

Go GC: Latency Problem Solved

Rick Hudson Google Engineer

> GopherCon Denver July 8, 2015

My Codefendants: The Cambridge Runtime Gang



Google

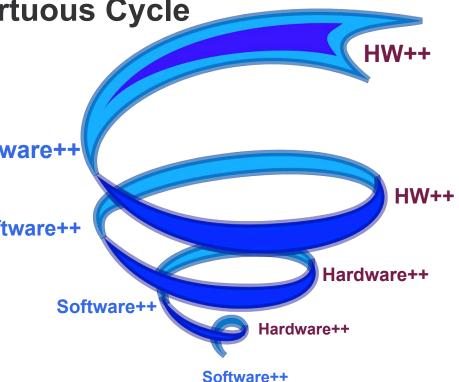


https://upload.wikimedia.org/wikipedia/commons/thumb/2/2f/Sato_Tadanobu_with_a_goban.jpeg/500px-Sato_Tadanobu_with_a_goban.jpeg

Making Go Go: Establish A Virtuous Cycle

News Flash: 2X Transistors != 2X Frequency More transistors == more cores Software++ Only if software uses more cores Software++ Long term Establish a virtuous cycle Short term

Increase Go Adoption

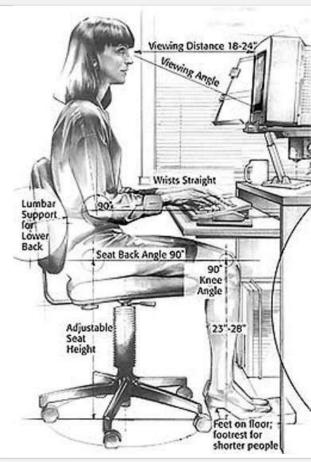


#1 Barrier: GC Latency

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.





Pop up a network wait icon



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

Or Trade Throughput for Reduced GC Latency

Google

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
- 300: Eye blink



Go isn't Java: GC Related Go Differences

Go

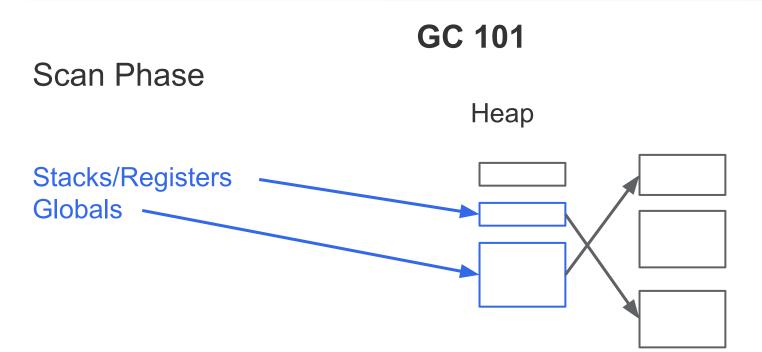
Google

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) Java Tens of Java Threads Synchronization via objects/locks Runtime written in C

