Graphite@Scale:
How to store millions metrics per second

Booking.com

Vladimir Smirnov
System Administrator

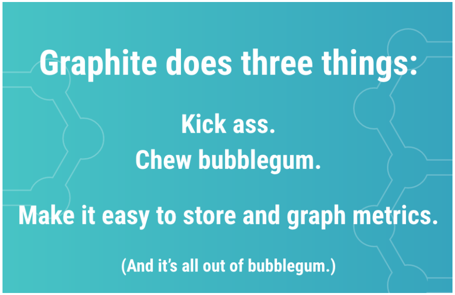
FOSDEM 2017
5 February 2017

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.

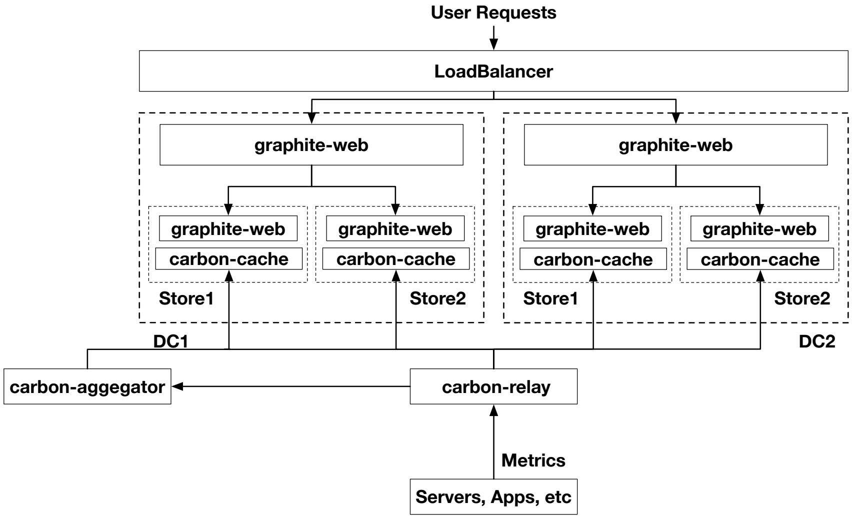
Make it easy to store and graph metrics.

(And it's all out of bubblegum.)

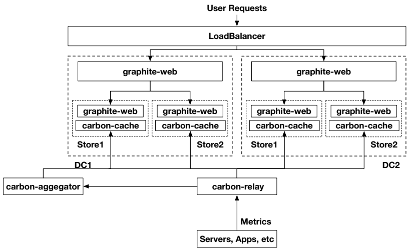
From the graphiteapp.org

- 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



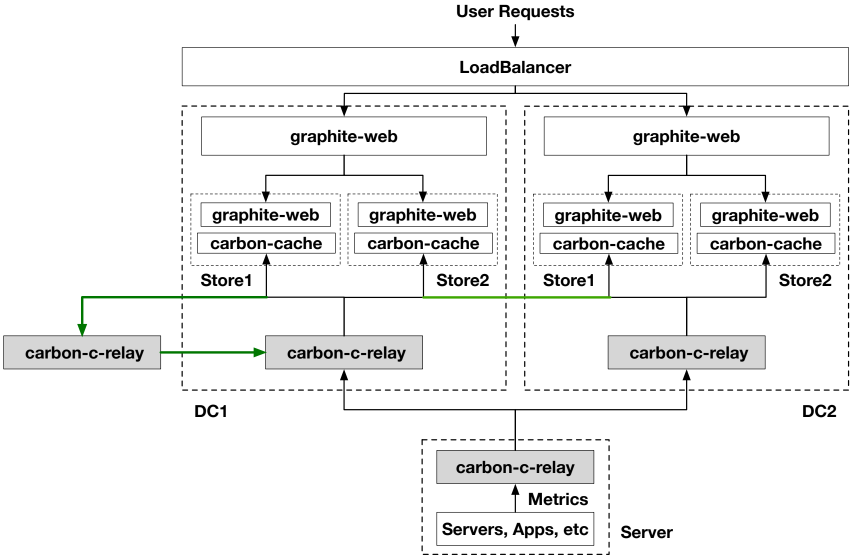
Breaking graphite: our problems at scale



What's wrong with this schema?

