15-418: Parallel Computer Architecture and Programming

Lock-Free Radix Trees for IP Routing

Damian Ho, Jessica Ruan

Background and Objective

Background
- Routing tables determine the next-hop for a packet based on its destination → find most specific matching route
- Often implemented with Radix Trees: space-optimized binary tries, O(k) per operation where k = # bits

Challenge
- Want to enable concurrent operations (lookups/updates) while ensuring that tree structure is never inconsistent
- Coarse-grained/fine-grained locking limits concurrency

Objective
- Write correct sequential implementation of Radix Tree
- Develop a high-performance CAS-based lock-free Radix Tree that supports concurrent lookups and updates
- Compare performance of lock-free radix tree to lock-based approaches

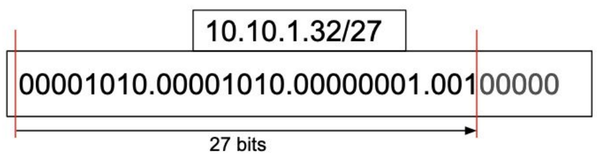
Benchmarks

Workload Configuration
- t: number of threads
- N: number of operations per thread
- r1: ratio of update (`Insert`/`Delete`) to lookup operations
- r2: ratio of `Insert` to `Delete`

Varying Update-to-Lookup Ratio
- Fix t = 8, N = 50000, r2 = 0.75
- Vary update-to-lookup ratio r1 from {0, 0.25, 0.5, 0.75, 1.00}
- Measure throughput as ops/μs

Varying Number of Threads
- Fix N = 50000, r1 = 0.25, r2 = 0.75
- Vary number of threads t from {1, 2, 4, 8}
- Measure throughput as ops/μs

Approach



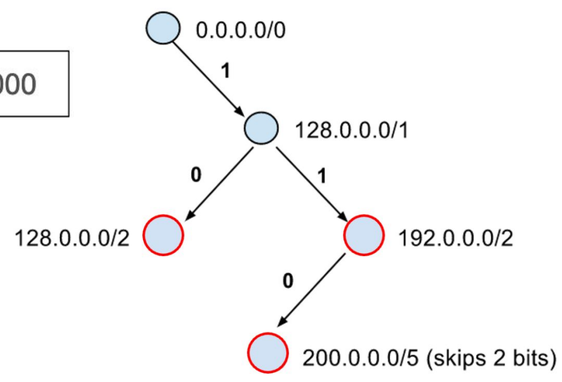


Figure 1 (Left): CIDR notation
Figure 2 (Right): Radix Tree

Routing Table API
- `Insert(ip, cidr, router)`: associates ip/cidr with router
- `Delete(ip, cidr)`: remove router associated with ip/cidr
- `Lookup(ip)`: find next-hop router for ip (best match)

Sequential Radix Tree
- Each node represents a route
- `Insert`: find insert point, create new nodes if needed

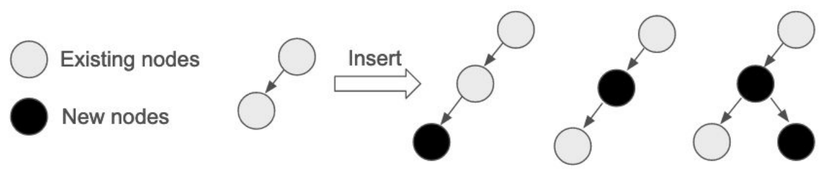


Figure 3: Possible outcomes of an `Insert` operation

- `Delete`: remove router, merge if only one child

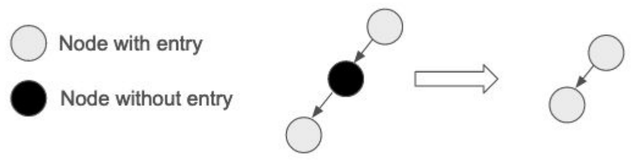


Figure 4: Space optimization: merge after removal

- `Lookup`: traverse trie, find deepest matching route with entry

Lock-Free Radix Tree
- Uses C++23's `atomic` library (`compare_exchange_strong`)
- Lower-order bits of child pointers store metadata
  - *flag*: node is undergoing removal
  - *freeze*: parent node is undergoing removal
- Metadata ensures that nodes can be safely removed
- Threads cooperate to complete removal operations

