Brief Announcement

Hyaline: Fast and Transparent Memory Reclamation

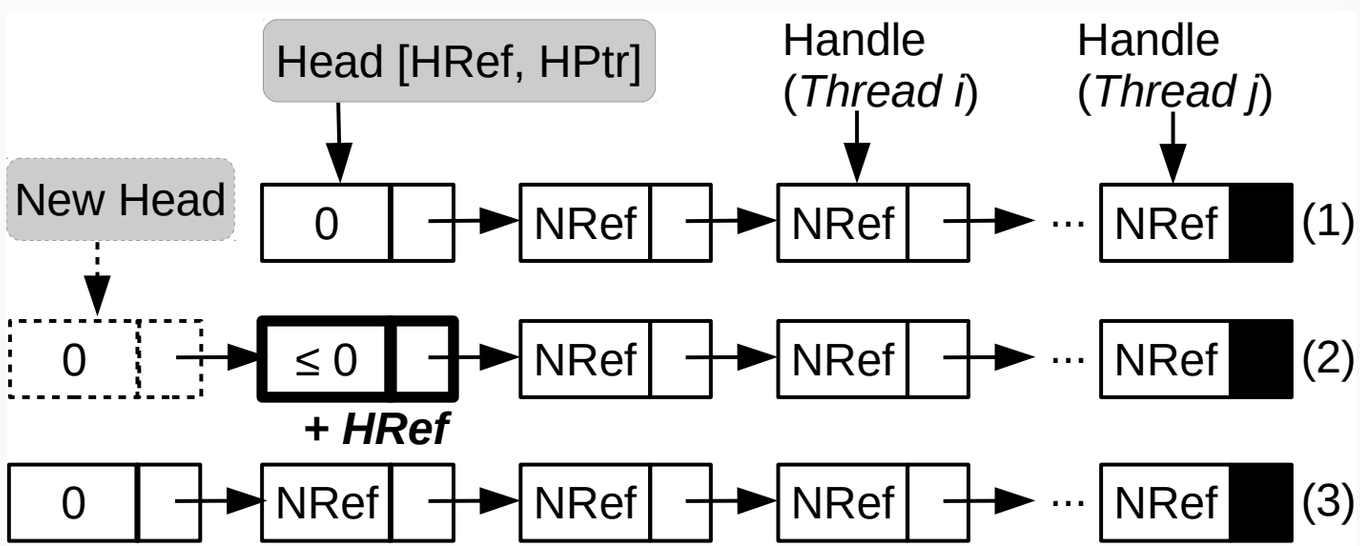
Ruslan Nikolaev and **Binoy Ravindran**
rnikola@vt.edu, binoy@vt.edu
Systems Software Research Group
Virginia Tech, USA

Memory Reclamation Problem

- Concurrent programming is hard
  - Non-blocking (lock-free) data structures require special treatment of deleted memory objects
  - Garbage collectors are often impractical in C/C++
- Desirable properties for memory reclamation
  - *Non-blocking*: protecting non-blocking data structures
  - *Robust*: bound memory usage even when threads are stalled or preempted
  - *Transparent*: avoid implicit assumptions about threads; they can be created/deleted dynamically

Hyaline

- General idea
  - Distributed reference counting, triggered only when deleting objects
  - Maintains multiple global lists of deleted objects
  - Each list is used by a subset of threads

