Introduction

- Concurrent data structures require special treatment of deleted memory objects – garbage collectors are impractical in C/C++ and lack suitable progress/performance properties.
- Desirable properties for memory reclamation:
  - Non-blocking progress: not using locks
  - Robustness: bounding memory usage even when threads are stalled or preempted
  - Transparency: avoiding implicit assumptions about threads; threads can be created/deleted dynamically
- Snapshot-freedom: not taking snapshots of the global state to alleviate contention

Problem: how do we create a safe memory reclamation scheme which satisfies all these properties?

Hyalone’s Main Idea

- Use special reference counting, which is triggered only when deleting objects
- Update Head’s reference counter (HRef) when entering and leaving thread operations
- Append deleted objects to a global list and propagate reference counters
- When leaving, a thread traverses a sublist from the beginning to the object pointed to by a handle
- The handle points to the part of the list when the thread entered its operation
- Treat the very first list element specially: HRef rather than NRef reflects its reference counter
- When appending to the list, adjust the predecessor’s NRef (previously 0) with the HRef value
- Maintain multiple global lists to alleviate contention
  - Each list is for a subset of threads
  - Delete an entire batch of objects rather than just one object
  - One reference counter for the entire batch

Reference counting is very slow
- Robust schemes typically lack snapshot-freedom
- Hyalone-S has all the desirable properties while retaining good performance and reasonable memory overhead

Evaluation

- We tested Hyalone variants on x86(64), ARM32/64, PowerPC, and MIPS
- All Hyalone variants exhibit very high throughput on various data structures, and ensure that the number of retired, but not-yet-reclaimed objects is small
- Hyalone’s advantages are especially visible in certain read-dominated workloads
- In oversubscribed scenarios, Hyalone obtains up to 2× throughput boost
- We present results for x86-64, read-dominated tests (90% get, 10% put)

Availability and Acknowledgments

- Hyalone’s code and the benchmark are open-source and available at https://github.com/rusnikola/fsmr
- The work is supported by AFOSR under grants FA9550-15-1-0098 and FA9550-16-1-0371, and ONR under grants N00014-18-1-2022 and N00014-19-1-2493