Lock-free Binary Search Tree and Skip List

Chenhao Li

April 29, 2022

Dictionary

- Dictionary is an important abstract data type in real-world applications.
  - `search`, `insert`, `remove`.
- Binary search tree and skip list are efficient implementations for the dictionary when a partial order relationship is defined.
- It is important (and interesting) and to build lock-free binary search tree and skip list.

Binary Search Tree and Skip List

- BST associates each key with a node and organizes them in a tree structure. The key in the parent node should be strictly greater than keys in the left subtree, and strictly smaller than keys in the right subtree.
- Skip list consists of multiple layers of ordered linked lists. Each node stores a set of `next` pointers, whose number is determined by an exponential distribution. The bottom layer contains all nodes, and higher layers skip multiple nodes.

Lock-free Binary Search Tree

- It does not strictly follow the original BST definition.
- Leaf nodes store all keys, internal nodes store parts of keys for routing. Leaf nodes have zero children, internal nodes have two children.
- Use the least significant two bits of child pointers for marking.
  - `flag`: both the parent and this child are going to be removed.
  - `tag`: the parent is going to be removed from the tree.
- Once an edge has been flagged or tagged, it cannot point to another node.
- Please refer to [2] for more details.

Lock-free Skip List

- Use the least significant bit in each level of `next` pointers for marking.
- The `mark` bit is set before physically removing the node to prevent concurrent operations from inserting after this node and losing the inserted node.
- Search physically remove marked nodes between a level's `predecessor` and `successor`.
- Please refer to [1] for more details.

Garbage Collection

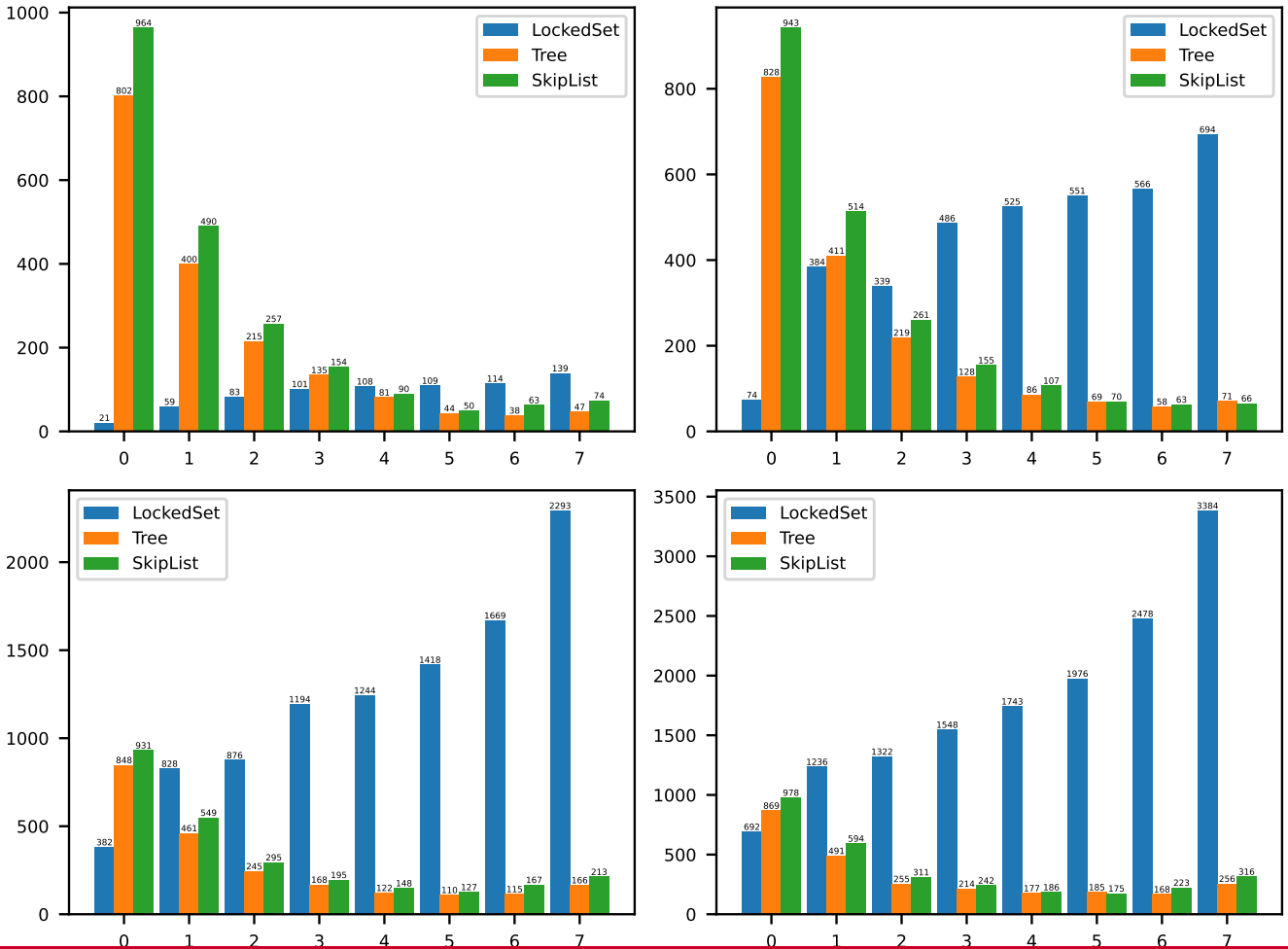
- Garbage Collection is non-trivial for lock-free data structures in a language without builtin GC.
- When a thread decides a node can be freed, it may still be reachable in another thread.
- ABA problem: a pointer is freed and reclaimed by the memory allocator, causing CAS instructions to make a wrong judgment that pointer fields have not changed.