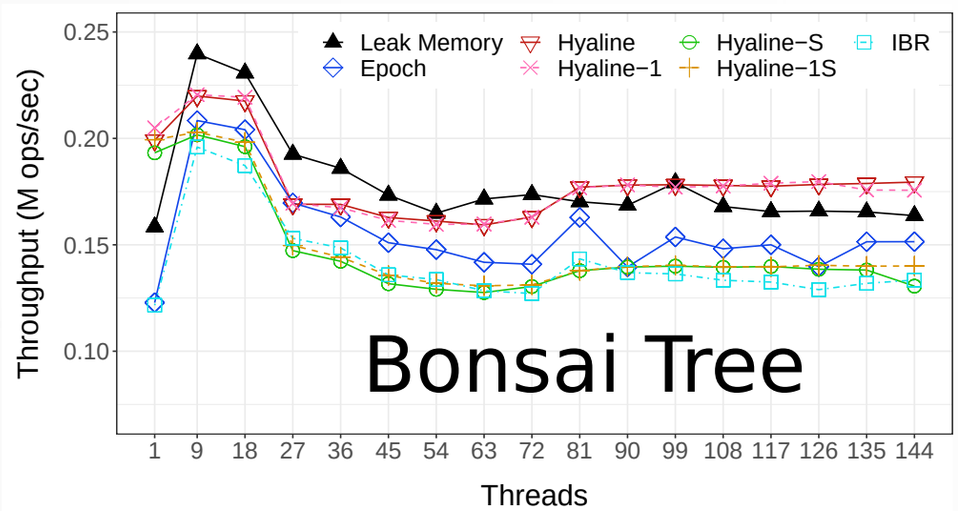


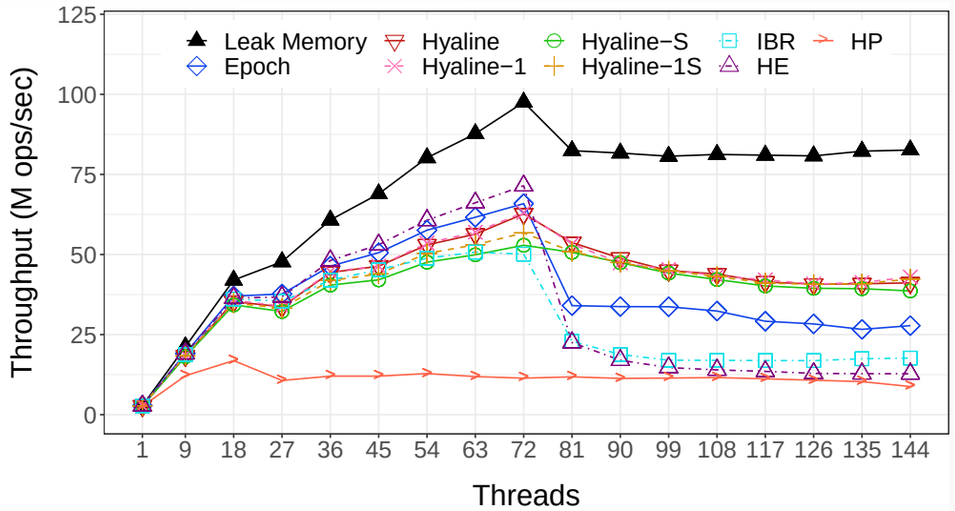
Comparison

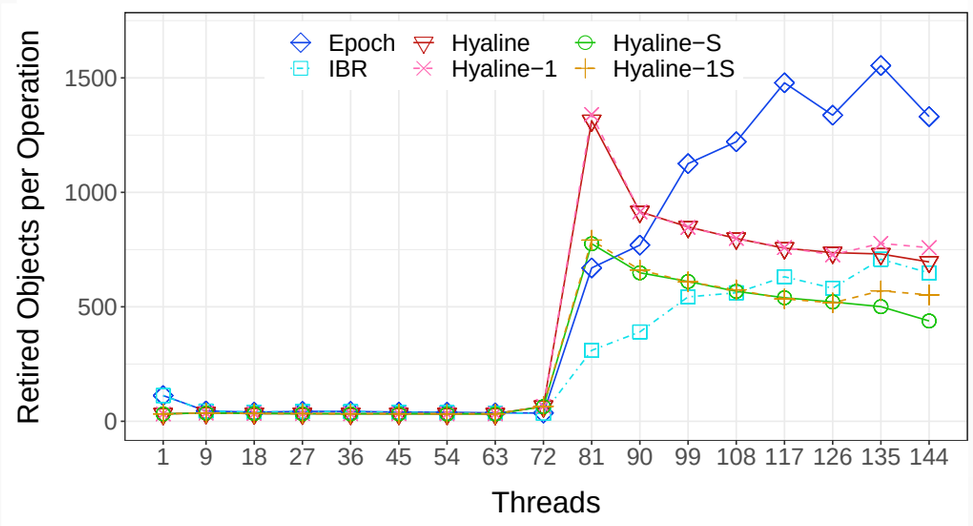
Scheme	Performance	Robust	Transparent	Extra Memory	API complexity
Reference Counting	Very Slow	Yes	Partially (swap)	Double each pointer	Intrusive
Hazard Pointers	Slow	Yes	No (deletion)	1 word	Hard
Epoch Based Reclamation	Fast	No	No (deletion)	1 word	Easy
Hazard Eras	Fast	Yes	No (deletion)	3 words	Hard
Interval Based Reclamation (2GEIBR)	Fast	Yes	No (deletion)	3 words	Medium
Hyaline	**Very Fast**	**No**	**Yes**	3 words	**Easy**
Hyaline-1	Very Fast	No	Almost	3 words	Easy
Hyaline-S	**Fast**	**Yes**	**Yes**	3 words	**Medium**
Hyaline-1S	Fast	Yes	Almost	3 words	Medium

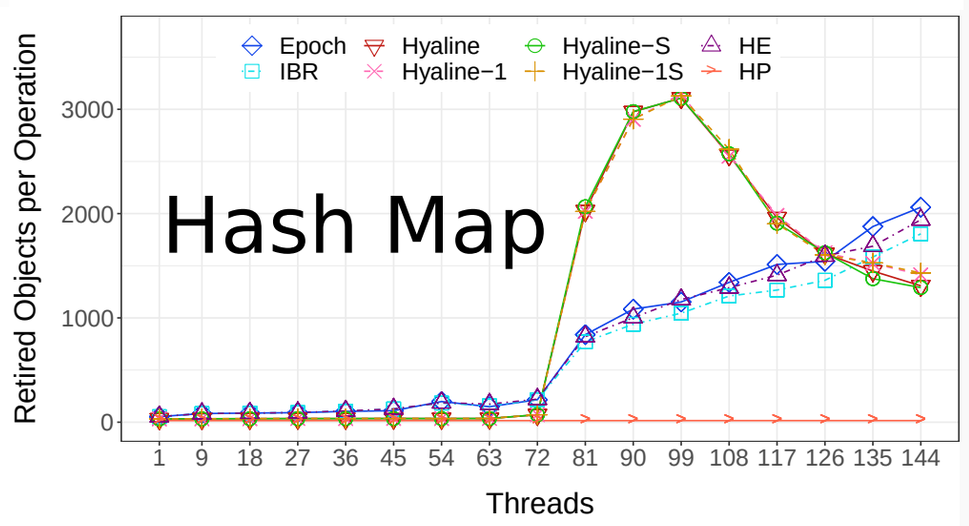
Evaluation

Xeon E7-8880 v3 2.30 GHz, 72 cores









More details

- Code is open-source and available at:
  - https://github.com/rusnikola/lfsmr
- Full paper is available as an arXiv report:
  - https://arxiv.org/pdf/1905.07903.pdf

Thank you!