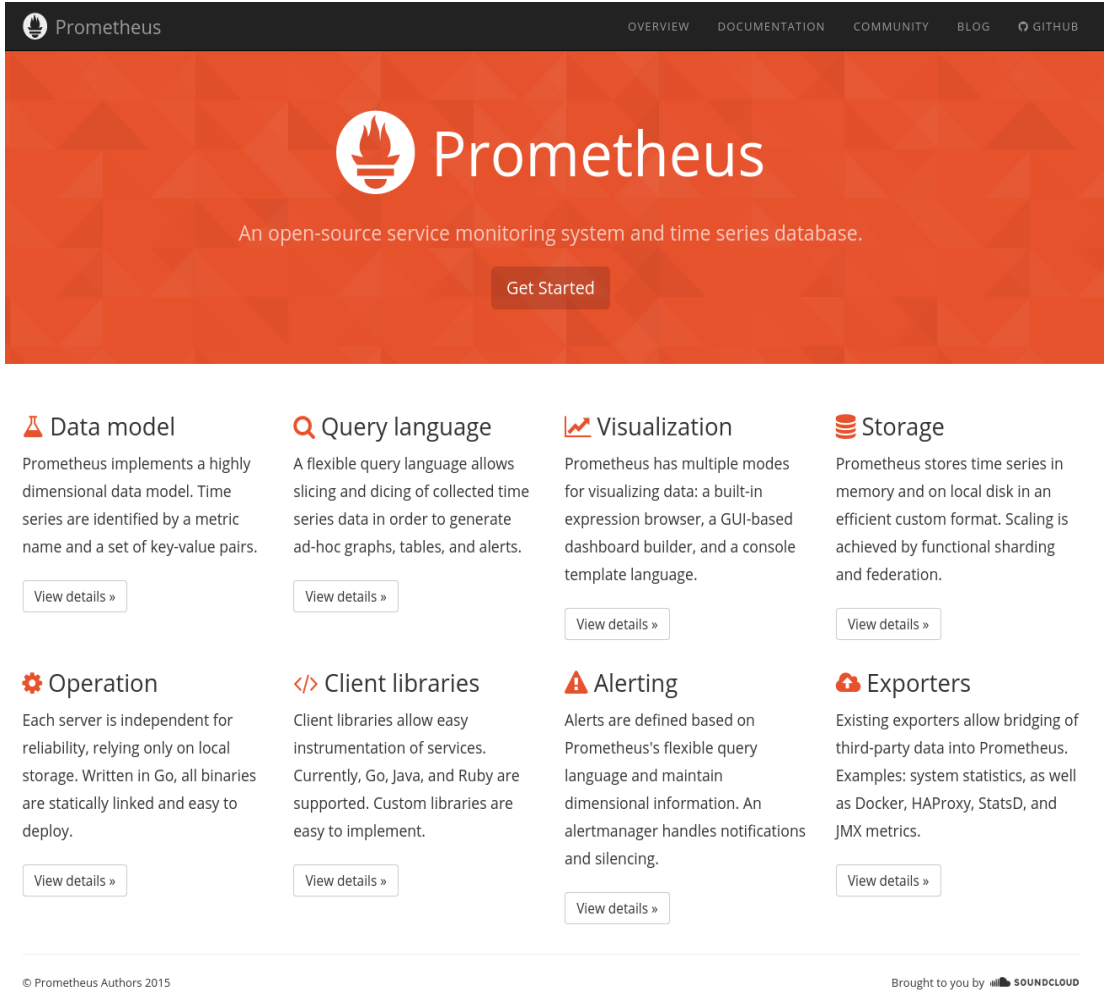
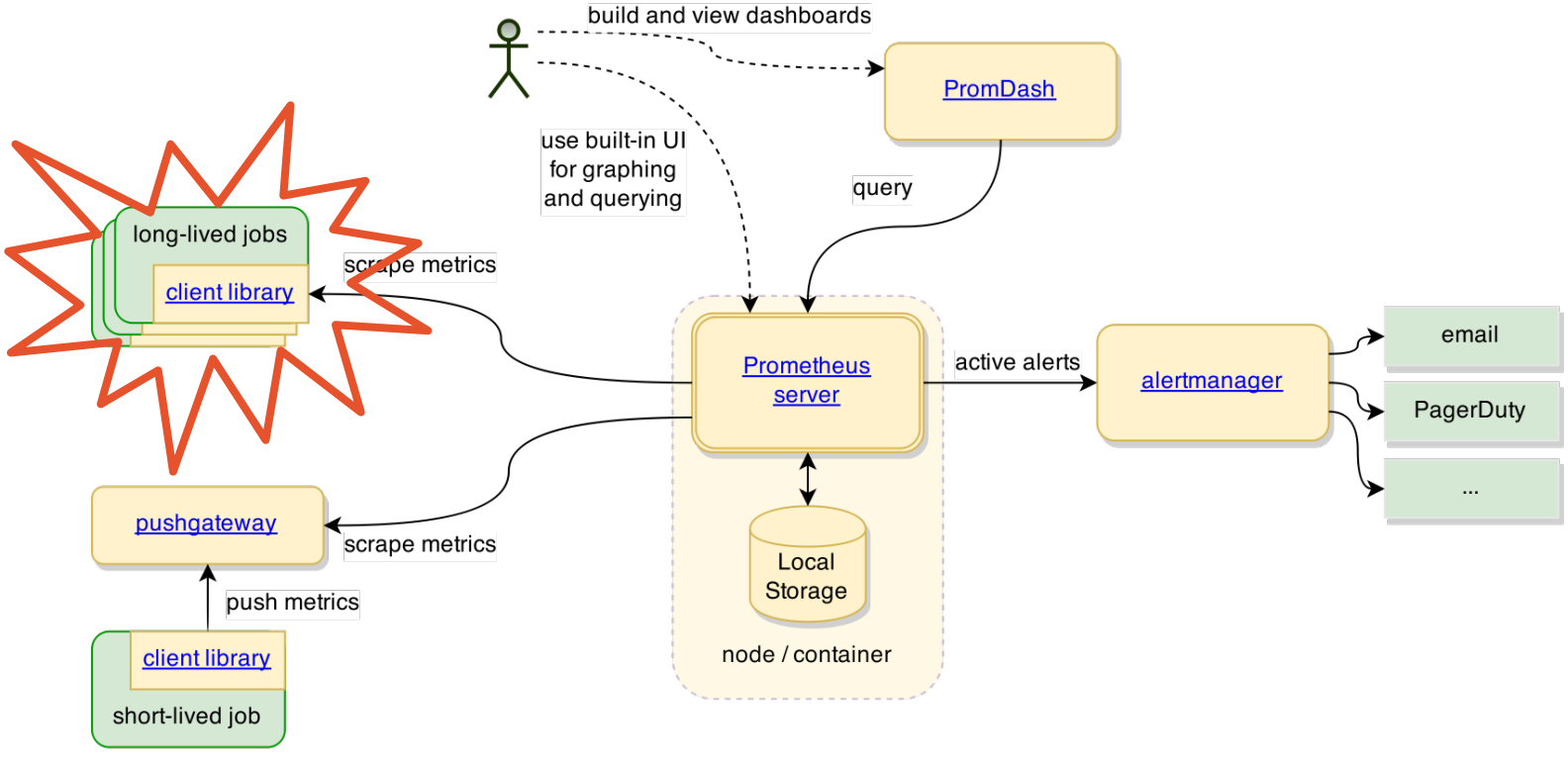
Prometheus: Designing and Implementing a Modern Monitoring Solution in Go

