



GopherCon 2023 recap

The Go gopher was designed by Renee French.

自己紹介

名前

- 渋谷拓真
 - X/GitHub: @sivchari
- 所属
 - CyberAgent
 - CIU
 - 22卒
 - Next Experts
- OSS
 - Go
 - golangci-lint
 - etc...



今日話す内容

- **AST操作による自動計装 by Datadog**
- **大量トラフィックをさばく Byte Dance社のBalanced GCについて**

AST操作による自動計装 by Datadog

Datadogとは

- SaaS形式のモニタリングツール
- こんなことができる
 - ログを集計してアラートとかを出す
 - マシン自体のCPUやRPSなどをみる
 - Integration
 - Dashboard
 - Spanを対象としたパフォーマンスの計測 (APM)
 - Profiling (e.g. pprof)



Datadog sample repository

- <https://github.com/sivchari/datadog-sample>
 - simpleが手動実装
 - orchestrionが紹介するツール
 - orchestrion-reqがorchistrion + magic annotationの例
- Datadogが出しているsample-app
 - <https://github.com/DataDog/go-sample-app>


















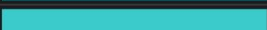






Datadog simple

```
package main
import (
    "log"
    "net/http"
    httptrace "gopkg.in/DataDog/dd-trace-go.v1/contrib/net/http"
    "gopkg.in/DataDog/dd-trace-go.v1/ddtrace/tracer"
)
func main() {
    tracer.Start(
        tracer.WithService("service"),
        tracer.WithEnv("env"),
    )
    defer tracer.Stop()
    mux := httptrace.NewServeMux()
    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello World!"))
    })
    if err := http.ListenAndServe(":8080", mux); err != nil {
        log.Fatal(err)
    }
}
```

Datadog simple

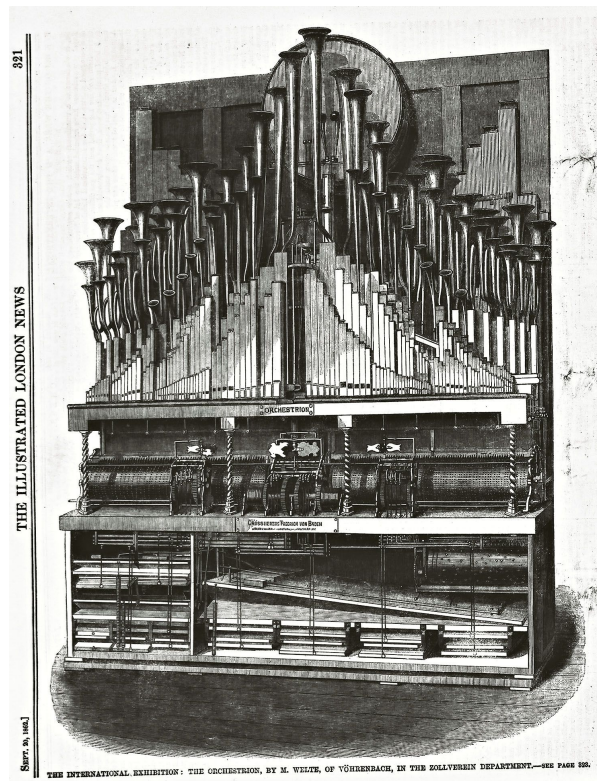
- `docker compose up --build -d`
- `go run simple/main.go`
- `curl localhost:8080`

