A Relaxed Balanced Non-Blocking Binary Search Tree

Manish Singh*, Lindsay Groves, Alex Potanin

manish.singh@weltec.ac.nz, lindsay.groves@ecs.vuw.ac.nz, alex.potanin@ecs.vuw.ac.nz

Abstract

We present a new relaxed balanced concurrent binary search tree in which all operations are non-blocking.

We utilise the notion of separating balancing operation and update operations in a concurrent environment and design a non- blocking balancing operation in addition to the regular insert, search and relaxed delete operations. Our design uses a single-word CAS supported by most modern CPUs.

Introduction

The evolution of multi-core/many-core systems has necessitated the design of scalable and efficient concurrent data structures. The asynchronous model of computation in such systems makes it notoriously hard to design a correct and efficient concurrent data structure that synchronises concurrent access to shared memory. A considerable amount of research has been done towards making a concurrent version of sequential data structures. Performance has been always an important factor that will always drive the design of new concurrent data structures.

A concurrent design is non-blocking(or lock-free) if it ensures that no thread/core accessing the data structure is postponed indefinitely while waiting for other threads that operate on that structure.

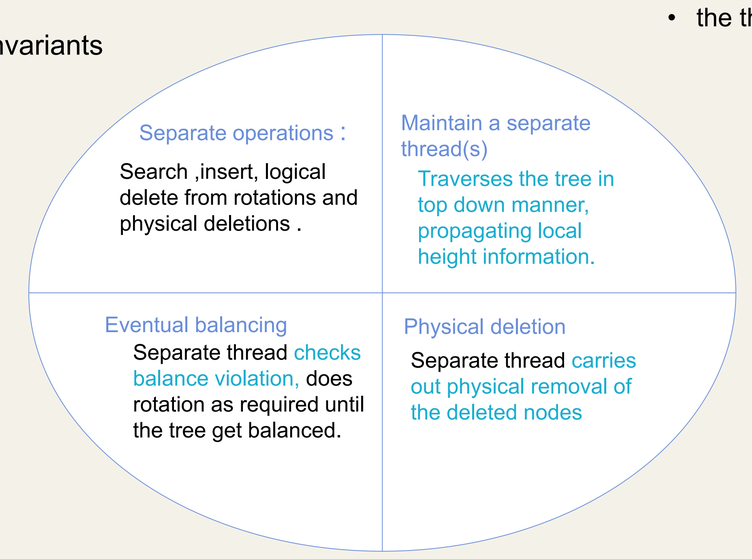
The Binary Search Tree (BST) is used in many practical real-world systems as a search structure. In recent years, some implementations of concurrent BST (both blocking and non-blocking versions) have been proposed[1-7], but a non-blocking concurrent BST that relaxes strict sequential invariants with balancing operation is yet to be seen. In an unbalanced BST, traversal could take O(n) time, when the inputs are in an ordered sequence. Rotation operations are used to balance the tree. Our work focuses on designing a non-blocking, scalable, and highly concurrent relaxed balanced BST.

Observations

In a concurrent environment,

- effect of some rotations might cancel out each other.
- doing rotation with every insert and delete increases contention. Locking large portion of BST results in decrease in concurrency. Design all operations non-blocking.
- balancing are Not for correctness!

Hence relax the invariants



Existing concurrent BSTs

BST	Type	BalOpn	SyncTech	Add Info
Howley[5]	Internal	no	non-blocking	Descriptor based
Natrajan[1]	External	no	non-blocking	Edge marking
Chaterjee[2]	Internal	no	non-blocking	threaded
Bronshon[4]	Partially External	yes	blocking	lock coupling
Crain[6]	Partially External	yes	blocking	Eager abstract n lazy structural modification
Drachsler[3]	Internal	yes	blocking	logical ordering of updates, threaded
Our tree	Partially Internal	yes	non-blocking	AVL based relaxed and delete operation

Table 1: Existing concurrent BST's.

Design Overview

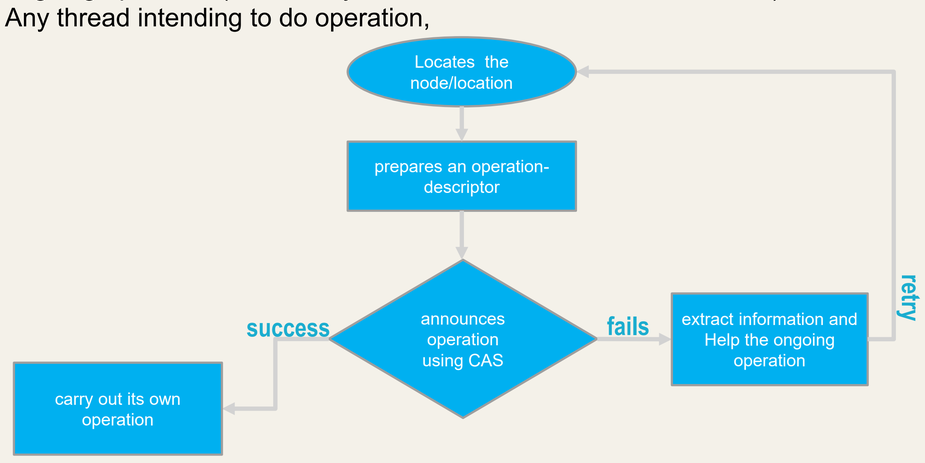
We have designed an abstract concurrent set in which the underlying data structure is a BST. Our implementation supports:

- search(k): to check if the key is in the set or not,
- insert(k): to add the key into the abstraction,
- delete(k): to remove the membership of key if it is present in that dataset.

