Statistics for Engineers

Monitorama PDX, June 29th 2016

Heinrich Hartmann, Circonus

CIRCONUS

Hi, I am Heinrich

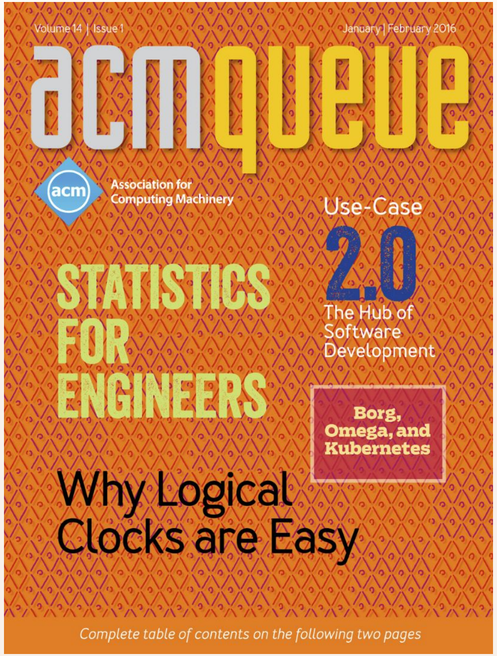
heinrich.hartmann@circonus.com

@HeinrichHartman(n)

- Lives in Munich, EU 🇪🇺
- Refugee from Academia (Ph.D.)
- Analytics Lead at Circonus,

  Monitoring and Analytics Platform

#StatsForEngineers has been around for a while


Volume 14 | Issue 1
January | February 2016
acmqueue
acm Association for Computing Machinery
STATISTICS FOR ENGINEERS
Use-Case 2.0 The Hub of Software Development
Borg, Omega, and Kubernetes
Why Logical Clocks are Easy
Complete table of contents on the following two pages

[1] Statistics for Engineers @ ACM Queue

[2] Statistics for Engineers Workshop Material @ GitHub

[3] Spike Erosion @ circonus.com

[4] T. Schlossnagle - The Problem with Math @ circonus.com

[5] T. Schlossnagle - Percentages are not People @ circonus.com

[6] W. Vogels - Service Level Agreements in Amazon's Dynamo/Sec. 2.2

[7] G. Schlossnagle - API Performance Monitoring @ Velocity Bejing 2015

Upcoming

[8] 3h workshop "Statistics for Engineers" @ SRECon 2016 in Dublin

A tale of API Monitoring

“Attic” - a furniture webstore

- Attic is a (fictional) furniture webstore
- Web API serving their catalog
- Loses money if requests take too long

Monitoring Goals

1. Measure user experience / quality of service
2. Determine (financial) implications of service degradation
3. Define sensible SLA-targets for the Dev- and Ops-teams

{1} External Monitoring

{1} External API Monitoring

Method

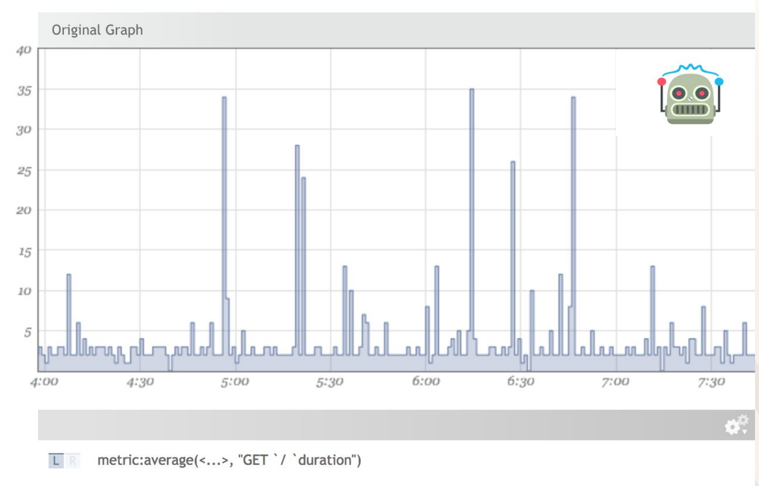
1. Make a synthetic request every minute
2. Measure and store request latency

Good for

- Measure Availability
- Alert on outages

Bad for

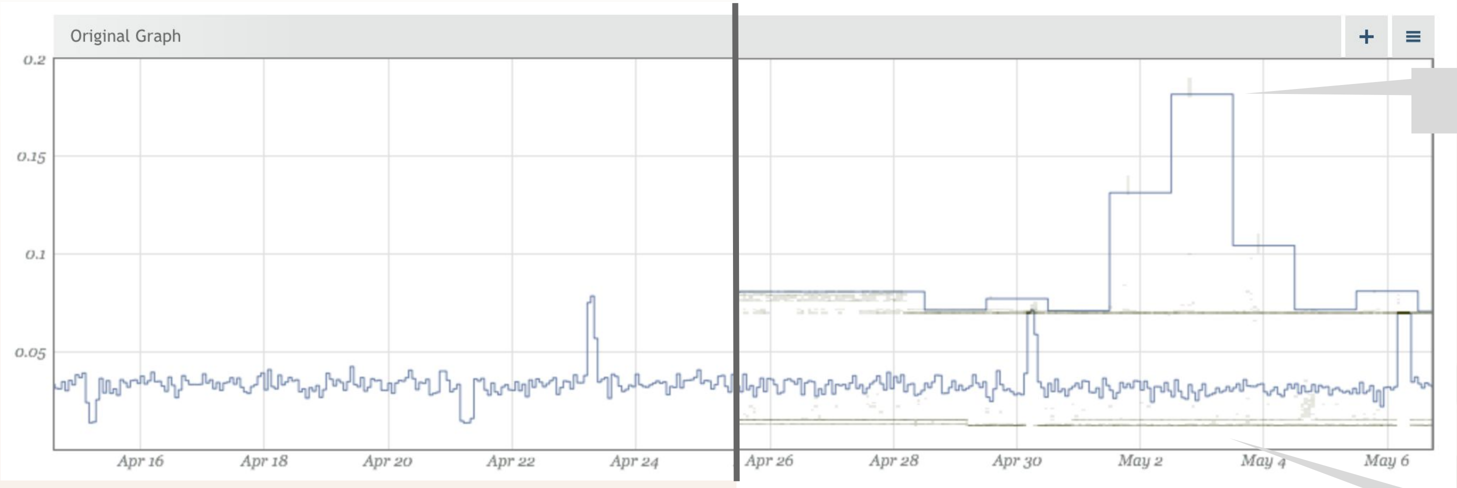
- Measuring user experience



metric:average(<...>, "GET `/ `duration")