Björn “Beorn” Rabenstein, Production Engineer, SoundCloud Ltd.

http://prometheus.io



Architecture



Go client library

Counter interface

(almost complete)

```
type Counter interface {
   Metric

   Inc()
   Add(int)
}

type Metric interface {
   Write(*dto.Metric) error
}
```

```
type counter struct {
    value int
}

func (c *counter) Add(v int) {
    if v < 0 {
        panic(errors.New("counter cannot decrease in value"))
    }
    c.value += v
}

func (c counter) Write(*dto.Metric) error {
    // ...
}
```

```
type counter struct {
    value int
    mtx   sync.Mutex
}

func (c *counter) Add(v int) {
    c.mtx.Lock()
    defer c.mtx.Unlock()
    if v < 0 {
        panic(errors.New("counter cannot decrease in value"))
    }
    c.value += v
}

func (c *counter) Write(*dto.Metric) error {
    c.mtx.Lock()
    defer c.mtx.Unlock()
    // ...
}
```

Performance matters

It's a library, run with a large number of unknown use-cases.

```
func benchmarkAddAndWrite(b *testing.B, c Counter) {
    for i := 0; i < b.N; i++ {
        if i%1000 == 0 {
            c.Write(&dto)
            continue
        }
        c.Add(42)
    }
}

func BenchmarkNaiveCounter(b *testing.B) {
     benchmarkAddAndWrite(b, NewNaiveCounter())
}

func BenchmarkMutexCounter(b *testing.B) {
     benchmarkAddAndWrite(b, NewMutexCounter())
}
```

```
$ go test -bench=Counter
```

Results are in.