Garbage Collection

- Use an atomic integer to represent the current state.
  - The least significant bit: whether there is a pending GC.
  - Higher bits count the number of active operations.
- Push freed pointers into a private list to the thread. When any operation ends, if the number of freed pointers reaches a threshold, set the `pending` bit with CAS.
- When any operation starts, wait until `pending` is cleared.
- The thread successfully setting `pending` waits until there is no active operations and performs GC by freeing all pointers in all threads, and clear `pending`.

Performance

- 1000000 ramdom operaions. Modification (`insert+remove`) ratios are 0.01, 0.1, 0.5, 0.9 respectively.

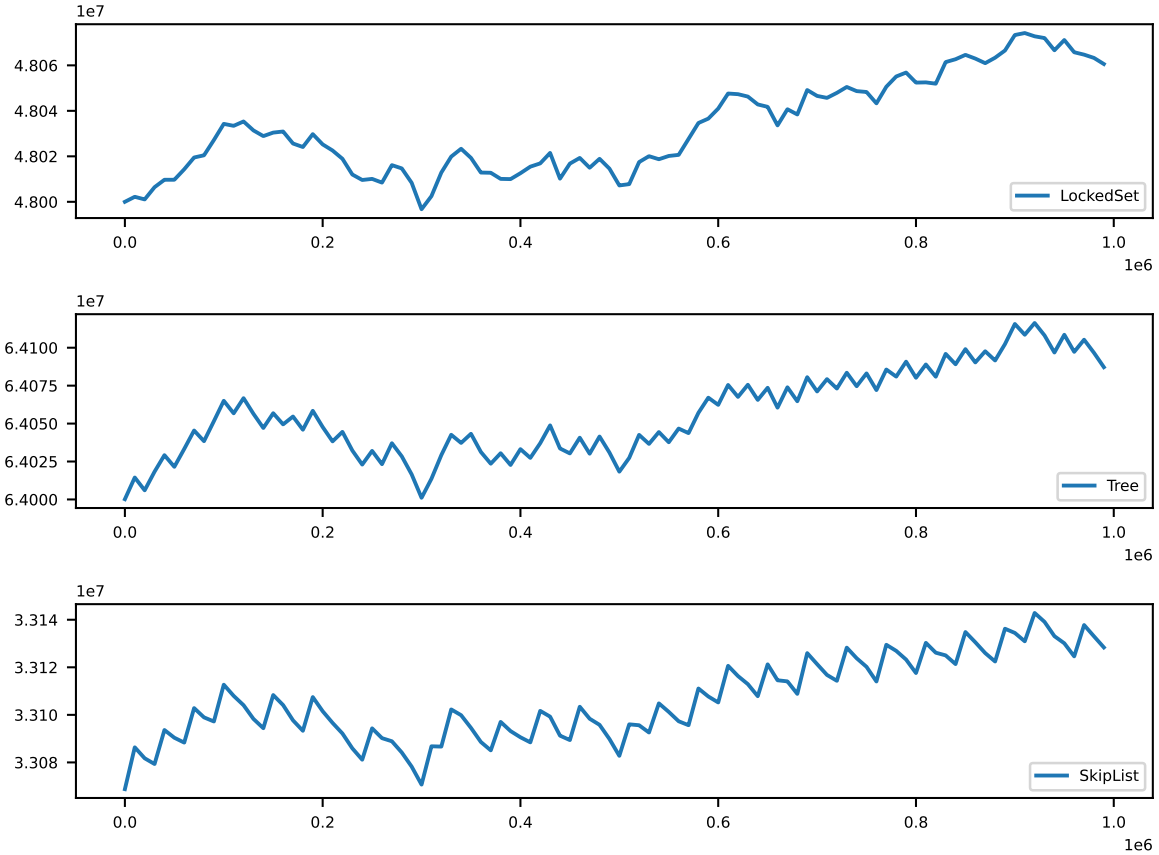


Performance

- `LockedSet` is the fastest when parallism is low and modifications are very few.
- `Tree` and `SkipList` can benefit from multithreading well, but more modifications also limit their acceleration.
- Overall, `Tree` is slightly faster than `SkipList` (average 15.88%).