Latencies of synthetic requests over time

<!> Spike Erosion </!>



1d max

all samples as Heatmap / ‘dirt’

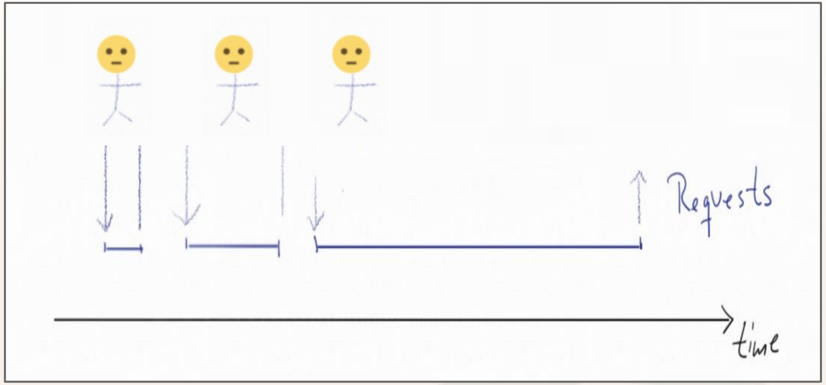
- On long time ranges, aggregated / rolled-up data is commonly displayed
- This practice “erodes” latency spikes heavily!
- Store all data and use alternative aggregation methods (min/max) to get full picture, cf. [3].

{2} Log Analysis

{2} Log Analysis

Method

Write to log file:
- time of completion,
- request latency,
and further metadata.



Internal view of an API - “UML” version.

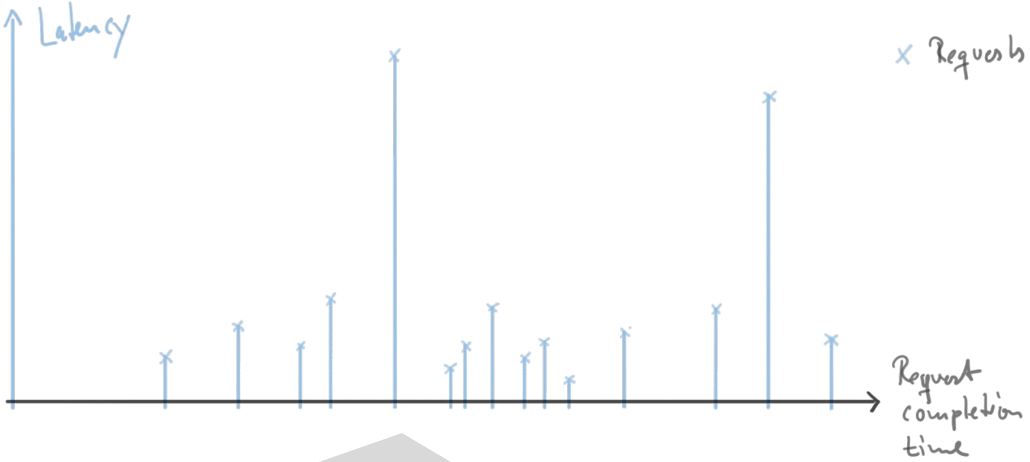
Discussion

- Rich information source for all kinds of analysis
- Easy instrumentation (printf)
- Slow. Long delay (minutes) before data is indexed and becomes accessible for analysis
- Expensive. Not feasibile for high volume APIs

Numerical Digest: The Request-Latency Chart

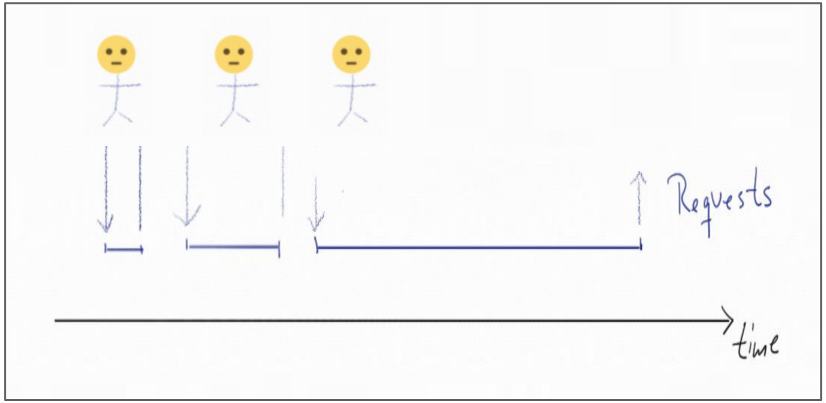
a concise visualization of the API usage

Latency on the y-axis

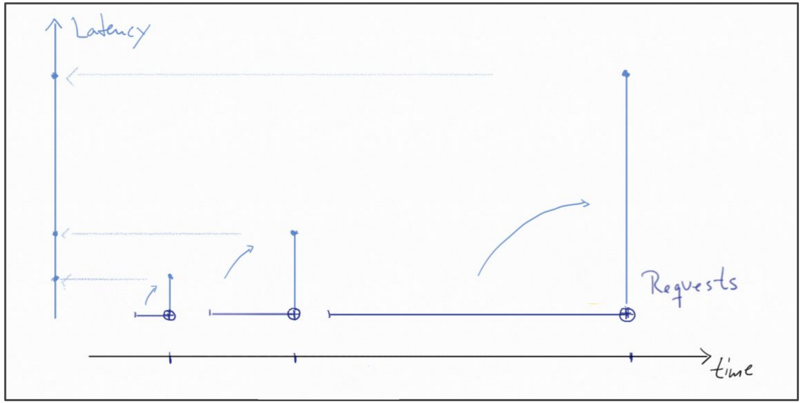


time the request was completed

Construction of the Request-Latency Chart (RLC)

