Current state and challenges

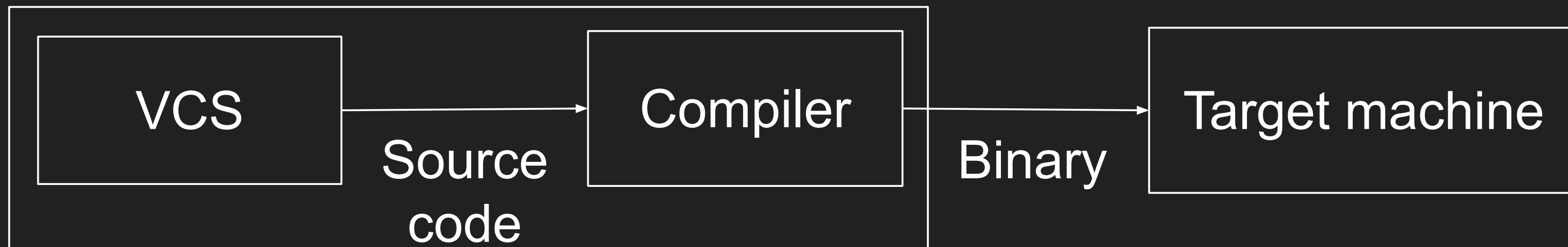
Few words about me

- Used to be a C++ engineer (now my C++ skills are Rust-y ;), later worked as an architect
- **Not a Go engineer!**
- Spent several years on “hacking” LLVM compiler/static analyzers/C++ standard library (libc++), etc.
- Awesome PGO author
- “Software Performance” devroom organizer
- Interested in data-driven software optimizations
- Like **rapid** software!

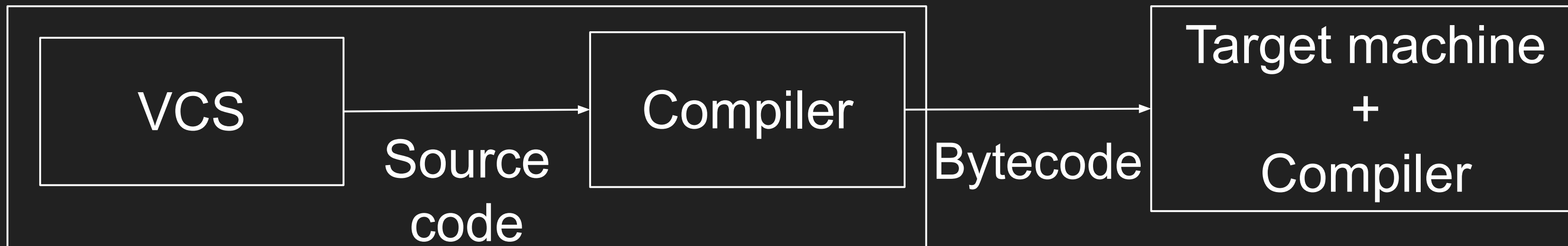
A **bit** of theory about PGO

Very quick, I promise

Ahead-of-Time (AOT)



Just-in-Time (JIT)



Compiler optimizations and runtime information

- **Inlining**
- **Devirtualization**
- Loop roll/unroll
- Hot/cold code splitting
- And many other funny things!

Many compiler optimizations can be improved by providing runtime execution statistics!

The solution: Profile-Guided Optimization

- Collect runtime statistics on a target machine aka a **PGO profile**
- Pass the profile to a compiler
- Use the profile during the compilation phase
- …
- Our binary is faster*
  - Is not true for all cases
  - Only for CPU-intensive workloads**
  - …