Datadog simple

↓ DATE	SERVICE	RESOURCE	DURATION	@HTTP.MET...	@HTTP.STAT...	SPANS	LATENCY BREAKDOWN
Oct 28 23:53:40.664	 service	GET /	47.0 μs	GET	200	1	
Oct 28 23:53:39.759	 service	GET /	150 μs	GET	200	1	
Oct 28 23:53:05.004	 service	GET /	91.0 μs	GET	200	1	
Oct 28 23:53:03.875	 service	GET /	143 μs	GET	200	1	
Oct 28 23:53:02.714	 service	GET /	42.0 μs	GET	200	1	
Oct 28 23:52:58.621	 service	GET /	42.0 μs	GET	200	1	
Oct 28 23:52:57.676	 service	GET /	87.0 μs	GET	200	1	
Oct 28 23:52:56.857	 service	GET /	146 μs	GET	200	1	
Oct 28 23:52:47.288	 service	GET /	135 μs	GET	200	1	
Oct 28 23:52:46.581	 service	GET /	36.0 μs	GET	200	1	
Oct 28 23:52:45.885	 service	GET /	28.0 μs	GET	200	1	
Oct 28 23:52:45.049	 service	GET /	170 μs	GET	200	1	

orchstrion

- <https://github.com/DataDog/orchestrion>
- 今回のsessionで紹介されたcli-tool
 - 自動計装を行ってくれる
 - 現在サポートしているのは以下
 - net/http
 - database/sql
 - google.golang.org/grpc
 - github.com/gin-gonic/gin
 - github.com/labstack/echo/v4
 - github.com/go-chi/chi/v5
 - github.com/gorilla/mux
 - 復活してよかった





orchestration

```
package main
import (
    "log"
    "net/http"
)
func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello World!"))
    })
    if err := http.ListenAndServe(":8080", mux); err != nil {
        log.Fatal(err)
    }
}
```

orchestrion (-w)

```
//dd:startinstrument
defer instrument.Init()()
//dd:endinstrument
mux := http.NewServeMux()
//dd:startwrap
mux.HandleFunc("/", instrument.WrapHandlerFunc(func(w http.ResponseWriter, r *http.Request)
{
    w.Write([]byte("Hello World!"))
}))
//dd:endwrap
if err := http.ListenAndServe(":8080", mux); err != nil {
    log.Fatal(err)
}
```

orchestration

↓ DATE	SERVICE	RESOURCE	DURATION	METHOD	STATUS CODE
 Oct 31 02:15:50.824	 main	GET /	86.0 μs	GET	200
 Oct 31 02:15:50.117	 main	GET /	90.0 μs	GET	200
 Oct 31 02:15:49.165	 main	GET /	113 μs	GET	200
 Oct 31 02:15:47.780	 main	GET /	251 μs	GET	200

orchestrion -rm

```
package main
import (
    "log"
    "net/http"
)
func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello World!"))
    })
    if err := http.ListenAndServe(":8080", mux); err != nil {
        log.Fatal(err)
    }
}
```

orchestrion custom span tag

```
//dd:span my:tag
func HandleRequest(ctx context.Context) ([]byte, error) {
    client := &http.Client{}
    req, err := http.NewRequestWithContext(ctx, "Post", "http://example.com",
strings.NewReader("Hello Worl!"))
    if err != nil { return nil, err }
    resp, err := client.Do(req)
    if err != nil { nil, err }
    defer resp.Body.Close()
    return io.ReadAll(resp.Body)
}
```

orchestrion custom span tag

```
//dd:span my:tag
func HandleRequest(ctx context.Context) ([]byte, error) {
    //dd:startinstrument
    ctx = instrument.Report(ctx, event.EventStart, "function-name", "GetSomeData", "my", "tag")
    defer instrument.Report(ctx, event.EventEnd, "function-name", "GetSomeData", "my", "tag")
    //dd:endinstrument
    //dd:startwrap
    client := instrument.WrapHTTPClient(&http.Client{
        Timeout: time.Second,
    })
    //dd:endwrap
    req, err := http.NewRequestWithContext(ctx,
        http.MethodPost, "http://example.com",
        strings.NewReader("Hello, World!"))
}
```


dave/dst

- <https://github.com/dave/dst>
 - Decorated Syntax Tree
 - go/astにファイルの情報を付加したもの
- <https://go.dev/play/p/DA1370qzhuV>
 - go/astを使用
 - コメントの位置情報がないため順序が崩れる
- <https://go.dev/play/p/XciwaHUJWSE>
 - dave/dstを使用
 - コメントの位置情報を保持しているため outputのtreeが崩れていない