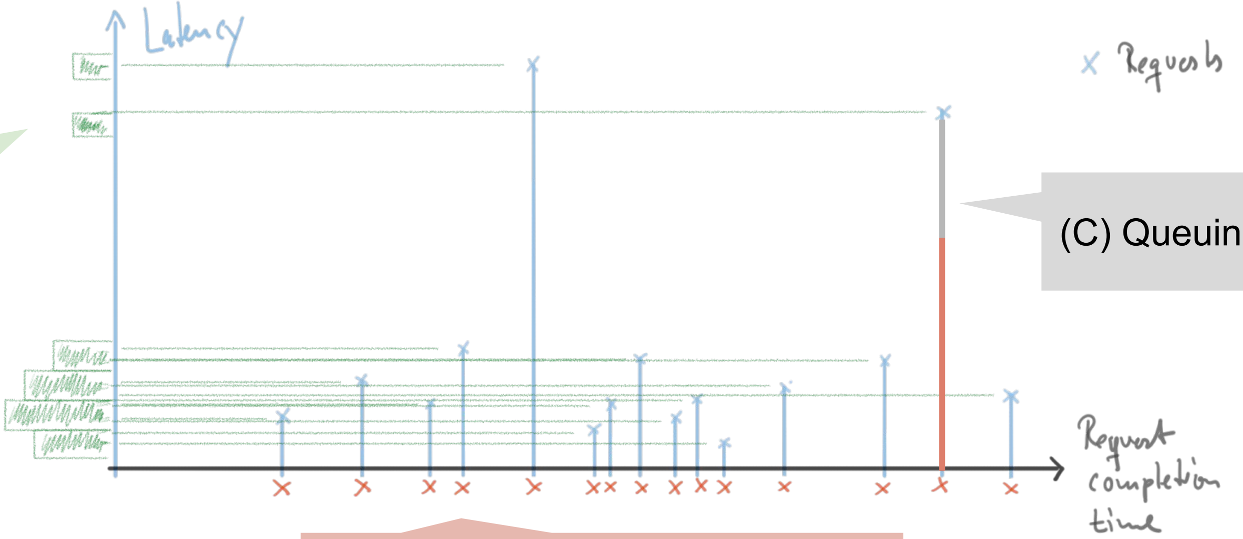


Request Latency UML Diagram



Request Latency Chart

Math view on APIs



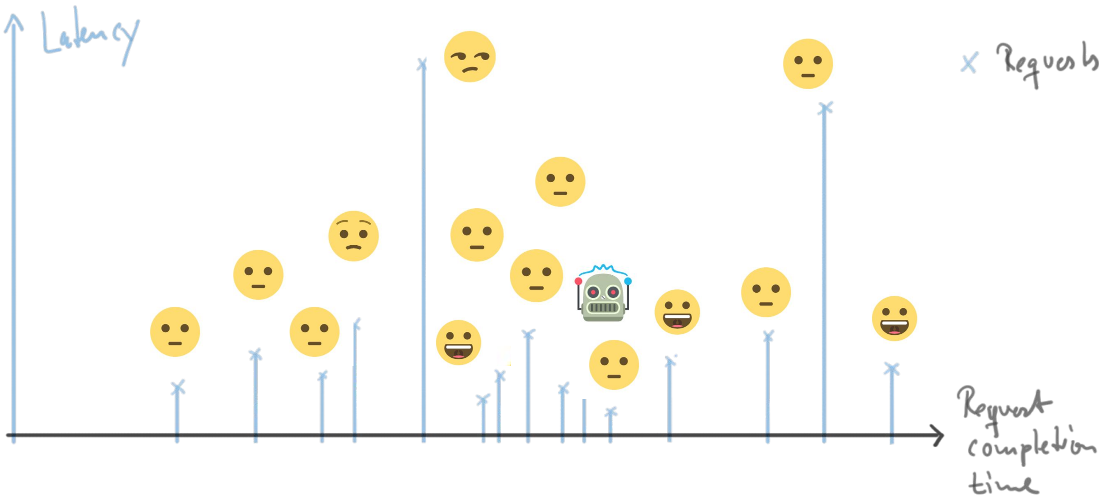
(A) Latency distribution

(C) Queuing theory

(B) Arrival/Completion times

@HeinrichHartman

“Requests are People”



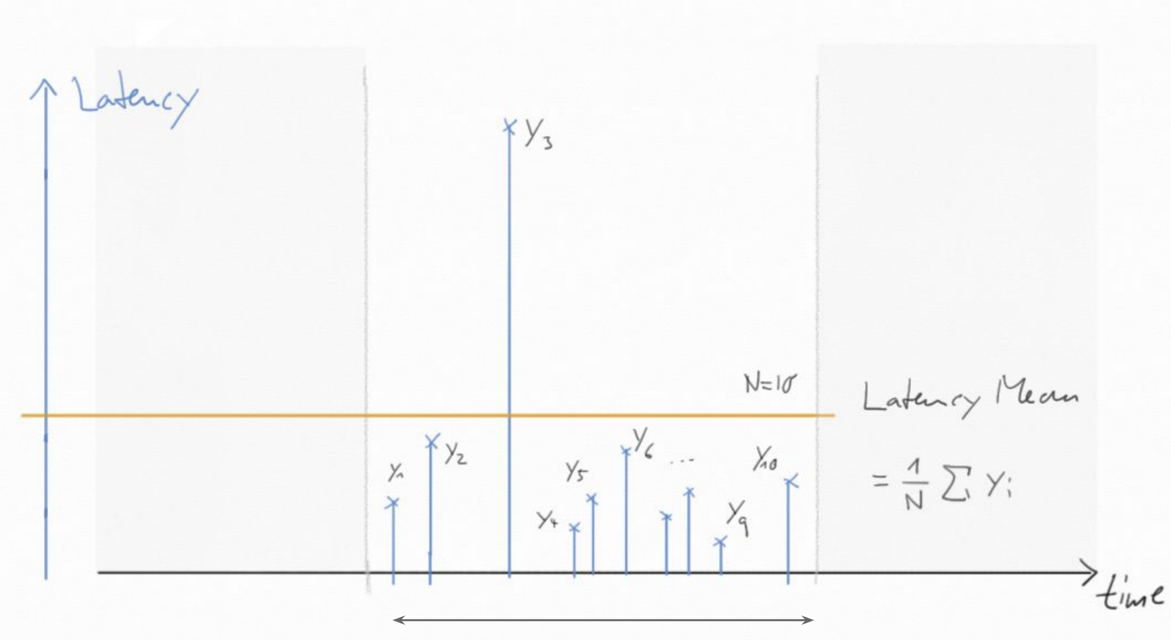
If you care about your users, you care about their requests.