GopherConUK

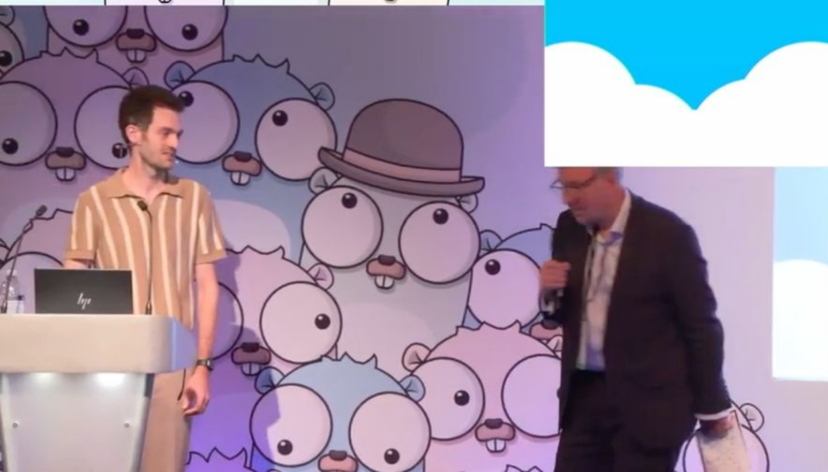
2024-08-15

Boost application performance with profile guided optimization

Michael Pratt

He/him

Google



Michael Pratt

Staff Software Engineer - Google

"Boost application performance with profile guided optimization"

Benchmarks (mostly non-Go - sorry!)

Application	Improvement	Library	Improvement
Official Go bench	2-14% improvement	serde_json	~15% improvement
Dolt	5% latency improvement	xml-rs	~35% improvement
Rust Analyzer	+20% speedup	quick-xml	~25% improvement
PostgreSQL	up to +15% faster queries	tonic	~10% improvement
SQLite	up to 20% faster queries	rustls	~6% improvement
ClickHouse	up to +13% QPS	axum	~10% improvement
MySQL	up to +35% QPS	tantivy	~30% improvement
MongoDB	up to **2x** faster queries	wgpu	~25% improvement
Redis	up to +70% RPS	tracing libs	~35-40% improvement

PGO in Go - current state

- Implemented in Go since 1.20 (preview), GA since 1.21 (August 2023) - **much** later compared to C and C++
- Only Sampling PGO (sPGO aka AutoFDO) is available
  - No Instrumentation PGO and some fancy stuff like Temporal PGO
- Only the main compiler supports PGO
  - No support in llgo or TinyGO
- Uses pprof CPU profiles with reusing all existing ecosystem!
  - Other formats are supported via conversion to pprof with nuances
- Mainly designed for service-like workloads
  - HTTP handlers for gathering PGO profiles, service-like examples everywhere in the documentation, etc.

Possible PGO issues and how to mitigate them

- Mismatched / outdated PGO profiles:
  - The “default” option - just regenerate them for each build*
  - cmd/compile: consider giving a warning or an error if a PGO profile looks mismatched #70291 - no activity since November 2024 :(
- Performance regressions from PGO
  - Yes, they are possible
  - Debugging them is a challenging thing
- What about small “one-shot” utilities like CLIs
  - Dumping a PGO profile at the program exit? Rejected :(
  - No available support tooling like cargo-pgo in Rust

To add equivalent profiling support to a standalone program, add code like the following to your main functio

```
var cpuprofile = flag.String("cpuprofile", "", "write cpu profile to `file`")
var memprofile = flag.String("memprofile", "", "write memory profile to `file`")

func main() {
    flag.Parse()
    if *cpuprofile != "" {
        f, err := os.Create(*cpuprofile)
        if err != nil {
            log.Fatal("could not create CPU profile: ", err)
        }
        defer f.Close() // error handling omitted for example
        if err := pprof.StartCPUProfile(f); err != nil {
            log.Fatal("could not start CPU profile: ", err)
        }
        defer pprof.StopCPUProfile()
    }

    // ... rest of the program ...
```