- carbon-relay — SPOF
- Hard to scale
- Data is different after failures
- Render time increases with more servers

Replacing carbon-relay



Replacing carbon-relay

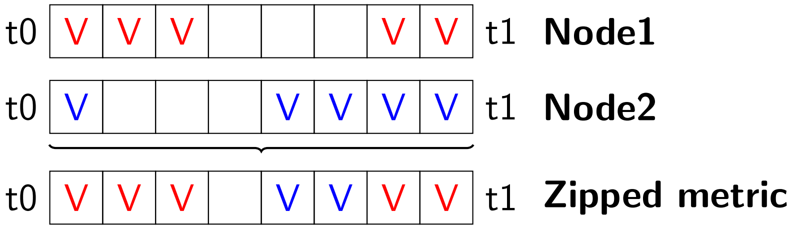
carbon-c-relay:

- 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

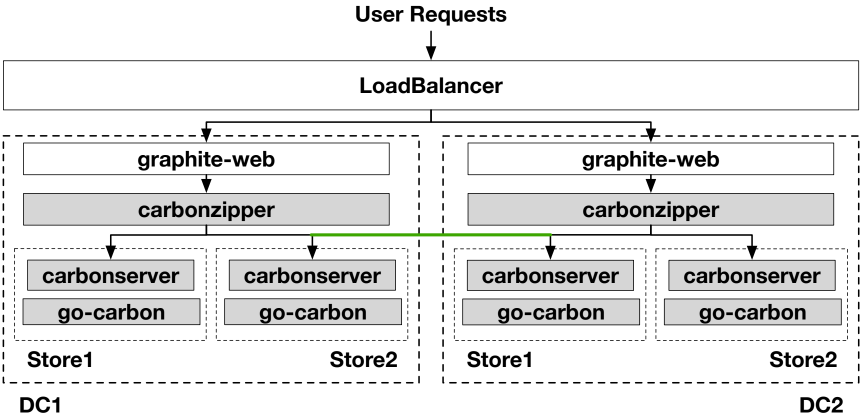
Zipper stack: Solution

Query: target=sys.server.cpu.user

Result:



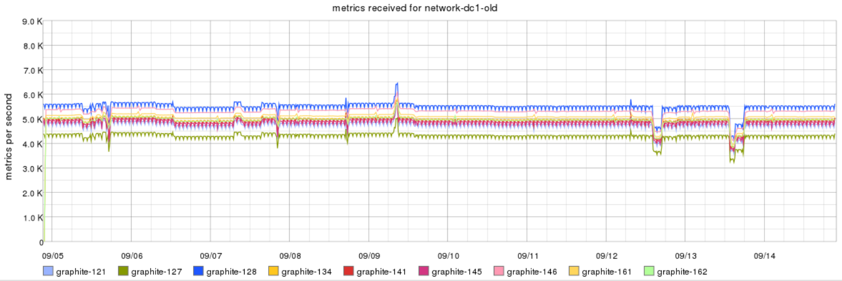
Zipper stack: architecture



Zipper stack: results

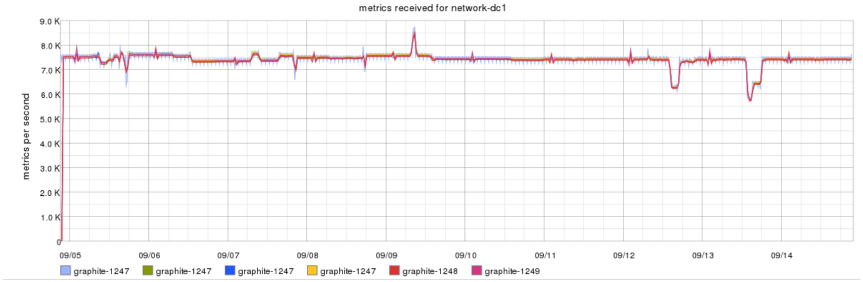
- Written in **Go**
- Can query store servers in **parallel**
- Can "Zip" the data
- carbonzipper ⇔ carbonserver — **2700** RPS
  graphite-web ⇔ carbon-cache — **80** RPS.
- carbonserver is now part of go-carbon (since December 2016)

