Profiling in Go

Boston Golang 3/3/2020

Bjoern Poetzschke
@kaenplan

Agenda

- **What is profiling?**
- **Collecting profiles**
- **Analyze profiles**
- **Profile types**
- **Execution tracer**

Photo by JESHOOTS.COM on Unsplash

Repository

http://bit.ly/profiling-bos-go

What is profiling?

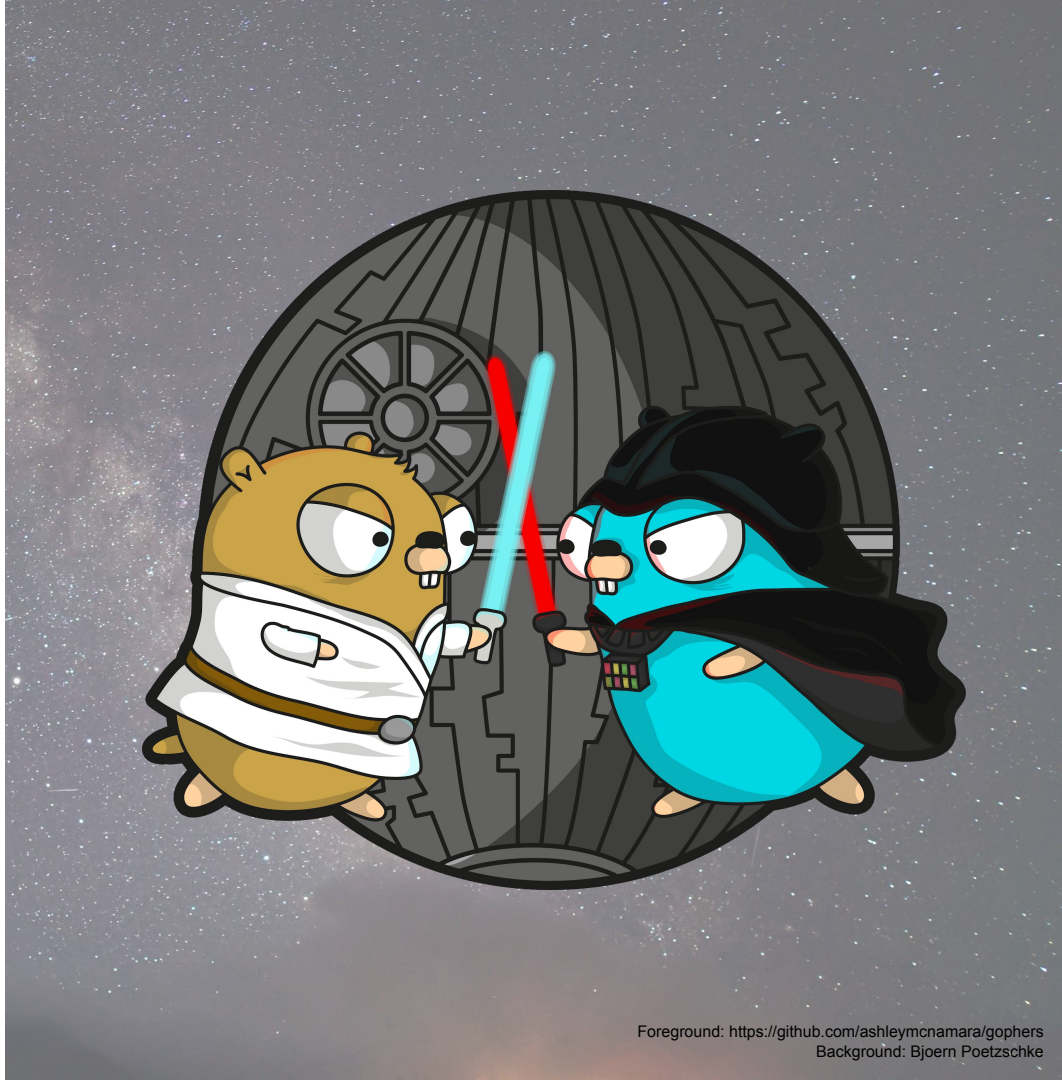
What is profiling?

