Graphite@Scale:

How to store million metrics per second

Booking.com

Vladimir Smirnov
System Administrator

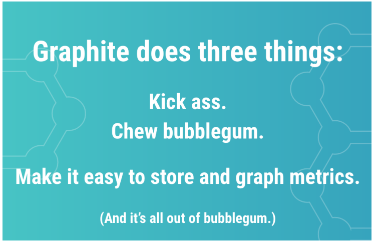
GrafanaCon 2016
1 December 2016

Why you might need to store your metrics?

Most common cases:

- Capacity planning
- Troubleshooting and Postmortems
- Visualization of business data
- And more...

Graphite and its modular architecture



Graphite does three things:

Kick ass.

Chew bubblegum.

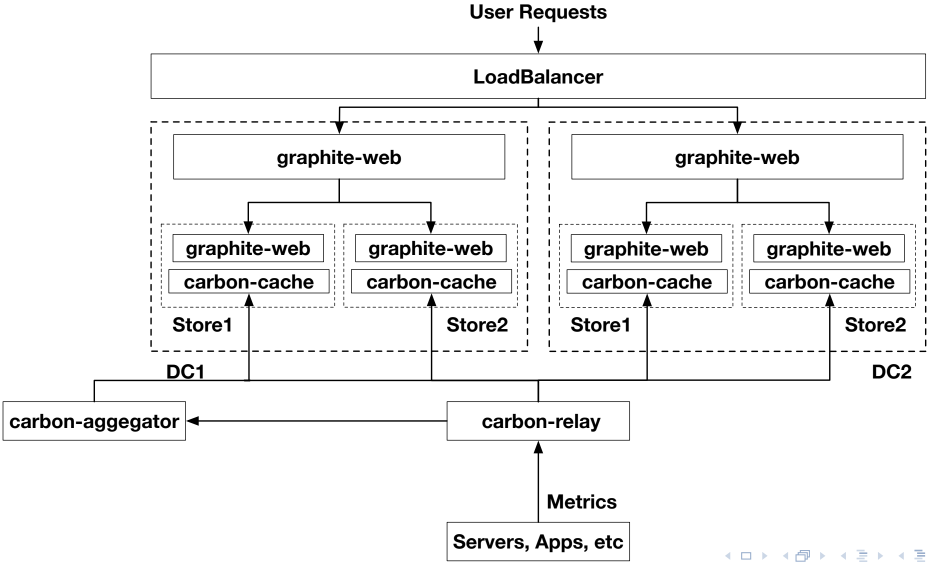
Make it easy to store and graph metrics.

(And it's all out of bubblegum.)

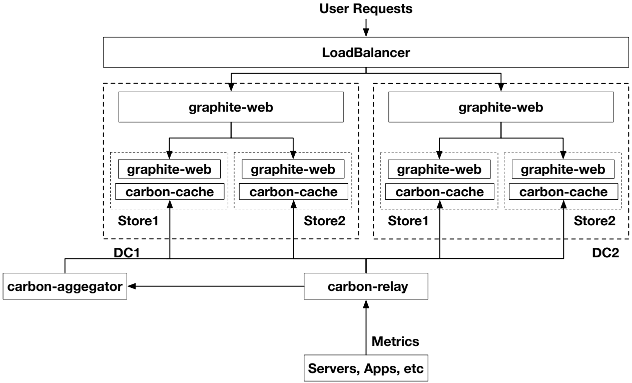
From the graphiteapp.org

- Allows to store time-series data
- Easy to use — text protocol and HTTP API
- You can create any data flow you want
- Modular — you can replace any part of it