Metric distribution: how it works



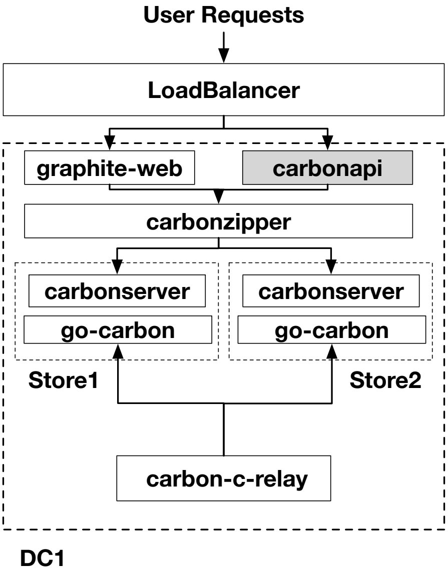
Up to **20%** difference in worst case

Metric distribution: jump hash

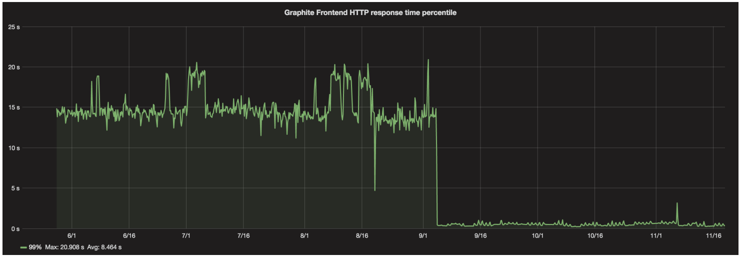


arxiv.org/pdf/1406.2294v1.pdf

Rewriting Frontend in Go: carbonapi

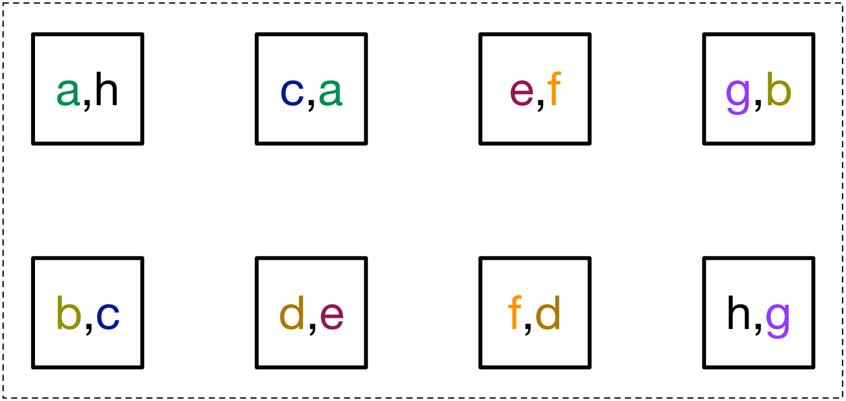


Rewriting Frontend in Go: result



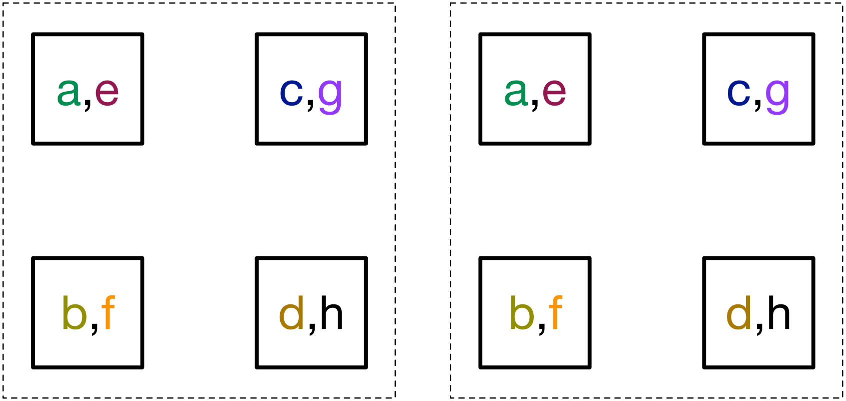
- Significantly reduced response time for users (**15s** $\Rightarrow$ **0.8s**)
- Allowes more complex queries because it's faster
- Easier to implement new heavy math functions
- Also available as Go library