内部コードを追ってみる

- **entrypoint**
 - **main.go**
 - **cobra**などには依存していない
 - **write**
 - **remove**
 - **httpmode**
 - HTTP HandlerのRequest/ResponseWriter自体もtraceするか(default: true)
 - report -> wrapの上書きはできたが、wrap -> reportでスコープを絞るのは現状できない

内部コードを追ってみる

- `type ProcessFunc func(string, io.Reader, config.Config) (io.Reader, error)`
 - [InstrumentFile](#)
 - `net/http`など対応している箇所に `tracer`を挟みながら `annotation`をコメントとして付与する
 - [UninstrumentFile](#)
 - `annotation`があるかどうかを確認し、存在していれば初期の状態に変更する
- `type OutputFunc func(string, io.Reader)`
 - `fmt.Println`で結果を表示
 - `Parse`したファイルに書き込む

内部コードを追ってみる

- Datadog magic annotations

```
const (  
    dd_startinstrument = "//dd:startinstrument"  
    dd_endinstrument   = "//dd:endinstrument"  
    dd_startwrap       = "//dd:startwrap"  
    dd_endwrap         = "//dd:endwrap"  
    dd_instrumented    = "//dd:instrumented"  
    dd_span            = "//dd:span"  
    dd_ignore          = "//dd:ignore"  
)
```

対応packageを増やしたい場合

- orchestrion/instrumentを覗いてみる
 - <https://github.com/DataDog/orchestrion/tree/main/instrument>
 - dd-traceを使用していることがわかる
 - これを足してあげればサポートが増えるし、足りなければ contribに増やす必要が現状の構成ではありそう

rewrite

- <https://github.com/jonbodner/rewrite>
- rewrite専用のルールファイルを書いて使用する
- -toolexecでコンパイルをhookする

vars:

path string

handlerFunc func(http.ResponseWriter, *http.Request)

rules:

r.Get(path, handlerFunc) -> r.Get(path,
instrument.WrapHandlerFunc(handlerFunc))

Balanced GC by ByteDance

ByteDance社とGo

- 数万のマイクロサービスが様々なビジネスを支えるために開発されている
- そのほとんどがGoで開発されている
- マイクロサービスのためのツールチェーンもある
 - 例としてあがっていたのは tango
 - パフォーマンス向上とカスタマイズされた機能をもつダウンストリーム (GitHubにはないので internal ??)
- 複数のマイクロサービスが通信するため当然レイテンシーを気にする
- Tiktokをはじめとした大規模サービスにとってユーザー体験を支えるためにパフォーマンスの最適化は重要

ByteDance社とGo

- 数万のマイクロサービスが様々なビジネスを支えるために開発されている
- そのほとんどがGoで開発されている
- マイクロサービスのためのツールチェーンもある
 - 例としてあがっていたのは tango
 - パフォーマンス向上とカスタマイズされた機能をもつダウンストリーム (GitHubにはないので internal ??)
- 複数のマイクロサービスが通信するため当然レイテンシーを気にする
- Tiktokをはじめとした大規模サービスにとってユーザー体験を支えるためにパフォーマンスの最適化は重要

登場人物

- **Concurrent Mark & Sweep GC**
- **Balanced GC**
- **Copying GC**
- **GAB(Goroutine allocation buffer)**

GoのGCについて

- Goは標準でConcurrent Mark & Sweep (CMS)を実装している
- GCの挙動は-gcflags '-m' でエスケープ解析を確認できる

concurrent mark and sweep (GC)