Open Source stack



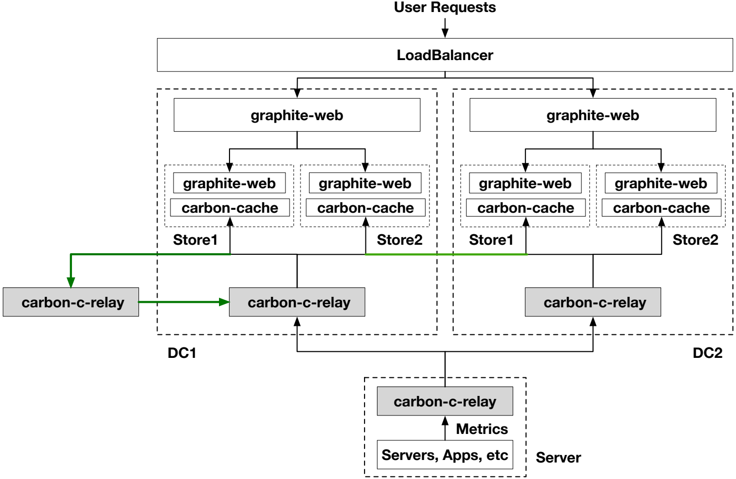
Breaking graphite: our problems at scale



What's wrong with this schema?

- carbon-relay — SPOF
- Doesn't scale well
- Stores may have different data after failures
- Render time increases with more store servers

Replacing carbon-relay



Replacing carbon-relay

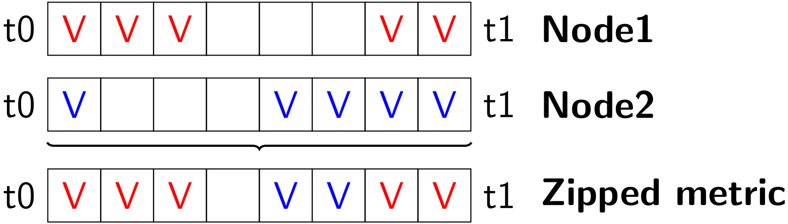
carbon-c-relay:

- Written in **C**
- Routes **1M** data points per second using only **2** cores
- L7 LB for graphite line protocol (RR with sticking)
- Can do aggregations
- Buffers the data if upstream is unavailable

Zipper stack: Solution

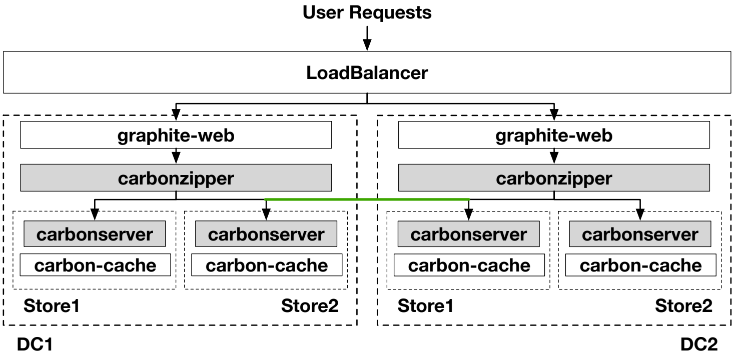
Query: target=sys.server.cpu.user

Result:



t0	V	V	V				V	V	t1	**Node1**
t0	V				V	V	V	V	t1	**Node2**
t0	V	V	V		V	V	V	V	t1	**Zipped metric**

