Garbage Collection

Josh Leverette | 2017-7-20

Garbage Collection: what is it?

Any algorithm that performs dynamic run-time analysis of the heap to free unused memory

The Heap

- A region of memory often managed through free lists (basically linked lists)
- Memory is allocated by finding a large enough free block and dividing it up
- Slow, complex, and can become fragmented

The Stack

- A linear/sequential region of memory that holds each function call or “frame”
- Memory is allocated for local variables of a function when the function is called
- Simple and fast

(Non-)Compacting GC

Non-Compacting

- Lower latency because nothing is moved
- Simply marks unreachable objects for later reuse
- Heap becomes fragmented, especially with long-running processes
    - Cache misses more common
    - Free memory often can’t be released to the OS

Compacting

- Copies reachable objects to a sequential region of memory
- Updates all pointers to reachable objects to the new addresses
- Defragments the heap and improves cache-friendliness

Primary Types of Garbage Collection

Reference Counting

Mark and Sweep

Generational GC

Reference Counting

- Integer tag on each object

- Incremented each time a reference is created

- Decremented each time a reference is destroyed

- If zero, the object is immediately freed

Reference Counting

Pros:

- Pauseless
    - No Latency
    - Collection is done in real-time
- Simplest Algorithm

Cons:

- Reference cycles can leak memory
- Alloc/Free are expensive, some planning required

Mark and Sweep

Mark:

- Starts with known objects. (globals, locals)
- Marks them reachable
- Recursively follows pointers in reachable objects and marks them.

Sweep:

- Iterates through heap and frees the unreachable.

Mark and Sweep

Pros:

- Simple
- Easy to make concurrent and incremental
    - Reduces Latency / Shorter Pauses

Cons:

- Must mark every object, whether reachable or not
- Concurrent/Incremental
    - Increases CPU usage
    - Reduces throughput

Generational GC

"Most allocations die young"

Typically an enhanced version of Mark and Sweep. Each GC cycle checks young objects for unreachables.

As objects survive GC cycles, they are promoted to older generation pools

Only occasionally checks older objects for reachability

Generational GC

Pros:

- Less CPU than Mark/Sweep
- Higher throughput
- Typically compacting instead of fragmenting
- Alloc/Free is very cheap

Cons:

- Compacting requires higher latencies / longer pauses
- Long-lived allocs can sometimes be expensive

Some Collectors In Use

Reference Counting

Objective-C
Swift
Python
PHP

Mark and Sweep

<u>Non-compacting</u>
Go
Lua

<u>Compacting</u>
Java (CMS)

Generational

<u>Non-compacting</u>
Ruby

<u>Compacting</u>
Haskell
Erlang
C#
Java (G1)