- **Definition:**
  - Profiling is the dynamic analysis of a running software application
- **Improve speed and / or memory usage**
- **Profiling helps to understand:**
  - Number of allocations
  - Size of allocations
  - Slow method calls
  - Blocking code
- **Expensive**

What is profiling?

- **Sampling**
  - Data collected at certain points
  - Things can be missed
  - Less expensive

- **Tracing**
  - Data is collected continuously
  - Everything is collected
  - Very expensive



Foreground: https://github.com/ashleymcnamara/gophers
Background: Bjoern Poetzschke

Collecting profiles

Collecting profiles

- **runtime/pprof package**
  - low level tool
  - different interfaces to configure profiling
- **github.com/pkg/profile**
  - Convenience wrapper for runtime/pprof package
  - `defer profile.Start(ProfileType, Path).Stop()`
- **pprof is included in testing package**
- **net/http/pprof package**
  - Collection via http interface
  - Sampling time can be specified
  - https://golang.org/pkg/net/http/pprof/

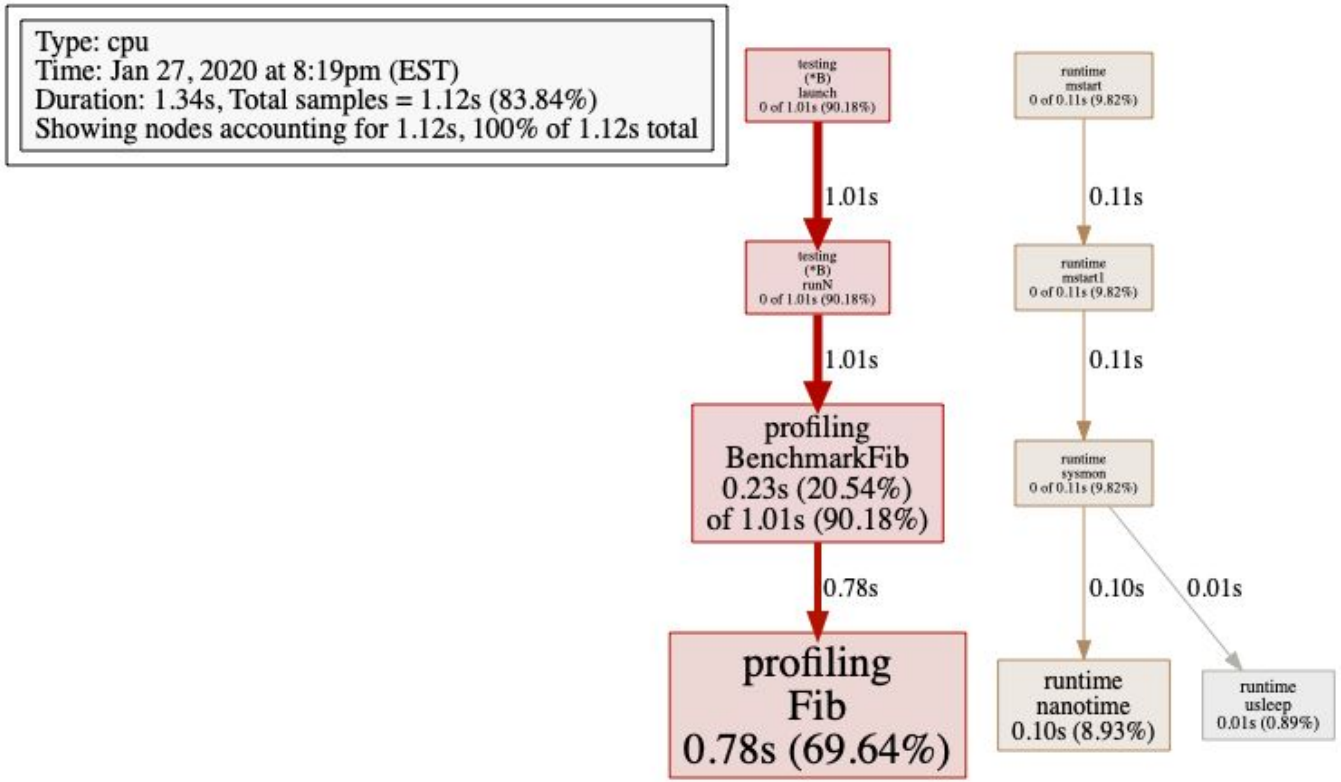
Analyze profiles

Analyze profiles

- **Profiles can be analyzed by pprof subcommand**
  - `go tool pprof`
    - `/path/to/profile`