Objects linked with pointers

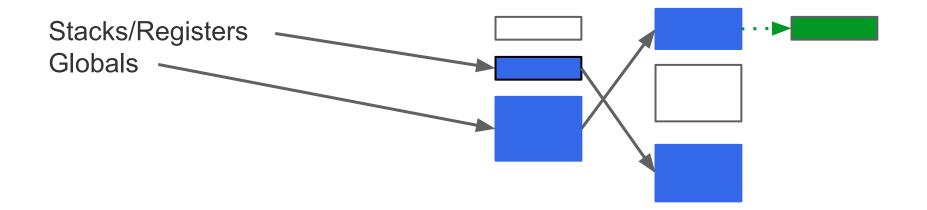
Let's Build a GC for Go







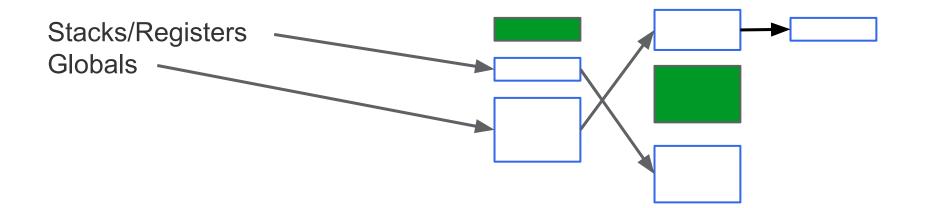
Mark Phase



Righteous Concurrent GC struggles with Evil Application changing pointers



Sweep Phase



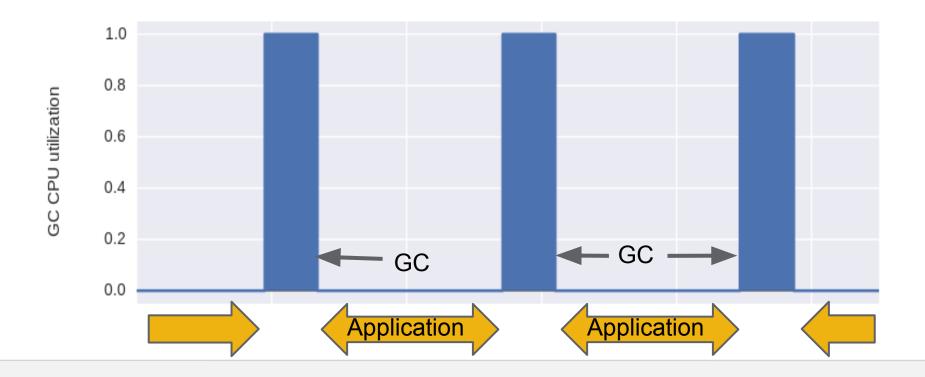
GC Algorithm Phases

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		Mark objects and follow pointers until pointer queue is empty Write barrier tracks pointer changes by mutator
Mark termination	STW	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)

