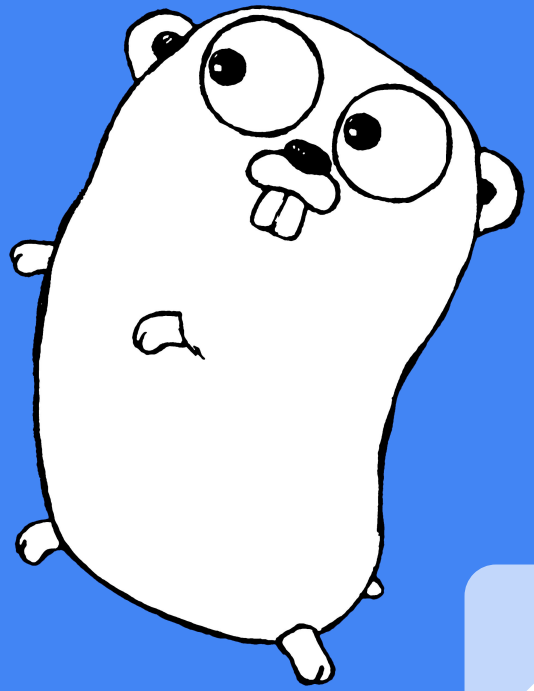
Real-Time Go

Experiences in using Go in a real-time environment



Andreas Krennmair, travel audience GmbH

What is “Real-Time”, anyway?

“A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure.”

“Any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period.”

“Finite and specified period”: deadline

Hard real-time

missing deadline means a catastrophic system failure

Firm real-time

Occasional deadline misses are tolerable, but degrade overall service quality, and the result is not useful

Soft real-time

The usefulness of a result degrades after the deadline, thereby degrading overall service quality

Real-Time Bidding

Fully automated second-price auctions for individual ad impressions, with strict deadlines of typically 100 to 120 ms

travel audience

an AMADEUS company


User interest
Showable offers
Available offers

One SSP (Google)
Two Servers
2000 QPS
memcached for data

Internal deadline + network RTT < SSP deadline
(75 ms + 17 ms < 100 ms)

```
resp := findOffers(req)
sendResponse(resp)
```

Old code

```
resultChan := findOffers(req)
select {
case <-time.After(75 * time.Millisecond):
   sendResponse(newEmptyBidResponse())
case resp := <-resultChan:
   sendResponse(resp)
}
```

New code

memcached
MongoDB

```
defer func(t0 time.Time) {
   log.Printf(“took %s”, time.Since(t0))
}(time.Now())
```