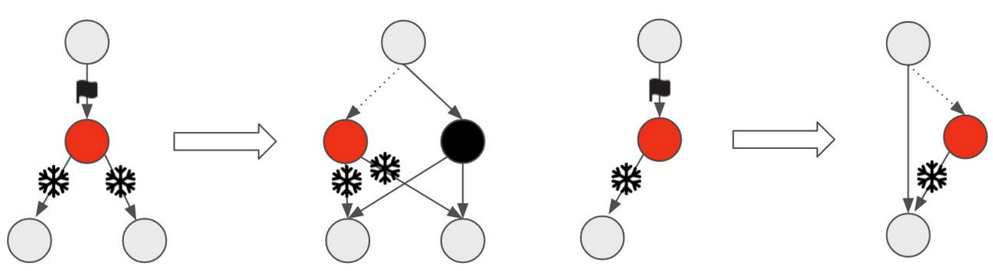
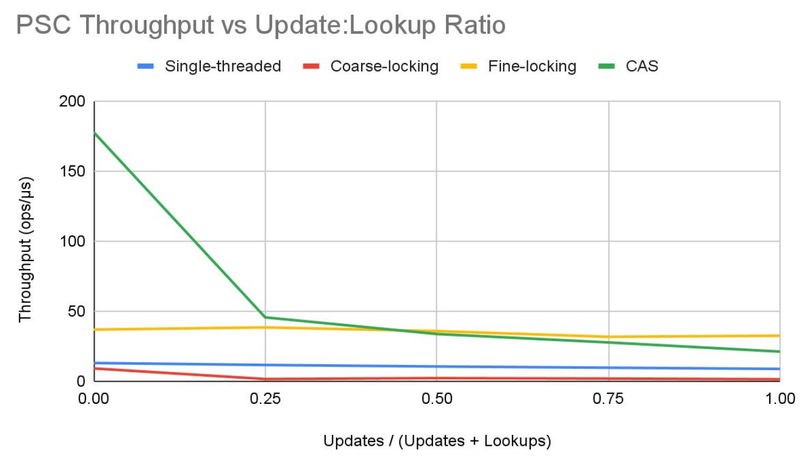


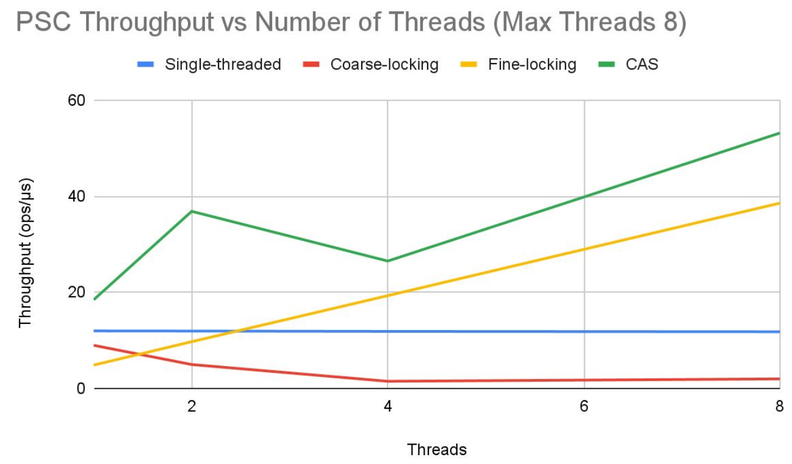
Figure 5: Possible outcomes when attempting to merge nodes

Results

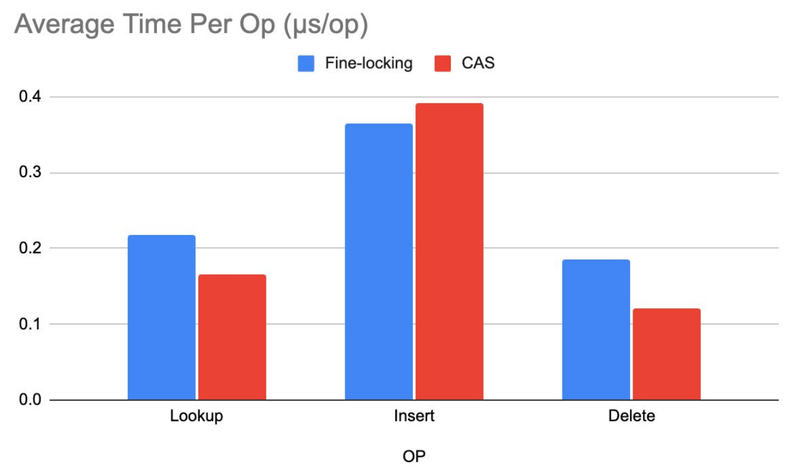
Varying Update-to-Lookup Ratio
CAS best for lookup-heavy workloads



Varying Number of Threads



Average Time per Operation



Conclusion

- Successfully implemented a CAS-based lock-free radix tree
- Lock-free radix trees shown to be viable for IP routing, best performance for lookup-heavy workloads
- Best alternative is the fine-grained locking radix trees, best performance for update-heavy workloads
- Sidestepped ABA problem by not reclaiming memory, possible solution is to use the read-copy-update (RCU) mechanism