Multilingua PGO with CGO

- This use case is ugly everywhere :(
- No build system can help you here for now
- You’ll need to pass corresponding PGO flags for your C / C++ / Rust dependencies
- And here you go:
    - Different PGO modes: sPGO in Go + sPGO in C are not compatible (remember different formats)
    - What about sPGO Go + Instrumentation PGO in C? Which instrumentation mode? There are many of them! Instrumentation brings even more issues to resolve…
    - C, C++, Rust -> different compilers -> kinda different PGO flags…
    - Your build scripts will be “hack-ish” at best, **awful** at practice

PGO with Go CGO

help

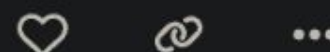
parags108 Feb 2023

Apologize if this was discussed before. A cursory search did not reveal anything substantial.

Problem:
I am trying to get PGO working for a Go -> Rust via CGO code flow. It requires access to the rust profile toolchain libraries, which I do provide using CGO_LDFLAGS, but for some reason it is now not able to find the Rust library changes. Go-based service app is calling methods in Rust via a compiled library.

If anyone has experience with doing this, that would be great!

Regards
Parag



434 views

3 months later

Closed on May 4, 2023

This topic was automatically closed 90 days after the last reply. We invite you to open a new topic if you have further questions or comments.

Reproducibility and PGO

- Top 1 blocker for using PGO in distributions!
- Two dedicated cases:
  - Reproducible build with a **saved** PGO profile
  - Reproducible PGO profile generation
- The first case can be resolved with a PGO profile caching and reusing between builds
  - However, it brings an additional problem with outdated PGO profiles!
- The second case generally speaking is unresolvable in practice (see LLVM discussion - it’s actual for all PGO implementations)

CI: Go 1.21.0 + default.pgo #468

New issue

Closed | #469

vadimalekseev opened on Aug 13, 2023 — Collaborator

Bump Go 1.21.0 and paste `default.pgo` to enable pgo

vadimalekseev added enhancement on Aug 13, 2023

vadimalekseev mentioned this on Aug 13, 2023

Go 1.21, default.pgo, alpine + ubuntu-22.04/20.04 images #469

vadimalekseev closed this as completed in #469 on Aug 14, 2023

zamazan4ik on Sep 2, 2023

@vadimalekseev Do you have performance benefits numbers from enabling PGO on file.d? If yes, could you please share the measured performance numbers? Another question - from which workload is the `default.pgo` file collected? We are trying to estimate - Do we need to prepare our own PGO profile or we can use the distributed with file.d one?

Thanks in advance!

