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

Khanh Nguyen
Advisor: Guoqing (Harry) Xu
University of California, Irvine

Motivation
Two paths, two hypotheses

- Data Loads and Feeds
- Queries and Results
- Data Publishing

Cloud

Partitioner
Cluster Controller
Node Controller
Approp.
Job
CIF
Approp.
Job
CIF

Control Path
Data Path

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

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

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 & C2), the production Parallel Scavenge garbage collector

Hyacks [Borker 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

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

Goals: REDUCE MEMORY MANAGEMENT COST

Yak Approach

Issues
- Huge manual effort to enable region-based techniques
- Escaping objects from Data Path to Control Path
- Real-world adaptation

Solutions
- Lightweight epoch annotations
- Concurrent Promotion Algorithm
- Oracle’s JVM OpenJDK 8

Heap layout in Yak

Control Space

Young Generation
Old Generation

Temured

Shared Set (SS)

.Data Path

Page

Page

Page

Page

Region #1

Region #2

User annotations

for (...) {
    epoch_start();
    while (...) {
        epoch_start();
        for (...) {
            epoch_start();
            if (r1 < r2) {
                epoch_end();
            } else if (r1 < r3) {
                epoch_end();
            } else {
                // more cases...
            }
        }
        epoch_end();
    }
}

Region semilattice

<CS, *>

<CS, *> 

<CS, *> 

<CS, *> 

<CS, *> 

Queryed by Promotion Algorithm to relocate escaping objects

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

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