Every single one.

@HeinrichHartman

{3} Monitoring Latency Averages

{3} What are latency mean values?



reporting period

@HeinrichHartman

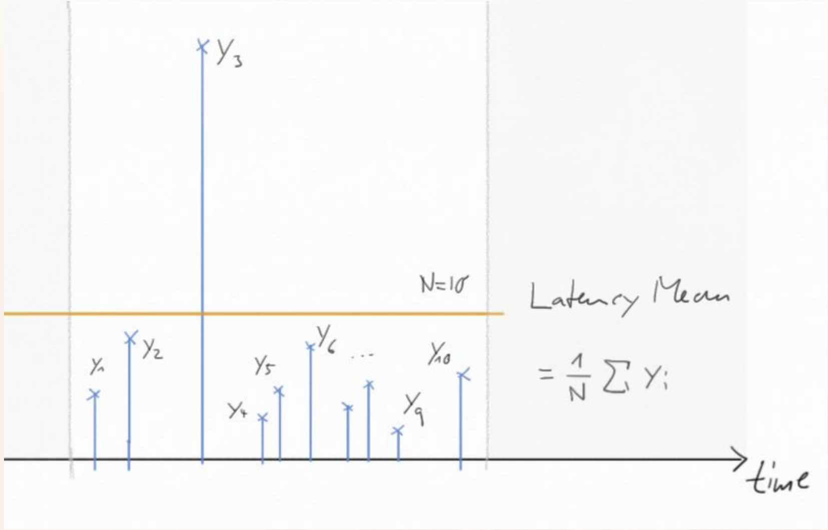
{3} Mean Request Latency Monitoring

Method

1. Select a reporting period (e.g. 1 min)
2. For each period report the mean latency

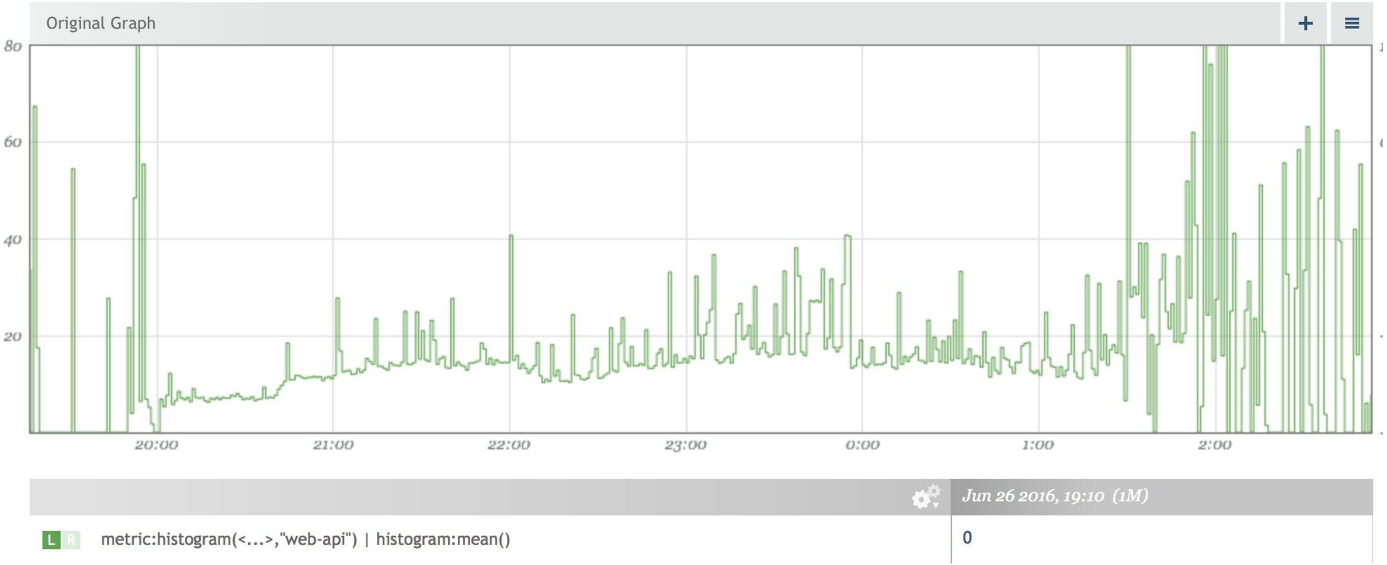
Pro/Con

+ Measure requests by actual people
+ Cheap to collect store and analyze
- Easily skewed by outliers at the high end (complex, long running requests)
- ... and the low end (cached responses)



“Measuring the average latency is like measuring the average temperature in a hospital.”
-- Dogan @ Optimizely

{3} Mean Request Latency in practice



Jun 26 2016, 19:10 (1M)

metric:histogram(<...>,"web-api") | histogram:mean()

