Concurrent Data Structures – Code Assignment 3

Parosh Aziz Abdulla
Sarbojit Das

December 14, 2021

Deadline: 2022–01–02 23.59.
Please, submit your source code and instructions to run them.

Task 1 Lazy synchronization Implement a concurrent list-based set with Lazy synchronization in a file named "LazyList.cpp".

Task 2 Experiment Performs the following experiments:

- Create n worker threads, where $n = 2, 4, 8, 16, 32, 64$ threads
- For each n, let each worker thread perform various operations `add`(), `rmv`(), and `ctn`(). The type of the operations are selected randomly, and the input values are selected randomly from the set $\{0, 1, \ldots, 7\}$. Measure the throughput of the system in 10 seconds, i.e., the total number of operations completed by the threads in 10 seconds. Perform the experiments for optimistic and lazy algorithm.
- Repeat the previous experiments when the the input values are selected randomly from the set $\{0, 1, \ldots, 1023\}$.
- Repeat all the above sets of experiments, now increasing the percentage of the `add()` operations. For each value $i = 10, 50$, and 90, conduct the above experiments such that i% of the operations are `cnt()`. From the remaining set, 90% should be `add`, and 10% should `rmv()`. For instance, for $i = 60$, we have 60% `ctn()`, 36% `add()`, and 4% `rmv()`.

Depict the results in tables or (preferably) curves, where the x-axis is the number of threads, and the y-axis is the throughput.

Task 3 The Extended Treiber Algorithm Modify and implement the Treiber algorithm so that it supports the following functions:

- `push(x)`: Adds element x to the top of the stack.
- `pop()`: Removes the top-most stack elements, and returns the value of its key. If the stack is empty, it returns the special value $*$.
- `size()`: Returns the size of the stack, i.e., the current number of elements in the stack.

Create 16 worker threads, each does 100 operations on stack. Mix various operations `push(x)`, `pop()`, and `size()`. Identify linearization policy and check if it is consistent with non-concurrent for the extended stack ADT. You can create a monitor thread exactly like the previous assignments and communicate. Create a shared sequence of operations. Whenever a worker does some operation, put it in the shared sequence at the linearization point. The monitor reads the sequence and performs the operations on the non-concurrent Stack ADT (that also includes the `size` method). Here you can find c++ stack: std::stack.