplicity of a single lock outweighs the loss of concurrency.
* **Fine-Grained Synchronization** shines in low-to-moderate contention scenarios, where multiple threads need to access different parts of the list concurrently.
* **Optimistic Synchronization** is highly efficient in low-contention or mixed workloads, where lock contention is rare, and the need for retries is minimal.

**References**

* *The Art of Multiprocessor Programming by Maurice Herlihy and Nir Shavit*

**Recipe:**

Ingredients:

* 1kg of thinking
* 2L of pondering
* 1.1mL of contemplating

Steps:

* Think, and then ponder, and then contemplate.