    - `http://localhost:6060/debug/pprof/heap`
  - `go tool pprof -http=:8080`
    - provides web interface
- **Useful commands inside pprof:**
  - Top
  - topN
  - list regexp
  - web / svg/ pdf / png / gif
- **Useful tools inside web interface**
  - Flame Graph
  - Graph
  - Source
  - Top

Analyze profile

Graph



Analyze profile

Top

```
bjoern@Caspar ~/repos/playground/profiling go tool pprof cpu.profile
Type: cpu
Time: Jan 27, 2020 at 8:19pm (EST)
Duration: 1.34s, Total samples = 1.12s (83.84%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top
Showing nodes accounting for 1.12s, 100% of 1.12s total
      flat  flat%   sum%        cum   cum%
     0.78s 69.64% 69.64%      0.78s 69.64%  playground/profiling.Fib
     0.23s 20.54% 90.18%      1.01s 90.18%  playground/profiling.BenchmarkFib
     0.10s  8.93% 99.11%      0.10s  8.93%  runtime.nanotime
     0.01s  0.89%   100%      0.01s  0.89%  runtime.usleep
         0     0%   100%      0.11s  9.82%  runtime.mstart
         0     0%   100%      0.11s  9.82%  runtime.mstart1
         0     0%   100%      0.11s  9.82%  runtime.sysmon
         0     0%   100%      1.01s 90.18%  testing.(*B).launch
         0     0%   100%      1.01s 90.18%  testing.(*B).runN
(pprof)
```