Naive counter: 5 ns/op.

(Probably mostly overhead: function call, for loop...)

Mutex counter: 150 ns/op.

```
func benchmarkAddAndWrite(b *testing.B, c Counter, concurrency int) {
	b.StopTimer()
	var start, end sync.WaitGroup
	start.Add(1)
	end.Add(concurrency)
	n := b.N / concurrency

	for i := 0; i < concurrency; i++ {
		go func() {
			start.Wait()
			for i := 0; i < n; i++ {
				if i%1000 == 0 {
					c.Write(&dto)
					continue
				}
				c.Add(42)
			}
			end.Done()
		}()
	}
	b.StartTimer()
	start.Done()
	end.Wait()
}

func BenchmarkMutexCounter10(b *testing.B) {
	benchmarkAddAndWrite(b, NewMutexCounter(), 10)
}
```

```
$ go test -bench=Counter -cpu=1,4,16  # -race
```

It’s getting worse.

Let’s talk about lock contention...

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	150	160	190
GOMAXPROCS=4	150	730	570
GOMAXPROCS=16	150	1100	1100

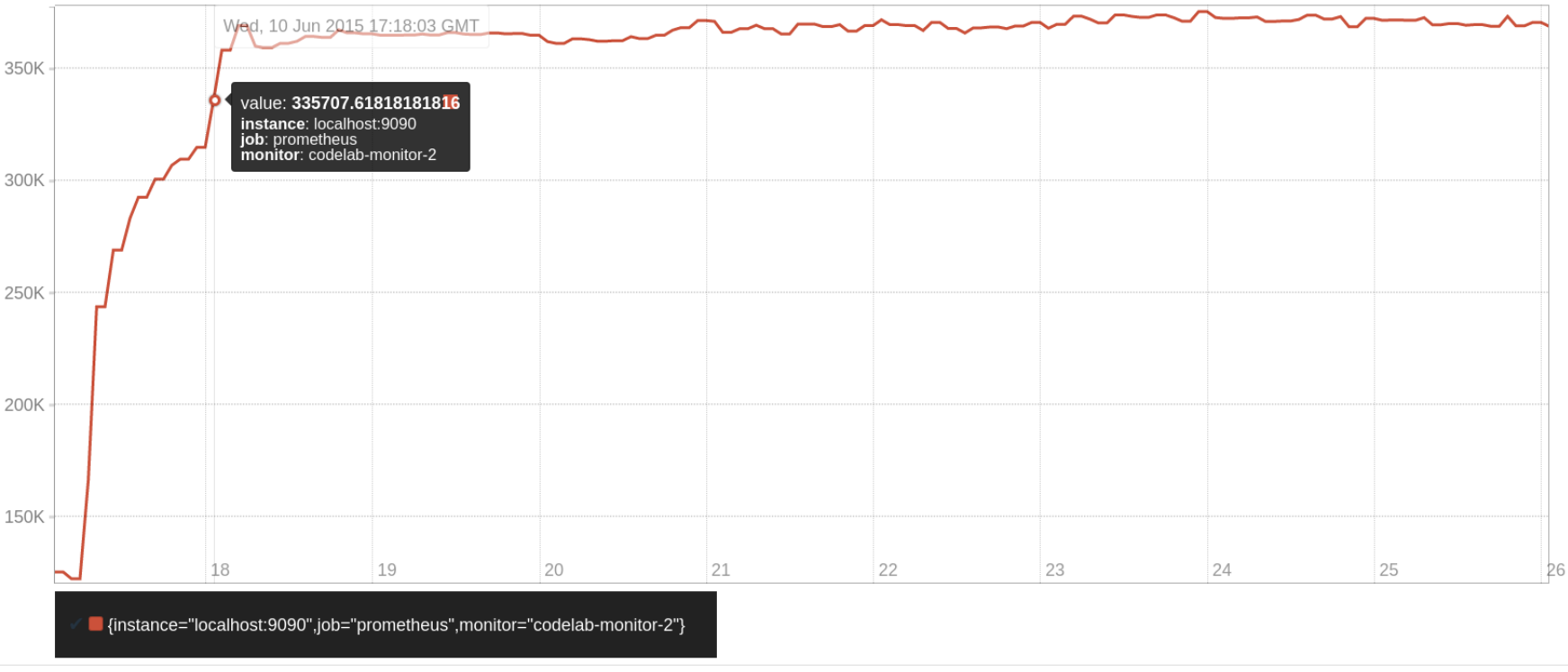
Prometheus Alerts Graph Status Help

rate(prometheus_local_storage_ingested_samples_total[1m]) Execute - Insert Metric at Cursor -

Load time: 511ms
Resolution: 3s

Graph Console

− 15m + Until Res. (s) stacked



Add Graph

Do not communicate by sharing memory;
share memory by communicating.