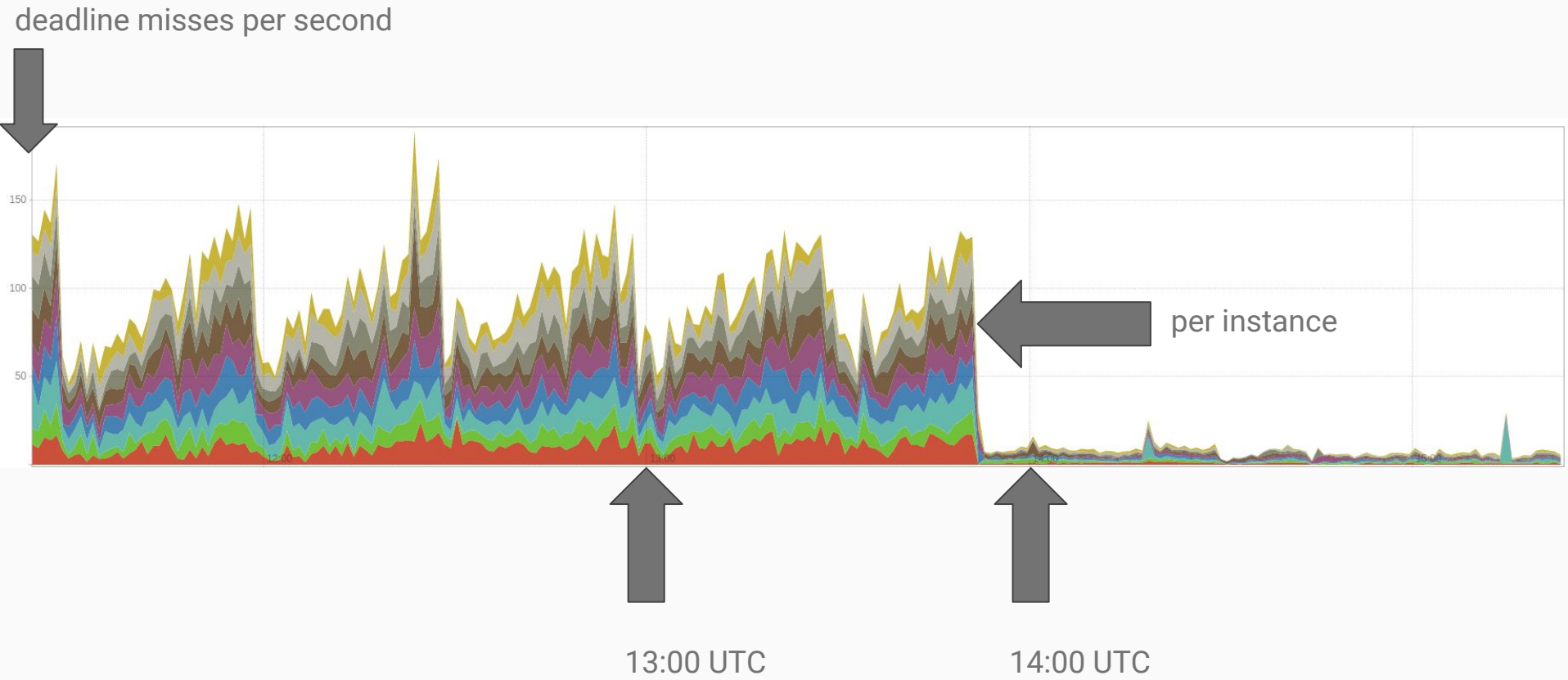
Tracking function duration

`github.com/golang/glog`

caused lock contention

MongoDB
Redis

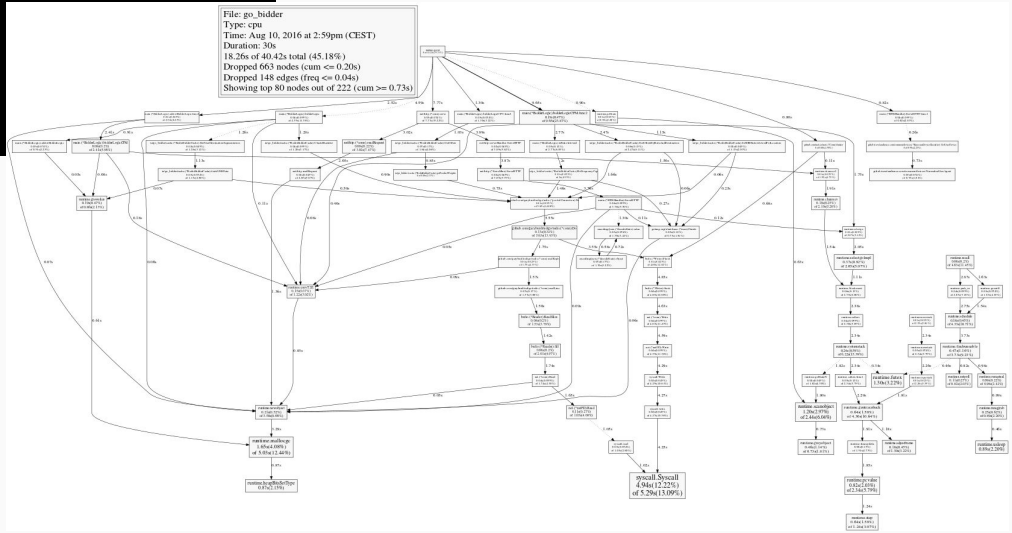
A rather impressive drop in deadline misses



Prometheus
Grafana

pprof

```
akrennmair@[redacted] λ go tool pprof go_bidder http://[redacted]/debug/pprof/profile
Fetching profile from http://[redacted]/debug/pprof/profile
Please wait... (30s)
Saved profile in /home/akrennmair/pprof/pprof.go_bidder[redacted].cpu.011.pb.gz
Entering interactive mode (type "help" for commands)
(pprof) top15 -cum
7.44s of 40.42s total (18.41%)
Dropped 663 nodes (cum <= 0.20s)
Showing top 15 nodes out of 222 (cum >= 4.59s)
      flat  flat%   sum%        cum   cum%
         0     0%     0%     32.22s 79.71%  runtime.goexit
     0.19s  0.47%  0.47%      9.65s 23.87%  main.(*[redacted]
     0.03s 0.074%  0.54%      7.77s 19.22%  net/http.(*conn).serve
     0.24s  0.59%  1.14%      6.22s 15.39%  runtime.systemstack
     0.01s 0.025%  1.16%      5.69s 14.08%  github.com/garyburd/redigo/redis.(*pooledConnection).Do
     0.13s  0.32%  1.48%      5.63s 13.93%  github.com/garyburd/redigo/redis.(*conn).Do
     4.94s 12.22% 13.71%      5.29s 13.09%  syscall.Syscall
     1.65s  4.08% 17.79%      5.03s 12.44%  runtime.mallocgc
     0.01s 0.025% 17.81%      4.86s 12.02%  bufio.(*Writer).Flush
     0.04s 0.099% 17.91%      4.85s 12.00%  bufio.(*Writer).flush
     0.04s 0.099% 18.01%      4.63s 11.45%  net.(*conn).Write
     0.08s   0.2% 18.21%      4.63s 11.45%  runtime.mcall
         0     0% 18.21%      4.62s 11.43%  main.(*[redacted]
     0.04s 0.099% 18.31%      4.59s 11.36%  main.(*[redacted]c
     0.04s 0.099% 18.41%      4.59s 11.36%  net.(*netFD).Write
(pprof)
```