Replication techniques and their pros and cons



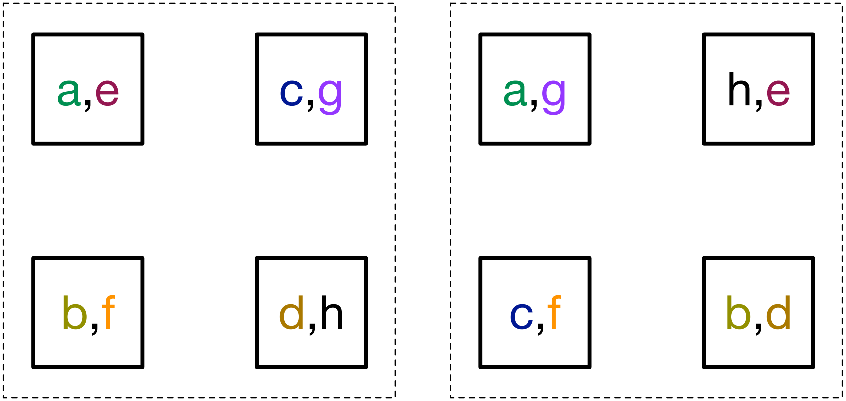
Replication Factor 2

Replication techniques and their pros and cons



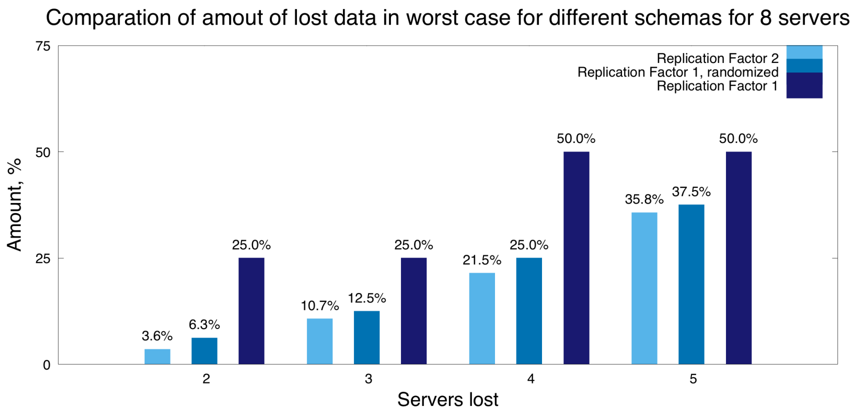
Replication Factor 1

Replication techniques and their pros and cons

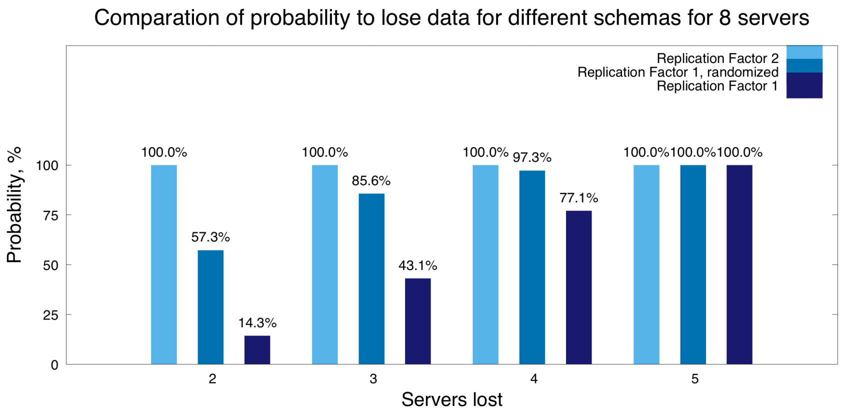


Replication Factor 1, randomized

Replication techniques and their pros and cons



Replication techniques and their pros and cons



Our current setup

- **32** Frontend Servers
- **400** RPS on Frontend
- **40k** Metric Requests per second
- **11 Gbps** traffic on the backend
- **200** Store servers in 2 DCs
- **2.5M** unique metrics per second (**10M** hitting stores)
- **130 TB** of Metrics in total
- Replaced **all** the components

What's next?

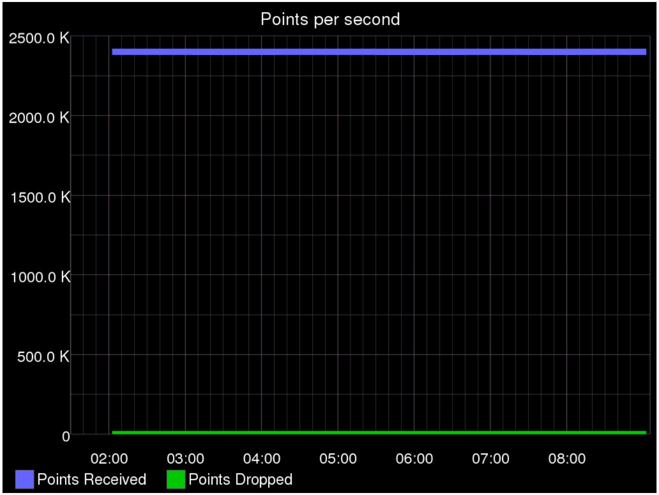
- Metadata search (in progress)
- Find a replacement for Whisper (in progress)
- Rethink aggregators
- Replace graphite line protocol between components

Bonus 0: carbonsearch — WIP tags support in graphite

Example:
target=sum(virt.v1.*.dc:datacenter1.status:live.role:graphiteStore.text-match:metricsReceived)

- Separate tags stream and storage
- No history (yet)
- No negative match support (yet)
- Only "and" syntax
- Just a few months old

Bonus 1: testing Clickhouse on a single server



It's all Open Source!

- carbonzipper — github.com/dgryski/carbonzipper
- go-carbon — github.com/lomik/go-carbon
- carbonsearch — github.com/kanatohodets/carbonsearch
- carbonapi — github.com/dgryski/carbonapi
- carbon-c-relay — github.com/grobian/carbon-c-relay
- carbonmem — github.com/dgryski/carbonmem
- replication factor test — github.com/Civil/graphite-rf-test

Questions?

vladimir.smirnov@booking.com

What's next?

Thanks!