Rob 12:3–4

```
type counter struct {
    in  chan int // May be buffered.
    out chan int // Must be synchronous.
}

func (c *counter) Add(v int) {
    c.in <- v
}

func (c *counter) Write(*dto.Metric) error {
    value <- c.out
    // ...
}

func (c *counter) loop() {
    var value int64
    for {
        select {
        case v := <-c.in:
            value += v
        case c.out <- value:
            // Do nothing.
        }
    }
}
```

Channel counter.

x / y: Synchronous vs. buffered *in* channel.

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	670 / 310	690 / 320	680 / 360
GOMAXPROCS=4	3600 / 940	2000 / 2000	1600 / 2200
GOMAXPROCS=16	3500 / 850	2300 / 2200	1800 / 2700

```
import "sync/atomic"

type counter struct {
    value int64
}

func (c *counter) Add(v int64) {
    if v < 0 {
        panic(errors.New("counter cannot decrease in value"))
    }
    atomic.AddInt64(&c.value, v)
}

func (c *counter) Write(*dto.Metric) error {
    v := atomic.LoadInt64(&c.value)
    // Process v...
}
```

Atomic counter.

Yay!

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	15	14	15
GOMAXPROCS=4	14	45	44
GOMAXPROCS=16	14	47	45

I lied!

Prometheus uses float64 for sample values.

```
type Counter interface {
   Metric

   Inc()
   Add(float64)
}

type Metric interface {
   Write(*dto.Metric) error
}
```

```
type counter struct {
    valueBits uint64
}

func (c *counter) Add(v float64) {
    if v < 0 {
        panic(errors.New("counter cannot decrease in value"))
    }
    for {
        oldBits := atomic.LoadUint64(&c.valueBits))
        newBits := math.Float64bits(math.Float64frombits(oldBits) + v)
        if atomic.CompareAndSwapUint64(&c.valueBits, oldBits, newBits) {
            return
        }
    }
}

func (c *counter) Write(*dto.Metric) error {
    v := math.Float64frombits(atomic.LoadUint64(&c.valueBits))
    // Process v...
}
```

Atomic “spinning” counter for floats.

Yes, it works...

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	25	23	24
GOMAXPROCS=4	24	97	100
GOMAXPROCS=16	24	120	130

One last thing.

Read the fine print at the bottom of the page...

Bugs

☞ On x86-32, the 64-bit functions use instructions unavailable before the Pentium MMX. On non-Linux ARM, the 64-bit functions use instructions unavailable before the ARMv6k core. On both ARM and x86-32, it is the caller's responsibility to arrange for 64-bit alignment of 64-bit words accessed atomically. The first word in a global variable or in an allocated struct or slice can be relied upon to be 64-bit aligned.

