A Transaction-Friendly Binary Search Tree

Tyler CRAIN    Vincent GRAMOLI    Michel RAYNAL
{tyler.crain|raynal}@irisa.fr
{vincent.gramoli}@epfl.ch

TM Theory Workshop 2011

ASAP team, IRISA, Rennes, France & Distributed Programming Laboratory,
EPFL, Lausanne, Switzerland
Existing Data Structures

Preserving structural invariants

- **Balanced Binary Trees**
  - Rotations keep the tree balanced

- **Skiplist**
  - Node levels follow some fixed distribution

- **Hashtable**
  - Bucket size must not exceed some threshold
Using data structures with TM

Simple!

- Copy-paste into transactions (more or less)
  + Easy to program/use
  + The TM system ensures safety
  - Are there disadvantages?
Concurrent Data Structures

Balanced Binary Tree

- Specially designed for concurrency
- Hand-over-hand locking

\[ O(\log n) \]
Data Structures in TM (So far)

Balanced Binary Tree

- (Mostly) Unmodified from their original versions
- Not designed for concurrency or transactions
  - Could lead to unnecessary conflicts and aborts.

\[ \Omega(r \log n) \]
\( \Omega(r \log n) \)

What is \( r \)?

- The number of restarts
  - Depends on the contention of the workload
  - Depends on the conflicts between transactions
  \[ \Rightarrow r \text{ depends on } n \]
Aborts and wasted work

- Still $O(\log n)$ operations?

<table>
<thead>
<tr>
<th>Update</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVL tree</td>
<td>29</td>
<td>415</td>
<td>711</td>
<td>1008</td>
<td>1981</td>
<td>2081</td>
</tr>
<tr>
<td>Sun red-black tree</td>
<td>31</td>
<td>573</td>
<td>965</td>
<td>1108</td>
<td>1484</td>
<td>1545</td>
</tr>
</tbody>
</table>

Table: Maximum #reads/op in $2^{12}$ sized trees

- Can we relax some invariants in order to reduce conflicts?
Example

- 3 operations
- $1 \rightarrow \text{insert}$, $2 \rightarrow \text{delete}$, $3 \rightarrow \text{contains}$
Where they can conflict

- Along their entire path
Goal

- Minimize conflicts
Rotations
Correctness & Conflicts

- Rotations are not required for correctness
- There will be concurrent insertions/deletions
  - Concurrent insertions/deletions might have conflicting rotations
    - They might cancel each other out
    - A later insert/deletion might invalidate these rotations
- Idea: relax the balance requirement in order to allow more concurrency
Rotations cont.

Solution

- Separate rotations from insert/delete operations
- Perform rotations in their own thread
- Each rotation is a single transaction

Deletions
Reducing conflicts further

- A delete operation can still modify the tree structure
- A successor must be found to replace the node being deleted
Deletions cont.

Solution

- Logical deletions
  - Each node has a *deleted* boolean flag
  - Initialized as *false*
  - Set to *true* on deletion

- Allows concurrent operations to traverse the node being deleted without conflicting
Removals

- Logically deleted nodes must be removed from the tree
  - Done in a separate thread
  - Only nodes with 1 or 0 children are removed

Bronson N., Casper J., Chafi H., Olukotun K., A Practical Concurrent Binary Search Tree, PPoPP ’10
• Each diagram is a single transaction
Insert

- Each diagram is a single transaction
Delete

- Each diagram is a single transaction
Now we have

Abstract transaction conflicts
# Impact on read size

<table>
<thead>
<tr>
<th>Update</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVL tree</td>
<td>29</td>
<td>415</td>
<td>711</td>
<td>1008</td>
<td>1981</td>
<td>2081</td>
</tr>
<tr>
<td>Sun red-black tree</td>
<td>31</td>
<td>573</td>
<td>965</td>
<td>1108</td>
<td>1484</td>
<td>1545</td>
</tr>
<tr>
<td>Tx-friendly tree</td>
<td>29</td>
<td>75</td>
<td>123</td>
<td>120</td>
<td>144</td>
<td>180</td>
</tr>
</tbody>
</table>

**Table:** Maximum #reads/op in $2^{12}$ sized trees
Conclusion

Benefits of a Transaction Friendly Data Structure

- Improved Performance
- No difference to the programmer using the tree as a library
- Uses normal transactional reads/writes
  - Compatible with many TMs
    - Tested on TinySTM and $\varepsilon$-STM
  - Independent of TM specifications
    - Tested using CTL/ETL, different contention managers
Reusability

Move operation

Algorithm 3 Move operation

1: \textbf{move} (\texttt{old\_key}, \texttt{new\_key})_p:
2: \hspace{1em} \textbf{transaction} \{ 
3: \hspace{2em} \texttt{ret} \leftarrow \texttt{false} 
4: \hspace{2em} \textbf{if} \neg \texttt{contains}(\texttt{new\_key}) \texttt{then} 
5: \hspace{3em} \texttt{if} \ \texttt{v} \leftarrow \texttt{delete}(\texttt{old\_key}) \texttt{then} 
6: \hspace{4em} \texttt{insert}(\texttt{new\_key}, \texttt{v}) 
7: \hspace{2em} \texttt{ret} \leftarrow \texttt{true} 
8: \hspace{1em} \} // \texttt{current transaction tries to commit} 
9: \hspace{1em} \textbf{return} \ \texttt{ret}
Future Work

Other structures

- Transaction friendly skip list
- Transaction friendly hash table
- Transaction friendly . . .
• There’s more?
Certain TMs give mechanisms for improved performance at the cost of safety
- Early-release
- $\varepsilon$-STM
- View transactions
- Unit reads
Unit Reads

- Returns the latest value written by a committed transaction
- Does not add the location to the read set or perform validation
• How can unit reads be used to improve performance of the algorithm?
Current Situation

- Rotations can still conflict with concurrent insert/delete/contains operations
Solution

- Use unit reads during the tree traversal
- Advantages:
  - Faster traversals (unit reads are cheaper)
  - Avoid during traversal
  - Smaller read set
Result
Abstract + Structural Transactions
• What about safety?
• Algorithm becomes a bit more complicated to ensure safety
New rotations

(a) Initial tree

(b) Result of usual right rotation

(c) Result of new right rotation
New removals

(a) Before removal
NULL

(b) After removal
Traversals

- Mostly unit reads
- Transactional reads performed at bottom to ensure safety
- Each node has a *removed* flag
  - Used to ensure traversal does not finish on a node that no longer in the tree
Impact on read size

<table>
<thead>
<tr>
<th>Update</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVL tree</td>
<td>29</td>
<td>415</td>
<td>711</td>
<td>1008</td>
<td>1981</td>
<td>2081</td>
</tr>
<tr>
<td>Sun red-black tree</td>
<td>31</td>
<td>573</td>
<td>965</td>
<td>1108</td>
<td>1484</td>
<td>1545</td>
</tr>
<tr>
<td>Tx-friendly tree</td>
<td>29</td>
<td>75</td>
<td>123</td>
<td>120</td>
<td>144</td>
<td>180</td>
</tr>
<tr>
<td>Unit read tree</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>13</td>
<td>15</td>
<td>18</td>
</tr>
</tbody>
</table>

Table: Maximum #reads/op in $2^{12}$ sized trees
• Performance Results: Some graphs from benchmarks