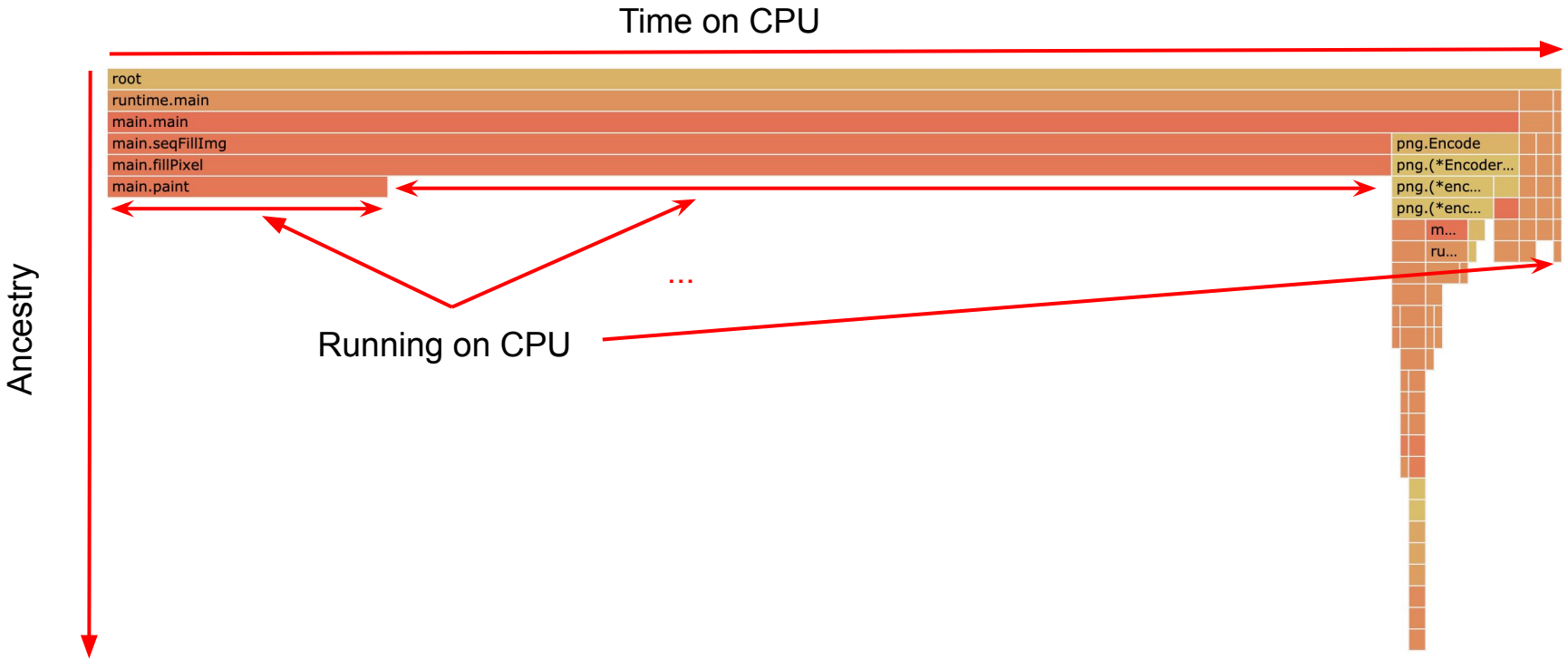
Analyze profile

List

```
bjoern@Caspar ~/repos/playground/profiling go tool pprof cpu.profile
Type: cpu
Time: Jan 28, 2020 at 2:52pm (EST)
Duration: 1.23s, Total samples = 990ms (80.18%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) list profiling.Fib
Total: 990ms
ROUTINE ======================== playground/profiling.Fib in /Users/bjoern/repos/playground/profiling/fib.go
     780ms      780ms (flat, cum) 78.79% of Total
         .          .     24:   }
         .          .     25:}
         .          .     26:
         .          .     27:func Fib(n int) int {
         .          .     28:   a, b := 0, 1
     470ms      470ms     29:   for i := 0; i < n; i++ {
     170ms      170ms     30:           a, b = b, a+b
         .          .     31:   }
     140ms      140ms     32:   return a
         .          .     33:}
(pprof)
```

Analyze profile

Flamegraph



Profile types

Profile types

- **CPU profile**
- **Memory profile**
- **Block profile**
- **Mutex profile**

Profile types

CPU

- **Most common profile**
- **10 ms sampling interval**
- **The more times a function appears in the sample the more time it takes on the runtime**
- **Shows how much time is spent in a specific function**
- **Usage:**
  - go test package: `go test [-bench=...] -cpuprofile`
  - profile package: `defer profile.Start(profile.CPUProfile, profile.Path(“.”)).Stop()`

Profile types

CPU

- **Example**
  - https://github.com/bpoetzschke/go_profiling/cpu

Profile types

Memory

- **Records the stack trace when a heap allocation is made**
- **1 sample / 1000 allocations**
- **Not recommended to find memory leaks**
- **Comes in two varieties**
  - alloc_objects / alloc_space
    - Number / Size of total bytes
  - inuse_objects / inuse_space
    - Number / Size of live bytes
- **Usage:**
  - go test package: `go test [-bench=...] -memprofile`
  - profile package: `defer profile.Start(profile.MemProfile, profile.Path(".")).Stop()`

Profile types

Memory

- **Example**
  - https://github.com/bpoetzschke/go_profiling/memory

Profile types

Block

- **Good for determining concurrency bottlenecks**
- **It shows you when goroutines could be faster but they were blocked**
- **Only as last resort**
- **Note: Needs to be enabled**
- **Usage:**
  - go test package: `go test [-bench=...] -blockprofile`
  - profile package: `defer profile.Start(profile.BlockProfile, profile.Path(“.”)).Stop()`