Prometheus on Raspberry Pi

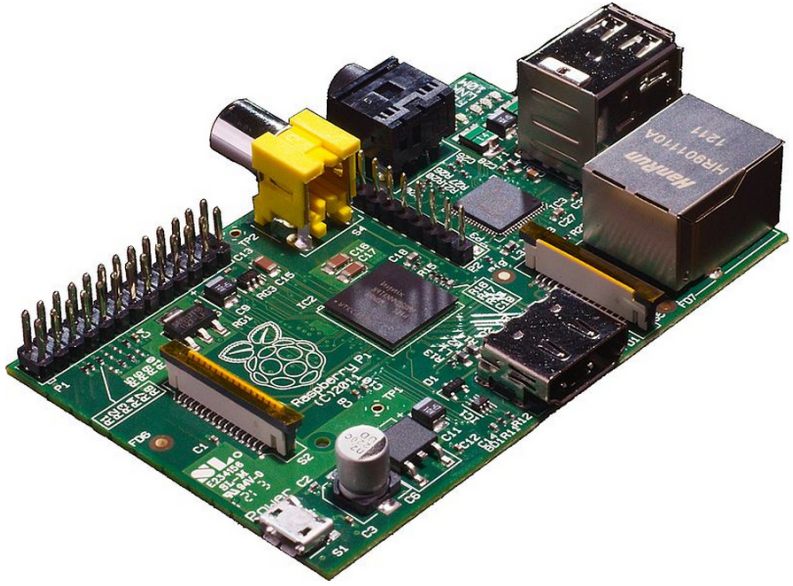
10 FEBRUARY 2015

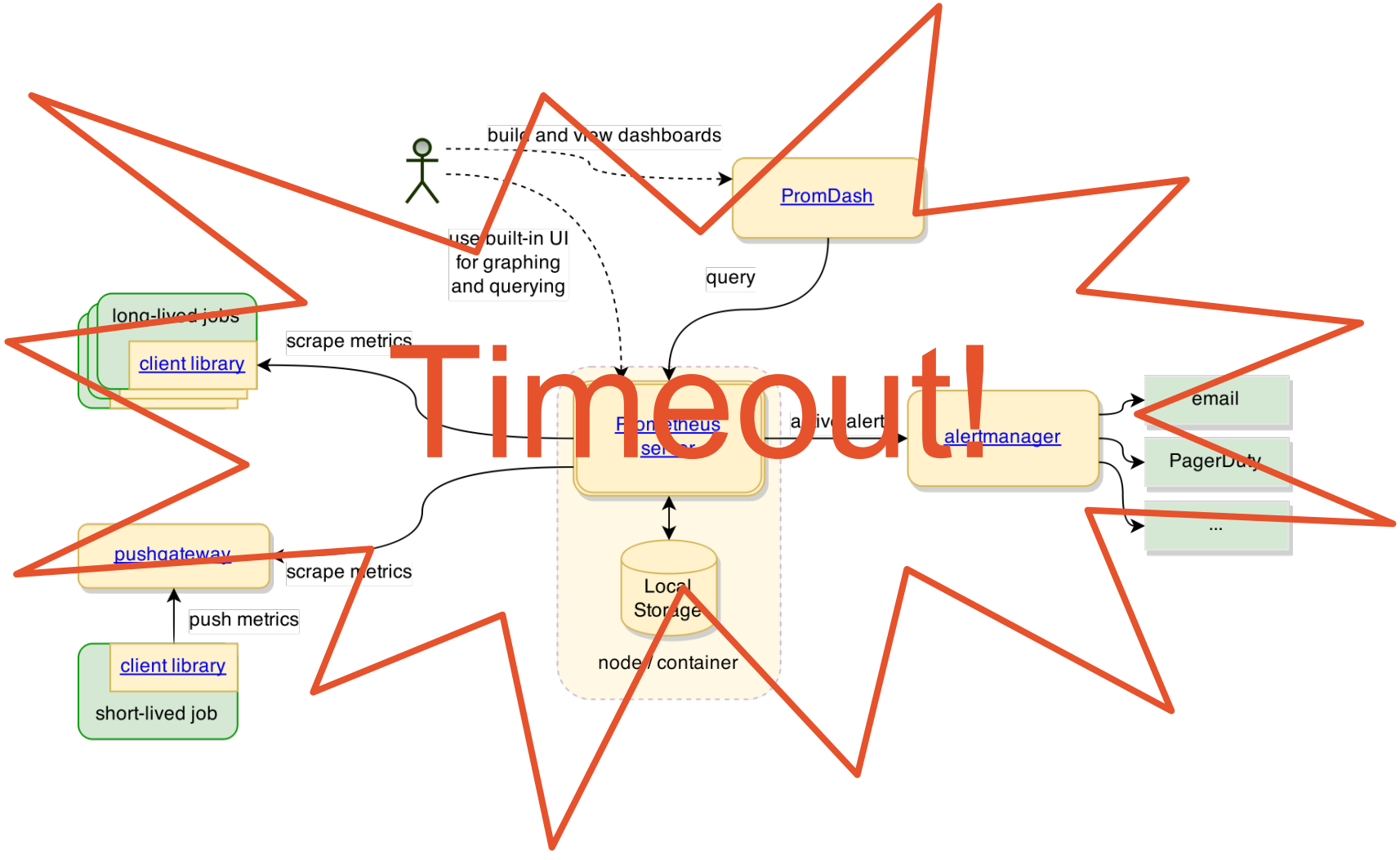
Prometheus

Prometheus is a new open-source service monitoring system and time series database written in Go.

Check out the announcement and my article about monitoring Docker Containers with Prometheus if you don't know what I'm talking about.

My Stack




PromDash
query
client library
scrape metrics
alertmanager
email
PagerDuty
...
pushgateway
scrape metrics
Local Storage
push metrics
client library
short-lived job
Timeout!

Prometheus: How to increment a numerical value

Björn “Beorn” Rabenstein, Production Engineer, SoundCloud Ltd.