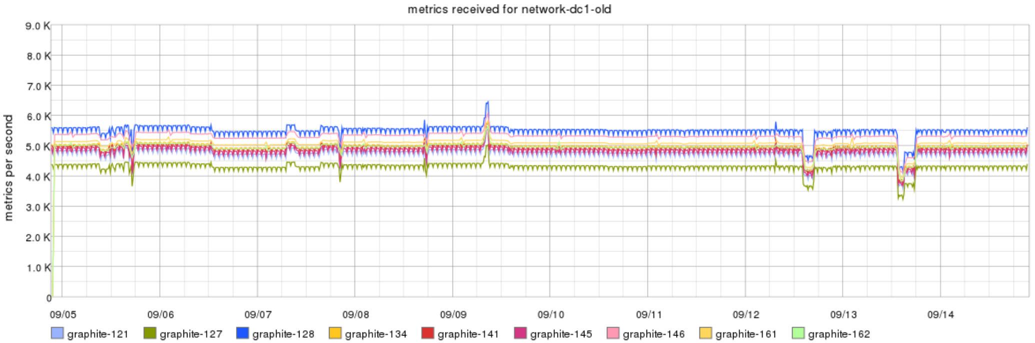
Zipper stack: architecture



Zipper stack: results

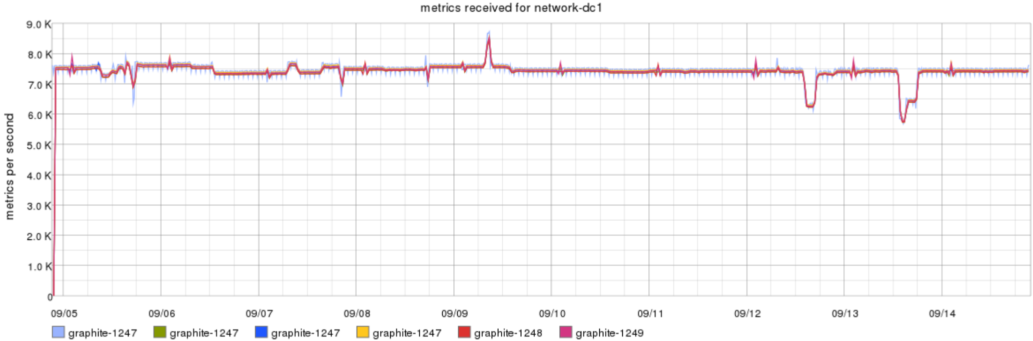
- Written in **Go**
- Can query store servers in **parallel**
- Can "Zip" the data
- carbonzipper ⇔ carbonserver — **2700** RPS
  graphite-web ⇔ carbon-cache — **80** RPS.

