Yak: A High-Performance Big-Data-Friendly Garbage Collector

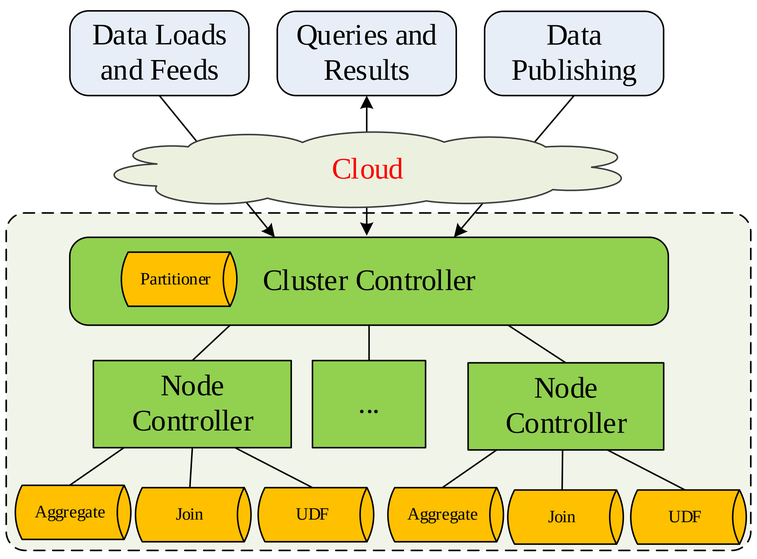
Khanh Nguyen

Advisor: Guoqing (Harry) Xu

University of California, Irvine

Motivation

Two paths, two hypotheses



Control Path

- ❖ Complicated logic
- ❖ Creates a small number of objects
- ❖ Generational Hypothesis holds
- ❖ State-of-the-Art Garbage Collectors are very efficient

Data Path

- ❖ Simple logic
- ❖ Creates most objects
- ❖ Insight: *Epochal Hypothesis*
- ❖ Suitable for Region-based Memory Management techniques

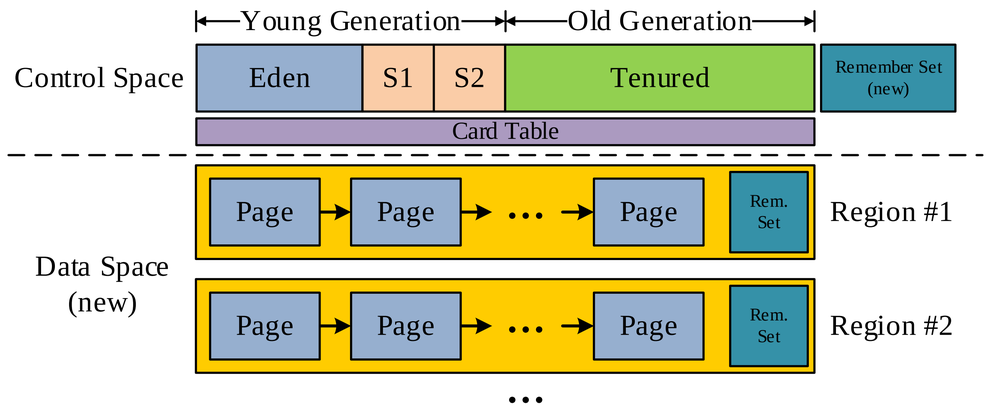
GOAL: REDUCE MEMORY MANAGEMENT COST

Challenge: Region-based Memory Management coexists in harmony with State-of-the-Art Garbage Collectors

Yak Approach

Issues		Solutions
• Huge manual effort to enable region-based techniques	➡	✓ Lightweight epoch annotations
• Escaping objects from Data Path to Control Path	➡	✓ Concurrent Promotion Algorithm
• Real-world adaptation	➡	✓ Oracle's JVM OpenJDK 8

Heap layout in Yak



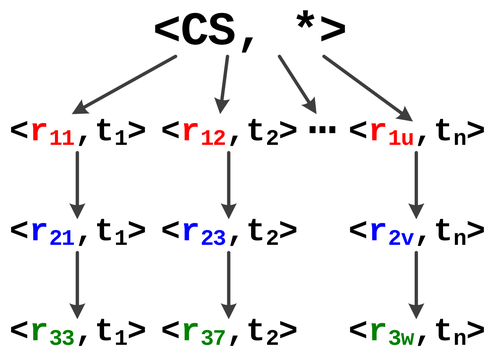
User annotations

```
for (…) {
  epoch_start();
  while (…) {
    epoch_start();
    for (…) {
      epoch_start();
      …
      epoch_end();
    }
    epoch_end();
  }
  epoch_end();
}
```

(r1 = outer epoch, r2 = middle epoch, r3 = inner epoch)

- ❖ Easy task for even novices
- ❖ Iterations are explicitly defined in existing systems