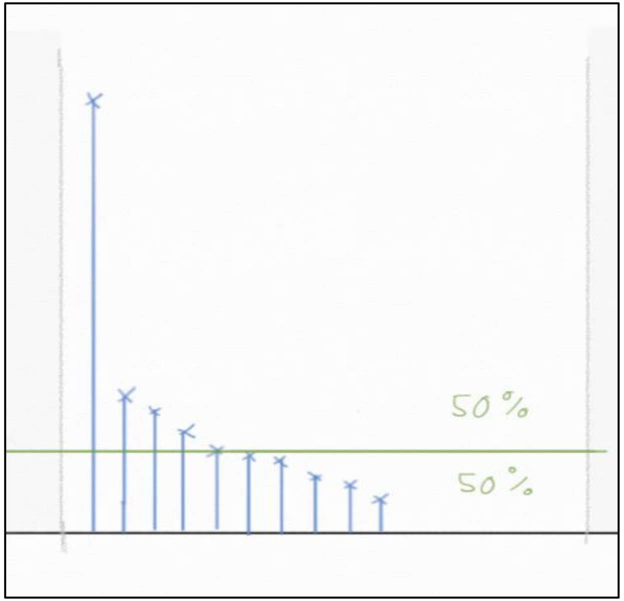
0

{3} Mean Request Latency - Robust Variants

1. Median Latency
   - Sort latency values in reporting period
   - The median is the ‘central’ value.

2. Truncated Means
   - Take out min and max latencies in reporting period (k-times).
   - Then compute the mean value

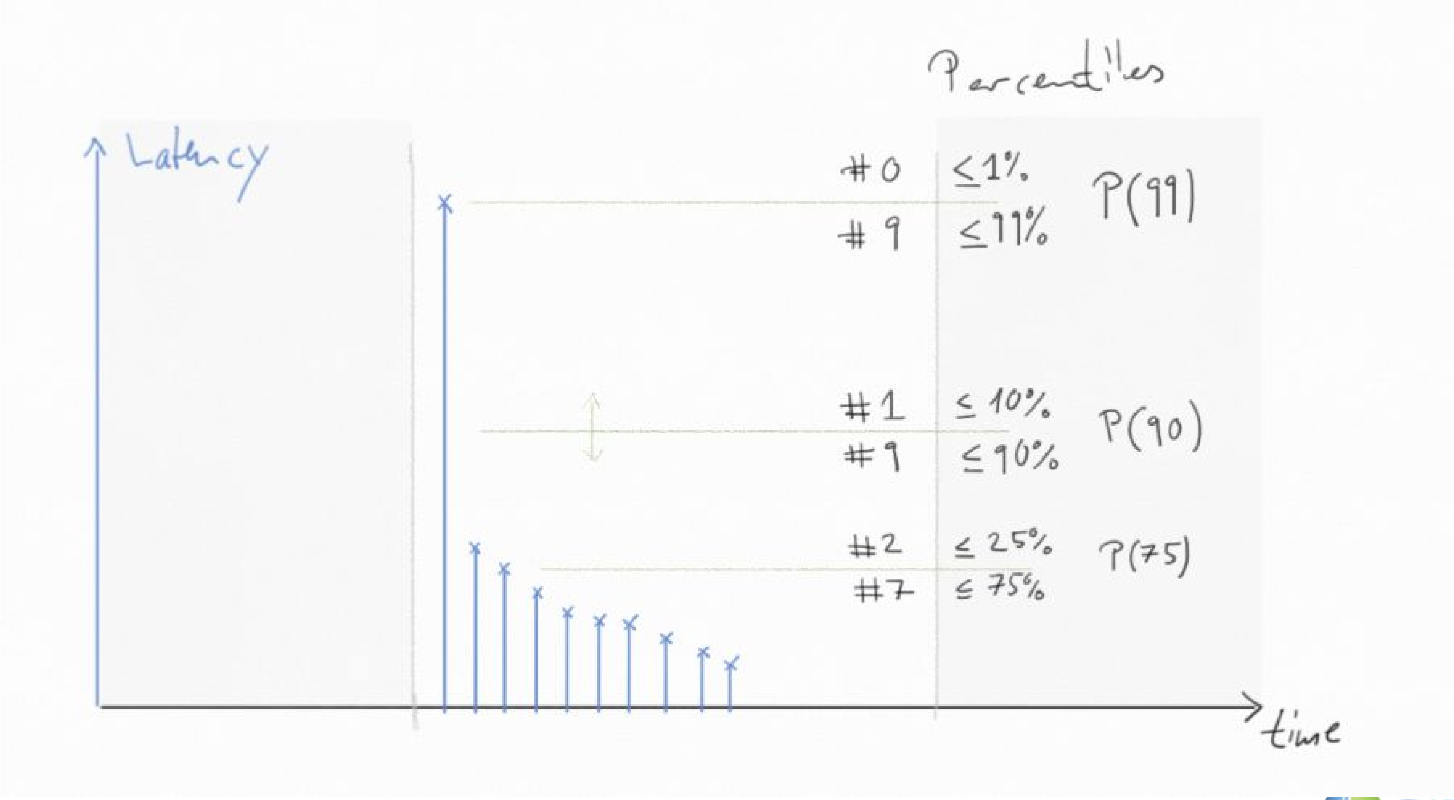
3. Collect Deviation Measures
   - Avoid standdard deviations, use
   - Use Mean absolute deviation



Construction of the median latency

{4} Percentile Monitoring

{4} What are Percentiles?



