algorithm, language-agnostic, data-structures, binary-tree, skip-lists

Skip List vs. Binary Tree

selected 15年5月

I recently came across the data structure known as a **Skip list**. They seem to have very similar behavior to a binary search tree... my question is - why would you ever want to use a skip list over a binary search tree?

selected 15年5月

Skip lists are more amenable to concurrent access/modification. Herb Sutter wrote an **article** about data structure in concurrent environments. It has more indepth information.

The most frequently used implementation of a binary search tree is a **red-black tree**. The concurrent problems come in when the tree is modified it often needs to rebalance. The rebalance operation can affect large portions of the tree, which would require a mutex lock on many of the tree nodes. Inserting a node into a skip list is far more localized, only nodes directly linked to the affected node need to be locked.

Update from Jon Harrops comments

I read Fraser and Harris's latest paper **Concurrent programming without locks**. Really good stuff if you're interested in lock-free data structures. The paper focuses on **Transactional Memory** and a theoretical operation multiword-compare-and-swap MCAS. Both of these are simulated in software as no hardware supports them yet. I'm fairly impressed that they were able to build MCAS in software at all.

I didn't find the transactional memory stuff particularly compelling as it requires a garbage collector. Also **software transactional memory** is plagued with performance issues. However, I'd be very excited if hardware transactional memory ever becomes common. In the end it's still research and won't be of use for production code for another decade or so.

In section 8.2 they compare the performance of several concurrent tree implementations. I'll summarize their findings. It's worth it to download the pdf as it has some very informative graphs on pages 50, 53, and 54.

- **Locking skip lists** are insanely fast. They scale incredibly well with the number of concurrent accesses. This is what makes skip lists special, other lock based data structures tend to croak under pressure.

- **Lock-free skip lists** are consistently faster than locking skip lists but only barely.
- **transactional skip lists** are consistently 2-3 times slower than the locking and non-locking versions.
- **locking red-black trees** croak under concurrent access. Their performance degrades linearly with each new concurrent user. Of the two known locking red-black tree implementations, one essentially has a global lock during tree rebalancing. The other uses fancy (and complicated) lock escalation but still doesn't significantly out perform the global lock version.
- **lock-free red-black trees** don't exist (no longer true, see Update).
- **transactional red-black trees** are comparable with transactional skip-lists. That was very surprising and very promising. Transactional memory, though slower if far easier to write. It can be as easy as quick search and replace on the non-concurrent version.

Update
Here is paper about lock-free trees: **Lock-Free Red-Black Trees Using CAS**.
I haven't looked into it deeply, but on the surface it seems solid.

---

selected **15年5月**

Also, in addition to the answers given (ease of implementation combined with comparable performance to a balanced tree). I find that implementing in-order traversal (forwards and backwards) is far simpler because a skip-list effectively has a linked list inside its implementation.

---

selected **15年5月**

From the **Wikipedia** article you quoted:

> Θ(n) operations, which force us to visit every node in ascending order (such as printing the entire list) provide the opportunity to perform a behind-the-scenes derandomization of the level structure of the skip-list in an optimal way, bringing the skip list to O(log n) search time.
> [...]
> A skip list, upon which we have not
> recently performed [any such] Θ(n) operations, **does not
> provide the same absolute worst-case
> performance guarantees as more
> traditional balanced tree data
> structures**, because it is always
> possible (though with very low
> probability) that the coin-flips used

> to build the skip list will produce a
> badly balanced structure

EDIT: so it's a trade-off: Skip Lists use less memory at the risk that they might degenerate into an unbalanced tree.