Modern Considerations of Garbage Collection in the Java Hotspot Virtual Machine

Jeffrey D. Lindblom

December 3, 2011

Object Oriented Programming Languages

- Objects operate as containers of data
- Objects refer to each other
- Notable object oriented languages:
  - Java
  - C#
  - Ruby

Live Objects versus Dead Objects

- Object reachability with a program
  - Reachable objects are *live*
  - Unreachable objects are *dead*
- The Garbage Problem
  - Memory is finite
  - Dead objects serve no purpose

The Java HotSpot Virtual Machine

The Java Virtual Machine (JVM)

- Runs the Java program
- Manages the object memory space
- Houses Garbage Collection

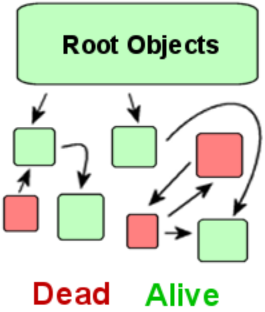
- The HotSpot™ implementation is developed by Oracle, formerly Sun

The Mark-and-Sweep Algorithm

Transitive reachability

If reachable Object A references Object B and Object B references Object C. Then Object C is reachable.

- The *Mark* Phase
  - Begins from root set
  - Traverses object references
- The *Sweep* Phase
  - Traverses all objects
  - Unmarked objects collected

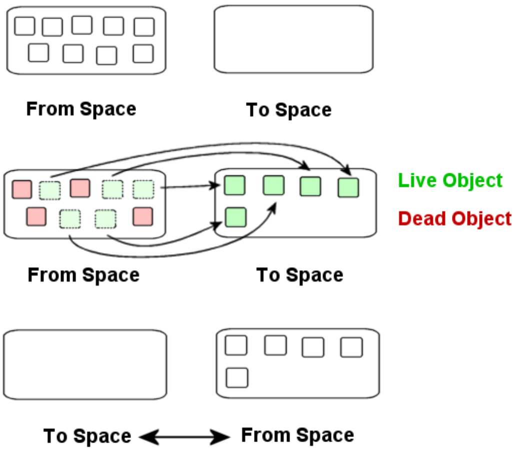


The Stop-and-Copy Algorithm

Initialization

Object Copy

Role Switch



Performance Considerations

- Frequent collection may cause processing overhead
- Infrequent collection may cause memory overhead
- Stop-the-World pauses
- Latency

Generational Garbage Collection

Three object life-time scenarios:

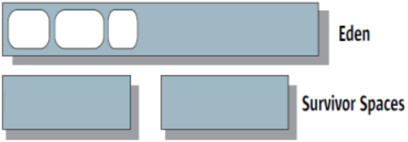
- Object dies soon after allocation
- Objects dies long after allocation
- Object never dies after allocation

Generation: A set of similarly aged objects

- Typically observed that most objects die young
- Java HotSpot generational garbage collection consists of:
  - The Young Generation
  - The Old Generation

The Young Generation

- The Young Generation consists of three spaces:



- Objects initially allocated to the Eden space
- When Eden space fills, *Minor Garbage Collection* occurs

Minor Garbage Collection

Uses Stop-and-Copy collection

- Efficient on small quantities of live objects
- Unaffected by large quantities of dead objects

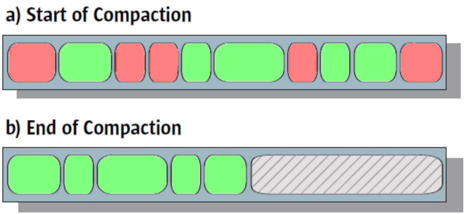
- Eden and one survivor spaces operate as From space
- Surviving objects receive a count
- Objects *tenured* to old generation when count meets threshold

The Old Generation

- Consists of just one space
- Slowly aggregates over time
- When space fills, *Major Garbage Collection* occurs

Major Garbage Collection

- By default uses Mark-Sweep-Compact algorithm
  - Requires less memory to operate
  - Defragments object space for allocation efficiency



Parallel Processing

Thread

A spawned process that is scheduled and functions independently of its parent.

- Single-threaded HotSpot Garbage Collectors:
  - Serial Collector
- Multi-threaded Generational HotSpot Garbage Collectors:
  - Throughput/Parallel Collector (Young Generation only)
  - Parallel Compacting Collector
  - Concurrent Mark-Sweep Collector

Parallel Young Generation Collection

- Uses parallel implementation of Stop-and-Copy

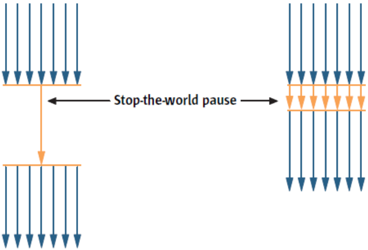


Figure: Single-threaded versus Multi-threaded

Parallel Old Generation Collection

Parallel Compacting Collector

- Uses parallel implementation of Mark-Sweep-Compact
- Incurs lower Stop-the-World pause-times

Concurrent Mark-Sweep Collector

- Uses concurrent implementation of Mark-and-Sweep
- Incurs even lower Stop-the-World pause-times
- Incurs high latency due to program CPU sharing

Real-time Applications

- Real-time applications operate within time-based deadlines:
  - E.g. military command-and-control operations, financial trading systems, on-the-fly audio processing

Three criteria of severity:

Strict	Missing a deadline compromises the entire application
Hard	Missing a deadline compromises that deadline result
Soft	Missing a deadline degrades that deadline result

- Collectors must have very low Stop-the-World pause times
- Only Java Hotspot collector to meet a real-time criteria is the *Garbage First Collector*, which satisfies Soft real-time processing.

Conclusion

- No ideal garbage collector

Main optimization criteria to consider:

- Low Stop-the-World pause times
- Low latency
- Processing power
- Memory capacity

- As computer architecture evolves, more leeway to delegate processing and memory footprints to the underlying computer system

Questions?

D. F. Bacon, P. Cheng, and V. T. Rajan.
A unified theory of garbage collection.
2004.

K. Barabash and E. Petrank.
Tracing garbage collection on highly parallel platforms.
2010.

S. M. Blackburn, P. Cheng, and K. S. McKinley.
Myths and realities: the performance impact of garbage collection.
2004.

D. Detlefs, C. Flood, S. Heller, and T. Printezis.
Garbage-first garbage collection.
2004.

T. Kalibera, F. Pizlo, A. L. Hosking, and J. Vitek.
Scheduling real-time garbage collection on uniprocessors.
2011.

S. Microsystems.
Memory management in the java hotspot™ virtual machine, 2006.

D. Vengerov.
Modeling, analysis and throughput optimization of a generational garbage collector.
2009.

B. Zorn.
Comparing mark-and sweep and stop-and-copy garbage collection.
1990.