{4} Percentile Monitoring

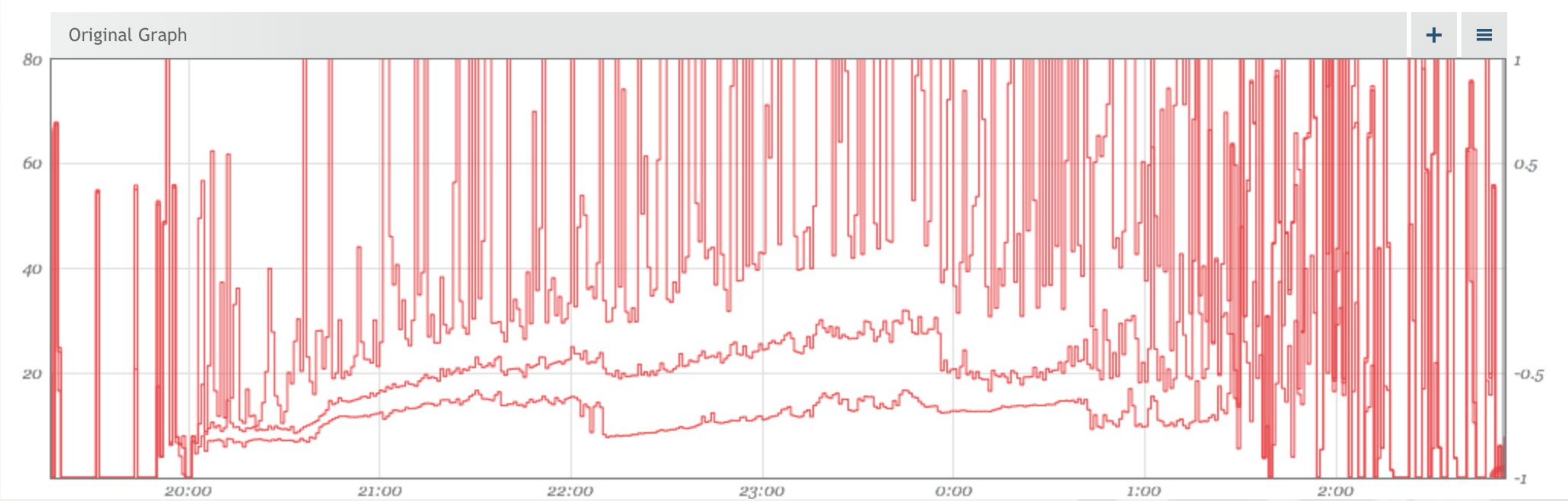
Method

1. Select a reporting period (e.g. 1 min)
2. For each reporting period measure the 50%, 90%, 99%, 99.9% latency percentile
3. Alert when percentiles are over a threshold value

Pro/Con

+ Measure requests by actual people
+ Cheap to collect store and analyze
+ Robust to Outliers
- Up-front choice of percentiles needed
- Can not be aggregated

{5} How it looks in practice



Latency percentiles 50,90,99 computed over 1m reporting periods

<!> Percentiles can’t be aggregated </!>

The median of two medians is NOT the total median.

If you store percentiles you need to:

A. Keep all your data. Never take average rollups!
B. Store percentiles for all aggregation levels separately, e.g.
   - per Node / Rack / DC
   - per Endpoint / Service
C. Store percentiles for all reporting periods you are interested in, e.g. per min / h / day
D. Store all percentiles you will ever be interested in, e.g. 50, 75, 90, 99, 99.9

Further Reading: [4] T. Schlossnagle - The Problem with Math @ circonus.com

{5} API Monitoring with Histograms

{5} API Monitoring with Histograms

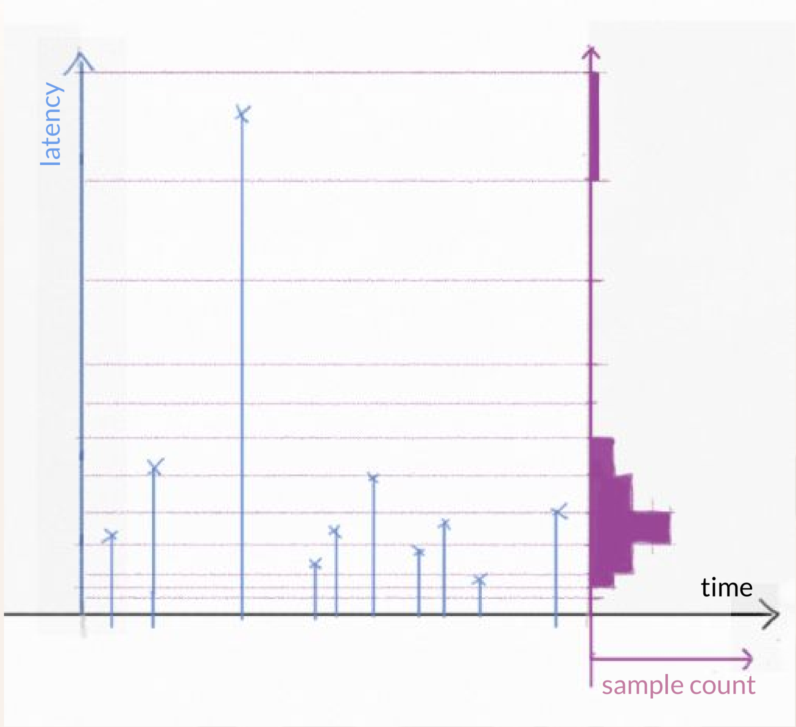
Method

1. Divide latency scale into bands
2. Divide the time scale into reporting periods
3. Count the number of samples in each

