Skip Lists: A concurrency-friendly data structure

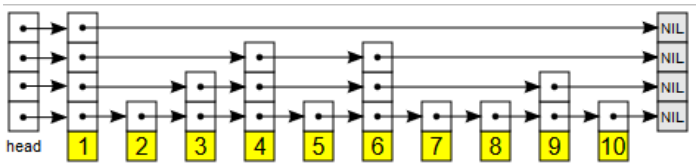


Fig 1: A Randomized Skip List

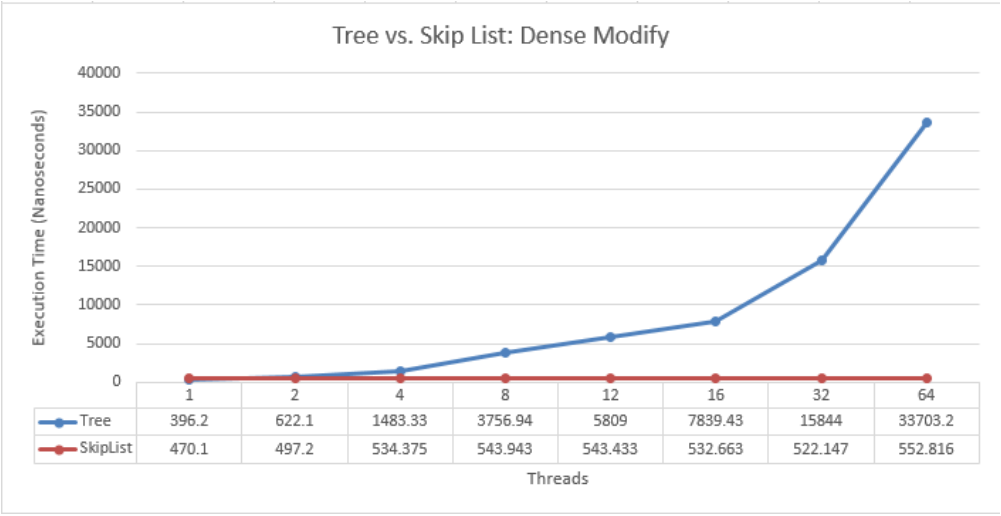
Summary:

Parallel execution of programs gives rise to problems that have no easy fixes. Shared resources are the biggest bottleneck holding back parallel programming.

Data is largest shared resource in most cases. Which is why concurrency-friendly data structures are more important than ever. But concurrent data structures need to be consistent as well – which requires locking. More sophisticated methods of locking like fine-grained locking are better when applied to list-like data structures (Skip Lists being one of them). Applying such methods to Balanced Search Trees are impractical.

Which brings us to Skip Lists. They try to take both the concurrency friendliness of Lists and the O(logn) performance guarantee of trees into one – albeit with compromises. Skip Lists are randomized multi-level linked list structures that give an expected time guarantee of O(logn).

In the absence of a concurrent tree structure, our aim is to show that the globally locked search tree performs very poorly when compared to skip lists in highly parallel environments. The main point being the lack of scalability of Search Trees, which leads to degrading performance when lots of threads try to access the data structure at the same time. Skip Lists maintain the same level of performance when confronted with the highly concurrent environment.



	1	2	4	8	12	16	32	64
Tree	396.2	622.1	1483.33	3756.94	5809	7839.43	15844	33703.2
SkipList	470.1	497.2	534.375	543.944	543.433	532.663	522.147	552.816