Assignees

No one assigned

Labels

enhancement

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

Go 1.21, default.pgo, alpine + ubuntu-22.04/20.04 images

ozontech/file.d

PGO at scale - a bunch of additional issues

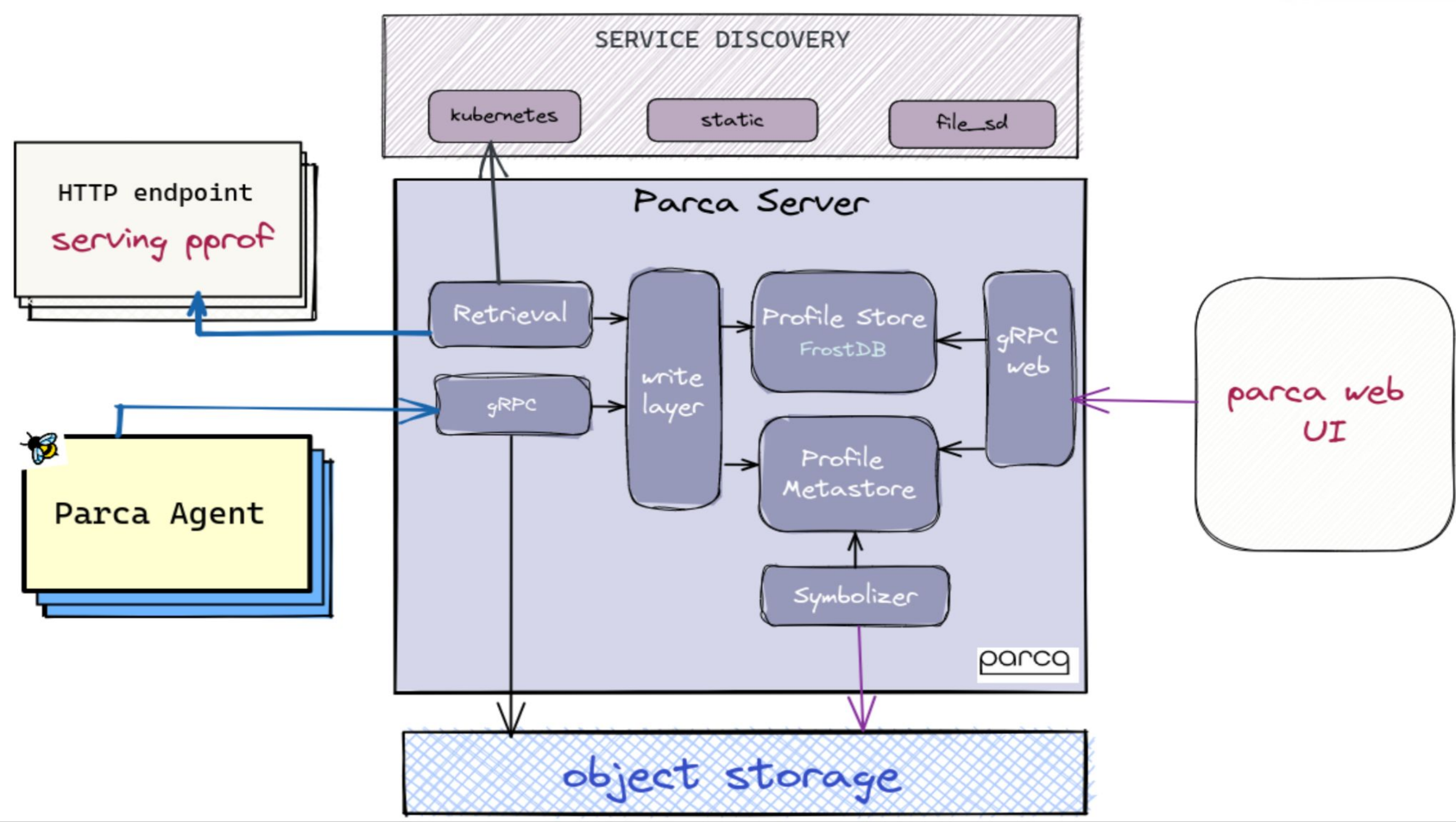
- Collect profiles **thousands** of profiles for hundreds of services
- Implement proper gather/symbolize/store/clean routines
- Make this process observable and robust **at scale**
- Tracking profile skew during time (and raising alerts)
- …