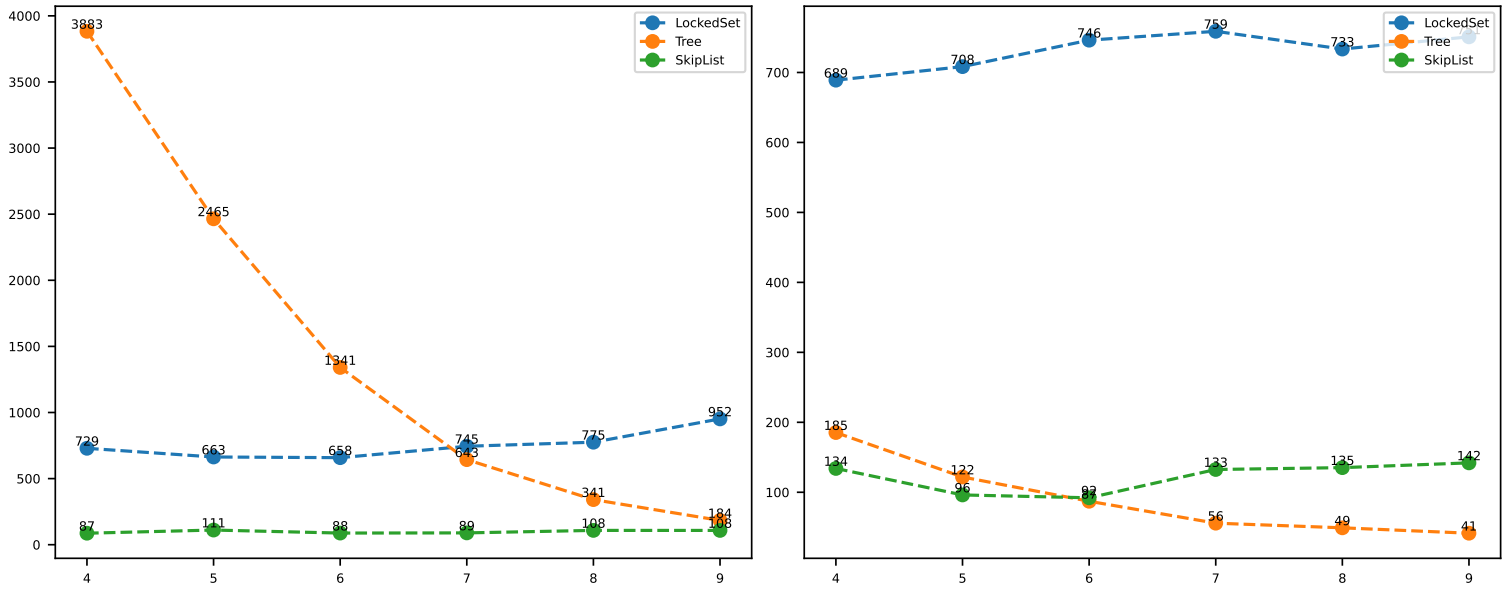
Memory Usage

- Intercept `malloc/free` and overload `new/delete`.
- `Tree > LockedSet > SkipList` in terms of memory usage.
- Some jitter in `Tree` and `SkipList` due to GC.



Imbalance Insertion

- Insert partially sorted data into dictionaries. Measure sortedness by the longest distance D between reverse pairs.
- `LockedSet` and `SkipList` are resistant to sorted data.
- `Tree` is significantly affected by the degree of sortedness.
- Parallel inserts may break the ordering of input data.



- If parallelism is low and the main workload is read, A simple STL data structure protected by a readers-writer lock gives the best performance.
- If parallelism is heavy, the lock-free binary search tree has slightly better performance while introducing many limitations.
- The lock-free skip list trades flexibilities for slightly lower performance, and is more useful in most systems.
- The amount of code of the two lock-free data structures is roughly the same, and the skip list is slightly less.

[1] K. Fraser. Practical lock-freedom. Technical report, University of Cambridge, Computer Laboratory, 2004.

[2] A. Natarajan and N. Mittal. Fast concurrent lock-free binary search trees. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, page 317328, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326568. doi: 10.1145/2555243.2555256. URL https://doi.org/10.1145/2555243.2555256.

Thanks!