   latency band x reporting period

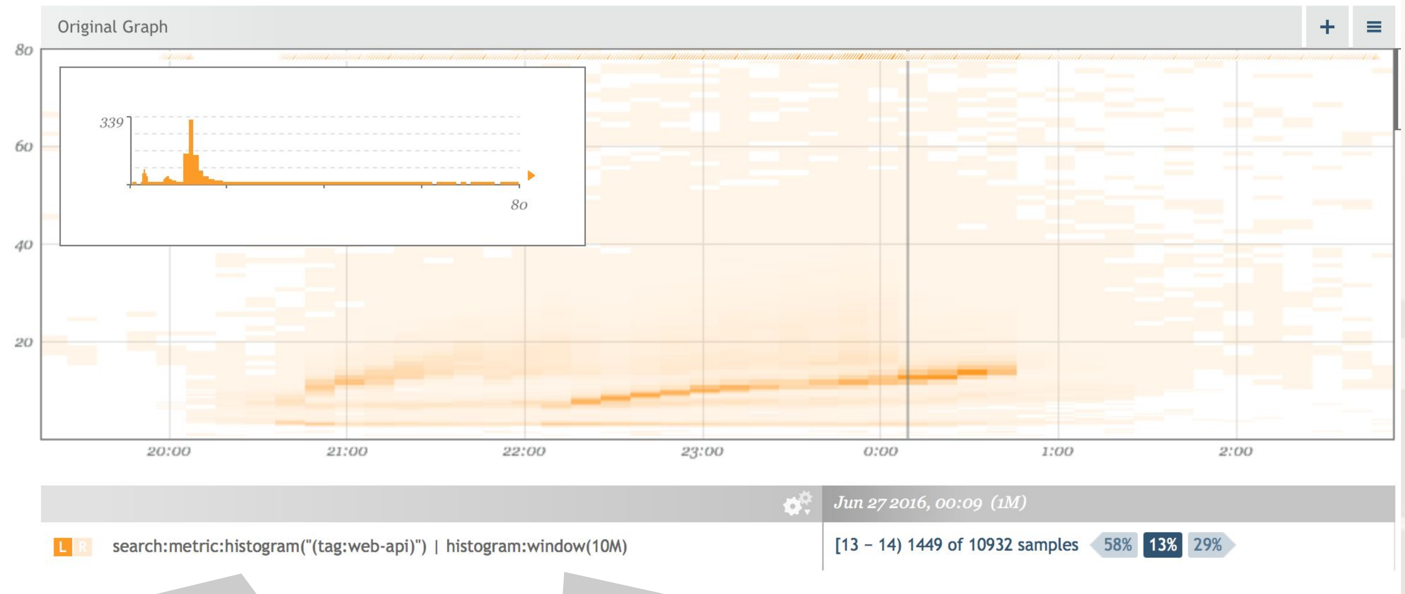
Discussion

- Summary of full RLC, with reduced precision
- Extreme compression compared to logs
- Percentiles, averages, medians, etc. can be derived
- Aggregation across time and nodes trivial
- Allows more meaningful metrics



{5} Histogram Monitoring in Practice

Histograms can be visualized as heatmaps.



search:metric:histogram("(tag:web-api)") | histogram:window(10M)

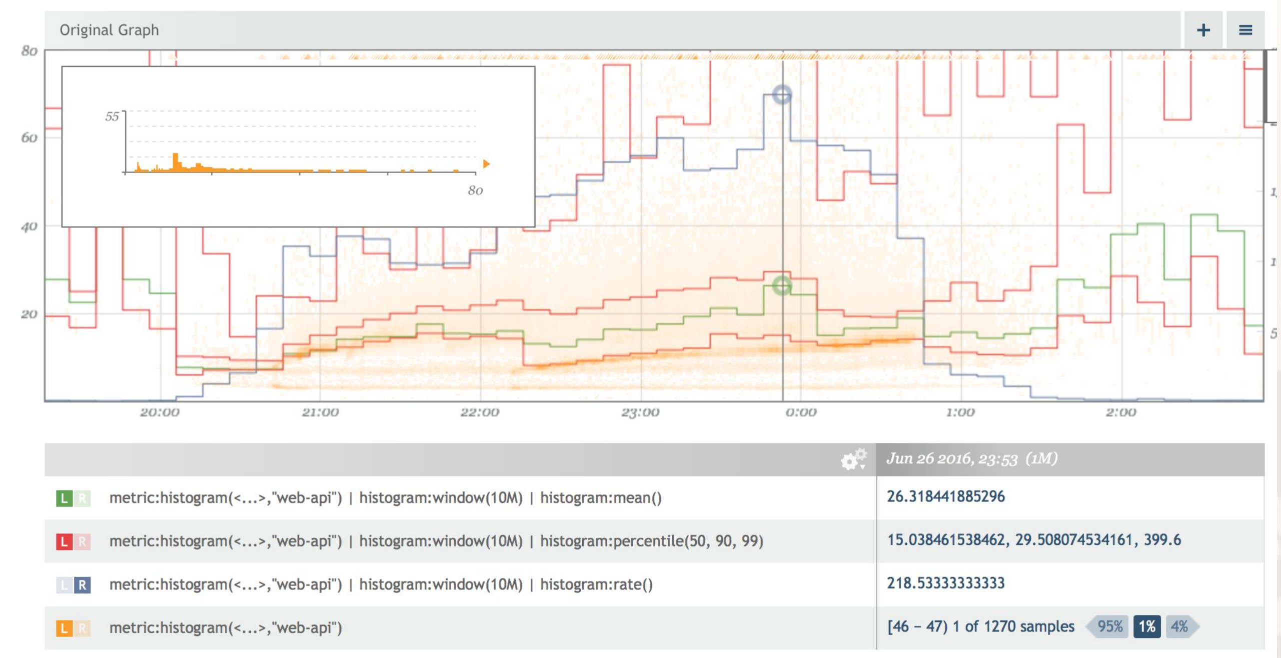
[13 – 14) 1449 of 10932 samples

Aggregate data from all nodes serving “web-api”

.. across windows of 10min.