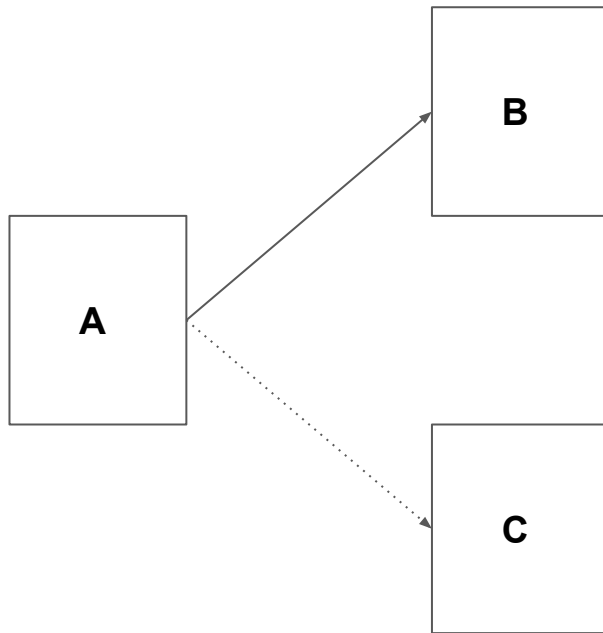
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

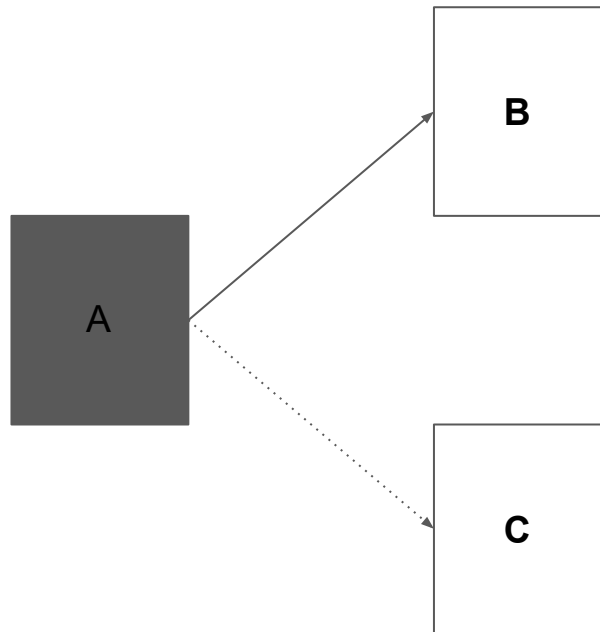
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

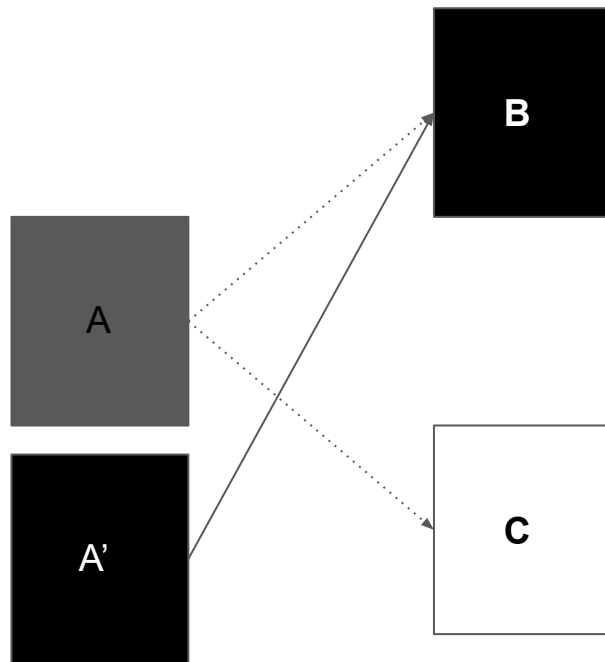
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