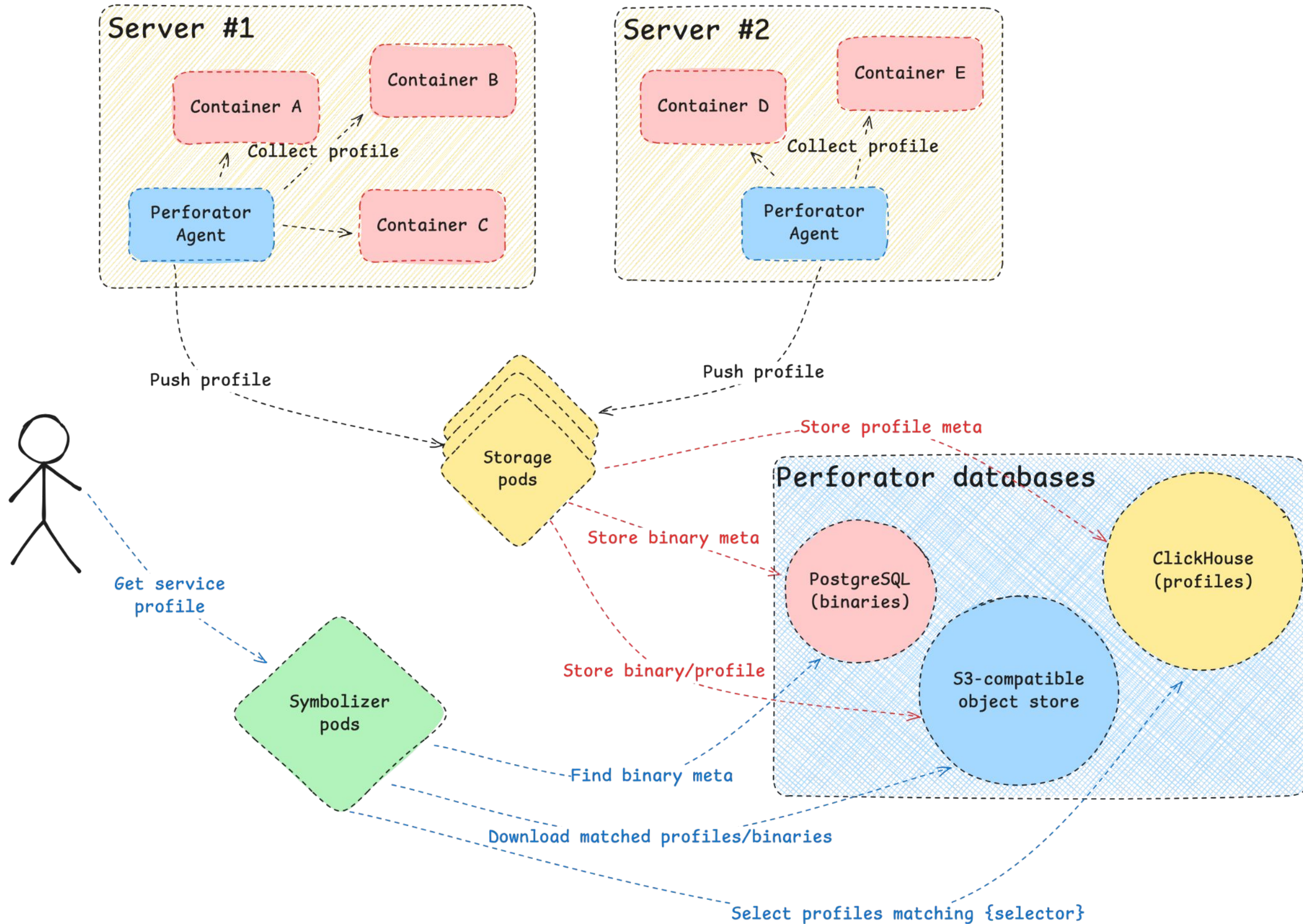
You don’t want to implement all these things from scratch!

PGO at scale - what to use?

- Don’t use PGO at all - not our way!
- Use PGO “manually” - not that scalable
- Parca
- Yandex.Perforator
- Extend other open-source solutions like Grafana Pyroscope or Elastic Universal Profiling - PGO-related issues exist but no activity :(
- Write your own form the scratch - quite a popular solution!
    - Google Wide Profiler (GWP) at Google
    - Ozon Vision at Ozon
    - And many other system-wide profilers

However - these solutions are all quite complicated to setup and admin.

