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 remove/delete operations. Our design uses a single-word CAS supported by most modern CPUs.

Existing concurrent BSTs

<table>
<thead>
<tr>
<th>BST</th>
<th>Type</th>
<th>BalDyn</th>
<th>SyncTech</th>
<th>Add Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Internal</td>
<td>no</td>
<td>non-blocking</td>
<td>Description based</td>
</tr>
<tr>
<td>Natural</td>
<td>External</td>
<td>no</td>
<td>non-blocking</td>
<td>Edge marking</td>
</tr>
<tr>
<td>Chatterjee</td>
<td>Internal</td>
<td>no</td>
<td>non-blocking</td>
<td>thread-1</td>
</tr>
<tr>
<td>Breachman</td>
<td>Partially External</td>
<td>yes</td>
<td>blocking</td>
<td>lock coupling</td>
</tr>
<tr>
<td>Grant</td>
<td>Partially External</td>
<td>yes</td>
<td>blocking</td>
<td>eager abstract lazy structural modification</td>
</tr>
<tr>
<td>Drachani</td>
<td>Internal</td>
<td>yes</td>
<td>non-blocking</td>
<td>AVL based relaxed and delete operation</td>
</tr>
</tbody>
</table>

Table 1: Existing concurrent BSTs.

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 marked 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 own 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.

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

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 linearity 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