1. Use -benchmem.

To detect allocation churn.

```
go test -bench=. -cpu=1,4,16 -benchmem
```

Escape analysis:

```
go test -gcflags=-m -bench=Something
```

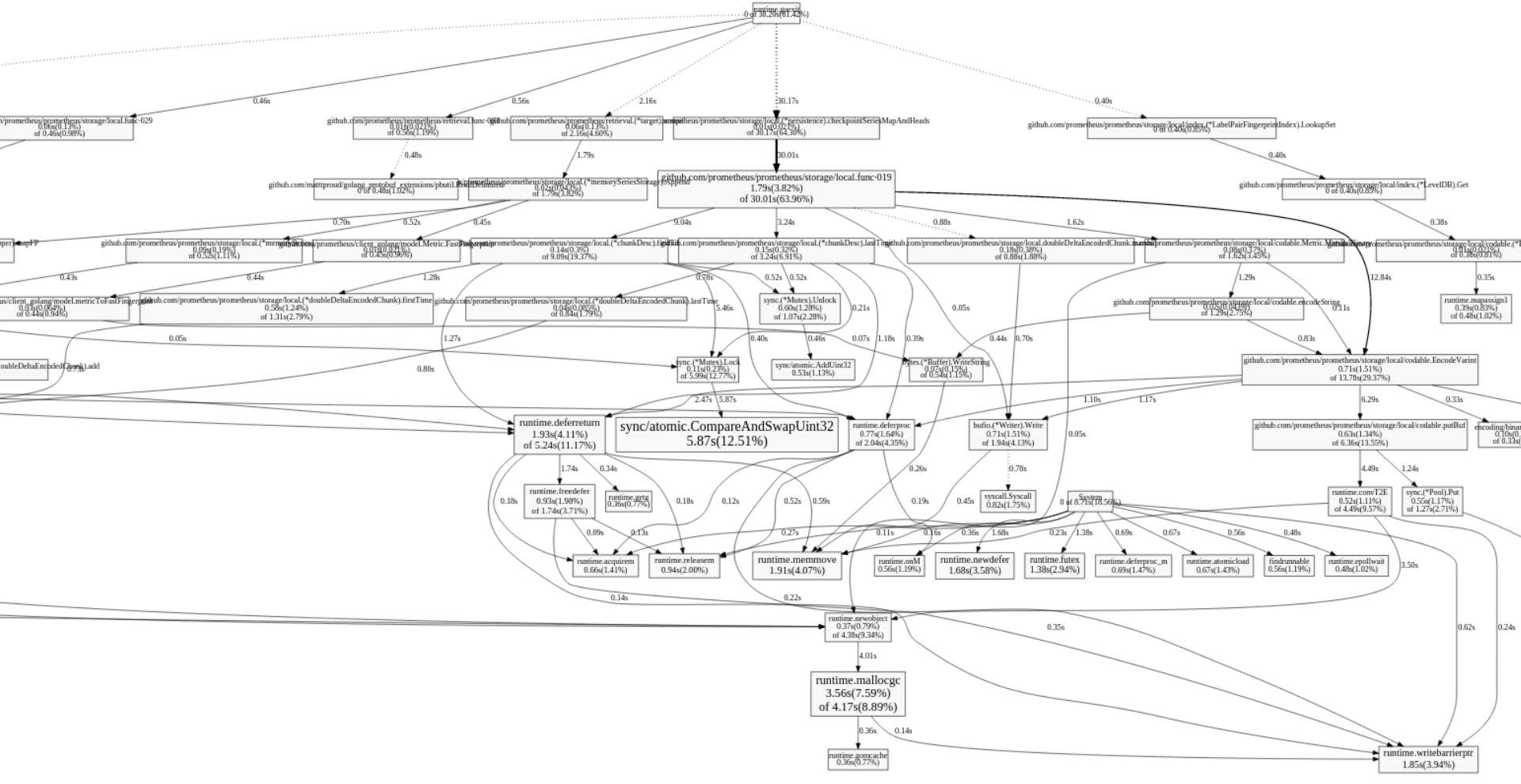
2. Use pprof.