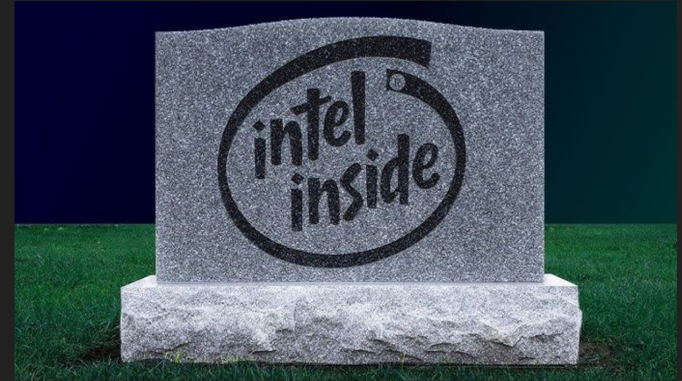

SERVICE DISCOVERY
kubernetes
static
file_sd
HTTP endpoint
serving pprof
Parca Server
Retrieval
gRPC
write layer
Profile Store
FrostDB
Profile Metastore
gRPC web
Symbolizer
parca web UI
Parca Agent
parca
object storage


Server #1
Container A
Container B
Collect profile
Perforator Agent
Container C
Server #2
Container D
Container E
Collect profile
Perforator Agent
Push profile
Push profile
Storage pods
Store profile meta
Perforator databases
Store binary meta
PostgreSQL (binaries)
ClickHouse (profiles)
Get service profile
Store binary/profile
S3-compatible object store
Symbolizer pods
Find binary meta
Download matched profiles/binaries
Select profiles matching {selector}

	Perforator	Parca
Is it alive?	Yes	Yes
Written in	C++	Go
License	Apache License 2.0	Apache License 2.0
Documentation	Could be better but sPGO is covered	Okay but no PGO part :(
Paid support	No (AFAIK)	Yes
PGO support for Go	Yes (but not tested yet)	Yes
PGO support for wrong (non-Go) languages	Yes (mostly used for C++)	No
PLO support	Yes*	No

Post-Link Optimization (PLO)

- "PGO on steroids" - can optimize binaries even after PGO
- TL;DR - reshuffles binaries to better utilize CPU I-cache (does actually more)
- Available open-source tools at the moment:
  - LLVM BOLT - the most popular PLO tool nowadays
  - Google Propeller
  - Intel Thin Layout Optimizer (TLO) - RIP



Performance Impact

- Up to **19%** of relative performance improvement on internal applications
- goweb "Light weight web framework based on net/http"
  - Repo: https://github.com/twharmon/goweb.git
  - Profile collected using BOLT instrumentation
  - Go 1.17
  - .text size ~3.5M
  - Performance Improvement (Xeon Gold 6230N): **+8.13%**
  - Performance Improvement (Kunpeng 920): **+11.74%**

name	old time/op	new time/op	delta	
GowebPlaintext-8	2.12µs ± 0%	1.89µs ± 0%	-10.54%	(p=0.008 n=5+5)
GinPlaintext-8	1.40µs ± 1%	1.31µs ± 2%	-6.36%	(p=0.008 n=5+5)
GorillaPlaintext-8	3.19µs ± 0%	3.09µs ± 0%	-3.25%	(p=0.008 n=5+5)
EchoPlaintext-8	1.39µs ± 0%	1.29µs ± 0%	-7.39%	(p=0.008 n=5+5)
MartiniPlaintext-8	17.9µs ± 0%	15.9µs ± 0%	-11.04%	(p=0.008 n=5+5)
GowebJSON-8	119µs ± 0%	99µs ± 0%	-16.81%	(p=0.008 n=5+5)
GinJSON-8	130µs ± 0%	110µs ± 0%	-15.61%	(p=0.008 n=5+5)
GorillaJSON-8	121µs ± 0%	101µs ± 0%	-16.26%	(p=0.008 n=5+5)
EchoJSON-8	117µs ± 0%	98µs ± 0%	-16.11%	(p=0.008 n=5+5)
MartiniJSON-8	169µs ± 0%	143µs ± 0%	-15.23%	(p=0.008 n=5+5)
GowebPathParams-8	5.97µs ± 0%	5.26µs ± 0%	-11.84%	(p=0.008 n=5+5)
GinPathParams-8	4.07µs ± 0%	3.67µs ± 0%	-9.81%	(p=0.008 n=5+5)
GorillaPathParams-8	7.18µs ± 0%	6.40µs ± 0%	-10.77%	(p=0.008 n=5+5)
EchoPathParams-8	4.24µs ± 0%	3.74µs ± 0%	-11.76%	(p=0.008 n=5+5)
MartiniPathParams-8	20.3µs ± 0%	17.9µs ± 0%	-12.09%	(p=0.008 n=5+5)
[Geo mean]	13.8µs	12.2µs	-11.74%	

