COP 6616 Fall 2015

Programming Assignment 3

Note 1:

Please, submit your work via Webcourses.

Submissions by e-mail will not be accepted.

Due date: Wednesday, Oct. 7th at 11:59pm.

Late submissions are not accepted.

Problem Definition

Abstract locking [1] or Semantic locking [2] refers to the synchronization technique, which applies mutual exclusion locks to object abstractions instead of concrete code blocks. For example, consider the following code snippet:

```
set.lock(x,y){if set.contain(y) then set.insert(x)}
```

It locks on two elements of the set data structure with specific values `x` and `y`. The use of abstract locks makes it possible to support compostable transactions for concurrent data structures. In the above example, two operations of the set are composed to form an `InsertIfContains`(x, y) transactional operation. This operation ensures that element y exists in the set (i.e., it cannot be removed by any concurrent delete operations) at least until element x has been successfully inserted.

In the existing approaches, abstract locks are implemented based on two phase-locking, which suffers from limited scalability and the possibility of deadlock. MRLock [3] is a centralized multi-resource locking algorithm that manages batch-locking request using a lock-free FIFO queue.

In this assignment you are tasked with the design of an abstract locking algorithm that is based on MRLock for the `tbb::concurrent_unordered_set` data structure from Intel's Thread Building Block (TBB) library (https://www.threadingbuildingblocks.org). This way you will implement an abstract locking algorithm that does not suffer from the hazards of deadlock. Your task is to:

1. Implement an abstract locking algorithm using the two-phase locking mechanism. The standard C++ library provides an `std::lock` method that implements two-phase locking (http://en.cppreference.com/w/cpp/thread/lock). Your algorithm should allocate

one mutual exclusion lock for each key. For simplicity limit the range of the keys to 1,000. It should also provide a lock and an unlock methods that accept an array of keys (e.g., lock(int key1, int key2, int key3, …) or lock (int[] keys)). The lock method acquires **all** the required mutual exclusion locks before returning to the caller.

2. Implement an abstract locking algorithm using MRLock. The source code of MRLock can be downloaded from http://cse.eecs.ucf.edu/gitlab/deli/mrlock . Your MRLock-based algorithm should have the same interface and functionality as the first implementation.
3. Compose a transactional `InsertIfContains` and `DeleteThenInsert` operation using the above two abstract lock algorithms. The semantics of `InsertIfContains` operation is explained at the beginning of the assignment. The semantics of `DeleteThenInsert(x, y)` require that element y is to be inserted into the set only if element x has been successfully deleted from the set. Optionally (bonus points), implement some large transactional operations that involve three or more elements.
4. Finally, conduct performance tests using multiple threads comparing the two abstract lock algorithms and a coarse-grained lock in the scenarios described below. Note that the threads in the coarse-grained lock approach should acquire the global lock on the data structure before performing the operations. The threads in the abstract lock approaches should acquire locks on the specific keys before performing the operations.
   a) Each thread performs a random operation on a single element.
   b) Each thread performs an `InsertIfContains` operation on random keys.
   c) Each thread performs a `DeleteThenInsert` operation on random keys.
   d) Optionally (bonus points), each thread performs a large transactional operation that you implemented in step 3.

Report the outcome of your experimental evaluation by presenting several graphs and your interpretation of the observed results.

Bibliography

[1] M. a. K. E. Herlihy, "Transactional boosting: a methodology for highly-concurrent transactional objects," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, 2008.

[2] G. a. R. G. a. S. M. a. Y. E. Golan-Gueta, "Automatic scalable atomicity via semantic locking," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2015.

[3] D. Zhang, B. Lynch and D. Dechev, "Fast and Scalable Queue-Based Resource Allocation Lock on Shared-Memory Multiprocessors," in *Principles of Distributed Systems*, 2013.