/x/net/trace

RTBHandler	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
RedisBidderCache	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
RedisBidderCache.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
RedisBidderCache.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
RedisBidderCache	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
RedisBidderCache.	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]
	[0 active]	[≥0s]	[≥0.05s]	[≥0.1s]	[≥0.2s]	[≥0.5s]	[≥1s]	[≥10s]	[≥100s]	[errors]	[minute]	[hour]	[total]

Latency (μs) of RTBHandler over last hour

Count: 7981465 Mean: 1131 StdDev: 7231 Median: 465

[32,	64)	188251	2.359%	2.359%	
[64,	128)	872778	10.935%	13.294%	
[128,	256)	396424	4.967%	18.260%	
[256,	512)	3109508	38.959%	57.220%	
[512,	1024)	2489609	31.192%	88.412%	
[1024,	2048)	461187	5.778%	94.190%	
[2048,	4096)	177907	2.229%	96.419%	
[4096,	8192)	119450	1.497%	97.916%	
[8192,	16384)	111333	1.395%	99.311%	
[16384,	32768)	41037	0.514%	99.825%	
[32768,	65536)	3428	0.043%	99.868%	
[65536,	131072)	5142	0.064%	99.932%	
[131072,	262144)	5167	0.065%	99.997%	
[262144,	524288)	234	0.003%	100.000%	
[524288,	1048576)	10	0.000%	100.000%	

Garbage collector

Go 1.2: alright…
Go 1.3: faster! Less latency!

But is it real-time?

1. Guarantee certain amount of mutator utilization in a given time window
2. Account precisely for all mutator interruptions
3. Ensure space bounds are not exceeded

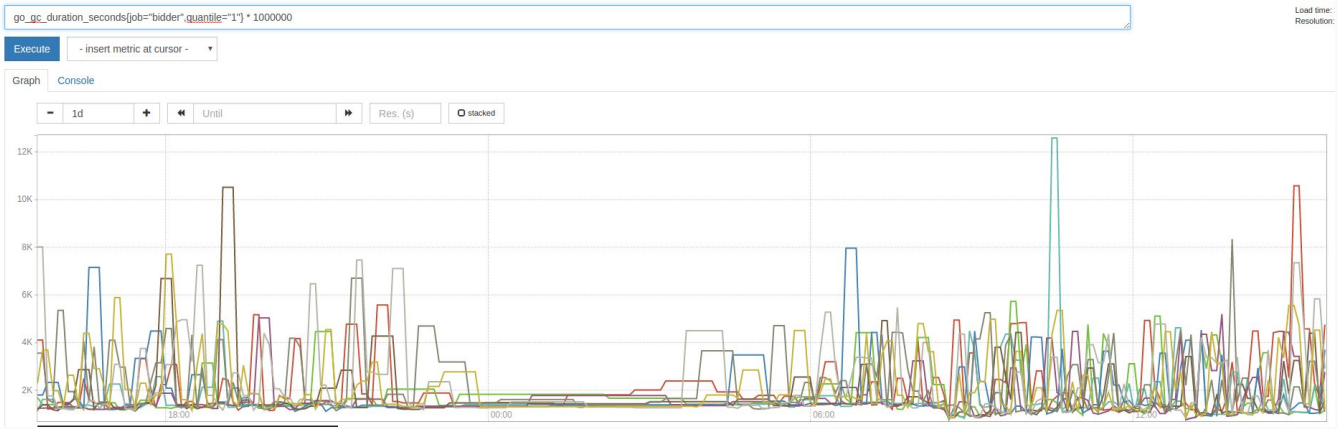
Go 1.5

Goal: 40 ms mutator utilization per 50 ms time window

Practical experience:

Pacing runs GC every few seconds
STW interruptions in sub-millisecond range

Median and maximum GC pause times



20000 QPS
9 servers, with LB in front
7 SSPs
< 10 deadline misses per second

Practical maximum: 6500 QPS per machine

- ➔ Understand what real-time means in your problem domain
- ➔ Track all your metrics: valuable, cheap to do
- ➔ Know the performance characteristics of your data store
- ➔ Introspect & profile your system, even production: pprof & /x/net/trace FTW!
- ➔ Know the GC’s capabilities & limitations

Thanks!

See https://goo.gl/OjFYdG for links & sources

(also: we hire Go developers! Just approach me after the talk)