Implementation of binary search tree with fine-grained locking and lock-free methods

Sean Zhou, Ye Lu

URL

https://louisluscu.github.io/15418_Project/

Summary

We are going to implement a fine-grained locking version and a lock-free version of binary search tree (BST) to achieve high performance in the face of concurrent tree operations. And benchmark their performance on different kinds of workloads and analysis on that.

Background

BST is one of the most commonly used data structures. Not surprisingly, in parallel programming, there is much need for BST that allows safe concurrent access while having good performance.

Fine-grained locking (for example: hand-over-hand lock) can improve the performance of concurrent BST by minimizing critical sections and reducing lock contention, allowing different threads to access different parts of the BST concurrently. However, fine-grained locking is vulnerable when the thread holding the lock crashes or is paged out, blocking any other thread from progressing.

In comparison, lock-free BST will not have this problem, as it relies on atomic operations rather than locks so that at least one thread is guaranteed to make progress at all times. This guarantee, however, typically comes at the cost of performance.

It is therefore meaningful to implement and benchmark fine-grained and lock-free BST under various work loads, so that we can better grasp the practicality of these concurrent BST structures.

Challenge

- Implement a concurrent fine-grained locking BST. We need to come up with a fine-grained locking schema suitable for BST (e.g. hand-over-hand lock) and implement it well to support all standard operations on trees.

- Implement a concurrent lock-free BST. We need to locate a good algorithm for lock-free BST and translate it correctly into code.
- We need to create a variety of workloads to benchmark our BST with.
- We need to come up with a valid correctness test for the results of concurrent operations. This is especially challenging because operations can be interleaved arbitrarily. We need to be able to tell the possible sequences of operations from the impossible.

Goals and deliverables

Plan to achieve

- Source code for fine-grained locking version of binary search tree
- Source code for lock-free version of binary search tree
- Utility program to generate specific workloads and construct different structured BST
- Correctness test harness to test the correctness of our implementation
- Detailed comparison of the two versions of BST's performance under different well-designed workloads

Hope to achieve

- Implement self-balanced BST and analysis how this influence concurrent performance

Platform & Resources

We are planning to use C++ as the main programming language, since C++ provides many useful prototypes like *std::mutex, atomic<T>* and multiple CAS operations that enable us to implement both fine-grained locking and lock-free data structures

We are planning to develop and benchmark our implementation on GHC machines. We might also utilize PSC machines to benchmark on heavy contention scenarios with a large number of threads.

We also find some helpful resources to help us understand and implement these two data structure:

- A Practical Concurrent Binary Search Tree
- A non-blocking internal binary search tree
- Fast concurrent lock-free binary search trees

Schedule

Period	Goal
11.3 - 11.8	paper reading and implementation decisions discussion
11.9 - 11.15	implementation of fine-grained locking BST
11.16 - 11.21	implementation of lock-free BST
11.22 - 11.29	workloads design and generation
11.30 - 12.6	benchmark on two versions of BST and analysis
12.6 - 12.9	writing final report and prepare for poster session