Monkey Hash Map: a highly performant thread-safe map without locks

Judismar Arpini Junior [1,*] Vinícius Gusmão Pereira de Sá [1]
[1] Universidade Federal do Rio de Janeiro

Keywords: concurrent data structures, hashing, wait-free

Hash tables are arguably the most powerful data structures ever known. A shared data structure is *lock-free* if infinitely often *some* thread completes its task within a finite number of steps. A shared data structure is *wait-free* if *each thread* completes its execution within a finite number of steps.

We exploit multiple-choice hashing to create a high-performance, wait-free hashing scheme with $O(1)$ worst-case time for lookup, getValue, insert, update and remove operations, a hash table that provides thread-safety without requiring any kind of thread synchronization. In short, our *monkey hashing* scheme consists of a single hash table and a family of $k \geq 1$ hash functions, meaning multiple alternative locations for each key in the same table. Unlike what happens in the well-known cuckoo hashing, elements will never be evicted from where they first landed, so new keys being inserted must always find an unoccupied spot to call their own. The actual counts of hash functions in use are kept track of, making it possible that lookups of absent keys fail before the entire family of hash functions has been exhausted. Dynamic memory allocation—and its inherent pauses, e.g. garbage collection—is avoided via a key-value object pool, and thread-safe is attained via (i) pre-allocation of the underlying array, meaning no rehashing will ever take place, and (ii) the fact that no collision-handling lists are called for, by design.

The proposed scheme works particularly well in scenarios with a single writer and multiple reader threads, dramatically outperforming popular solutions such as ConcurrentHashMap (Java) and Intel TBB concurrent_hash_map (C++) in heavily concurrent test scenarios. The prices to pay are (i) eventual consistency, which is dealt with well in numerous concurrent settings, and (ii) a non-zero probability that an insertion might fail, which can be made small enough, though, to suit all imaginable applications.