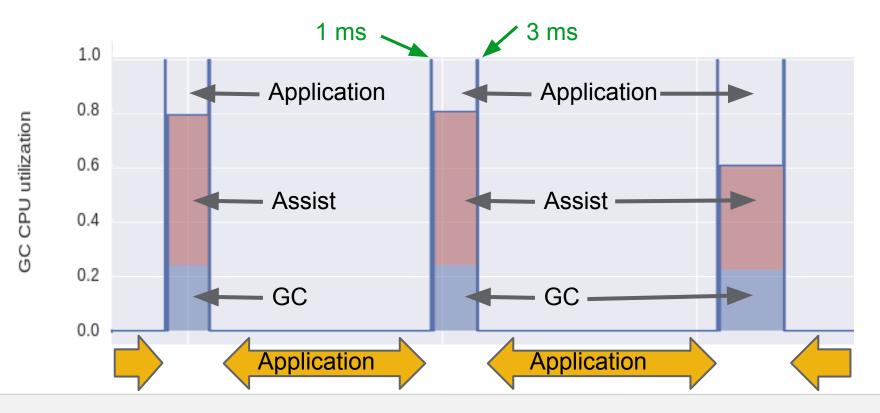
Google

1.4 Stop the World



Google

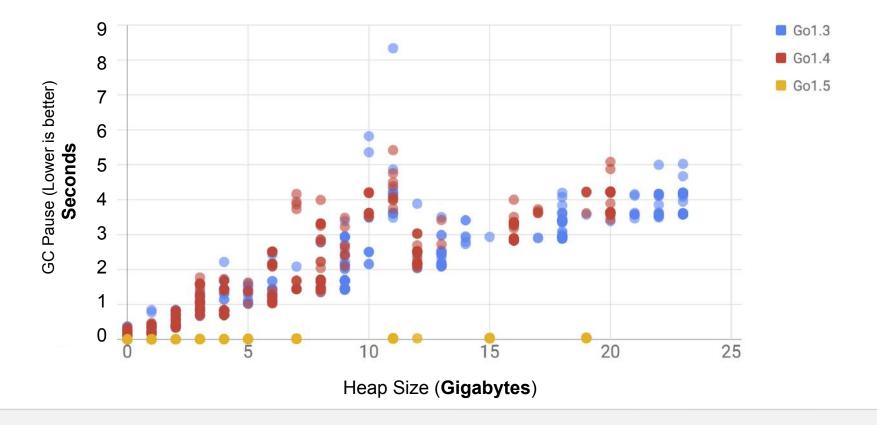
1.5 Concurrent GC





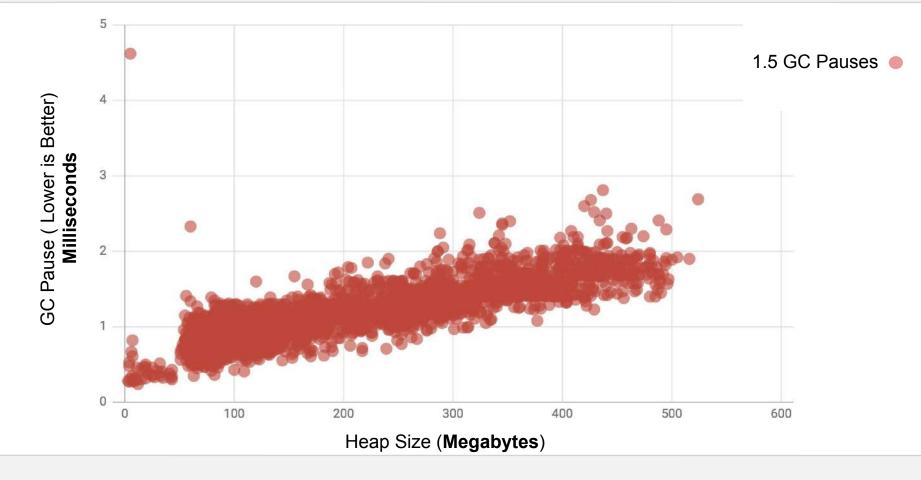
Garbage Benchmark

GC Pauses vs. Heap Size

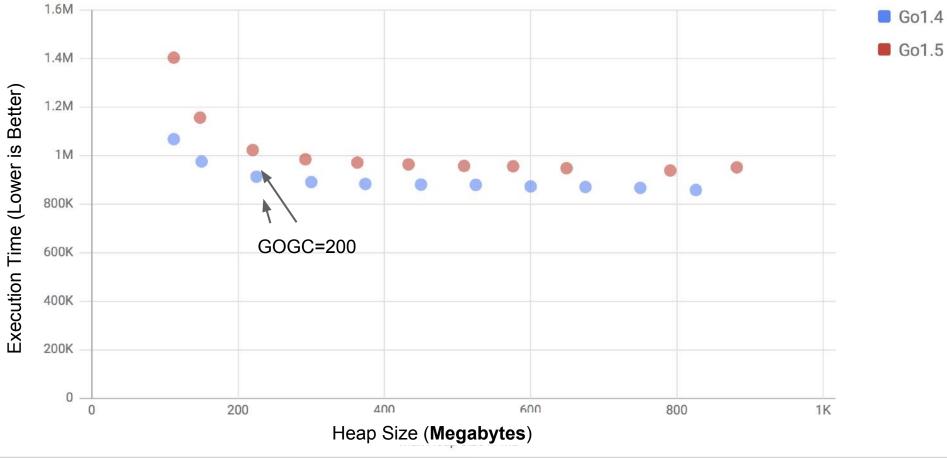




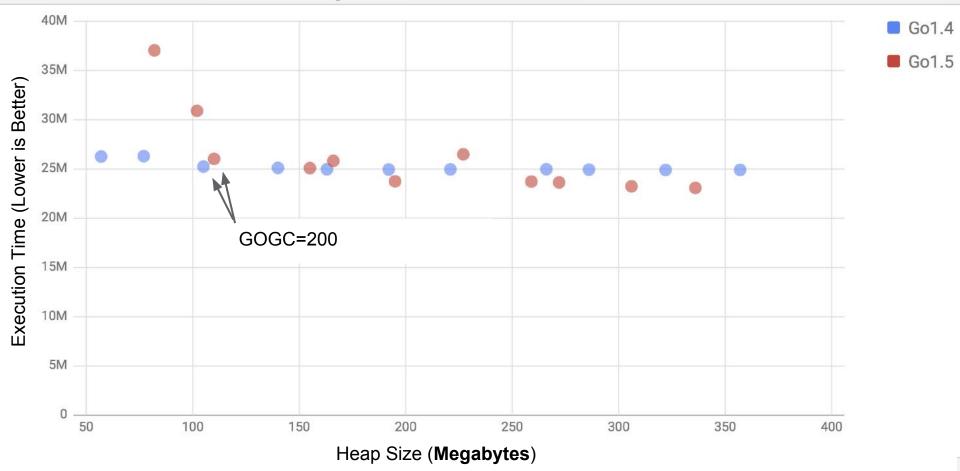
1.5 Garbage Benchmark Latency



Google Splay: Increasing Heap Size == Better Performance



Google JSON: Increasing Heap Size == Better Performance



Onward

Tell people that GC is not a barrier with Go's low latency GC

Tune for even lower latency, higher throughput, more predictability Find the sweet spot.

1.6 work will be use case driven: Let's talk.

> Increase Go Adoption Establish Virtuous Cycle



Questions