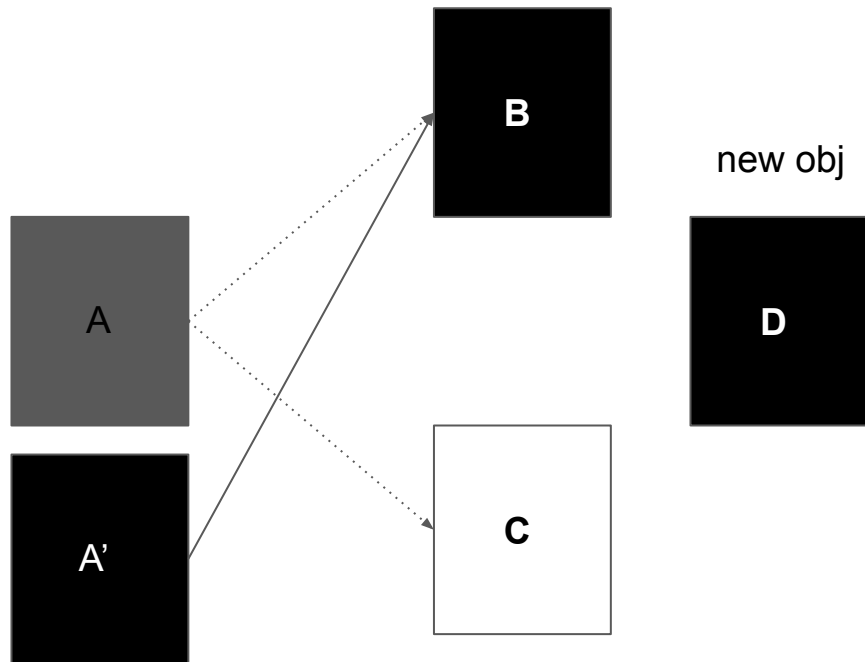
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

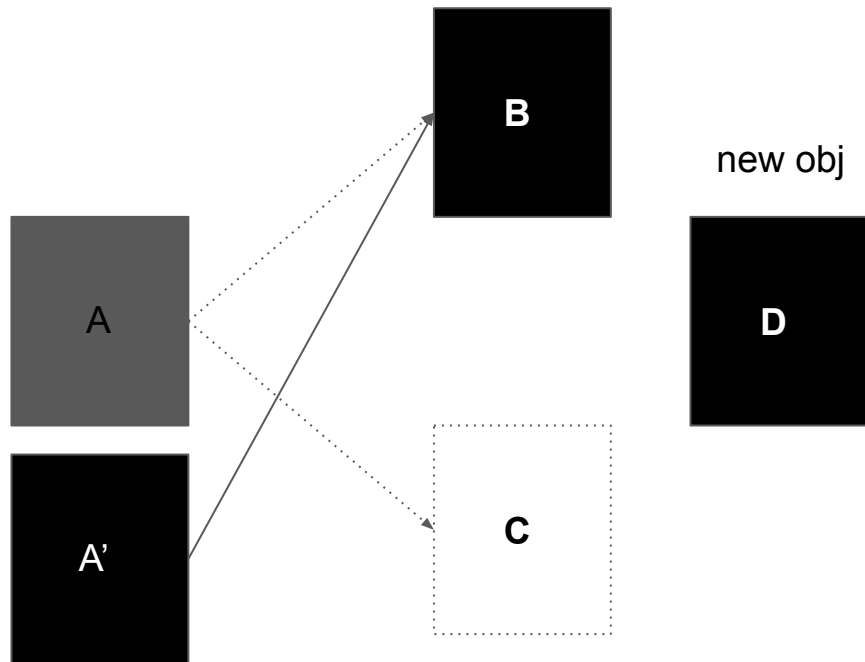
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



concurrent mark and sweep (GC)

