Go GC: Prioritizing Low Latency and Simplicity

Rick Hudson
Google Engineer

QCon San Francisco
Nov 16, 2015
My Codefendants: The Cambridge Runtime Gang
Go: A Language for Scalable Concurrency

Lightweight threads (Goroutines)
Channels for communication
GC for scalable APIs
Simple Foreign Function Interface

Simplicity: The Key to Success
Go: A Language for Scalable Open Source Projects

Do Less, Enable More
Learning
Implementation
Tooling
Reading
Understanding

Sharing
Go: A Runtime for Scalable Applications

This is the story of Go’s garbage collector
Making Go Go: Establish A Virtuous Cycle

News Flash:
2X Transistors != 2X Frequency
More transistors == more cores
Only if software uses more cores

Long term
Establish a virtuous cycle

Short term
Increase Go Adoption

#1 Barrier: GC Latency
When is the best time to do a GC?

When nobody is looking.

Using camera to track eye movement
When subject looks away do a GC.

https://upload.wikimedia.org/wikipedia/commons/3/35/Computer_Workstation_Variables.png
Pop up a network wait icon

Waiting

https://commons.wikimedia.org/wiki/File:WIFI_icon.svg#globalusage
Or

Trade Throughput for Reduced GC Latency
Latency

Nanosecond
   1: Grace Hopper Nanosecond 11.8 inches

Microsecond
   5.4: Time light travels 1 mile in vacuum

Millisecond
   1: Read 1 MB sequentially from SSD
   20: Read 1 MB from disk
   50: Perceptual Causality (cursor response threshold)
   50+: Various network delays
Saccades (ms)
  30 Reading
  200 Involuntary

Eye Blink
  300 ms
Root Scan Phase

GC 101

Heap

Stacks/Registers

Globals
Mark Phase

Righteous Concurrent GC struggles with Evil Application changing pointers
Sweep Phase

Stacks/Registers

Globals
Go isn’t Java: GC Related Go Differences

**Go**
- Thousands of Goroutines
- Synchronization via channels
- Runtime written in Go
  - Leverages Go same as users
- Control of spatial locality
  - Objects can be embedded
  - Interior pointers (&foo.field)
- Simpler foreign function interface

**Java**
- Tens of Java Threads
- Synchronization via objects/locks
- Runtime written in C
- Objects linked with pointers

Let’s Build a GC for Go
1.4 Stop the World
1.5 Concurrent GC

- Application
- Assist
- GC

1 ms
3 ms
## GC Algorithm Phases

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
</table>
| Off              | GC disabled  
Pointer writes are just memory writes: *slot = ptr |
| Stack scan       | Collect pointers from globals and goroutine stacks  
Stacks scanned at preemption points |
| Mark             | Mark objects and follow pointers until pointer queue is empty  
Write barrier tracks pointer changes by mutator |
| Mark termination | Rescan globals/changed stacks, finish marking, shrink stacks, ...  
Literature contains non-STW algorithms: keeping it simple for now |
| Sweep            | Reclaim unmarked objects as needed  
Adjust GC pacing for next cycle |
| Off              | Rinse and repeat |

Correctness proofs in literature (see me)
Garbage Benchmark

GC Pauses vs. Heap Size

GC Pause (Lower is better)

Seconds

Heap Size (Gigabytes)
Garbage Benchmark

- 2x Live heap size

Graph showing the relationship between GC pause time and heap size for two versions of Go (Go 1.5 and Go 1.6).

- Y-axis: GC pause time
- X-axis: Heap size (in GB)
- Two sets of data points (orange and green) indicating performance differences between the two versions.
GOGC knob: Space-Time Trade off
More heap space: less GC time, and vice-versa

Implementing a one knob GC is a challenge
Splay: Increasing Heap Size == Better Performance

Execution Time (Lower is Better)

Heap Size (Megabytes):
Live heap kept constant

GOGC=200
JSON: Increasing Heap Size == Better Performance

Execution Time (Lower is Better)

Heap Size (Megabytes)

GOGC=200
Onward: We’re not done yet....

Tell people that GC latency is not a barrier to Go’s adoption

Tune for even lower latency
  higher throughput
  more predictability
Tune for user’s applications
Fight devils reported by users

Increase Go Adoption
Establish Virtuous Cycle
Questions