Metric distribution: how it works

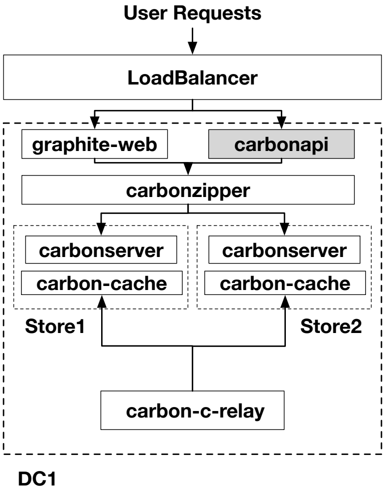


Up to **20%** difference in worst case

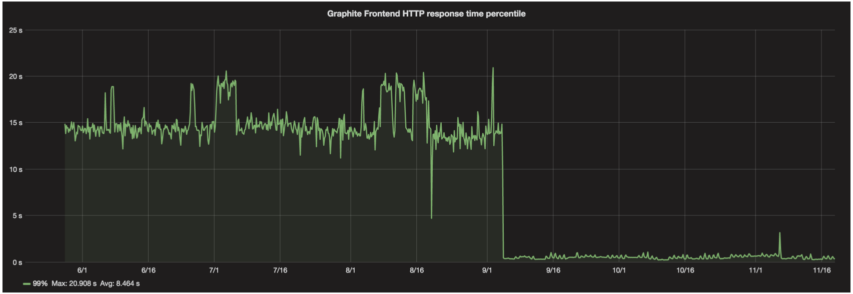
Metric distribution: jump hash



Rewriting Frontend in Go: carbonapi

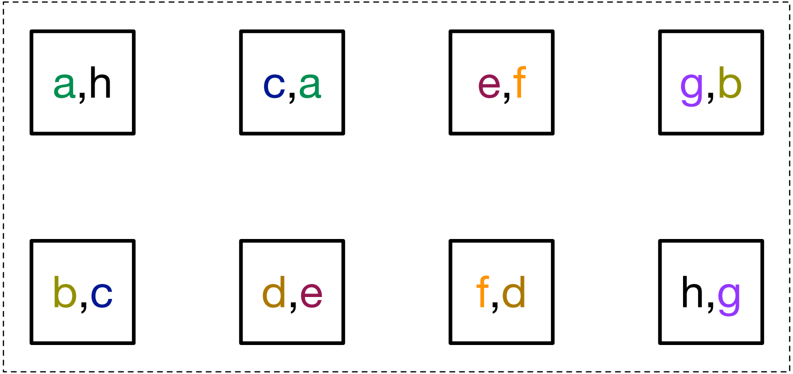


Rewriting Frontend in Go: result



- Significantly reduced response time for users (**15s** ⇒ **0.8s**)
- Allowes more complex queries because it’s faster
- Easier to implement new heavy math functions
- Also available as Go library