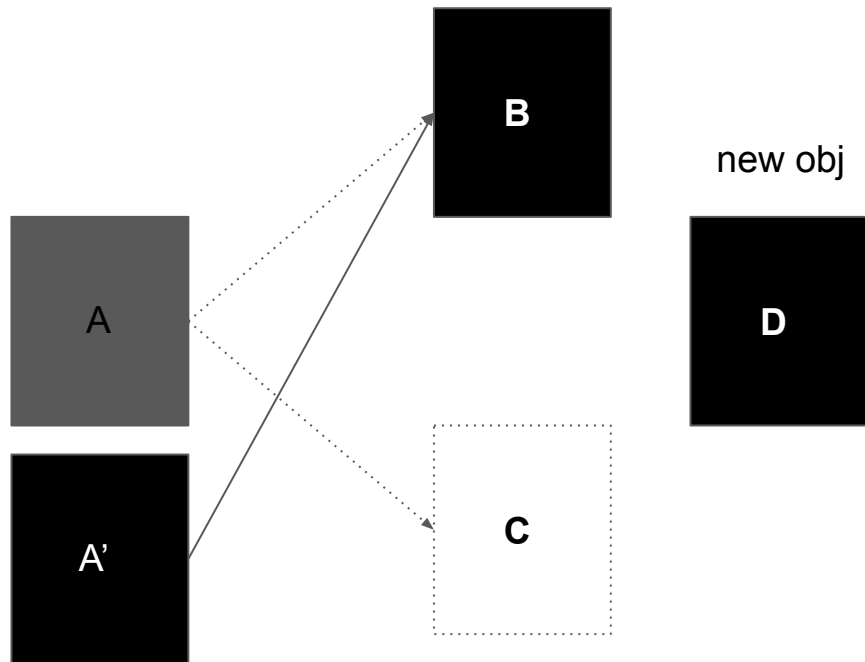
Initial Mark (STW)

Concurrent Mark

Mark Termination (STW)

Concurrent Sweep

Sweep Termination (STW)



GAB(Goroutine allocation buffer)

- いくつかのStatsを集計したところ多くのマイクロサービスでメモリ割り当てとGC時間でCPUリソースの30%以上を占めていることが多いことがわかった
- メモリ割り当て操作の頻度とヒープメモリを割り当てる際の実装が比較的重い
 - ポインター
 - グローバル変数
- runtime/mgc.goでM/gc lockをとってGCするが命令がとても多い

GAB(Goroutine allocation buffer)

- 多くの割り当てられるオブジェクトが比較的小さなオブジェクトだった
 - 88%のオブジェクトが128Bだった
- このことからGoroutine allocation bufferを設計
 - JVMのThread-local allocation bufferを参考にしている
- それぞれのgoroutineに大きなバッファを事前に割り当ててGABに収まるオブジェクトをすぐに割り当てるためにCopying GCとバンプアロケーションを使用する
- オブジェクトの閾値は128B

Copying GCとバンプアロケーション

- Copying GC

- 利用可能なメモリを2つの領域にわけ
- 片方がいっぱいになった場合もう片方に移していっぱいになった方を再利用可能な領域としてマークする
- GABを管理するために採用している

- Bump Pointer Allocation

- メモリ上の特定の位置を指すポインタを保持する
- 新しく割り当てる際にポインタを増加させる
- GABのメモリ割り当てで使っている

Balanced GC

- Copying GCを採用してConcurrent Mark & Sweepと互換性をもったGCを実現している
- 8バイトごとにアライメントしている
- バンプアロケーションを採用
- GABのサイズが足りない場合は
 - 使用している部分までを fillする
 - 現在のGABをすてて、新しくつくる

実際に導入したベンチマーク

- **Balanced GCを有効化するためのオプションを作成**
- **実際にByteDanceのマイクロサービスで有効化する**
- **Balanced GCを有効化する前と後でパフォーマンスを比較する**
 - CPU使用率は4%ほど平均して向上した
 - いくつかの効果が良くでたケースでは 10%ほどパフォーマンスが改善した

感想

- GopherCon自体の参加がはじめてだったが尊敬している方々がいたり非常に刺激的な経験だった
- コメントを利用したlinterなどを作る際にdstは有効そう。annotationをうまくつかう例としてorchestrionもいいケース
- Balanced GCは実際のコードが公開されていないため実際にどのような実装を行い、どのようにConcurrent Mark & Sweepとシンクロして実行しているのかみたくなった
 - 考え方としては1.20のArenaと似ているなど感じた