For debugging. For runtime and allocation profiling.

```
import _ "net/http/pprof"
```

```
$ go tool pprof http://localhost:9090/debug/pprof/profile
(pprof) web
```

```
$ go tool pprof http://localhost:9090/debug/pprof/heap
(pprof) web
```


runtime.gcBgMarkWorker
0 of 36.26s(61.42%)
github.com/prometheus/prometheus/storage/local.func-029
0.06s(0.13%)
of 0.46s(0.98%)
0.46s
github.com/prometheus/prometheus/retrieval.func-01
0.01s(0.02%)
of 0.56s(1.19%)
0.56s
github.com/prometheus/prometheus/retrieval.(*target).scrape
0.06s(0.13%)
of 2.16s(4.60%)
2.16s
github.com/prometheus/prometheus/storage/local.(*persistence).checkpointSeriesMapAndHeads
0.01s(0.02%)
of 30.17s(64.30%)
30.17s
0.40s
github.com/prometheus/prometheus/storage/local/index.(*LabelPairFingerprintIndex).LookupSet
0 of 0.40s(0.85%)
0.48s
1.79s
30.01s
0.40s
github.com/mattproud/golang_protobuf_extensions/pbutil.WriteDelimited
0 of 0.48s(1.02%)
github.com/prometheus/prometheus/storage/local.(*memorySeriesStorage).ApplyPatience
0.02s(0.043%)
of 1.79s(3.82%)
github.com/prometheus/prometheus/storage/local.func-019
1.79s(3.82%)
of 30.01s(63.96%)
github.com/prometheus/prometheus/storage/local/index.(*LevelDB).Get
0 of 0.40s(0.85%)
0.70s
0.52s
0.45s
9.04s
3.24s
0.88s
1.62s
0.38s
github.com/prometheus/prometheus/storage/local.(*memorySeriesStorage).fingerprintsForLabelPair
0.09s(0.19%)
of 0.52s(1.11%)
github.com/prometheus/client_golang/model.Metric.FastFingerprint
0.01s(0.021%)
of 0.45s(0.96%)
github.com/prometheus/prometheus/storage/local.(*chunkDesc).firstTime
0.14s(0.3%)
of 9.09s(19.37%)
github.com/prometheus/prometheus/storage/local.(*chunkDesc).lastTime
0.15s(0.32%)
of 3.24s(6.91%)
github.com/prometheus/prometheus/storage/local.doubleDeltaEncodedChunk.marshal
0.18s(0.38%)
of 0.88s(1.88%)
github.com/prometheus/prometheus/storage/local/codable.Metric.MarshalToBuf
0.09s(0.19%)
of 1.62s(3.45%)
github.com/prometheus/prometheus/storage/local/codable.(*
0.01s(0.021%)
of 0.38s(0.81%)
0.43s
0.44s
1.28s
0.78s
0.52s
0.52s
1.29s
12.84s
0.35s
github.com/prometheus/client_golang/model.metricToFastFingerprint
0.03s(0.064%)
of 0.44s(0.94%)
github.com/prometheus/prometheus/storage/local.(*doubleDeltaEncodedChunk).firstTime
0.58s(1.24%)
of 1.31s(2.79%)
github.com/prometheus/prometheus/storage/local.(*doubleDeltaEncodedChunk).lastTime
0.04s(0.085%)
of 0.84s(1.79%)
sync.(*Mutex).Unlock
0.60s(1.28%)
of 1.07s(2.28%)
5.46s
0.21s
0.05s
github.com/prometheus/prometheus/storage/local/codable.encodeString
0.02s(0.043%)
of 1.29s(2.75%)
0.11s
runtime.mapassign1
0.39s(0.83%)
of 0.48s(1.02%)
0.05s
1.27s
0.80s
0.40s
0.46s
0.07s
1.18s
0.39s
0.44s
0.70s
0.83s
sync.(*Mutex).Lock
0.11s(0.23%)
of 5.99s(12.77%)
sync/atomic.AddUint32
0.53s(1.13%)
bytes.(*Buffer).WriteString
0.07s(0.15%)
of 0.54s(1.15%)
github.com/prometheus/prometheus/storage/local/codable.EncodeVarint
0.71s(1.51%)
of 13.78s(29.37%)
2.47s
5.87s
1.10s
1.17s
6.29s
0.33s
runtime.deferreturn
1.93s(4.11%)
of 5.24s(11.17%)
sync/atomic.CompareAndSwapUint32
5.87s(12.51%)
runtime.deferproc
0.77s(1.64%)
of 2.04s(4.35%)
bufio.(*Writer).Write
0.71s(1.51%)
of 1.94s(4.13%)
0.05s
github.com/prometheus/prometheus/storage/local/codable.putBuf
0.63s(1.34%)
of 6.36s(13.59%)
encoding/binar
0.10s(0.
of 0.33s(
1.74s
0.34s
0.26s
0.78s
4.49s
1.24s
runtime.freedefer
0.93s(1.98%)
of 1.74s(3.71%)
runtime.gerg
0.36s(0.77%)
syscall.Syscall
0.82s(1.75%)
System
0 of 8.75s(18.56%)
runtime.convT2E
0.52s(1.11%)
of 4.49s(9.57%)
sync.(*Pool).Put
0.55s(1.17%)
of 1.27s(2.71%)
0.18s
0.18s
0.12s
0.52s
0.59s
0.19s
0.45s
0.09s
0.13s
0.27s
0.11s
0.16s
0.36s
1.68s
0.23s
1.38s
0.69s
0.67s
0.56s
0.48s
3.50s
runtime.acquirem
0.66s(1.41%)
runtime.releasem
0.94s(2.00%)
runtime.memmove
1.91s(4.07%)
runtime.onM
0.56s(1.19%)
runtime.newdefer
1.68s(3.58%)
runtime.futex
1.38s(2.94%)
runtime.deferproc_m
0.69s(1.47%)
runtime.atomicload
0.67s(1.43%)
findrunnable
0.56s(1.19%)
runtime.epollwait
0.48s(1.02%)
0.14s
0.22s
0.35s
0.62s
0.24s
runtime.newobject
0.37s(0.79%)
of 4.38s(9.34%)
4.01s
runtime.mallocgc
3.56s(7.59%)
of 4.17s(8.89%)
0.36s
0.14s
runtime.gomcache
0.36s(0.77%)
runtime.writebarrierptr
1.85s(3.94%)