- benchmark of graphql frameworks
  - Repo: https://github.com/appleboy/golang-graphql-benchmark.git
  - Profile collected using BOLT instrumentation
  - Go 1.17
  - .text size ~6M
  - Performance Improvement (Xeon Gold 6230N): **+11.36%**
  - Performance Improvement (Kunpeng 920): **+8.98%**

name	old time/op	new time/op	delta	
GinHttpRoute-8	1.92µs ± 0%	1.70µs ± 0%	-11.50%	(p=0.008 n=5+5)
GinGQLGenRoute-8	1.95µs ± 0%	1.76µs ± 0%	-9.84%	(p=0.008 n=5+5)
GinGoGraphQLRoute-8	22.5µs ± 0%	19.3µs ± 0%	-14.09%	(p=0.008 n=5+5)
GinGopherGraphQLRoute-8	754ns ± 0%	686ns ± 1%	-9.01%	(p=0.008 n=5+5)
GinThunderGraphQLRoute-8	1.30µs ± 0%	1.16µs ± 1%	-10.48%	(p=0.008 n=5+5)
GoGraphQLMaster-8	59.4µs ± 0%	51.9µs ± 0%	-12.67%	(p=0.008 n=5+5)
PlaylyfeGraphQLMaster-8	5.18µs ± 0%	4.70µs ± 0%	-9.20%	(p=0.008 n=5+5)
GophersGraphQLMaster-8	4.08µs ± 0%	3.56µs ± 0%	-12.77%	(p=0.008 n=5+5)
ThunderGraphQLMaster-8	2.31µs ± 0%	2.02µs ± 1%	-12.57%	(p=0.008 n=5+5)
[Geo mean]	3.96µs	3.51µs	-11.36%	

ADVANCED SOFTWARE TECHNOLOGY LAB HUAWEI

cmd/link: support option **-emit-relocs** for the golang linker #49031

yota9 on Dec 3, 2023 · edited by yota9 Edits ···

@zamazan4ik Hi! The process of upstreaming is stopped, I don't have time for this and didn't see enough interest from both go and llvm-bolt community. the change is quite huge and I have no idea how to complete it now :) I was speaking about it with the BOLT creator, he said that there might be some interest now in this field and we might continue this process together one day, but I'm not sure when. The PoC could be found here, but it was not updated for a long time https://reviews.llvm.org/D124347 , so it might contain some unresolved bugs and doesn't support later versions. Internally we're continue to use it successfully in production, currently the latest version supported is 1.18* .

1

Milestone

Unplanned

Due by December 31, 2099, 60% complete

Summary

- PGO in Go is already great - please use it (if it's worth it)
- Localhost-like PGO optimizations - be ready for some "hacking" with pprof
- Large PGO deployments - start with Parca or Perforator
- We have a lot of Go open-source projects without PGO - we can do better
- PLO is not available for Go yet - we need improvements here!
- Got any PGO performance improvements? Please share with Awesome PGO

Thank you! Questions?

The PGO Gate
https://github.com/zamazan4ik/awesome-pgo

- **Emails**:
  - zamazan4ik@tut.by (primary)
  - zamazan4ik@gmail.com (secondary)
- **Matrix**: @zamazan4ik:matrix.org
- **Telegram**: zamazan4ik
- **Discord**: zamazan4ik
- **GitHub**: zamazan4ik