Replication techniques and their pros and cons

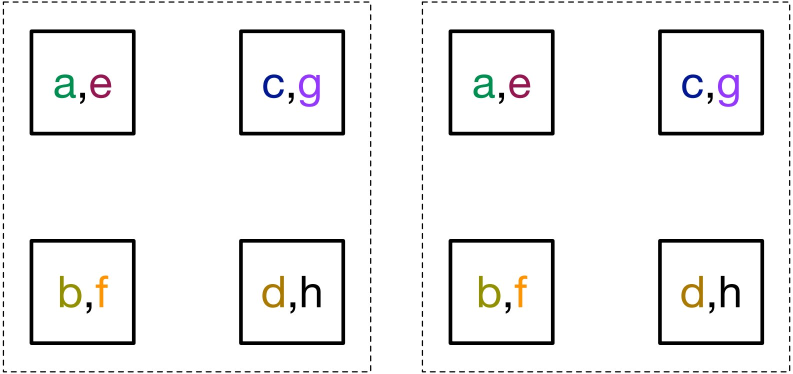


a,h c,a e,f g,b

b,c d,e f,d h,g

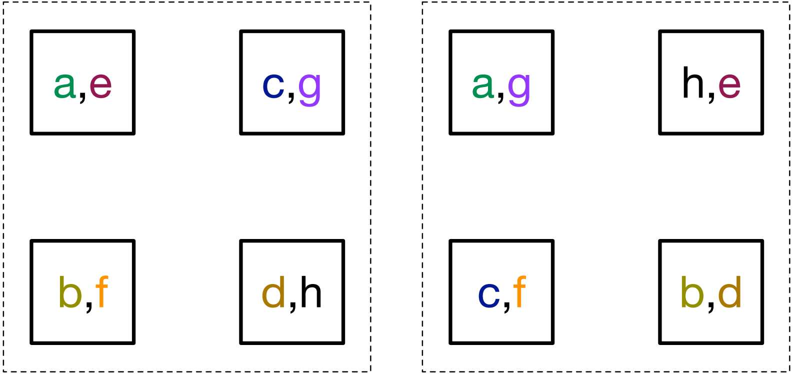
Replication Factor 2

Replication techniques and their pros and cons



Replication Factor 1

Replication techniques and their pros and cons



a,e

c,g

b,f

d,h

a,g

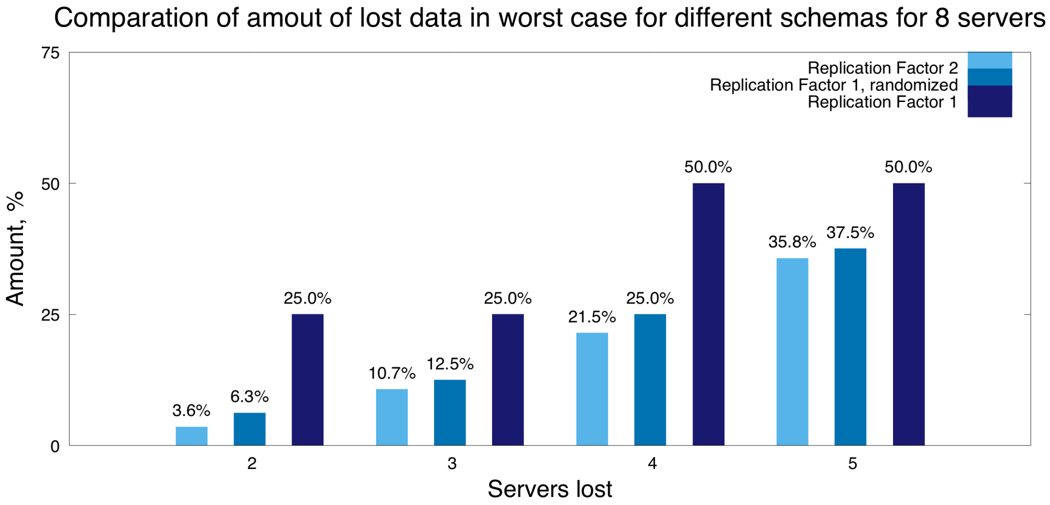
h,e

c,f

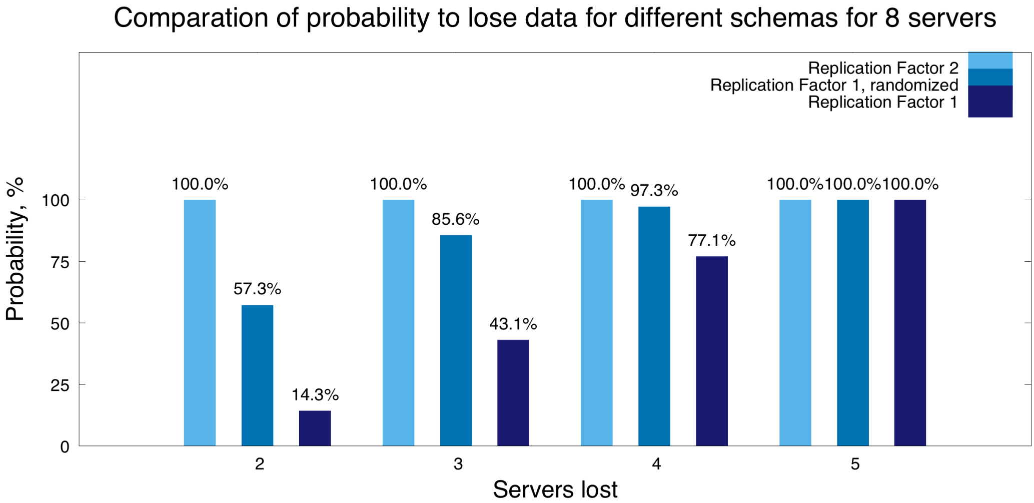
b,d

Replication Factor 1, randomized

Replication techniques and their pros and cons



Replication techniques and their pros and cons



Our current setup

- **32** Frontend Servers
- **200** RPS on Frontend
- **30k** Metric Requests per second
- **18 Gbps** traffic on the backend
- **200** Store servers in 2 DCs
- **2M** unique metrics per second (**8M** hitting stores)
- **53M** unique metrics stored
- **200+ TB** of Metrics in total
- Replaced **all** the components

What’s next?

- Metadata search (in progress)
- Solve problems with missing Cache (in progress)
- Find a replacement for Whisper
- Improve aggregators
- Replace graphite line protocol between components

It’s all Open Source!

- carbonzipper — github.com/dgryski/carbonzipper
- carbonserver — github.com/grobian/carbonserver
- carbonapi — github.com/dgryski/carbonapi
- carbon-c-relay — github.com/grobian/carbon-c-relay
- carbonmem — github.com/dgryski/carbonmem
- carbonsearch — github.com/kanatohodets/carbonsearch
- go-carbon — github.com/lomik/go-carbon (Not a Booking.com project)
- replication factor test — github.com/Civil/graphite-rf-test

Questions?

vladimir.smirnov@booking.com

Thanks!

We are hiring!
https://workingatbooking.com