3. Use cgo judiciously.

Highly optimized C libraries can be great. But there is a cost...

- Loss of certain advantages of the Go build environment.
- Per-call overhead – dominates run-time if C function runs for <1µs.
- Need to shovel input and output data back and forth.

http://jmoiron.net/blog/go-performance-tales/

Special thanks

Matt T. Proud & Julius Volz

founding fathers of the Prometheus project

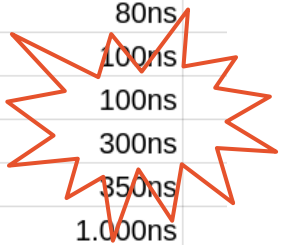
Supplementary slides

Numbers every Go + App Engine dev should know

File Edit View Insert Format Data Tools Add-ons Help View only

L1 cache reference

	A	B
1	L1 cache reference	1ns
2	Branch mispredict	5ns
3	L2 cache reference	7ns
4	Mutex lock/unlock	25ns
5	Linear seek in 100 long slice @ F1	80ns
6	Main memory reference	100ns
7	Go sync.Mutex lock/unlock @ F1	100ns
8	Go channel send/receive @ F1	300ns
9	Go native map retrieval @ F1	350ns
10	Go native map insertion @ F1	1.000ns
11	Call method reflectively @ F1	1.000ns
12	Compress 1kb /w cheap compression algorithm	3.000ns
13	Send 2kb over 1GBPS network	20.000ns
14	Read 1Mb sequentially from memory	250.000ns
15	Roundtrip within datacenter	500.000ns
16	Memcache access @ F1	3.000.000ns
17	Datastore put @ F1	6.000.000ns
18	Datastore get @ F1	6.000.000ns
19	Datastore query @ F1	6.500.000ns
20	Disk seek	10.000.000ns
21	Read 1MB sequentially from disk	20.000.000ns
22	Roundtrip over the atlantic	150.000.000ns
23		



```
type counter struct {
    value int
    mtx   sync.RWMutex
}

func (c *counter) Add(v int) {
    c.mtx.Lock()
    defer c.mtx.Unlock()
    if v < 0 {
        panic(errors.New("counter cannot decrease in value"))
    }
    c.value += v
}

func (c *counter) Inc() {
    c.Add(1)
}

func (c *counter) Write(*dto.Metric) error {
    c.mtx.RLock()
    defer c.mtx.RUnlock()
    // ...
}
```

RWMutex

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	170	180	210
GOMAXPROCS=4	170	820	680
GOMAXPROCS=16	170	1300	1200

```
func (c *counter) loop() {
    var value float64
    for {
        select {
        case v := <-c.write:
            value += v
        default:
            select {
            case v := <-c.write:
                value += v
            case c.read <- value:
                // Do nothing.
            }
        }
    }
}
```

Tricky channel counter.

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	117 ↓	130	164
GOMAXPROCS=4	389 ↑↑	707	1044 ↑↑
GOMAXPROCS=16	388 ↑↑	1297	1707 ↑

Channel counter without `Write`.

ns/op	1 Goroutine	10 Goroutines	100 Goroutines
GOMAXPROCS=1	240 / 73	254 / 75	260 / 82
GOMAXPROCS=4	1040 / 150	760 / 290	500 / 630
GOMAXPROCS=16	1040 / 150	700 / 360	510 / 460