- This design closely mirrors the sequential version of BST except for the delete operation, where the key is first mark as deleted and then physically removed later by the dedicated separate thread.

Achieving non-blocking or lock-freedom

- Threads do not lock any locations operations owns the locations instead.
- Thread synchronization
  - Each node in the BST has an operation pointer field of which last two bits give information about ongoing operation (In 32-bit system last two bit of address are zero). No extra overhead.
  - Any thread intending to do operation,



- Rotation can only be announced by dedicated thread but can be completed by any other thread. Physical removal can only be done by the dedicated thread.

Rotation operation is shown in figure1 (for right rotation, Similarly for left rotation). After grabbing the required nodes,

- a newnode having the same key as the node where balance violation has occurred is allocated.
- right and left pointers of the newnode are allocated to the respective children which the original node would get after the rotation(b).
- next step, is to insert the newnode to its position after rotation (c).
- the third and last step is to connect 'c' to the parent 'p' thereby removing node 'n' from the tree (d).

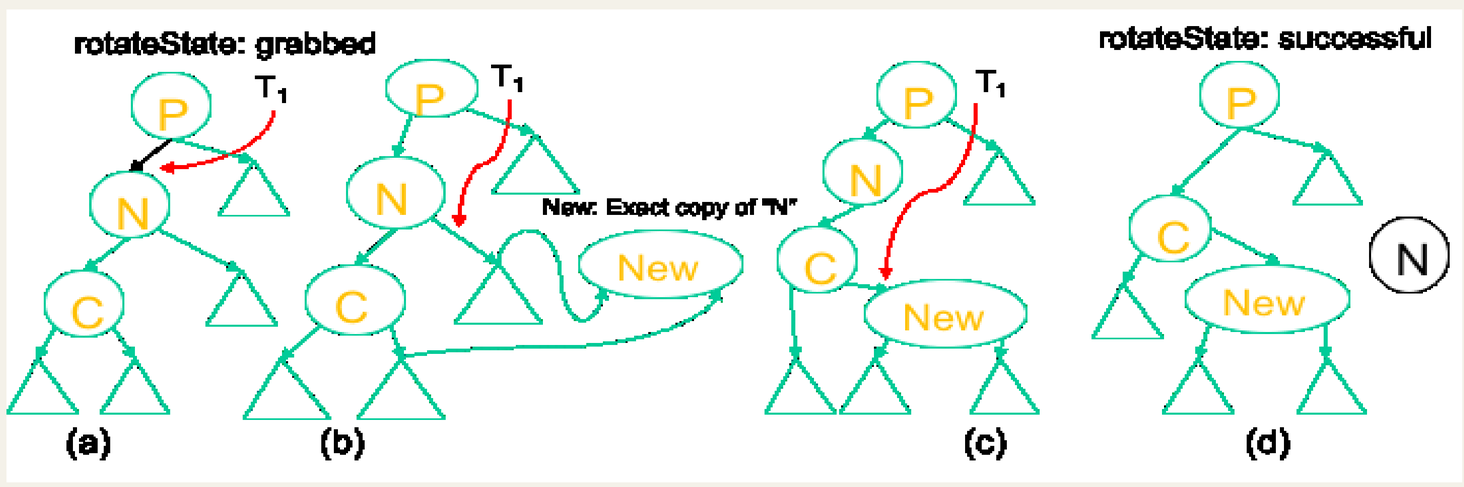


Figure 1: Right Rotation example. T1: a thread carrying out a search operation oblivious of ongoing concurrent rotation.

Summary

- In our design, all the operations are *non-blocking*.
- Search operation is free of any additional synchronization.
- We are evaluating performance of our algorithm on x86 _ 64 and SPARC multi-core machines against the concurrent BSTs shown in table 1.
- A mechanised proof outline has been done using linearisability as the correctness criteria.
- To the best of our knowledge, this is the first design which utilises decoupling of operations as well as rotations to balance the tree in a non-blocking concurrent set-up.

References

1) N. Mittal A. Natarajan. 2014. Fast concurrent lock-free binary search trees. *In Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Prac- tice of Parallel Programming (PPoPP)* (2014).
2) Bapi Chatterjee, Nhan Nguyen, and Philippas Tsigas. 2014. Efficient Lock-free Binary Search Trees. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC '14)*. ACM, New York, NY, USA, 322–331.
3) Dana Drachsler, Martin Vechev, and Eran Yahav. 2014. Practical Concurrent Binary Search Trees via Logical Ordering. *SIGPLAN Not.* 49, 8 (Feb. 2014), 343–356.
4) Hassan Cha Nathan G. Bronson, Jared Casper and Kunle Olukotun. 2010. A practical concurrent binary search tree. *ACM SIGPLAN Symposium on Principals and Practice of Parallel Programming* (2010).
5) Jeremy Jones Shane V. Howley. 2012. A non blocking internal binary tree. *SPAA* (June 2012).
6) M. Raynal T. Crain, V. Gramoli. 2013. contention-friendly binary search tree. In *In Euro-Par*. 229–249.
7) Yehuda Afek. 2012. A practical concurrent self-adjusting search tree. *Tel Aviv University* (2012).

*Corresponding author: Manish Singh, manish.singh@weltec.ac.nz, School of IT, Wellington Institute of Technology, Lower Hutt, NZ