HPC ASSIGNMENT 4

Group 14

1. Tanisha Rana SE20UCSE202
2. Jahnavi Siripurapu SE20UCSE178
3. Harshit Reddy SE20UCSE051
4. Ridhi Bigala SE20UCSE138
5. Amrutha Manne SE20UARI017
6. Agastya Gandra SE20UARI011

To develop a concurrent double linked list, we can consider the following synchronization techniques:

1. Coarse-Grain Synchronization: In this technique, a single lock is used to synchronize the entire list. Each thread acquires the lock before accessing the list and releases it after completing the operation.
2. Fine-Grain Synchronization: In this technique, we divide the list into smaller sections, and each section has its own lock. This allows multiple threads to access different sections of the list simultaneously without interfering with each other.
3. Optimistic Synchronization: In this technique, we assume that conflicts between threads are rare, and most operations can proceed without locking. A thread executes its operation without acquiring a lock and checks for conflicts afterward. If there is a conflict, the thread retries the operation with a lock.
4. Lazy Synchronization: In this technique, we defer the synchronization until it is necessary. We use a non-blocking algorithm that allows threads to make progress even in the presence of conflicts.

For each of these techniques, we can implement the concurrent double linked list and evaluate their performance using five different workloads:
Explanation:

1. 100C-0I-0D: 100% of operations are insertions.
2. 70C-20I-10D: 70% of operations are insertions, 20% are searches, and 10% are deletions.
3. 50C-25I-25D: 50% of operations are insertions, 25% are searches, and 25% are deletions.
4. 30C-35I-35D: 30% of operations are insertions, 35% are searches, and 35% are deletions.
5. 0C-50I-50D: 50% of operations are insertions, and 50% are deletions.

We will implement each synchronization technique and evaluate their performance using five different workloads and three different key

ranges. We will also vary the number of threads from 1 to 16 and report the speedup of our implementations.
To implement the concurrent double-linked list, we can use the following data structure:
Each node contains a key, a pointer to the previous node, a pointer to the next node, and a mutex for synchronization.

The three different key ranges are 2 × 105, 2 × 106, and 2 × 107. We can assume that the data structure is prefilled with 50% of keys range, and the execution time of each run is 100 seconds. To evaluate the performance of the synchronization techniques, we need to vary the number of threads from 1 to 16 and report the speedup of each implementation. We can draw graphs using gnuplot to visualize the performance of each implementation.
The ‘add’ method creates a new node with the given key and inserts it at the end of the list.

The ’remove’ method removes the node with the given key from the list.

The ‘contains’ method checks if a node with the given key exists in the list.

All these methods acquire the lock before accessing the list.