Introduction

● Matej Gera
● Software Engineer @ Coralogix
● Maintainer @ Thanos

● GitHub: @matej-g
● Twitter: @gmat90
Three things today

- Understanding Go strings behind the scenes
- String use cases prone to performance bottlenecks
- Optimization strategies
Inspiration behind this talk

- (Data-driven) performance optimization
  - Working on Thanos project (distributed time-series database)

Thanos

Open source, highly available Prometheus setup with long term storage capabilities.
Inspiration behind this talk

- (Data-driven) performance optimization
  - Don’t miss talk from Bartek Plotka today!

Five Steps to Make Your Go Code Faster & More Efficient

- Track: Go devroom
- Room: UD2.218A
- Day: Saturday
- Start: 15:00
- End: 15:30
- Video with Q&A: We've hit a snag. The Video only link still works!
- Video only: We're not quite ready yet
- Chat: We've hit a snag. The Video only link still works!
Inspiration behind this talk

- Focus on strings
Inspiration behind this talk

- **Focus on strings**

- **Source:** *PromCon EU 2022: Why Is It so Big? Analysing the Memory Consumption of Prometheus* by Bryan Boreham
Strings behind the scene

- Immutable, can be converted to []byte, concatenable, slicable...

- But strings are not “just” strings

- Runtime representation of strings (/src/runtime/string.go):
Strings behind the scene

- In actuality, it’s slice of bytes
- Size stays the same during lifetime (remember, immutable)
- Size of string will correspond to
  - String header overhead (16 bytes) + actual string (length of the slice of bytes)

```go
// 'FOSDEM 👋' string size
str := "FOSDEM 👋"
reflect.TypeOf(str).Size() // 16 bytes (8 bytes pointer, 8 bytes len)
len(str) // 11 bytes
```
Strings behind the scene

- Copying string will create shallow copy
  - But results in a new string header!

```go
str := "FOSDEM 🎊"
// `newStr` will reference 'FOSDEM 🎊' from `str`
newStr := str

fmt.Printf("%p\n", &str)        // 0xc000014240

// That's a brand new string header - another 16 bytes!
fmt.Printf("%p\n", &newStr)    // 0xc000014250
```
The problem zone

- In-memory stores
  - Can result in large number of strings being stored (billions)
  - Potential for repetition of strings (e.g. metadata, labels)
    - cluster=us-prod-1
  - Handling of incoming data
    - Often involves unmarshalling into structs
    - Strings from the request might be kept in memory long term
    - Garbage collection?
The problem zone

- One-off data processing
  - Documents that might require decoding (JSON, YAML)
  - Repeated keys
Optimization strategies

- Detaching strings from larger memory pools
  - To make sure we keep around only string
  - Rest of the struct can be garbage collected
  - Can be achieved by “detaching” of the string
  - This can be achieved by using `strings.Clone(s string) string`
    - Since Go 1.18
Optimization strategies

- **String interning**
  - Technique to store only one single copy of each distinct string value
  - At simplest, can be achieved by storing values in a `map[string]string`{} 
  - Each reference carries the string header overhead (16 bytes)
  - How to know when to drop a string from interning map?
    - Won't be garbage collected as long as map is around (possible DoS vector)
    - Possible solutions:
      - Periodically remove entries (akin to clearing cache)
      - Count references (see example: prometheus/prometheus/pull/5316)
Optimization strategies

- **String interning the “dark arts” way**
  - What if the unused string references could be dropped “automagically”?
  - Implementation in go4.org/intern
    - Enter the concept of finalizers
  - Boxes the interned values (string header) into a single pointer
    - 16 bytes -> 8 bytes overhead
Optimization strategies

- String interning the “dark arts” way

  What

  Package intern lets you make smaller comparable values by boxing a larger comparable value (such as a string header) down into a single globally unique pointer.

  Docs: https://pkg.go.dev/go4.org/intern

  Status

  This package is mature and stable. However, it depends on the implementation details of the Go runtime. Use with care.

  This package is a core, low-level package with no substantive dependencies.

  We take code review, testing, dependencies, and performance seriously, similar to Go’s standard library or the golang.org/x repos.
Optimization strategies

- **String interning the “dark arts” way**
  - What if the unused string references could be dropped “automagically”?
  - Implementation in [go4.org/intern](http://go4.org/intern)
    - Enter the concept of [finalizers](http://finalizers)
  - Boxes the interned values (string header) into a single pointer
    - 16 bytes -> 8 bytes overhead
  - Example of use: [thanos-io/thanos/pull/5926](https://thanos-io/thanos/pull/5926)
Optimization strategies

- String interning the "dark arts" way
  - What if the unused string references could be dropped "automagically"?
  - Implementation in go4.org/intern
  - Enter the concept of finalizers
  - Example of use: thanos-io/thanos/pull/5926
Optimization strategies

- **String interning with symbol table**
  - Structure with key-value pairs to lookup strings
  - E.g. each int will correspond to given unique string
  - Can be beneficial in scenarios with lot of duplicate strings
    - to decrease network costs and number of allocations
  - Example: thanos-io/thanos/pull/5906
Optimization strategies

- String concatenation
  - Combining strings into single bigger backing string
  - Saves the overhead of each string header
  - Requires look up of individual strings within the structure
  - Example: prometheus/prometheus/pull/10991
Conclusion

- Still a balancing act (memory vs CPU)
- More empirical data needed
Thank you for your attention!
More useful resources:

- https://go101.org/article/string.html
- https://commaok.xyz/post/intern-strings/
- https://crawshaw.io/blog/tragedy-of-finalizers