Region semilattice

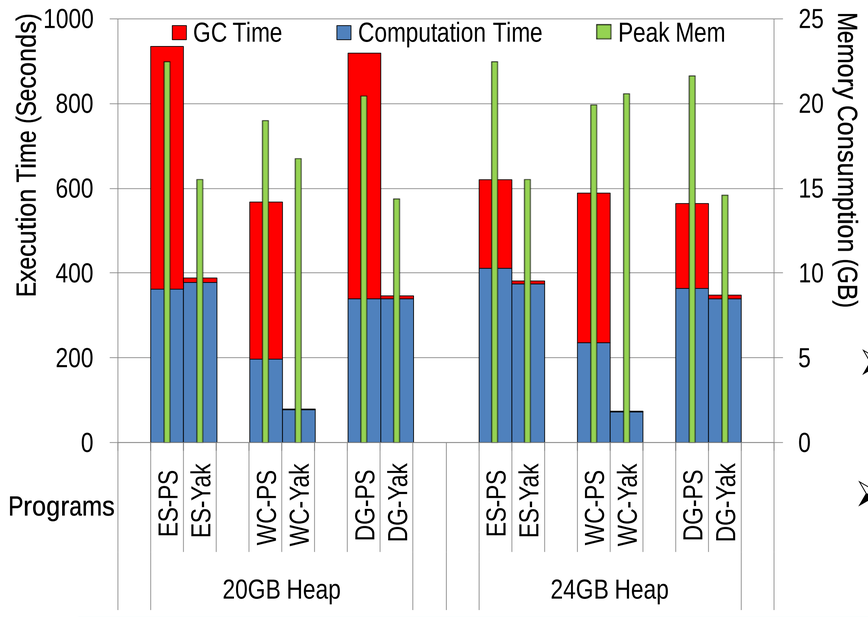


- ❖ Queried by Promotion Algorithm to relocate escaping objects

Implementation & Evaluation

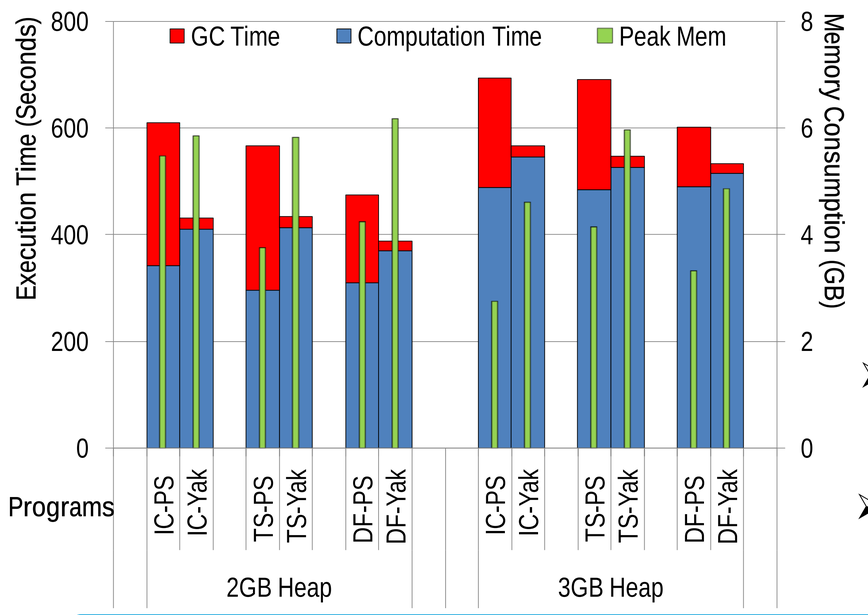
- ❖ Implemented in Oracle's production JVM OpenJDK 8 build 25.0-b70
- ❖ Across-the-stack modifications: the object and heap layout, the interpreter, two JIT compilers (C1 & Opto), the production Parallel Scavenge garbage collector

Hyracks [Borkar et al., ICDE'11]



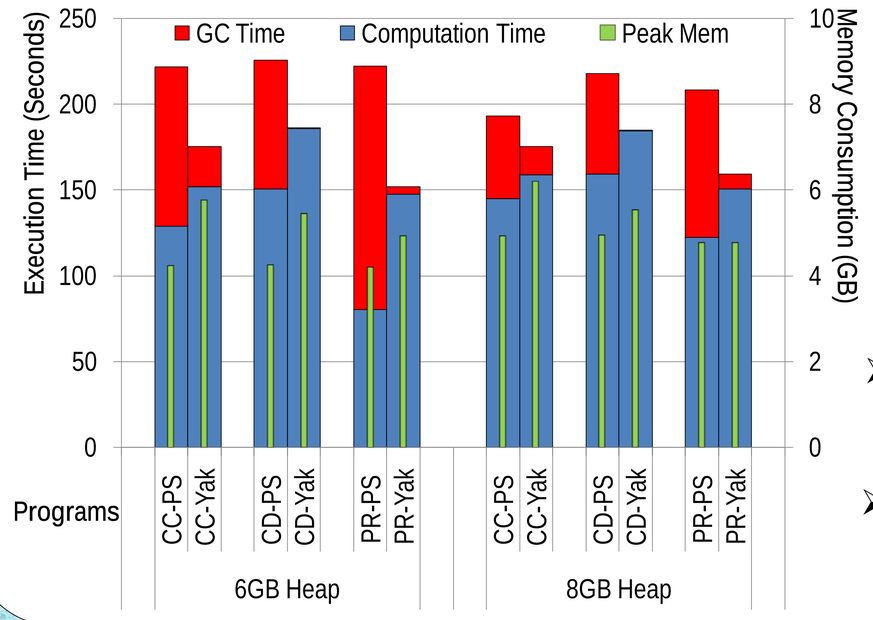
- ➢ 68.3X reduction in GC Time
- ➢ 58.6% reduction in Execution Time

Hadoop [Dean et al., OSDI'04]



- ➢ 14.1X reduction in GC Time
- ➢ 20.2% reduction in Execution Time

GraphChi [Kyrola et al., OSDI'12]



- ➢ 22.2X reduction in GC Time
- ➢ 18.4% reduction in Execution Time

Conclusions

- ❖ Yak is *novel*: the first hybrid garbage collector that employs generation- and region-based algorithms to automatically manage memory in Big Data systems
- ❖ Yak is *efficient*:
  - ✓ Outperforms the default production GC in OpenJDK: 35X GC cost reduction
  - ✓ Improves end-to-end performance: 32.4% savings
- ❖ Yak is *practical*: a JVM-based solution, applicable to all JVM-based languages while requiring almost zero user effort