Profile types

Block

- **Example**
  - https://github.com/bpoetzschke/go_profiling/block

Profile types

Mutex

- **Similar to Block profiling**
- **Shows how much time was spent for waiting for locks**
- **Helps to reduce mutex contention**
- **Note: Needs to be enabled**
- **Usage:**
  - go test package: `go test [-bench=...] -mutexprofile`
  - profile package: `defer profile.Start(profile.MutexProfile, profile.Path(“.”)).Stop()`

Profile types

Mutex

- **Example**
  - https://github.com/bpoetzschke/go_profiling/mutex

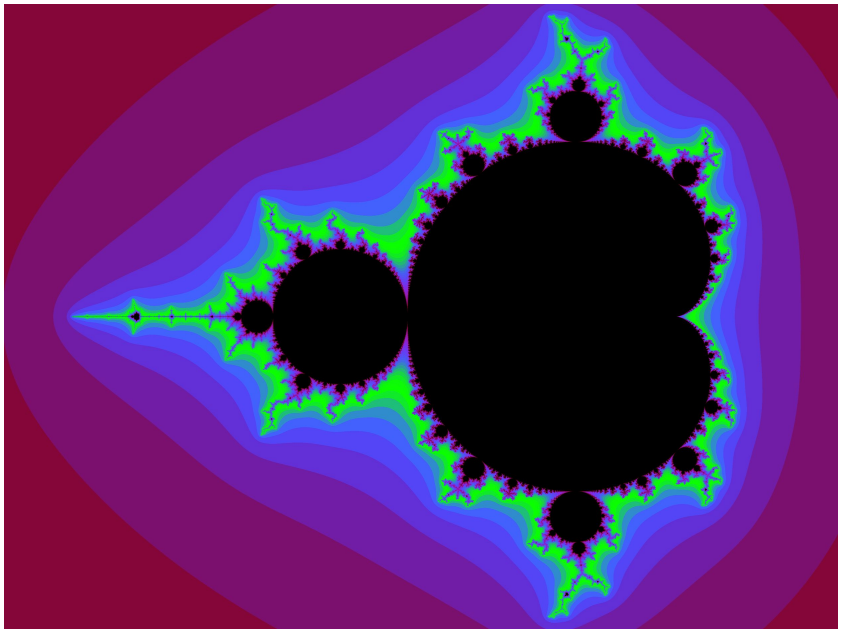
Execution tracer

Execution tracer

- **Captures a wide range of execution events:**
  - Goroutine information
  - heap allocations
  - CPU utilization
  - GC
- **Syscalls / channels / lock**
- **Provides nanosecond precision**
- **Helps finding contention when running concurrent applications**
- **Requires Chrome Browser < Version 80!**
- **Usage**
  - go test package: `go test [-bench=...] -trace`
  - profile package: `defer profile.Start(profile.TraceProf, profile.Path(".")).Stop()`
  - Analyze trace: `go tool trace`

Execution tracer

- **Mandelbrot Example:**
  - https://github.com/bpoetzschke/go_profiling/tracing



Source: https://github.com/ashleymcnamara/gophers

Resources

- **https://dave.cheney.net/high-performance-go-workshop/gophercon-2019.html**
- **https://jvns.ca/blog/2017/09/24/profiling-go-with-pprof/**
- **https://www.youtube.com/watch?v=N3PWzBeLX2M**
- **https://docs.google.com/presentation/d/1n6bse0JifemG7yve0Bb0ZAC-lWhTQjCNAclbInn2ANY/present?slide=id.g10679bf2dc_0_49**
- **https://matoski.com/article/golang-profiling-flamegraphs/**
- **https://rakyll.org/mutexprofile/**
- **https://segment.com/blog/allocation-efficiency-in-high-performance-go-services/**
- **https://flaviocopes.com/golang-profiling/**
- **https://golang.org/pkg/runtime/pprof/#Profile**