{5} Histogram Monitoring in Practice

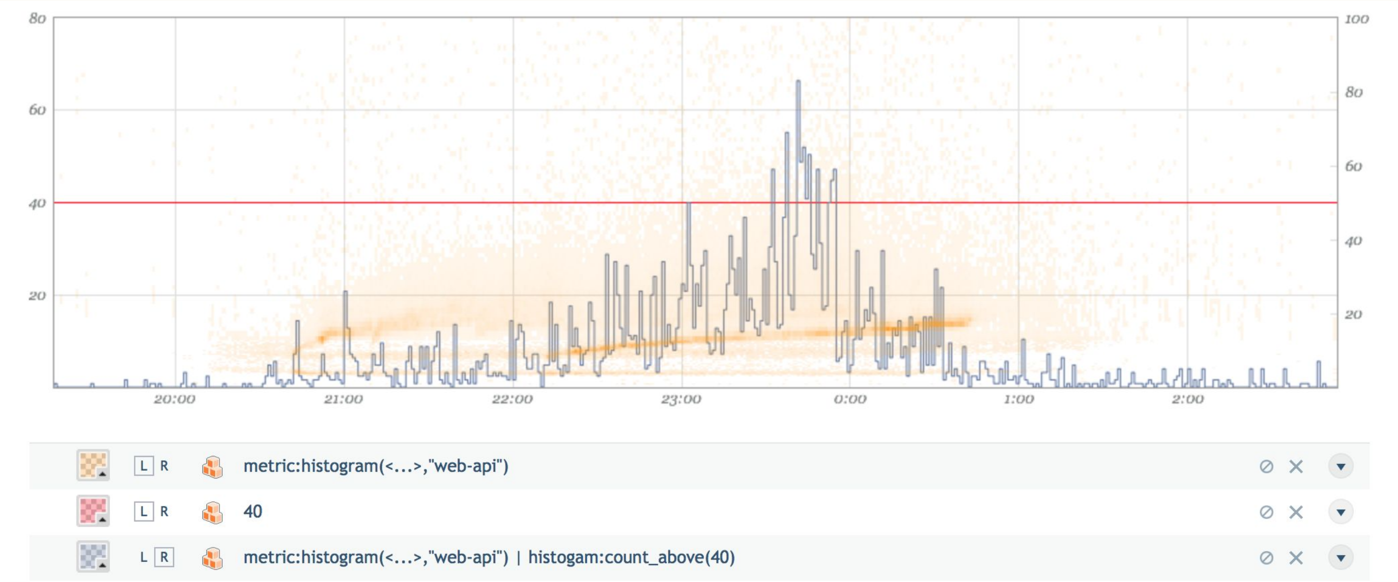
All kinds of metrics can be derived from histograms



		Jun 26 2016, 23:53 (1M)
L R	metric:histogram(<...>,"web-api") | histogram:window(10M) | histogram:mean()	26.318441885296
L R	metric:histogram(<...>,"web-api") | histogram:window(10M) | histogram:percentile(50, 90, 99)	15.038461538462, 29.508074534161, 399.6
L R	metric:histogram(<...>,"web-api") | histogram:window(10M) | histogram:rate()	218.53333333333
L R	metric:histogram(<...>,"web-api")	[46 – 47) 1 of 1270 samples 95% 1% 4%

{6} The search for meaningful metrics

{6} Users offended per minute

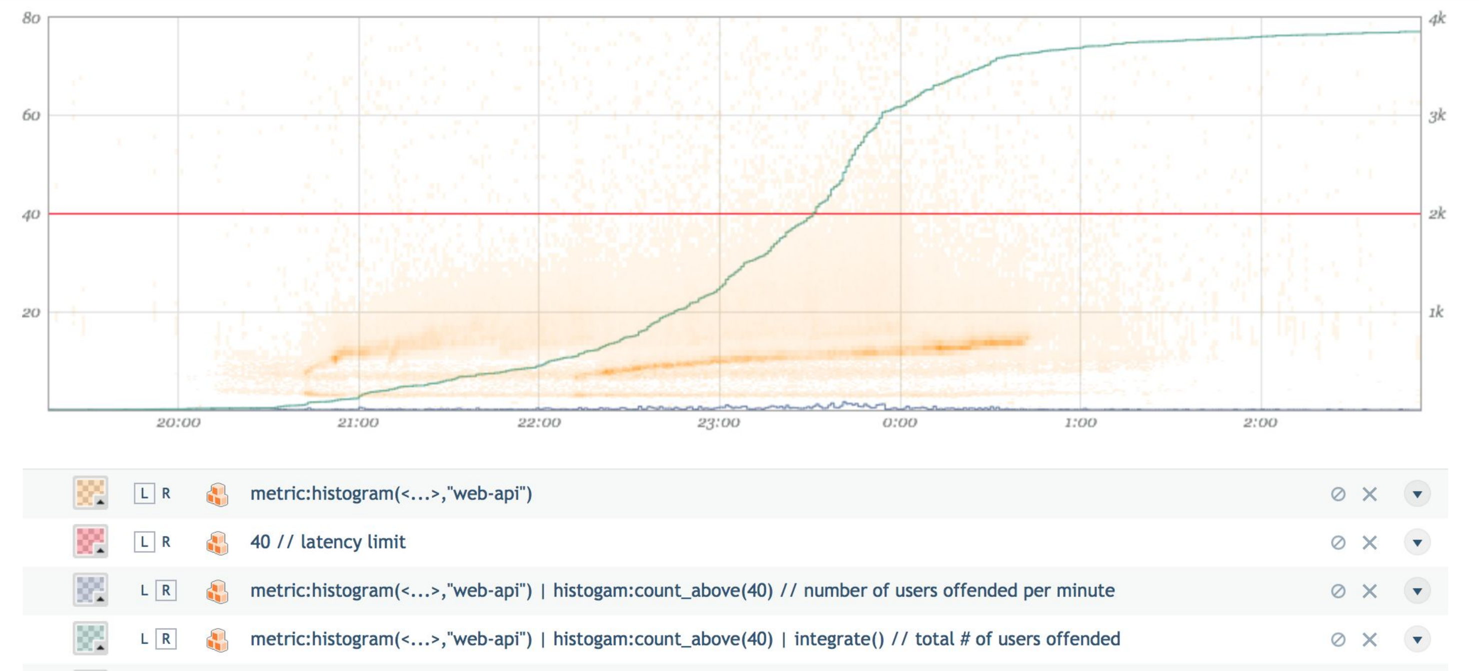


metric:histogram(<…>,"web-api")

40

metric:histogram(<…>,"web-api") | histogam:count_above(40)

{6} Total users offended so far



L R metric:histogram(<...>,"web-api")

L R 40 // latency limit

L R metric:histogram(<...>,"web-api") | histogam:count_above(40) // number of users offended per minute

L R metric:histogram(<...>,"web-api") | histogam:count_above(40) | integrate() // total # of users offended

Takeaways

- Don’t trust line graphs (at least on large scale)
- Don’t aggregate percentiles. Aggregate histograms.
- Keep your data
- Strive for meaningful metrics